Capturing knowledge about the instances behavior in probabilistic domains

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, ffernand@inf.uc3m.es, dborrajo@ia.uc3m.es

Abstract. When executing plans in real world, a plan that theoretically solves a problem, can fail because of special features of an object were not properly captured in the initial domain representation. We propose to capture this uncertainty about the world repeating cycles of planning, execution and learning. In this paper, we describe the Planning, Execution and Learning Architecture (PELA) that generates plans, executes those plans using the simulator of the International Planning Competition, and automatically acquires knowledge about the behaviour of the objects to strengthen future execution processes.

1 Introduction

Suppose you have just been engaged as a project manager in an organization and you are in charge 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. This is basically, what reinforcement learning does in the case of mdp models (managing with fully instantiated states and actions). It also 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 modeled in the initial state. And the robustness of each plan could be computed by cost-based planners. 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.

With an architecture that integrates planning, execution and learning [1] we want to achieve a system able to learn some knowledge about the success of actions execution, but also, able to manage a rich representation of the action

* Fernando Fernández is currently working with the Computer Science Department Carnegie Mellon University, funded by a MEC-Fullbright gant

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 [2].

Already exist previous works that learn information about the domain theory from the executing action in real world, such as [3] and [4], but we think our approach is different as we do not pretend to learn actions models but to learn useful heuristics to choose the best operators and instances to strengthen future execution processes.

In the last International Planning Competition(IPC)[1], a probabilistic track took place for the first time. In this track, a simulator of probabilistic worlds was developed to evaluate the participant planners. We think this simulator is also a valid framework for testing and evaluating other architectures proposal that aim to act and plan in realistic domains. In this paper we will describe our Planning, Exection and Learning Architecture and how we have integrated the IPC simulator in it.

2 Capturing operators and instances behavior

We would like to capture knowledge about the uncertainty associated to instantiated actions, assuming we don't have "a priori" any knowledge about the special features of the objects that affects to the success in executing an actions.

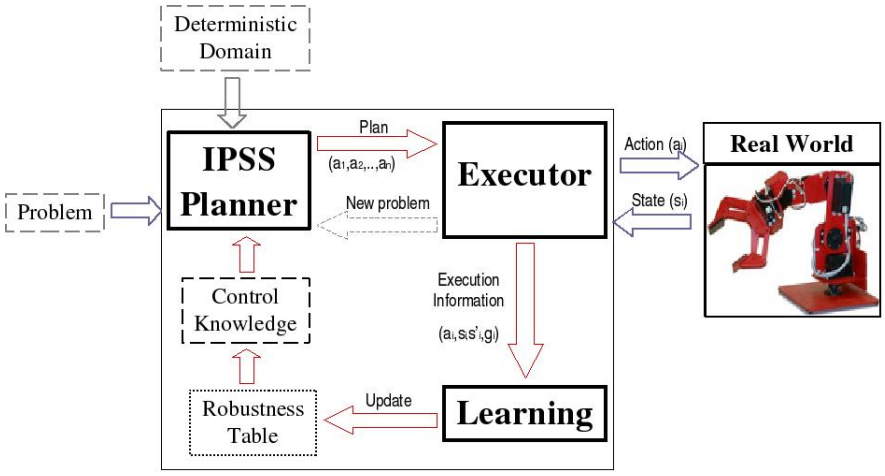

Fig. 1. High level view of the planning-execution-learning architecture.

To learn this knowledge, the system starts with a deterministic description of the world dynamics and plans using this description to solve problems. Once

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

it has found a plan that theoretically would solve the problem, it tries to execute it in the real world action by action. After every execution of an action the system observes the new state of the real world and learns. It learns whether the execution of the action was a success or a failure, that is, whether the effects caused the execution of the action in the real world are the theoretically expected or not. The learnt information is stored in a table. This table that we call the *robustness table*, registers an estimation of the probability of succeed on executing the actions in the real world. Table 1 is an example of a *robustness table* for the blocksworld domain. To use this information, the system defines control knowledge that decides the instantiation of the actions. So, when the planner decide which binding should use for a given action, it will choose the best one according to the acquired robustness knowledge. Figure 1 show the proposed planning-execution-learning architecture.

