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 the 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 (PEL) Architecture 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. As you "a priori" don't know them, it would be common sense to evaluate their work in order to assign them tasks according to their worth. So, when you have to decide how to assign tasks to the programmers, you will only take into account which worker performs which task, that is how actions are instantiated because the values of their skills are unknown. Otherwise, they could be modeled 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 probability of the task to be carried out successfully by the programmer.

Traditionally, two communities have been working in the field of planning in environments where there is incomplete or faulty information: One community consists of Markov Decision Process researchers [1] interested in developing algorithms that apply to powerfully expressive representations of environments. And the other community consists of planning researchers incorporating probabilistic and decision theoretic concepts into their planning algorithms [2]. Both approaches suppose we have at our disposal a representation of the action model

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

with the exact probabilities of every action effect. But in fact, there are very few real domains where this happens. So it is necessary to have a preceding phase where the systems learn 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 [4]. In [5] it is proposed to used Adaptive Dynamic Programming, which 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.

With an architecture that integrates planning, execution and learning [6] we want to achieve a system able to capture some uncertainty about the success of action execution, as Reinforcement Learning [7] does, but also able to manage 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 effects.

2 Capturing instances behavior

The Planning, Execution and Learning architecture assumes there is no "a priori" knowledge about the special features of the objects that can influence the success in executing actions, and tries to capture this uncertainty repeating cycles of planning, execution and learning. Figure 1 show a high level view of the Planning, Execution and Learning Architecture.

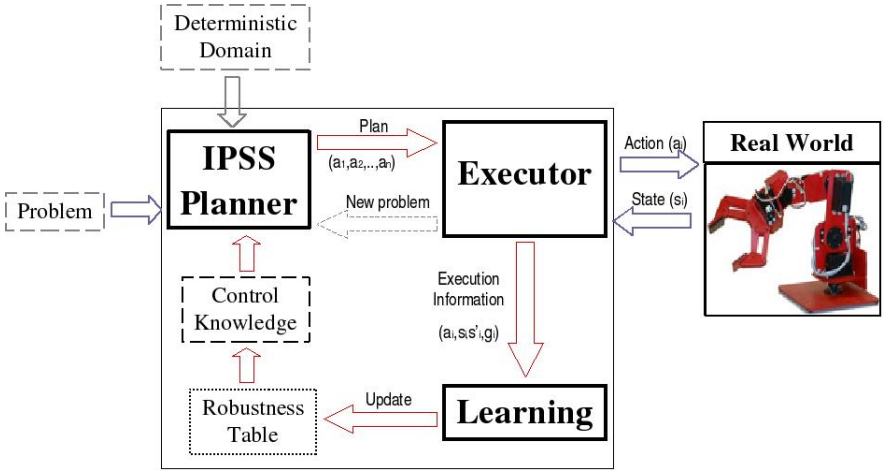

Fig. 1. High level view of the planning-execution-learning architecture.

The proposed architecture starts with a deterministic description of the world dynamics and plans to solve problems using this description. Once 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 success in executing an action in the real world. Table 1 is an example of a *robustness table* for a three blocks and two operators 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

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

To use this information, the system automatically generates control knowledge that decides the instantiation of the actions. So, when the planner has to decide which binding to use for a given action, it will choose the best one according to the acquired robustness knowledge.

2.1 Planning

For the planning task we have used the non-linear backward chaining planner IPSS [8]. 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. The 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 totally-ordered plan.

2.2 Execution

The executor module receives the sequence of actions proposed by the planner to solve a problem and tries to execute it step by step. When the execution of an action in the real world is a failure, the executor will wait till the planner develops a new plan for solving the problem in the new situation (replaning). An action execution is considered a failure when it causes a new state different from the expected state according to the deterministic domain and moreover the proposed plan is not yet valid to reach the solution from this new state.

