Machine learning of plan robustness knowledge about instances

Sergio Jiménez, Fernando Fernández[⋆], and Daniel Borrajo

Departamento de Informática
Universidad Carlos III de Madrid
Avda. de la Universidad, 30. Leganés (Madrid). Spain
sjimenez@inf.uc3m.es, fernando@cs.cmu.edu, dborrajo@ia.uc3m.es

Abstract. Classical planning domain representations assume all the objects from one type are exactly the same. But when solving problems in the real world systems, the execution of a plan that theoretically solves a problem, can fail because of not properly capturing the special features of an object in the initial representation. We propose to capture this uncertainty about the world with an architecture that integrates planning, execution and learning. In this paper, we describe the PELA system (Planning-Execution-Learning Architecture). This system generates plans, executes those plans in the real world, and automatically acquires knowledge about the behaviour of the objects to strengthen the execution processes in the future.

1 Introduction

Suppose you have just been engaged as a project manager in an organization. The organization staff consists of two programmers, **A** and **B**. Theoretically **A** and **B** can do the same work, but probably they will have different skills. So it would be common sense to evaluate their work in order to assign them tasks according to their worthy.

In this example, it seems that the success of fulfilling a task depends on which worker performs which task, that is how actions are instantiated, rather than depending on what state the action is executed. The latter would be, basically, what reinforcement learning does in the case of MDP models (fully instantiated states and actions). It does not depend either on the initial characteristics of a given instance, because the values of these characteristics might not be known "a priori". Otherwise, they could be modelled in the initial state. For instance, one could represent the level of expertise of programmers, as a predicate `expertise-level(programmer,task,prob)` where `prob` could be a number, reflecting the uncertainty of the `task` to be carried out successfully by the `programmer` . Then, the robustness of each plan could be computed by cost-based planners. So, we would like to acquire knowledge only about the uncertainty associated to instantiated actions, without knowing "a priori" the facts from the

[⋆] Fernando Fernández is currently with the Computer Science Department Carnegie Mellon University, funded by a MEC-Fulbright grant state that should be true in order to influence the execution of an action. Examples of this type of situations arise in many real world domains, such as project management, workflow domains, robotics, etc. We are currently working in the domain of planning tourist visits. In this domain, we want to propose plans that please the tourist as much as possible, and we have to deal with the uncertainty about which is the best day to visit a place.

With an architecture that integrates planning, execution and learning we want to achieve a system that is able to learn some knowledge about the effects of actions execution, but managing a rich representation of the action model. Thus, the architecture can be used for flexible kinds of goals as in deliberative planning, together with knowledge about the expected future reward, as in Reinforcement Learning [1]. A similar approach is followed in [2], but they learn the operators (actions models), while our goal is to acquire heuristics (as control knowledge) to guide the planner search towards more robust solutions. In a classical AI setting, our approach tries to separate the domain model that might be common to many different problems within the domain, from the control knowledge that can vary over time. And we propose to gradually and automatically acquire this type of control knowledge through repeated cycles of planning, execution and learning, as it is commonly done in most real world planning situations by humans.

2 The Planning-Execution-Learning Architecture

The aim of the architecture is to automatically acquire knowledge about the objects behaviour in the real world to generate plans whose execution will be more robust. Figure 1 shows a high level view of the proposed architecture.

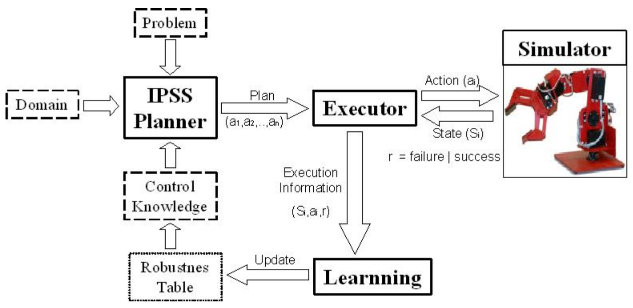

Fig. 1. High level view of the planning-execution-learning architecture.

To acquire this knowledge, the system begins with a deterministic knowledge of the world dynamics and observes the effects that the execution of its actions causes in the real world. The system registers whether an action execution is successful or not in the real world. So it has information about the possibility of succeed on executing an action in the real world. This is what we call the *robustness* of an action. To use this information, the system defines control knowledge that decides the instantiation of the actions. So, the planner will choose the best bindings for the actions according to the acquired robustness information. We have developed a preliminary prototype of such architecture, that we call PELA (Planning-Execution-Learning Architecture). To make this prototype come true, we have made the following assumptions (we describe how we plan to relax them in the future work section).