Table 1. An example of a *Robustness Table* for a two operators and three blocks blocksworld domain.

Action	**Parameters**	**Robustness**
pick-up-block-from	(block0 table)	0.945
pick-up-block-from	(block1 table)	0.654
pick-up-block-from	(block2 table)	0.83
put-down-block-on	(block0 table)	0.2534
put-down-block-on	(block1 table)	0.744
put-down-block-on	(block2 table)	0.753
pick-up-block-from	(block1 block0)	0.43
pick-up-block-from	(block2 block0)	0.85
put-down-block-on	(block1 block0)	0.36
put-down-block-on	(block2 block0)	0.42
pick-up-block-from	(block0 block1)	0.43
pick-up-block-from	(block2 block1)	0.85
put-down-block-on	(block0 block1)	0.36
put-down-block-on	(block2 block1)	0.154
pick-up-block-from	(block0 block2)	0.27
pick-up-block-from	(block1 block2)	0.45
put-down-block-on	(block0 block2)	0.32
put-down-block-on	(block1 block2)	0.265

2.1 Planning

For the planning task we have use the nonlinear backward chaining planner IPSS [5]. 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, and they are the main reason we have used this planner, given that they provide an

easy method for declarative representation of automatically acquired knowledge. 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 planner is executed with control rules that make the planner prefer the more robust bindings for an action in order to guide the planner towards solutions that guarantee successful execution of plans according to the acquired knowledge on robustness. The output of the planner, as we have used it in this paper, is a total-ordered plan.

2.2 Execution

The executor module receives the sequence of actions proposed by the planner to solve a problem and executes it step by step in the real world. When the execution of an action in the real world is a failure (the new state is different from the state expected according to the deterministic domain), the executor stops and the planner develops a new plan for solving the problem in the new situation.

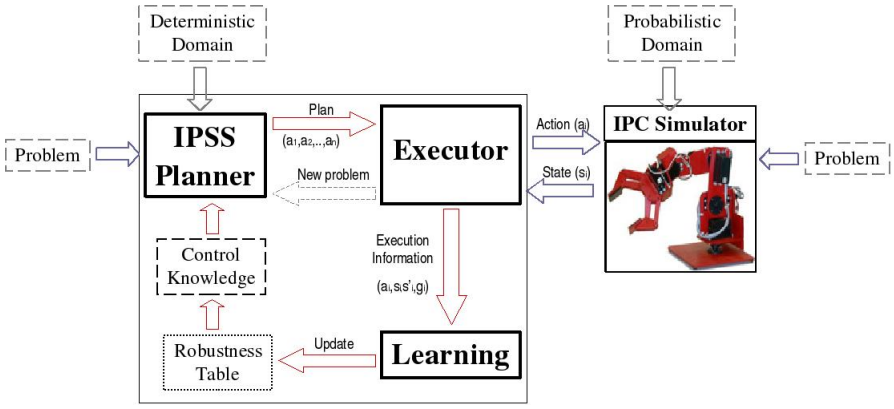

Fig. 2. High level view of the planning-execution-learning architecture.

To test the architecture, instead of executing the actions in the real world we simulate them. Specifically, we have used the simulator provided by the International Planning Competition to evaluate probabilistic planners. This simulator uses PPDDL 1.0 [6] to describe the world we want to simulate. This language allows to describe actions with probabilistic and conditional effects. It is based on PDDL 2.1.

The task of this simulator in our architecture is to keep a representation of the current state and to update it when it receives an action from the executor module. Figure 2 shows how the IPC simulator is integrated in our architecture.

2.3 Learning

The learning process can be seen as a process of updating the *robustness table*. In this table the estimation of success of an instantiated actions in the real world is registered using tuples of the form `(op-name, op-params, r-value)`. Where `op-name` is the action name, `op-params` is the list of the instantiated parameters and `r-value` is the robustness value.

