Extending DTGOLOG with Options[4]

A. Ferrein, Ch. Fritz, and G. Lakemeyer
Department of Computer Science
RWTH Aachen
{ferrein, fritz, gerhard}@cs.rwth-aachen.de

1 Introduction

Recently Boutilier et al. (2000) proposed the language DTGOLOG which combines explicit agent programming with decision theory. The motivation is that a user often has some idea about how to go about solving a particular problem yet at the same time does not want to or cannot commit in advance to an exact course of action. Instead, certain choices are left to the agent running the program and determining an optimal action selection policy involves solving a Markov Decision Process (MDP) [Puterman, 1994]. In a sense, a DTGOLOG program can be thought of as a factored representation of an MDP. As an example, Boutilier et al. consider a mail-delivery scenario where the task of delivering mail to a particular person is hand-coded and fixed, while the agent chooses the order in which the various people are served according to some reward function. They note that this approach allows solving problems which are much larger than those solvable using the traditional dynamic-programming approach for MDPs.

Nevertheless, it is not very difficult to find examples where DTGOLOG also shows poor computational behavior. Roughly, the complexity is determined by the number of choices the agent needs to consider when computing a policy and the number of actions with uncertain outcomes (nature's choices) that need to be considered along the way. To better control this potential blow-up we were inspired by work on so-called macro actions or options previously developed in the MDP framework [Hauskrecht *et al,* 1998; Sutton *et al.,* 1999]. The idea is that the policies for certain subtasks like leaving a room are pre-computed and then simply used when working on the global policy.

This is, roughly, what we propose: for a given domain identify subtasks and compute local policies using standard MDP techniques like value iteration. From those local policies generate two representations: (1) a (deterministic) DTGOLOG program directly encoding the local policy in terms of what action to take while the local policy is applicable and (2) so-called *abstract stochastic actions* which only address the expected outcome of a local policy. It turns out that, while both representations can be used to compute a global policy, the use of *abstract stochastic actions* may result in an exponential speed-up.— In the next section we briefly introduce MDPs and DTGOLOG. Section 3 describes options in more detail and how to map them into DTGOLOG. We end with a discussion of experimental results.

*This work was partly supported by the German Science Foundation (DFG), Grant No. LA 747/9-1, and a grant by the NRW Ministry of Education and Research (MSWF).

2 Decision Theory and GOLOG

A fully observable MDP is represented as a tuple $M = (\mathcal{S}, \mathcal{A}, \Pr, \mathcal{R})$, where S is a finite set of states, A is a finite set of actions, Pr is a probability distribution $\Pr : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0,1]$ ($\Pr(s_j|A, s_i)$ denotes the probability of the agent ending up in state S_j after performing action A in state s_t, and $\mathcal{R} : \mathcal{S} \to \mathbb{R}$ is a bounded reward function. The objective is then to construct a policy $\pi : \mathcal{S} \to \mathcal{A}$ that maximizes the expected total reward over some horizon. A simple algorithm for constructing optimal policies is value iteration (cf. [Puterman, 1994]).

DTGOLOG [Boutilier *et al,* 2000] can be thought of, roughly, as a programming language which allows a user to combine pre-defined primitive actions into complex programs using the usual constructs like sequence, if-then-else, while, and recursive procedures. In addition there are *nondeterministic actions* to model that an agent can choose among alternatives. The semantics is based on the situation calculus and, in particular, the so-called basic action theories described in [Reiter, 2001], which define when primitive actions are executable and how they change and do not change what is true in the world. As a special form of primitive action DTGOLOG allows so-called *stochastic actions,* which have probabilistic outcomes, just as in MDPs. For example, it is possible to define an action r (move to the right), after which a robot has moved one step to the right with probability 0.9, but has moved to the left, up, or down with probability 0.1. MDP-style reward and cost functions are incorporated as well. Given a DTGOLOG program, the idea is then to compute a policy in the sense that each nondeterministic choice is resolved in a decision theoretic manner, that is, by choosing the action with maximal expected reward. Assuming that there are not too many nondeterministic or stochastic actions, a policy can be computed even in cases where an MDP representation seems infeasible due to the size of the state space.

3 Mapping Options into DTGOLOG

As in [Sutton *et al.,* 1999], we define *options* as triples (I, π, β), where I is the set of initial states, π is a policy, and β is the set of terminating states. To illustrate options consider the example in Figure 1 from [Hauskrecht *et al,* 1998].

Figure 1: Mazc66 from [Hauskrecht *et al,* 1998].

The task is to find an optimal policy to get from position *S* to position *G.* Performing an action has cost 1, the goal position has a high positive reward. The agent can perform the stochastic basic or primitive actions *r,l,u,d* succeeding with probability 0.9. With probability 0.1 it will be in any other adjacent position. For each room options arc defined to leave the room through a certain door (one for each room/door combination). The gray dots correspond to the termination positions of these options. For the left lower room in Figure 1 there are two options *OE* and O_N (leave through the east or north door, respectively) with $I_E = \{(x,y) \mid 0 < x \leq 3, 0 < y \leq 3\}$ and β_E = {(2,4), (4,2)}, and similarly for O_N. (In order to mark state (4,2) in β_E as success and (2,4) as failure, they are assigned positive and negative reward, respectively.)

