CABALA: Case-based State Lookaheads

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

Abstract

We present in this work the CABALA system, a learning-based planner submitted to the first learning track of the International Planning Competition, (IPC-6). CABALA uses a case-based strategy for ordering applicable actions of the relaxed plan when lookahead states are built during the search algorithm. The lookahead strategy is one of the techniques used to speed up heuristic planning when relaxed plans of some domains are of good quality or at least contain most of the actions that belong to the solution plan. However, this technique has only been used in variations of best-first search and it has not been exploited probably because applying actions of the relaxed plan in a wrong order leads to generate lookahead states that do not help the search process. In this work we enhance the lookahead technique using sequences of abstracted state-transitions as an advise for building lookahead states. Previous work has shown that these sequences are useful for some benchmarks since they allow to reduce the number of evaluations of the heuristic function. We argue that the same CBR advise used for ordering nodes can be used for ordering actions in the lookahead state construction.

Introduction

At the sixth edition of the International Planning Competition (IPC-6) a learning track has been proposed to encourage the AI Planning community to develop different techniques that can improve state-of-the-art in planning. Although domain-independent planning has achieved considerable improvements in the last decade, IPC-3 showed that domain-dependent planners can significantly outperform the current domain-independent techniques. The main issue with domain-dependent knowledge is how to automatically acquire this knowledge, so a planning system can improve its performance after learning from experience. In this work we present the basic ideas of CABALA , one of the participants of the IPC-6 Learning Track. CABALA uses a case-based strategy developed in (De la Rosa, García-Olaya, & Borrajo 2007) in order to decide the ordering of applicable actions from the relaxed plan when lookahead states are generated. The lookahead strategy introduced in the YAHSP planner (Vidal 2004) uses the actions in the relaxed plan to compute reachable states in order to speed up the search process. This improvement is due to the reduction of the real depth of the search tree which produces as a result less evaluations of the heuristic function. Since the computations of the heuristic function spend most of the total planning time, the lookahead strategy improves planning in terms of search time. On the other hand, in some domains, applying actions of the relaxed plan in a wrong way can produce useless lookahead states that do not help in the target of decreasing the number of evaluations of the heuristic function. CABALA produces a different construction of lookahead states, because it tries to execute the action that replays the same abstracted state transition of the learned sequence retrieved according to a CBR approach (Aamodt & Plaza 1994). We will present in this paper the system overview and a summary of techniques and algorithms used for the version submitted to the competition.

CABALA Overview

CABALA is part of the set of learning-based planners integrated with SAYPHI, a heuristic planner developed with the aim of building different learning strategies on top of it, using a similar philosophy architecture addressed within PRODIGY (Veloso *et al.* 1995). For the version submitted to the Learning Track, the CABALA system includes:

- **SAYPHI Planner**: A collection of heuristic search algorithms that use an implementation of heuristic function of the FF planner (Hoffmann & Nebel 2001). The collection of algorithms includes the Lookahead-BFS algorithm used as the default one for CABALA
- **CBR Advisor**: A module for storing, merging and retrieving domain control knowledge in the form of typed sequences, as used in (De la Rosa, García-Olaya, & Borrajo 2007).
- **Incremental Trainer**: An extension of the CBR Advisor for generating the domain-dependent knowledge in a incremental way using a given training set.

In the following section we will explain how the CBR Advisor works and how the learned knowledge is used in Lookahead-BFS.

CBR advisor overview

In CABALA, the domain-specific knowledge is stored as cases, called *typed sequences*. A typed sequence is an abstracted sub-state transition relative to an object type. A typed sequence is formed by an ordered list of pairs (typed sub-state, action to reach the state) which partially collects a planning episode from an object instance perspective. So, we say that this type of knowledge is object-centered or type-centered. A typed sub-state is the set of all properties that an object has in a particular state. A property, first introduced in the work of domain analysis in TIM (Fox & Long 1998), is defined as a predicate subscripted with the object position of a literal; e.g., on_1 is a property of object *blockA* in the literal *(on blockA blockB)*. In addition, an object sub-state is the set of the state literals in which the object is present. Then, the set of object properties that forms the typed sub-state is computed from the object sub-state. For instance, suppose we have the state s_1 =[*(on blockA blockB) (ontable blockB) (clear blockA) (holding blockC)* ...]. Then, the object sub-state of *blockA* would be [*(on blockA blockB) (clear block A)*]. This is generalized to $(on_1\ clear_1)$ which is a typed sub-state of type *block*. If the action *stack(blockC, blockA)* is applied in s_1, the new object sub-state for *blockA* would be [*(on blockA blockB) (on blockC blockA)*], thus the typed sub-state becomes $(on_1\ on_2)$.