1. A domain consists of a set of operators or actions, and a set of specific instances that will be used as parameters of the actions. This is something relatively different from the way in which planning domains are handled, since they usually do not include specific instances, which appear in the planning problems. This assumption is only needed for learning and it is not really needed for deterministic planning purposes.
2. As we are working in a preliminary prototype, the robustness of the execution of plan actions only depends on the instantiation of the actions parameters, and it does not depend on the states before applying the actions.

2.1 Planning

For the planning task we have used the nonlinear backward chaining planner IPSS [3]. The inputs to the planner are the usual ones (domain theory and problem definition), plus declarative control knowledge, described as a set of control rules. These control rules act as domain dependent heuristics. They are the main reason we have used this planner, given that they provide an easy method for declarative representation of automatically acquired knowledge [4]. IPSS planning-reasoning cycle involves as decision points: select a goal from the set of pending goals and subgoals; choose an operator to achieve a particular goal; choose the bindings to instantiate the chosen operator and apply an instantiated operator whose preconditions are satisfied or continue subgoaling on another unsolved goal. The output of the planner, as we have used it in this paper, is a total-ordered plan.

2.2 Execution

The system executes step by step the sequence of actions proposed by the planner to solve a problem. When the execution of a plan step is a failure the execution process is aborted. To test the architecture, we have developed a module that simulates the execution of actions in the real world. This simulator module receives an action and returns whether the execution succeeded or failed. It is very simple for the time being as it doesn't take care of the current state of the world. The simulator keeps a probability distribution function as a model of execution for each possible action. When the execution of an action has to be simulated, the simulator generates a random value following its corresponding distribution probability. If the generated random value satisfies the model, the action is considered successfully executed.

2.3 Learning

The process of acquiring the knowledge can be seen as a process of updating the robustness table. This table registers the estimation of success of an instantiated action in the real world. It is composed of tuples of the form `<op-name, op-params, r-value>` `op-name` is the action name, `op-params` is the list of the instantiated parameters and `r-value` is the robustness value. In the planning tourist visits domain, as we want to capture the uncertainty about which is the best day for a tourist to visit a fixed place we register the robustness of the operator `PREPARE-VISIT` with the parameters `PLACE` and `DAY`. An example of the robustness-table for this domain is Table 1.

Action	**Parameters**	**Robustness**
prepare-visit	(PRADO MONDAY)	5.0
prepare-visit	(PRADO TUESDAY)	6.0
prepare-visit	(PRADO WEDNESDAY)	8.0
prepare-visit	(PRADO THURSDAY)	4.0
prepare-visit	(PRADO FRIDAY)	2.0
prepare-visit	(PRADO SATURDAY)	1.0
prepare-visit	(PRADO SUNDAY)	1.0
prepare-visit	(ROYAL-PALACE MONDAY)	2.0
prepare-visit	(ROYAL-PALACE TUESDAY)	2.0
	...	

Table 1. An example of a Robustness-Table for the planning tourist visits domain.

We update the robustness value of the actions using the learning algorithm shown in Figure 2. According to this algorithm [5], when the action execution is successful, we increase the robustness of the action, but if the action execution is a failure, the new robustness value is the square root of the old robustness value.

```
Function Learning (ai, r,Rob-Table):Rob-Table
ai: executed action
r: execution outcome (failure or success)
Rob-Table: Table with the robustness of the actions
if r=success
    Then
        robustness(ai,Rob-Table) = robustness(ai,Rob-Table) +1
    Else
        robustness(ai,Rob-Table) = $\sqrt{robustness(ai, Rob - Table)}$
Return Rob-Table;
```

Fig. 2. Algorithm that updates the robustness of one action.

2.4 Exploitation of acquired knowledge

Control rules guide the planner among all the possible actions, choosing the action bindings with the greatest robustness value in the Robustness Table. In the planning tourist visits domain, these control rules will make the planner prefer the most `'robust'` day to prepare the visit of the tourist `<user-1>` to the place `<place-1>`. An example of these control rules is shown in Figure 3. Suppose that a tourist called Mike wants to visit the Prado museum, the system will decides to prepare the visit to the Prado museum on Wednesday, among all the possible instantiations, because `PREPARE-VISIT PRADO WEDNESDAY 8.0` is the tuple with the greatest robustness value in the Robustness-Table for the PRADO. To achieve a balance between exploration and exploitation the system only use the control rules in 80% of the times.

```
(control-rule prefer-bindings-prepare-visit
 (IF
   (and
          (current-goal (prepared-visit <user-1> <place-1>))
          (current-operator prepare-visit)
          (true-in-state (current-time <user-1> <day-1> <time-1>))
          (true-in-state (current-time <user-1> <day-2> <time-2>))
          (diff <day-1> <day-2>)
          (more-robustness-than
               (list 'prepare-visit <place-1> <day-1>)
               (list 'prepare-visit <place-1> <day-2>))))
(THEN prefer bindings ((<day> . <day-1>))((<day> . <day-2>))))
```