Applying standard value iteration techniques, the following can be computed for an option *O:*

1. the optimal policy π, that is, the most appropriate action A_i for each $s_t \in$ J;

2. for each $s_t \in I$ and $S_j \in \beta$, the probability to terminate the option with the outcome S_J when starting in s_i

These results can now be translated into a form suitable for DTGOLOG. Given I = *{s1,..., sn),* suppose that each s_t can be uniquely characterized by a logical formula φ_t. (In our example, <*pi* simply expresses the coordinates of the location of .s\.) Then the policy π can be translated in a straightforward fashion into the following DTGOLOG program:

```
senseState(O);
while(φ^I)
  if φ1 then A1 else if ... else if φn then An;
  senseState(O);
endwhile
```

where $\varphi^I = \bigvee \varphi_i$ and senseState(O) is a so-called sensing action. Roughly, executing senseState(O) establishes the truth values of the φ_i so that they can be tested in the following while and if conditions. (The sensing actions are necessary to account for the MDP assumption of full observability.) All in all, the program simply prescribes that the optimal action (according to π) should be executed as long as the agent is in one of the initial states of *O.*

Given (2.) above we can also generate for each $s_t \in I$ an abstract stochastic action *o(si).* For example, for the option *OE* and state s_1 = (2,2) we would define an abstract stochastic action $o_e(s_1)$ with nature's choices $o_e^e(s_1)$ and $o_e^n(s_1)$ for leaving through the east or north door, respectively, together with the probabilities of these actually occurring as computed in (2.) $(\mathrm{Prob}(o_e^e(s_1) \mid o_e(s_1)) = 0.99898$, $\mathrm{Prob}(o_e^n(s_1) \mid o_e(s_1)) = 0.00102)$.

Just as options in the MDP framework can be treated like primitive actions in MDPs that use these options, the translation into abstract stochastic actions in the DTGOLOG framework has the same effect, that is, we can now write DTGOLOG programs treating abstract stochastic actions just like any other primitive actions. A global policy computed by DTGOLOG for such a program usually mentions abstract actions. These are not readily executable (leaving a room through the east door is not a primitive action). Hence, in a final step we need to replace the abstract actions by the programs which we derived above.

4 Experimental results

Using our running example we conducted a number of experiments, the goal being at different distances from the initial position (Fig.l shows the special case of distance 8). The results are given in Figure 2. The x-axis depicts the initial distance to the goal, the y-axis the running time. We compared three different approaches: (A) calculating the optimal policy in DTGOLOG nondeterministically choosing only from the primitive actions, (B) using a set of procedures like the one in the previous section for leaving each room towards a certain neighboring room, choosing from primitive actions only in the goal room, and (C) using options in the form of abstract stochastic actions, choosing from primitive actions only in the goal room.

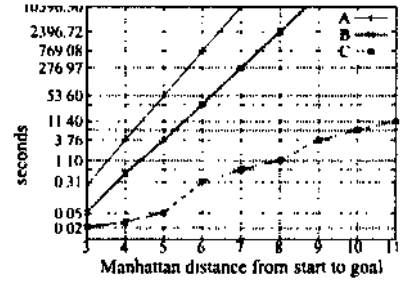

Figure 2: Runtimes of the three test programs in the maze.

Note that the y-axis of the diagram has a logarithmic scale. The speed-up from (A) to (B) shows the benefit using DTGOLOG to constrain the search space by providing fixed programs for certain subtasks. Interestingly, (C), that is, using abstract actions, clearly outperforms (B). Roughly, this is because each abstract action has only two outcomes, whereas the corresponding program provides a very fine-grained view with a huge number of outcomes that need to be considered.

Taking the time of calculation of all options into account (here: 10.51 seconds) the use of options pays off at horizons greater than 5. Also, calculating options can be done off-line and can be neglected in case of frequent reuse.

We remark that while method (A) guarantees optimality, this is not necessarily so for (B) ana (C), for essentially the same reasons as in [Hauskrecht *et ai,* 1998]. Certainly in the case of (C), this price seems worth the computational gain. Finally, while we currently assume options as given, Hauskrecht et al. (1998) discuss ways of automatically generating options with good solution qualities, an issue we intend to investigate in the future as well.

References

[Boutilier *et al,* 2000] C. Boutilier, R. Reiter, M. Soutchanski, and S. Thrun. Decision-theoretic, high-level agent programming in the situation calculus. In *Proc. AAAI-2000,* 2000.

[Hauskrecht *et al.,* 1998] M. Hauskrecht, N. Meuleau, L. Kaelbling, T. Dean, and C. Boutilier. Hierarchical solutions of MDPs using macro-actions. In *Proc. UAI98,* 1998.

[Puterman, 1994] M. Puterman. *Markov Decision Processes: Discrete Dynamic Programming*. Wiley, New York, 1994.

[Reiter, 2001] R. Reiter. *Knowledge in Action.* MIT Press, 2001.

[Sutton *et al.,* 1999] R. Sutton, D. Precup, and S. Singh. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Journal of Artificial Intelligence,* 1999.