ECAI 2014
T. Schaub et al. (Eds.)

doi:10.3233/978-1-61499-419-0-1007

Planning with Ensembles of Classifiers

Alberto Garbajosa and **Tomás de la Rosa** and **Raquel Fuentetaja** [1]

Abstract. Learning search control for forward state planning has been previously addressed as a relational classification task, where predictions are used to generate action policies. In this paper, we describe a new bagging approach to learn and apply ensembles of relational decision trees to generate more robust policies for planning. Preliminary experimental results demonstrate that new policies produce on average plans of better quality.

1 INTRODUCTION

In automated planning, machine learning techniques have been used to discover and exploit knowledge about the structure of the planning task that is not explicitly encoded in the domain model. The acquired knowledge can be used to modify the execution of a search algorithm by pruning, sorting or selecting actions. Specifically, one alternative for building learning-based planners is to learn generalized action policies [7, 3]. A generalized policy is a mapping of planning contexts into the preferred actions to apply. The learning of this mapping can be modelled as a relational classification task and solved by inductive learning algorithms. 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 [4]. 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. Regarding policies for planning, we consider that two policies are diverse if they achieve different plans for the same task. In this paper we describe a bagging approach for learning ensembles of relational decision trees, and then we propose two ways of exploiting the control knowledge provided by these models.

2 DECISION TREE LEARNING IN PLANNING

ROLLER [3] is a 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 in PDDL 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 [1], an off-the-shelf relational classification tool. 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. Binding trees are used to sort the instantiations of each operator. There is one binding tree for each domain operator. Leaves of binding trees suggest to select or reject an instantiation. 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] Universidad Carlos III de Madrid, Spain, email: trosa,rfuentet@inf.uc3m.es

3 BAGGING FOR PLANNING

Bagging is a machine learning technique for building ensembles of classifiers through the manipulation of the training set [2]. 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 ensemble of classifiers is by simple voting.

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). In our case, we consider all instances generated from each training problem as a whole, therefore 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 bootstrap problems assigned to it. 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 ROLLER models, each composed by one operator tree and several binding trees, one per operator. We have developed two new strategies to consider these ensembles in the planning phase, namely *Aggregated* bagging policy and *Multiple-Queue* bagging policy.

3.1 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 the priority of each successor is computed in the following way. The context of the current state determines a path to a leaf node

in the operator tree. This leaf node associates to each (ungrounded) action an *operator priority* representing the number of covered training examples. Given an instantiation of an operator, the current context also determines a path to a leaf node in the corresponding binding tree. This leaf node provides the *selection ratio*, 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 the priority of single ROLLER to deal with multiple models. A simple voting might not be a good option since the number of voters is quite small compared to the number of alternatives. Also, we wanted to maintain, as in ROLLER, the scheme where operator trees participate in recommendations with a higher weight than binding trees. Thus, for multiple models, we define the *operator priority* as the sum of operator priorities for all models. The *selection ratio* is defined as the overall ratio of successful bindings considering all models, i.e. the sum of the number of selected bindings for all models, divided by the total number of examples matching the corresponding leaf of the corresponding binding tree. The algorithm for sorting successors receives the set of applicable actions, the context of the current state and the ensemble of trees. Then, it returns a sorted list of applicable actions.

3.2 Multiple-Queue Bagging Policy

The Multiple Queue Bagging Policy (MQBP) algorithm exploits DCK from different ROLLER bags separately. Each bag is used to sort successors in independent open lists, rather than aggregate them into a single value. For this strategy we were inspired by recent search algorithms exploiting several open lists [5, 6], which have shown to be effective for exploring different regions of the search space. The MQBP algorithm performs depth-first search as the H-context Policy algorithm, but it extracts nodes alternatively from different open lists. Each open list is associated to a ROLLER bag. When a node is expanded its successors replace the extracted node in all open lists, but in a different order determined by the priority of single ROLLER for the corresponding bag. Successors are not included in open lists not containing the expanded parent node. As a result, the different open lists contains a fairly similar set of states, but structured in different search trees, providing diversity to the search. If the algorithm does not backtrack, the plan found with MQBP is guaranteed to be the shortest one of those found when using just one bag. However, any sort of backtracking will lead to a "parallel" exploration of different state space regions sorted by different ROLLER bags, which provide diversity to the search.

4 EXPERIMENTAL EVALUATION

We have evaluated the performance of the two bagging approaches considering training and testing phases with 2, 3, 5 and 8 bags. The benchmarks for the evaluation were the domains from the Learning Track of IPC-2011 (*International Planning Competition*). Experiments were run on a 2.93Ghz machine CPU with 7.5Gb of RAM.

