Learning to Evaluate Conditional Partial Plans

Sławomir Nowaczyk and Jacek Malec
Department of Computer Science, Lund University
{slawek|jacek}@cs.lth.se

We study agents situated in partially observable environments, who do not have sufficient resources to create conformant (complete) plans. Instead, they create plans which are conditional and partial, execute or simulate them, and learn from experience to evaluate their quality. Our agent employs an incomplete symbolic deduction system based on Active Logic and Situation Calculus for reasoning about actions and their consequences. An Inductive Logic Programming algorithm generalises observations and deduced knowledge so that the agent can choose the best plan for execution.

To be fully rational, situated agents need to consciously alternate between reasoning, acting and observing their environment, or even do all those things in parallel. We aim to achieve this by making the agents create short partial plans and execute them, learning more about their surroundings throughout the process. They create several partial plans and reason about usefulness of each one, including what knowledge can it provide. They generalise their past experience to evaluate the likelihood of plans leading to the goal. The plans are conditional (i.e. actions to be taken depend on observations made during execution), which makes quality estimation less situation specific.

The architecture of our agent consists of four main functional modules. Each of them is responsible for a different part of agent’s rationality, but the overall intelligence is only achievable by the interactions of them all. The Deductor module is the one responsible for classical “reasoning”. It uses a logical formalism based on combination of Active Logic [1] and Situation Calculus [2] (as introduced in [3]) to find out consequences of the agent’s current beliefs. Based on the domain knowledge and previous observations, it analyses possible actions and predicts the effect of their execution.

The second module is a Planner, which generates partial, conditional plans applicable in the agent’s current situation. The third main module, Actor, oversees Deductor’s reasoning process and evaluates plans the latter has come up with, trying to find out which is the most useful one to perform. For this paper, Actor waits until Deductor terminates and only executes plans after this happens, but in general it is Actor’s responsibility to balance acting and deliberation.

Finally, the Learner module analyses the agent’s past experience and induces rules for estimating quality of plans. Results of learning process are used both by Deductor and by Actor. In particular, since the plans Deductor reasons about are partial, it can be very difficult to estimate whether a particular plan is a step in the right direction or not. Using machine learning techniques is one way in which this could be achieved.

Our initial experiments concerned learning how to detect “bad” plans early, so that Deductor does not need to waste time deliberating about them. We have used the Inductive Logic Programming algorithm called PROGOL [4], since it is among the best known ones. PROGOL is based on the idea of inverse entailment and it employs a covering approach similar to FOIL, in order to generate hypothesis consisting of a set of clauses which cover all positive examples and do not cover any negative ones.
In the first experiment (curve marked “Without mode declarations”), we used as little domain-specific knowledge as possible, in particular we have not provided any mode declarations for PROGOL. The goal of mode declarations is to reduce the hypothesis search space by limiting types of predicate arguments. The second curve (“With mode declarations”) clearly shows that providing even such a small amount of domain knowledge greatly improves quality of learned hypothesis. It can be also easily seen that the accuracy in the Wumpus domain is significantly higher than the one in the Chess domain. Nevertheless, the learning is still not fully successful. This is due to overfitting and the fact that the search space is too large for PROGOL to handle sufficiently well.

Because of that, we have limited the amount of knowledge used for learning: seemingly, presenting all of the agent’s knowledge to the ILP algorithm is not the best idea. As a start, we have decided to use only the initial domain definition and the observations that the agent made in previous situations. The results of learning can be seen on curve marked “Excluding Deductor”, so named since they roughly correspond to an agent who does not have a specialised deduction module and uses learning only. Finally, in our fourth and final experiment, we have selected only the most relevant parts of knowledge generated by Deductor and presented them to PROGOL. In the Wumpus case this included maybeWumpus, noWumpus and knowsWumpus predicates, while in Chess it included notProtected, distanceTwo, and distanceTwo predicates. As can be seen from the curve marked “Including Deductor”, the agent managed to perfectly identify bad plans from as few as 30 examples chosen at random, in both domains.

It is interesting to note that as few as five hand-chosen example plans suffice for PROGOL to learn the correct definition for the Wumpus domain, which opens up interesting possibilities for an agent to select learning examples in an intelligent way.

References