2.3 Learning

The learning process lies in the updating of the *robustness table* (table 1). This table registers the estimation of the probability of success in executing instantiated actions in the real world. Each register of the table is a tuple of the form `(op-name, op-params, r-value)`, where `op-name` is the name of the action, `op-params` is the list of the instantiated parameters and `r-value` is the robustness value. So, the number of different instantiated actions determines the size of the *robustness table*. This number can be computed:

$$Instances = \sum_{i=1}^{N} \prod_{j=1}^{arguments(i)} instances(i,j)$$

where `N` is the number of operators, `arguments(i)` is the number of arguments of the operator `i` and `instances(i,j)` is a function that computes the number of different values that the argument `j` can take in the operator `i`.

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, the robustness value of an action symbolizes the frequency of successful executions of the action. If the probabilities of the action's effects would vary with time, a more complex learning strategy will be needed, such as

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

where α means the significance of the recent executions.

3 Experiments and Results

The final aim of the carried out experiments is to analyze the correction of the estimation of the action robustness we get using the proposed architecture. To test the architecture, we simulate the actions instead of executing them in the real world. Specifically, we have used the simulator provided by the last International Planning Competition (IPC)[1], to evaluate probabilistic planners. This simulator uses PPDDL 1.0 [9] to describe the world we want to simulate. This language based on PDDL 2.1., allows us to describe actions with probabilistic and conditional effects. The task of this IPC simulator in our architecture is to keep a representation of the simulated current state and to update it when it receives an action from the executor module. Figure 2 shows how the IPC simulator is integrated into our architecture.

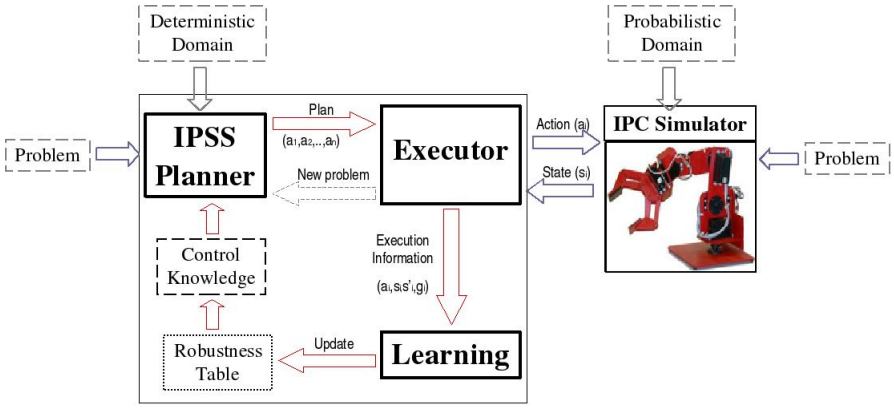

Fig. 2. High level view of the planning-execution-learning architecture integrated with the simulator of the IPC probabilistic track.

We have performed the experiments in the blocksworld domain from the probabilistic track of the IPC, we have modified the probabilities of the effects of the operators `pick-up` in order to get a domain a bit more complex where the frequency of success of the action depends on the instantiation of the operators. Thus, each block has a different behavior when they are picked-up or put-down. In these experiments, our system will try to solve problems of different complexity. Our system will use a deterministic domain description where all the blocks are initially the same and will try to estimate the probability of success in executing the actions learning from its own experience. To evaluate the correction of the estimation of the action robustness we will measure the mean quadratic error between the estimation of our system and the real values.

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

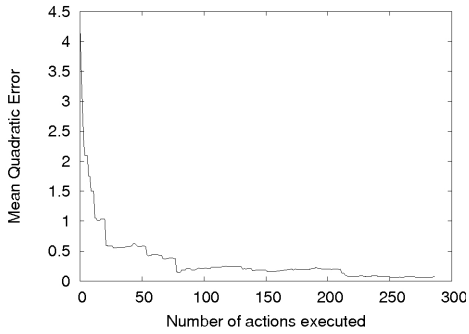

Fig. 3. Evolution of the error in estimating actions robustness with 5 different blocks

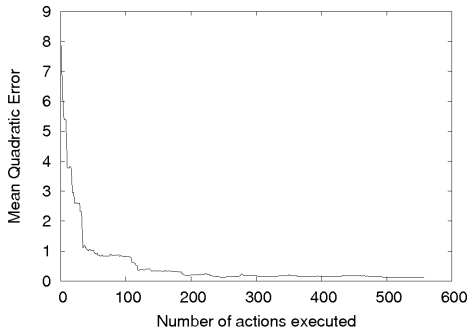

Fig. 4. Evolution of the error in estimating actions robustness with 8 different blocks