Training: For each domain we have generated a set of 50 training problems with the random generators provided by the IPC-2011 organizers. These problems were solved with a time bound of 120 seconds. The random selection of problems for bagging only distributes training problems on different bags. Therefore, it is not necessary to solve them again for different distributions. With this regard, it makes no sense to measure learning times because the remaining time after solving training problems is consumed by the learning algorithm (TILDE) and it is proportional to the number of bags.

Table 1. Quality scores obtained by different policies.

	HP	ABP				MQBP			
		b2	b3	b5	b8	b2	b3	b5	b8
blocks	10.0	1.6	18.1	20.8	**27.8**	0.9	19.7	16.4	22.8
depots	7.7	2.0	0.3	1.6	2.9	3.3	1.8	**10.6**	4.1
gripper	29.4	29.8	29.8	**29.9**	29.4	29.4	29.2	29.4	27.6
parking	26.1	24.4	26.2	23.8	23.9	24.5	25.9	26.0	**27.4**
satellite	27.2	**27.8**	23.5	4.0	2.9	20.5	11.5	2.6	2.7
rovers	4.8	25.3	21.7	23.7	**25.4**	24.7	24.4	24.6	22.3
spanner	**30.0**	**30.0**	**30.0**	**30.0**	**30.0**	**30.0**	**30.0**	20.0	15.0
tpp	16.3	18.3	21.9	13.2	10.6	20.1	22.9	**24.3**	17.2
Total	151	158	**172**	147	153	153	165	154	139

Testing: For the evaluation we compared the H-context Policy (HP) algorithm with the Aggregated Bagging Policy (ABP) and the Multiple Queue Bagging Policy (MQBP) algorithms, each one configured with the different number of bags. Each domain has a test set of 30 problems. The time bound to solve a problem was set to 900 seconds. The generation of the ensembles of classifiers implies randomness in the evaluation process, therefore we have executed each configuration for 5 times. For each problem we have considered the median of the plan length and its associated CPU time.

Table 1 shows the quality scores of each configuration following the same scoring of IPC-2011. The barman domain is omitted since no problems were solved by any configuration. For the rest of domains, always one or more bagging configurations improve the quality score of HP. However, there is no clear dominance between ABP or MQBP, not even between the number of bags. Our results support empirically the idea that these ensembles of relational classifiers improve the behavior of single classifiers. But to some extend, the right number of bags depends on the domain and problem structure, since here we are facing the *utility problem*. For instance, the spanner domain has a simple *key knowledge* (e.g., pick all spanners on the way), so there is no classifier diversity. Adding more bags only increases the tree matching time and then it degrades the overall performance, as shown for MQBP (b5 and b8). On the other hand, blocksworld problems have multiple tower layouts, therefore single classifiers are only accurate in some problems. Diversity is important for this domain, and higher numbers of bags tend to improve the performance.

On average, MQBP wastes more time than ABP in solving the same problems. This is not surprising due to the overload of handling multiple lists. Also, as for HP, the heuristic evaluation for computing ROLLER contexts has a great impact in the total time. The tree matching extra time is worth when additional bags provide more diversity to the current knowledge.

Acknowledgment: This work has been partially supported by the Spanish project TIN2011-27652-C03-02.

REFERENCES

[1] Hendrik Blockeel and Luc De Raedt, 'Top-down induction of first-order logical decision trees', *AI Journal*, **101**(1-2), 285–297, (1998).
[2] Leo Breiman, 'Bagging predictors', *Machine Learning*, **24**, 123–140, (1996).
[3] Tomás De la Rosa, Sergio Jiménez, Raquel Fuentetaja, and Daniel Borrajo, 'Scaling up heuristic planning with relational decision trees', *JAIR*, **40**, 767–813, (2011).
[4] Thomas Dietterich, 'Ensemble methods in machine learning', in *1st. International Workshop in Multiple Classifier Systems*, (2000).
[5] Malte Helmert, 'The fast downward planning system', *JAIR*, **26**, 191–246, (2006).
[6] Gabriele Röger and Malte Helmert, 'The more, the merrier: Combining heuristic estimators for satisficing planning', in *ICAPS*, (2010).
[7] Sungwook Yoon, Alan Fern, and Robert Givan, 'Learning control knowledge for forward search planning', *JMLR.*, **9**, 683–718, (2008).