Combining ILP and Parameter Estimation to Plan Robustly in Probabilistic Domains

Sergio Jiménez[1] and James Cussens[2]

[1] Departamento de Informática
Universidad Carlos III de Madrid
Avda. de la Universidad, 30. Leganés (Madrid). Spain
sjimenez@inf.uc3m.es
[2] Department of Computer Science
University of York
Heslington, York YO10 5DD
jc@cs.york.ac.uk

Abstract. When developing planning applications, designing a domain theory to describe a realistic problem is not an easy task. Since the outcome of actions is usually not deterministic, it is likely that 'a priori' the preconditions or the effects of actions are unknown or hard to fully specify. Even if we are able to specify this information, the outcome of actions could vary over time. In this paper we describe how to combine ILP and parameter estimation to automatically acquire probabilistic information for a planning domain thus simplifying the process of generating realistic planning applications.

1 Introduction

In this paper we describe how to complete a deterministic planning domain description with probabilistic information about the outcome of actions, where this information is acquired from experience. We propose to solve probabilistic planning problems using a deterministic planner together with replannning techniques, so we can reason with a richer representation of the domain than in Reinforcement Learning. We can flexibly solve a greater number of problems with different goals and infinite number of potential states, and also handle resources and the costs of actions. Also, we learn knowledge about the dynamics of the world, which allows us to be flexible enough to deal with domains where the outcomes of actions are not deterministic. The acquired knowledge is expressed in a symbolic representation, thus, it can be reused to solve other problems in the same domain and is understandable by humans.

2 Capturing the world uncertainty

The probabilistic information for the planning domain is captured as rules of two kinds which explain the successes and failures of actions depending on a given

state. Although our approach is flexible enough to learn other concepts, we just try to induce these two kinds of rules from observing the executions of actions because we consider they give us enough information to deal with the domains traditionally used as test benches in probabilistic planning.

2.1 Learning the Rules about the actions performance with ILP

To induce these rules we use the ILP system ALEPH[3] based on the Stephen Muggleton's ideas of inverse entailment. ALEPH proposes hypothesis, as PROLOG programs, that cover a set of examples described using first order logic predicates and is able to deal with noisy learning examples. It receives as inputs:

- Background knowledge, which contains information about the types, the objects and the predicates of the planning domain. Also here we could introduced extra information about the structure of the planning domain extracted using domain analysis tools.
- The positive and negative learning examples, which are tuples of the form: `(action, state, result)`, where `action` is the fully instantiated executed action; `state` is a description of the current state in which the action was executed and `result` is the result of the execution: `success` or `failure`.

2.2 Estimating rule probabilities with MLE

We use Maximum Likelihood Estimation (MLE). This estimation is implemented by using a PRISM program that models the classification of the observations as positive or negative examples for the induced rules.

We assume the following generative model for the observed data: (1) the unlabelled example is generated with a probability equal to its relative frequency in the data; (2), a rule is chosen according to an unknown distribution over rules; (3), the process checks to see if the rule thus chosen covers the example: if not we have a failure and the process returns to step (2) otherwise it continues and (4) the example is now classified as positive or negative according to an unknown probability associated with the chosen rule.

MLE simultaneously estimates both the probabilities of choosing rules and the classification probabilities. This is a missing data problem, since we do not know which rule classified each example. PRISM handles MLE in such situations using the EM (Expectation-Maximisation) algorithm. The missing data includes an unknown number of failures at step (3) so a particular version of EM: Failure-Adjusted Maximisation (FAM) is used which PRISM implements using a First Order Compiler.

3 Planning in Probabilistic domains

We propose three different ways to automatically integrate the new acquired information with the initial knowledge about the world. Further studies are needed to decide which is the most convenient one.

[3] http://web.comlab.ox.ac.uk/oucl/research/areas/machlearn/Aleph

1. This information can be used to define heuristics that modify the initial behaviour of a deterministic planner.
2. This information can be use to define a cost model for the planning actions.
3. This information together with the initial deterministic description of the domain can be compiled into a probabilistic domain description.

4 Experiments

The experiments have been carried out in a modified version of the blocksworld domain from the probabilistic track of the last International Planning Competition(IPC[4]). In this domain, a robot arm can: `put-down` and `pick-up` a block. Besides that, more than one robot arm that can handle the blocks. Therefore, the operators `pick-up` and `put-down` have another extra argument indicating which robot arm is carrying out the action. Actions tried by a robot which is dirty fail 75% of the times. Initially the planner does not know this; this knowledge has to be automatically acquired from the execution of actions.

```
success_pick_up_block_from(A,B,C,D) :-
  true_in_state_not_holding(A,B,E).
failure_pick_up_block_from(A,B,C,D) :-
  true_in_state_dirty(A,B).
success_put_down_block_on(A,B,C,D) :-
  true_in_state_not_dirty(A,B).
success_put_down_block_on(A,B,C,D) :-
  true_in_state_holding(A,E,F),
  true_in_state_not_dirty(A,E).
success_put_down_block_on(A,B,C,D) :-
  true_in_state_holding(A,B,C).
failure_put_down_block_on(A,B,C,D) :-
  true_in_state_dirty(A,B).
```

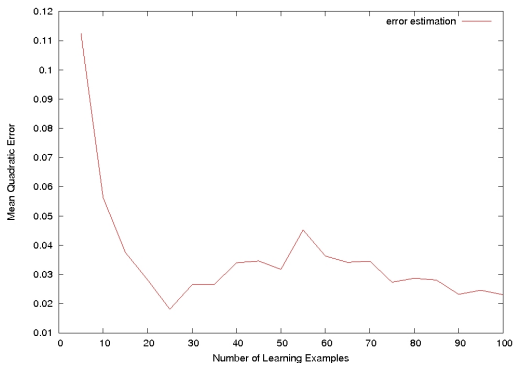

Fig. 1. The induced Rules after solving 5 blocksworld problems and the variation of the error as a function of the number of samples used in the estimation

We have solved five *8-blocks* problems and analysed the induced rules to test whether this knowledge has been captured or not. Figure 1 shows the set of acquired rules: rules 2,3,4 and 6 state that actions are going to fail when the robot trying the action is dirty. Rule 1 and 5 have just captured preconditions of the actions that we initially knew from the deterministic description of the domain. Also, we have measured the mean quadratic error between our estimation and the real probability values. The graph in Fig 1 shows the evolution of this error as we increase the number of samples used in the estimation. In this domain the error value is stabilised when we are using more than 25 learning examples.

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