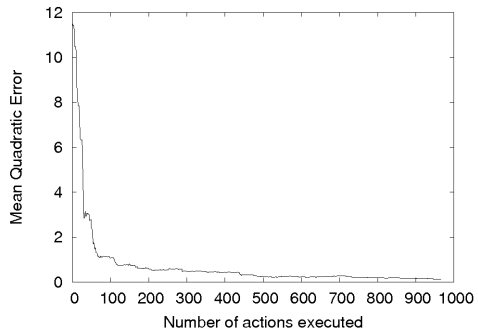

Fig. 5. Evolution of the error in estimating actions robustness with 11 different blocks

Figure 3 shows the evolution of the mean quadratic error of our estimation as the system executes actions to solve 25 random problems with 5 blocks. Figure 4, shows how this error evolves when we have 8 blocks with a different behavior each. And Figure 5 shows how this error evolves when the system executes actions to solve 25 random problems with 11 blocks. Each block with a different behavior when it is picked-up or put-down.

In all the experiments the system achieves very low values of mean quadratic error so the system can achieve a good estimation of the robustness of the actions. But as the number of blocks is increased, the learning process is much slower. That is because we increase the number of different instantiated actions so the system has to learn a greater amount of concepts.

4 Related Work

There are another architectures that integrate planning, executing and learning to act in domains with incomplete information: In [10] is described an automaton

that is able to move in an unknown deterministic environments by learning the desirability of its actions. This approach uses a very simple planning strategy to decide future actions, so it is not able to manage a rich representation of the action model and the goals. [11] describes an architecture that 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 their architecture and ours is that we guide the planner in choosing the instantiations of the actions rather than in choosing goals.

5 Conclusions and future work

In this paper we have presented an architecture that automatically acquires knowledge about the uncertainty associated with instantiated actions, assuming we have no "a priori" knowledge about the special features of the objects that cause the failure when executing actions.

The experiments show 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. So this would be a problem in really big domains. This problem also happens in other machine learning techniques such as Q-Learning [12], and in other current planning algorithms such as GraphPlan [13] or FF [14] that need a whole instantiation of the domain. Otherwise we are working in capturing the special behaviour of instances using a state based representation and trying to induce the similar features of the objects

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 but have duration. Then we can interpret the robustness value as a duration or (a quality) value that depends on the instantiation of the action.

6 Acknowledgments

This work has been partially supported by the Spanish MCyT under project TIC2002-04146-C05

References

1. Boutilier, C., Dean, T., Hanks, S.: Planning under uncertainty: structural assumptions and computational leverage. Journal of Artificial Intelligence Research (1998)
2. Blythe, J.: Decision-theoretic planning. AI Magazine, Summer (1999)

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. Garca-Martnez, R., Borrajo, D.: An integrated approach of learning, planning, and execution. Journal of Intelligent and Robotic Systems **29** (2000) 47–78
5. 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
6. Jimnez, S., Fernndez, F., Borrajo, D.: Machine learning of plan robustness knowledge about instances. Proceedings of the 16th European Conference on Artificial Intelligence (2005)
7. Kaelbling, L.P., Littman, M., More, A.: Reinforcement learning: A survey. Journal of Artificial Intelligence Research (1996)
8. 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)
9. 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)
10. Doran, J.: Experiments with a pleasure-seeking automaton. Machine Intelligence 3 (1968) 195–216
11. Haigh, K.Z., Veloso, M.M.: Planning, execution and learning in a robotic agent. AIPS (1998) 120–127
12. Watkins, C.J.C.H., Dayan, P.: Technical note: Q-learning. Machine Learning **8** (1992) 279–292
13. Blum, A., Furst, M.: Fast planning through planning graphs analysis. Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI) (1995) 1636–1642
14. Hoffmann, J.: Ff:the fast forward planning system. AI Magazine, 22(3) (2001) 57–62