Ensemble-Roller: Planning with Ensemble of Relational Decision Trees

Tomás de la Rosa and **Raquel Fuentetaja**
Departamento de Informática, Universidad Carlos III de Madrid
Avda. de la Universidad, 30. Leganés (Madrid). Spain
rfuentet@inf.uc3m.es, trosa@inf.uc3m.es

Abstract

In this paper we describe the ENSEMBLE-ROLLER planner submitted to the Learning Track of the International Planning Competition (IPC). The planner uses ensembles of relational classifiers to generate robust planning policies. As in other applications of machine learning, the idea of the ensembles of classifiers consists of providing accuracy for particular scenarios and diversity to cover a wide range of situations. In particular, ENSEMBLE-ROLLER is a bagging approach to learn ensembles of relational decision trees. The control knowledge from different sets of trees is aggregated as a single prediction which is used to sort candidates in a depth-first search.

Introduction

ENSEMBLE-ROLLER is learning-based planner that learns for each domain a generalized action policy in a similar way as previously done by (Khardon 1999; Martin and Geffner 2000; Yoon, Fern, and Givan 2008; De la Rosa et al. 2011). A generalized policy is a mapping of planning contexts into the preferred actions to apply. The learning of this mapping can be modeled as a relational classification task and solved by inductive learning algorithms and tools.

Given a planning task, we say that an action policy is accurate if it is able to reach goals by repeatedly applying the selected action without reconsidering its decision. Finding an accurate action policy for a domain is a hard task since there can be a wide range of problem distributions and it is difficult to encode all conceivable action selection strategies in a single policy.

In the machine learning community it is well-known that ensemble methods improve the accuracy of single models (Dietterich 2000). The idea of ensemble-based classifiers is to build predictive models by integrating multiple single classifiers. The key to the success of a classifier ensemble is that the base components should be accurate and diverse. Two classifiers can be considered diverse if they perform different on the same data or if they they make different errors when classifying new instances. Regarding action policies for planning, we consider that two policies are diverse if they achieve different plans for the same problem.

ENSEMBLE-ROLLER is an upgrade of the ROLLER planner, for learning ensembles of relational classifiers in order to generate accurate action policies for a wide range of problems in a planning domain. Specifically, we have developed a bagging approach for learning ensembles of relational decision trees. We propose two ways of exploiting the search control knowledge provided by the ensemble of trees. In the first one, individual predictions are aggregated in a single prediction, therefore the resulting action policy is a combination of individual ones. In the second one, individual predictions are considered separately in a multiple-queue search algorithm. However, only the aggregated approach was submitted to the competition.

Decision Tree Learning in Planning

ROLLER (De la Rosa et al. 2011) is a inductive learning system that learns relational decision trees for planning. These decision trees contain control knowledge useful to sort the successors of a given node for state space search planners.

Roller receives as inputs a domain, expressed in PDDL (STRIPS) (Fox and Long 2003) and a set of training problems. Then, it extracts training instances from the search trees generated to solve the training problems. These training instances are used to train TILDE (Blockeel and De Raedt 1998), an off-the-shelf relational classification tool.

Decision trees are binary trees in which each node represents a query about a feature expressed as a positive predicate logic literal. The features considered by ROLLER, that define the *helpful context* of a given search state, are:

- *helpful actions*: whether a particular action is helpful or not at the current state. The notion of helpful actions was first introduced in the FF planner (Hoffmann and Nebel 2001).
- target and achieved goals: whether a goal is pending or has been achieved in the current state.
- static facts, whether a fact is static, i.e. it is defined at the initial state and any action changes it.

ROLLER generates two types of decision trees: operator trees and binding trees. There is one operator tree per domain. The leaves of operator trees provide an order to sort applicable operators at a given state, depending on the features indicated by the path from the root node to the leaf. This order is given by the number of times each operator was selected successfully in training problems for helpful

contexts with the same features. Binding trees are used to sort the instantiations of each operator. There is one binding tree for each operator of the domain. Leaves of binding trees recommend to select or reject an instantiation, considering how many times that binding was chosen when solving previous problems.