The main idea of a CBR cycle is that past experience is stored in a case base and when a new problem needs to be solved, the most similar case is retrieved. Then, the retrieved case is adapted and reused to solve the new problem. CABALA obtains the experience from solved problems as follows: once a problem is solved, for each object instance in the problem, a typed sequence is generated. Each step in the sequence is created with the corresponding object sub-state from the solution path. If we consider $U(S, o)$ the transformation function that obtains the typed sub-state (properties of object o) from the state S, and a plan $P = \{a_1, \ldots a_n\}$, then we generate a typed sequence as $Q = (U(o, S_0), \emptyset), \ldots, (U(o, S_n), a_n)$ where state S_i is the resulting state of applying action a_i to state S_{i-1}. Sequences are grouped in the case base by domain types, so a new case is inserted in the type of object from which it was generated. If the object sub-state does not change when an action is applied, a `no-op` is saved to represent a void action from the object perspective. A merge process verifies that equivalent sequences are not repeated in the case base. We consider that two sequences are equivalent when they only vary in the number of `no-op`. Figure 1 shows a complete example of a typed sequence generated from a solution plan in the blocksworld domain. The typed sequence has one step more because the typed sub-state of the initial state is included without any action.

The next step in the CBR cycle is the retrieval. A typed sequence is retrieved for each object instance in the new problem. For this purpose we have a retrieval scheme that only considers the first step of the sequence referred to the initial state and the last step of the sequence referring to the goal state. The system performs two matches to retrieve a sequence. The first one matches the typed sub-state generated from the goals of the new problem, G, against the last step of all sequences of the corresponding type. For the second match the typed sequence generated from the initial state of the new problem, I is matched against the first step of the sequences resulting from the first match. Thus, if we consider an object o of type t and $Q = \{(q_0, \emptyset), \ldots, (q_n, a_n)\}$ an arbitrary typed sequence in the case base of type t, the first match holds when $U(G, o) \subseteq q_n$, and the second match holds when $q_0 \subseteq U(I, o)$.

Initial State:	(ontable A) (clear A)
	(ontable B) (clear B)
	(ontable C) (clear C)
Goals:	(on A B) (on C A)

Plan	Typed Sequence (Block A)
initial state	$[(clear_1\ ontable_1),\ nil]$
0: (PICKUP A)	$[(holding_1),\ pickup]$
1: (STACK A B)	$[(clear_1\ on_1),\ stack]$
2: (PICKUP C)	$[(clear_1\ on_1)$, `no-op`$]$
3: (STACK C A)	$[(on_1\ on_2),\ stack]$

Figure 1: A typed sequence example in the blocksworld domain.

Typed sequences are used during the search as control knowledge that supports exploring decisions together with the heuristic function. The search control is performed with recommendations given by the retrieved sequences, which are replayed while the search is advancing. We say that successor node is recommended when it matches the next step of a sequence retrieved for any object involved in the applied action to reach the node state. Thus, we say that a child node S' achieved by applying action a' is a recommended node when $q_{k+1} \subseteq U(S, o_j)$ where o_j is the parameter (object instance), j of the action a' and k is the current step number for the sequence retrieved for o_j. The notion of *recommended node* is independent of the search algorithm, so it can be used for different search control tasks like selecting, pruning or ordering nodes. In CABALA these recommendations guide the generation of lookahead states as explained in a further section.

Incremental Trainer

The incremental trainer is an extension to the CBR Advisor developed for the CABALA version submitted to the competition. The underlying idea is that using subsets of problems from the training set, we iteratively build case bases that are validated with a test set. If one iteration improves a previous iteration, the subset of problems are considered useful and this subset is used to build the final case base.

The algorithm in Figure 2 shows the pseudo-code for the incremental trainer. CABALA uses a training and a test set in order to perform the incremental learning. This may correspond to the bootstrap and the target distributions of the problems given for the competition. The np parameter is the number of training problems used for each iteration and *split_k* is a constant for building additional problems (explained in detail in the next section). Function `populate_case_base` is a call to the CBR Advisor when it solves a set of problems and generates the case base from these problems. The function `quality_metric` computes the proposed quality metric for the competition using the test set. At the end of the algorithm the final case base is built with the set of problems that belongs to iterations that

```
Cabala-Training ($train\_set, test\_set, np, split\_k$)

$train\_set, test\_set$: training and test sets.
$np$: number of problems per iteration.
$split\_k$: constant for splitting phase.

$inc\_train\_set \leftarrow$ prepare_split($train\_set, split\_k$)
$best\_score = 0$; $learned\_set = \emptyset$
while there are new problems in $inc\_train\_set$ do
    $iter\_set \leftarrow$ next $np$ problems in $inc\_train\_set$
    populate_case_base($iter\_set \cup learned\_set$)
    if quality_metric($test\_set, iter\_set \cup learned\_set$) $> best\_score$ then
        $learned\_set \leftarrow iter\_set \cup learned\_set$
        $best\_score \leftarrow$ quality_metric($test\_set$)
populate_case_base($learned\_set$)
```

Figure 2: Algorithm for incremental training of the case-base.

improved the overall quality metric.