Fig. 3. Control rule for preferring the best day to visit a museum

3 Experiments and Results

The experiments carried out to evaluate the proposed architecture have been performed in the planning tourist visits domain. The used domain is a simplification of the SAMAP project domain [6], but given that this is preliminary work, we have left out the part of path planning. We assume a tourist is always able to move from any zone in the city to another. The operators in this domain are `MOVE`, `VISIT-PLACE` and `PREPARE-VISIT`. In order to test the system we have developed a simulator that emulates the execution of the planned actions in the real world. The simulator decides whether visiting a place is a failure or not. It decides with probability 0.1 that the visit was successfully in Mondays, Tuesdays, Wednesdays or Thursdays. And with probability 0.5 when the visit happens on Fridays, Saturdays or Sunday.

We have used a test problem set with 100 random generated problems with different complexity. The state of the random problems represents the free time of the tourist for each day in the week, its available money and its initial location. The problem goals describe the places the tourist wants to visit. We measure the complexity of the problems in terms of the available time the user has to visit all the goals. For that purpose we have defined the following ratio:

```
complexity = goals-time / available-time
```

Where goals-time represents the time needed to visit all the goals and available-time represents the sum of the tourist free time. So, when a problem have a complexity ratio over 1.0 the planner will not be able to find a solution.

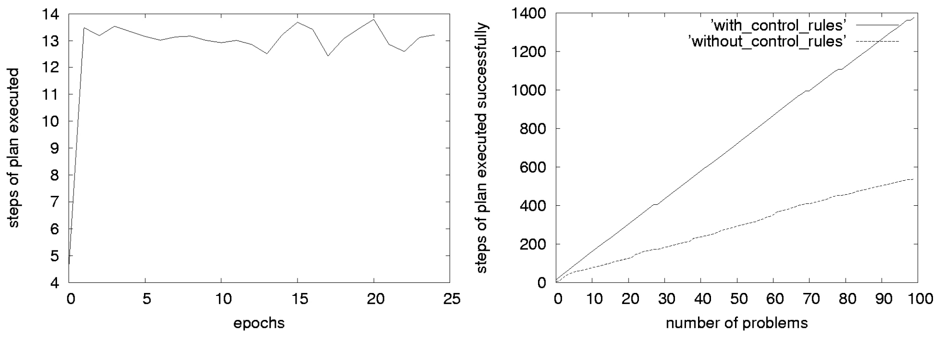

Fig. 4. Steps of plan successfully executed in the planning tourist visits domain.

The first graph in figure 4 shows the evolution of the learning process. This graph presents the number of successfully executed actions for 25 epochs. This number converges quickly to approximately 13-14 steps. The average length of the plans that solve the problems from the test set is 19,5. So, in terms of percentage, 13-14 steps executed succesfully represents approximately 66-72% percentage of plan executed succesfully. The fast convergence is because of failure and success probabilities for an action don't change with time. The second graph in figure 4 compares the behaviour of our system to the behaviour of a system that does not use the learned knowledge in the planning process. The number of successful actions is computed after 25 epochs of learning with a ten random problems train set.

4 Related Work

Learning to plan and act in uncertain domains is an important kind of machine learning task. Most of literature in the field separates this task in two different phases: A first phase to capture the uncertainty and a second phase to plan dealing with it.

1. A first phase when the uncertainty is captured, [2] propose to obtain the world dynamics by learning from examples representing action models as probabilistic relational rules. A similar approach was previously used in

propositional logic in [7]. [8] proposes using Adaptive Dynamic Programming, this technique allows reinforcement learning agents to build the transition model of an unknown environment whereas the agent is solving the Markov Decission Process through exploring the transitions.

2. A second phase when problems are solved using planners able to handle actions with probabilistic effects. This kind of Planning is a well studied problem [9]. We can also include in this second phase the systems that solve Markov Decision Processes. The standard Markov Decission Process algorithms seek a policy (a function to choose an action for every possible state) that guarantees the maximum expected utility. So, once the optimal policy is found planning under uncertainty can be considered as following the policy starting from the initial state [10].