Given a search node, ROLLER assigns a priority to each successor, calculated considering the operator tree and the corresponding binding tree. This priority is used to sort successors.[1] Successors with a priority of zero can be considered as non-recommended.

Bagging for Planning

Bagging is a machine learning technique for building ensembles of classifiers through the manipulation of the training set (Breiman 1996). The base learning algorithm is trained k times to obtain k different models. The training set of each of these models consists of m training instances randomly sampled with replacement from the original training set of m examples. Thus, the training set of each iteration contains on average 63.2% of the instances in the original training set, with some of them repeated several times. The standard way of aggregating the prediction of the resulting ensemble of classifiers is by simple voting.

A ROLLER model consists of several decision trees obtained after training on a set of training problems. Bagging can be applied almost directly to obtain an ensemble of ROLLER models. However, there are two decision points that should be solved: how to select training instances and how to combine recommendations.

Regarding the selection of training instances, ROLLER receives as input a set training problems rather than a set of training instances. Thus, there are several alternatives for building the *bootstrap replicates* (i.e., training sets generated for each individual classifier). One possibility is to solve all training problems and to take as the "original training set" the set of all training instances derived from the solved problems. In this case, examples will be randomly drawn with replacement from the original training set, regardless the problem they come from. A second alternative is to consider all instances generated from each training problem as a whole, so the random selection is done at problem level. Instead of selecting instances, the bootstrap problems are built by selecting problems randomly with replacement from the original set of problems. Then, the training instances to learn each individual model are generated from the solutions of the boostrap problems assigned to it. In this work we selected the second alternative. Intuitively, it seems interesting to maintain the notion of problem to generate different ROLLER models more specialized in particular types of problems.

A training phase that applies bagging to ROLLER will end with an ensemble of generalized policies or ROLLER models, each composed by one operator tree and several binding trees, one per operator. To combine these policies in a single planning process that exploits this multiple control knowledge we have developed the Aggregated Bagging Policy, explained in detail in the following section.

[1] Ties are broken arbitrarily.

DT-Filter-Sort-Aggregation (A,$\mathcal{H}$,$\mathcal{B}$): *action list*

A: applicable actions, $\mathcal{H}$: Helpful context, $\mathcal{B}$:ensemble

---

HA $=$ helpful-actions of A
NON-HA $= A \setminus$ *HA*
selected-actions $= \emptyset$
for each a **in** *HA* **do**
  if $opPriority_{\mathcal{B}}(a, \mathcal{H}) > 0$ **then**
    $priority_{\mathcal{B}}(a, \mathcal{H}) = opPriority_{\mathcal{B}}(a, \mathcal{H}) + selRatio_{\mathcal{B}}(a, \mathcal{H})$
    selected-actions = *selected-actions* $\cup \{a\}$
max-HA-priority = $\max_{a \in \textit{selected-actions}} priority_{\mathcal{B}}(a, \mathcal{H})$
for each a **in** *NON-HA* **do**
  if $opPriority_{\mathcal{B}}(a, \mathcal{H}) >$ *max-HA-priority* **then**
    $priority_{\mathcal{B}}(a, \mathcal{H}) = opPriority_{\mathcal{B}}(a, \mathcal{H}) + selRatio_{\mathcal{B}}(a, \mathcal{H})$
    selected-actions = *selected-actions* $\cup \{a\}$
return sort (*selected-actions*, *priority*)

Figure 1: Algorithm for sorting actions for ABP.

Aggregated Bagging Policy

The Aggregated Bagging Policy (ABP) algorithm combines the domain control knowledge (DCK) from an ensemble of decision trees by aggregating their policies into a single generalized policy. ABP uses exactly the same search algorithm as single ROLLER, the H-context Policy algorithm, but now the computation of action priority considers all available ROLLER models (ROLLER bags). H-context Policy performs depth-first search sorting successors by their priority, assigned from ROLLER decision trees.