In this work, as the probabilities of the actions we want to estimate don't vary with time, a very simple statistical algorithm is enough to update the *robustness table*. In this case robustness value of an action symbolizes the frequency of the action successful executions. If the probabilities of the actions effects would vary with time a more complex learning strategy will be needed, such as

$$robustness(t+1) = \alpha * robustness(t) + (1-\alpha) * robustness(t-1)$$

to give more significance to recent happenings.

3 Experiments and Results

The experiments carried out to evaluate the proposed architecture have been performed in the domain of the blocksworld belonging to the probabilistic track of the International Planning Competition.

In the first experiment, we have tried to solve twenty five five-blocks problems randomly generated with the programs of the IPC. In this case the operator `pick-up-block-from` will success on picking up a block with probability `0.75` and `put-down-block-on` that success on putting down a block with probability `0.75`. The first graph (Figure 3) shows the evolution of the mean quadratic error of our estimation of the probability of successful execution of the two operators.

For the second experiment, as the previous actions model were very simple, we have modified it in order to get a domain a bit more complex where the success of the actions depends on the instantiation of the operators. If we try to `pick up` or to `put down` the `block0` we success `90%` of the times, if we try to `pick up` or to `put down` the `block1` we success `80%` of the times, if we try to `pick up` or to `put down` the `block2` we success `70%` of the times, if we try to `pick up` or to `put down` the `block3` we success `60%` of the times, and finally if we try to `pick up` or to `put down` the `block4` we success `50%` of the times. Again, we have tried to solve twenty five random problems with five blocks, the second graph (Figure 4) shows the evolution of the mean quadratic error of our estimation of the probability of success execution of the actions.

In both experiments the system achieve very low values of mean quadratic error so the system can achieve a good estimation of the robustness of the actions. In the second experiment the error reduction is slower than in the first.

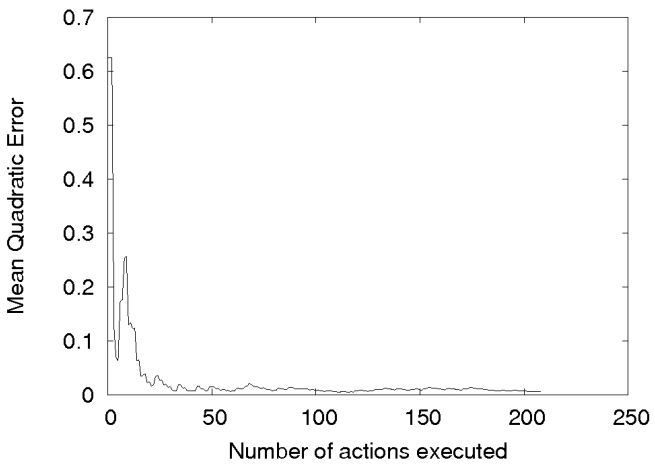

Fig. 3. Evolution of the mean quadratic error of our estimation of the probability of success execution of the operators `pick-up-block-from` and `put-down-block-on`

That is because in the first experiment we are only estimating the robustness of two operators whereas in the second one we are estimating probabilities for 10 different actions.

4 Related Work

Trying to plan in the real world is a very complex task, so to solve problems in realistic domains , normally it is useful to make some assumptions that make easier the handling of the problem. Usually these assumptions simplify the way of perceiving the environment, the way the environment can be modified, and the way of defining the goals to reach. According to this assumptions normally there is a *full observability* of the current state, the actions executions are the only way of changing the current state (*static word*), these executions always cause the same effects in the environment (*deterministic actions*), and goals are sets of states, thus, solving a problem consists on building a plan that reaches one of these states (*reachability goals*).

We are interested in works that try to relax the *deterministic actions* assumption. Two communities have been mainly working in this field with two different approaches: One community consists of Markov decision process (MDP) [7] researchers interested in developing algorithms that apply to powerfully expressive representations of environments. And the other consists of planning researchers incorporating probabilistic and decision theoretic concepts into their planning algorithms [8].

Both approaches suppose we have at our disposal the exact probabilities of every action effect. But in fact, there are very few real domains where this happens.