With the aim of having sufficient training problems, some problems of the training set can be split into some new problem. Thus, we try to guarantee that at least a distribution of small problems can be solved with a good plan quality, so the stored cases are of good quality. The *split_k* parameter represents the estimated heuristic distance for solving one goal, and its value is used for building the new problems. The constant determines the number of goals of the new problems, and the number of problems built as shown in the next equations, where I is the initial state and G the goals of the seeding problem.

$$num\_new\_goals = (split\_k * |G|)/h_{\mathrm{FF}}(I) \quad (1)$$

$$num\_new\_problems = |G|/num\_new\_goals \quad (2)$$

The function `prepare_split` in the incremental trainer code builds the list of problems to be solved. It adds first all problems in the training set and then, it generates new problems for those problems that the heuristic value of the initial state exceeds the *split_k*. The set of goals of a new problem are randomly selected from the set of goals of the seeding problem.

Case-based Lookahead Search

In this section we will explain the algorithm used by CABALA in the competition. It is based on a modification of a best-first search (BFS) in which each node expansion generated a lookahead state that is inserted at the beginning of the open list. Due to scalability reasons we skip the rest of node evaluations since in previous experiments we saw that preferring the lookahead state could lead to a solution in most benchmarks. Thus, we define an artificial evaluation function as:

$$f(n) = \begin{cases} 0 & \text{if } n \text{ is a lookahead node} \\ depth(n) & \text{otherwise} \end{cases} \quad (3)$$

We could substitute the function $depth(n)$ by $h_{\mathrm{FF}}(n)$ for using a standard heuristic evaluation of nodes. This substitution in particular produces analogous behavior of the BFS lookahead search of YASHP (Vidal 2004) in domains in which it is always possible to build a lookahead node from an arbitrary state.

CABALA focuses on the quality metric of the competition, so after finding a solution, we continue to search for better solutions pruning nodes that exceed the best cost found so far. Thus, CABALA algorithm could be interpreted as lookahead BFS with a Branch & Bound technique for refining solutions.

Rather than trying to apply as most applicable actions as CABALA can, the idea of case-based lookahead is to decide in which order the applicable actions of the relaxed plan may be applied.

Figure 3 shows the algorithm for computing case-based lookahead nodes. From the applicable actions of the relaxed plan, CABALA selects the action matching more cases. Then, applicable actions are recomputed and another action is selected an so on. In the standard algorithm for computing lookahead states, actions are inserted into the applicable list as they appear, but in the case-based approach, the selected applicable action must match one or more sequences in their current step. The implemented algorithm keeps track of visited states to avoid generating lookahead nodes with repeated states. The arg variable represents the problem objects used as parameters of the applicable actions.

```
Case-base Lookahead State ($S, RP$)

$S$: the current state.
$RP$: relaxed plan of $S$.

$lh \leftarrow S$
while ($C \leftarrow$ applicable($lh, RP$)) $\neq \emptyset$ do
    for each $c$ in $C$ do
        for each $arg$ in $c$ do
            if apply($c, lh$) matches case_pointer($arg$) then
                matching[$c$] = matching[$c$] + 1
    $next \leftarrow argmax$(matching[$i$]), $i \in C$
    $lh \leftarrow$apply($c, lh$)
return $lh$
```

Figure 3: Algorithm for computing case-based lookahead states

Conclusions and Future Work

We have presented the CABALA system, a competitor in the first version of a Learning Track within IPC. Our system was submitted focusing on two main goals.

- Showing a CBR approach as an alternative technique for ordering applicable actions of the relaxed plan in a lookahead search.
- Verifying the possibility of finding refined solutions after the first solution is found, using a branch and bound strategy, also guided by CBR recommendations.

We are currently working on some extensions to the system. We have noticed that in some domains EHC with lookahead states obtains better results than our current algorithm. This confirms that case-based recommendations are suitable for different algorithms, but on the other hand, this reveals us that we need to research on algorithm diversity to determine a more robust algorithm for the most available benchmarks.

Acknowledgments

This work has been partially supported by the Spanish MEC project TIN2005-08945-C06-05. We want to thank Planning and Learning Group members that helped us in any kind of work related to the IPC Learning Track.

References

Aamodt, A., and Plaza, E. 1994. Case-based reasoning: Foundational issues, methodological variations, and system approaches. *AI Communications* 7, no.1:39–59.

De la Rosa, T.; García-Olaya, A.; and Borrajo, D. 2007. Using cases utility for heuristic planning improvement. In Weber, R., and Richter, M., eds., *Case-Based Reasoning Research and Development: Proceedings of the 7th International Conference on Case-Based Reasoning*, 137–148. Belfast, Northern Ireland, UK: Springer Verlag.

Fox, M., and Long, D. 1998. The automatic inference of state invariants in TIM. *Journal of Artificial Intelligence Research* 9:317–371.

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

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

Vidal, V. 2004. A lookahead strategy for heuristic search planning. In *Proceedings of the Fourteenth International Conference on Automated Planning and Scheduling*, 150–160.