So, as our system propose the integration of these two phases, it presents several differences with the previous systems:

– Our system does not learn a probabilistic action model, the system starts with a deterministic description of the actions. Then it explores the environment not to learn the whole world dynamics but to complete the domain theory.
– We don't assume completely the object abstraction as we are interested in domains where the execution of an action depends on the identity of the instances rather than on their type.
– Our system uses the learnt information about instances as control knowledge so it keeps separately the domain model from the control knowledge.

We have also found another architecture that integrates planning, executing and learning to in a similar way. [11] interleaves high-level task planning with real world robot execution and learns situation-dependent control rules from selecting goals to allow the planner to predict and avoid failures. The main differences between this architecture and ours, are that: we don't learn control rules, control rules are part of the initial domain representation, what we learn is the robustness of the actions. And we don't guide the planner choosing the goals but choosing the instantiations of the actions.

5 Future work

We plan to remove, when possible, the initial assumptions, mentioned in the introduction section. Relaxing the first assumption requires generating robustness knowledge with generalized instances and then mapping new problems instances to those used in the acquired knowledge. As we described in the introduction section, we believe this is not really needed in many domains, since one always has the same instances in all problems of the same domain. In that case, we have to assure that there is a unique mapping between real world instances and instance names in all problems. When new instances appear, their robustness values can be initialized to a specific value, and then gradually be updated with the proposed learning mechanism. To relax the second assumption, we will use a more complex simulator that considers not only the instantiated action, but also the state before applying each action. We are planning to test the system with the simulator and the domains of the probabilistic track of the International Planning Competition[1]. Thus, during learning, the reinforcement formula should also consider the state where it was executed. One could use standard reinforcement learning techniques [12] for that purpose, but states in deliberative planning are represented as predicate logic formulae. One solution would consist on using relational reinforcement learning techniques [13].

And finally, for the time being the learning algorithm and the exploration-exploitation strategy we use are very simple, both of them must be studied deeper [14] in order to obtain better results in more realistic domains.

References

1. Kaelbling, L.P., Littman, M.L., Moore, A.W.: Reinforcement learning: A survey. Journal of Artificial Intelligence Research **4** (1996) 237–285
2. Pasula, H., Zettlemoyer, L., Kaelbling, L.: Learning probabilistic relational planning rules. In: Proceedings of the Fourteenth International Conference on Automated Planning and Scheduling. (2004)
3. Rodrguez-Moreno, M.D., Borrajo, D., Oddi, A., Cesta, A., Meziat, D.: Ipss: A problem solver that integrates planning and scheduling. Third Italian Workshop on Planning and Scheduling (2004)
4. Veloso, M., Carbonell, J., Pérez, A., Borrajo, D., Fink, E., Blythe, J.: Integrating planning and learning: The PRODIGY architecture. Journal of Experimental and Theoretical AI **7** (1995) 81–120
5. Nareyek, A.: Choosing search heuristics by non-stationary reinforcement learning (2003)
6. Fernández, S., Sebastiá, L., Fdez-Olivares, J.: Planning tourist visits adapted to user preferences. Workshop on Planning and Scheduling. ECAI (2004)
7. Garca-Martnez, R., Borrajo, D.: An integrated approach of learning, planning, and execution. Journal of Intelligent and Robotic Systems **29** (2000) 47–78
8. Barto, A., Bradtke, S., Singh, S.: Real-time learning and control using asynchronous dynamic programming. Technical Report, Department of Computer Science, University of Massachusetts, Amherst (1991) 91–57
9. Blythe, J.: Decision-theoretic planning. AI Magazine, Summer (1999)
10. Koening, S.: Optimal probabilistic and decision-theoretic planning using markovian decision theory. Master's Report, Computer Science Division University of California, Berkeley (1991)
11. Haigh, K.Z., Veloso, M.M.: Planning, execution and learning in a robotic agent. In: AIPS. (1998) 120–127
12. Watkins, C.J.C.H., Dayan, P.: Technical note: Q-learning. Machine Learning **8** (1992) 279–292
13. Dzeroski, S., Raedt, L.D., Driessens, K.: Relational reinforcement learning. Machine Learning **43** (2001) 7–52
14. Thrun, S.: Efficient exploration in reinforcement learning. Technical Report C,I-CS-92-102, Carnegie Mellon University (1992)

[1] http://ipc.icaps-conference.org/