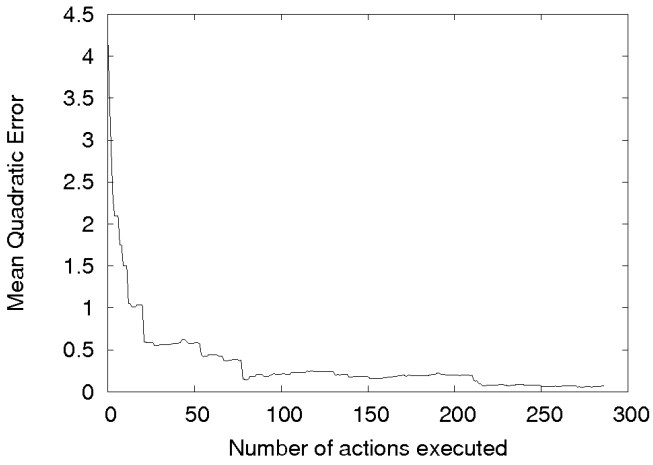

Fig. 4. Evolution of the mean quadratic error of our estimation of the probability of success execution of the instantiated actions

So it is necessary a previous phase where the systems learns these probabilities. In [3] it is proposed 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 [9]. In [10] it is proposed to used 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 Decision Process through exploring the transitions.

Our approach presents an integrated architecture that as the same time it plans it also discovers the behavior of the objects of the world. We have also found another architecture that integrates planning, executing and learning in a similar way [11]. This architecture also 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 difference between this architecture and ours is that we don't pretend to guide the planner choosing goals but choosing the instantiations of the actions.

5 Conclusions and future work

In this paper we have presented an architecture that, automatically acquires knowledge about the uncertainty associated to instantiated actions, assuming we don't have "a priori" any knowledge about the special features of the objects that cause the failure when executing actions.

The carried out experiments show us that the system can learn the probabilities of success of the instantiated actions in the proposed domain.

From a critical point of view, our approach present scaling limitations. As we try to learn the robustness of every instantiation of the operators, it will not scale well when the number of objects in the world is too big. This problem also happens in others machine learning algorithms such as Reinforcement Learning. But in case of planning we can easily think in domains where the set of the objects in the world is limited, such as all the domains of the probabilistic track in the International Planning Competition.

In this work we have only considered the failure or success of an action execution, so we have interpreted the robustness as a probability of success. A pending extension to this work for future efforts, is working in domains where actions are not instantaneous, have duration, and we can interpret the robustness value as a duration or (a quality) value that depends on the instantiation of the action.

References

1. Jimnez, S., Fernndez, F., Borrajo, D.: Machine learning of plan robustness knowledge about instances. Proceedings of the 16th European Conference on Artificial Intelligence (2005)
2. Kaelbling, L.P., Littman, M., More, A.: Reinforcement learning: A survey. Journal of Artificial Intelligence Research (1996)
3. Pasula, H., Zettlemoyer, L., Kaelbling, L.: Learning probabilistic relational planning rules. Proceedings of the Fourteenth International Conference on Automated Planning and Scheduling (2004)
4. Zettlemoyer, L., Pasula, H., Kaelbling, L.: Learning planning rules in noisy stochastic worlds. proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI-05) (2005)
5. 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)
6. Younes, H.L.S., Littman, M.L.: Ppddl1.0: An extension to pddl for expressing planning domains with probabilistic effects. Technical Report CMU-CS-04-167, School of Computer Science, Carnegie Mellon University, Pittsburgh, Pennsylvania. (2004)
7. Boutilier, C., Dean, T., Hanks, S.: Planning under uncertainty: structural assumptions and computational leverage. Journal of Artificial Intelligence Research (1998)
8. Blythe, J.: Decision-theoretic planning. AI Magazine, Summer (1999)
9. Garca-Martnez, R., Borrajo, D.: An integrated approach of learning, planning, and execution. Journal of Intelligent and Robotic Systems **29** (2000) 47–78
10. 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
11. Haigh, K.Z., Veloso, M.M.: Planning, execution and learning in a robotic agent. AIPS (1998) 120–127