For a single ROLLER model and a given search node, the priority of each successor is computed in the following way. The current helpful context determines a path to a leaf node in the operator tree. This leaf node associates to each (lifted) action an *operator priority* (*opPriority*) representing the number of covered examples for which this action was the best option in training instances. Given an instantiation of an operator, the current helpful context also determines a path to a leaf node in the corresponding binding tree. This leaf node provides the *selection ratio* (*selRatio*), i.e. the ratio of successful bindings covered by that leaf. The priority of a successor is computed as the sum of its operator priority and selection ratio.

We have adapted the scheme for computing priority of single ROLLER to deal with multiple models. A simple voting could not be a good option since voters are few compared with the number of alternatives. Also, we wanted to maintain the idea of the ROLLER scheme that prioritizes the recommendation of the operator tree over those of binding trees. Thus, for multiple models, the operator priority is computed as the sum of operator priorities for all models. The selection ratio is the overall ratio of successful bindings considering all models, i.e. it is the sum of the number of selected bindings for all models, divided by total number of examples matching the corresponding leaf of the corresponding binding tree. Formally, given an ensemble $\mathcal{B}$ of b ROLLER

bags, and for an applicable action a in a helpful context $\mathcal{H}$, we define:

$$opPriority_{\mathcal{B}}(a, \mathcal{H}) = \sum_{b \in \mathcal{B}} opPriority_b(a, \mathcal{H})$$

$$selRatio_{\mathcal{B}}(a, \mathcal{H}) = \frac{\sum_{b \in \mathcal{B}} selected_b(a, \mathcal{H})}{\sum_{b \in \mathcal{B}} selected_b(a, \mathcal{H}) + rejected_b(a, \mathcal{H})}$$

The complete algorithm for selecting and sorting successors is shown in Figure 1. It receives the set of applicable actions, the current helpful context and the ROLLER ensemble. Then, it returns a sorted list of applicable actions. Priorities are computed using the described equations.

Competition Details

In this section we describe specific details for the ENSEMBLE-ROLLER submitted to the IPC. Planners competing in the learning track are supposed to have a base performance without learning. To have a fair comparison with the acquired domain-knowledge we have set the base algorithm to be depth-first search over the helpful actions sorting successors arbitrarily. Even though this is enough for solving problems in many domains, we expect this configuration will show very bad performance given the large size of the test distributions.

As in ROLLER, training problems for generating DCK should be small enough, so a *BFS+Branch and Bound* algorithm can explore the search space completely. Thus, we have used the random problem generators provided by the organizers to generate appropriate training sets for our system.

The number of bags is a parameter of the planner. Intuitively, a domain will require many bags when problems in this domain present a wide diversity. However, this fact can be only evaluated empirically. Therefore, to select the final number of bags, we have trained ENSEMBLE-ROLLER several times and evaluated its performance in a set of problems of the expected size for the test phase.

The ENSEMBLE-ROLLER search algorithm do not stop at first solution, but continues searching until the time bound, trying to improve the best solution found so far. The search prunes any branch that exceeds the current best cost found. This branch and bound strategy tries to optimize costs, but ENSEMBLE-ROLLER is a system designed to discover the domain and problem structure in terms of predicate logic. Thus, the first solution may present hopeless quality. We expect this to be an important issue in domains where the best decision for a policy depends on the cost structure of problems (e.g., a graph with random arc costs for the navigation in a city).

References

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

Breiman, L. 1996. Bagging predictors. *Machine Learning* 24:123–140.

De la Rosa, T.; Jiménez, S.; Fuentetaja, R.; and Borrajo, D. 2011. Scaling up heuristic planning with relational decision trees. *JAIR* 40:767–813.

Dietterich, T. 2000. Ensemble methods in machine learning. In *1st. International Workshop in Multiple Classifier Systems*.

Fox, M., and Long, D. 2003. PDDL2.1: An extension to PDDL for expressing temporal planning domains. *JAIR* 20:61–124.

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

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

Martin, M., and Geffner, H. 2000. Learning generalized policies in planning using concept languages. In *International Conference on Artificial Intelligence Planning Systems, AIPS00*.

Yoon, S.; Fern, A.; and Givan, R. 2008. Learning control knowledge for forward search planning. *J. Mach. Learn. Res.* 9:683–718.