Integrating decision-theoretic planning and programming for robot control in highly dynamic domains

Christian Fritz

Thesis, Final Presentation
Introduction

Goals:

- combine:
  - programming
  - decision-theoretic planning
  - on-line!
- extend planning with options
- evaluate in three diversified example domains
  - grid world
  - RoboCup Simulation
  - RoboCup Mid-Size
ICPGOLOG

- based on situation calculus
- extends basic GOLOG:
  - on-line: incremental, sensing (active and passive)
  - continuous change
  - concurrency
  - progression
  - probabilistic projection
  - nondeterminism
- problems:
  - decision making: explicit, missing utility theory
  - projection comparatively slow
Markov Decision Processes (MDPs) standard model for decision-theoretic planning problems

- Formally: $M = \langle S, A, T, R \rangle$, with
  - $S$ a set of states
  - $A$ a set of actions
  - $T : S \times A \times S \rightarrow [0, 1]$ a transition function
  - $R : S \rightarrow \mathbb{R}$ a reward function

- Here: fully observable MDPs

- Planning task: find an optimal policy, maximizing expected reward

- **Note**: $S$ and $A$ are usually finite!
New Golog derivative DTGolog [Boutilier et al.]

Combines explicit agent programming with planning

Uses MDPs to model the planning problem:
- $S =$ situations
- $A =$ primitive actions
- $T =$ for each action $a \in A$, a list of outcomes and their respective probability
- $R :$ situations $\rightarrow \mathbb{R}$

applies decision-tree search to solve MDP up to a given horizon
Disadvantages:

- offline
- situations = states
  - infinite state space
  - inefficient
Contributions:

- re-added nondeterminism with decision-theoretic semantics
  → on-line decision-theoretic Golog
- added options to speed up MDP solution
- preprocessor to minimize interpretation on-line
Extending DTGolog with Options
Options? what’s that?

with cell-to-cell primitive actions

```
Iteration #0
```

```
Iteration #1
```

```
Iteration #2
```

with room-to-room options

```
Iteration #0
```

```
Iteration #1
```

```
Iteration #2
```
Options

Idea:

- construct complex actions from primitive ones
- options: solutions to sub-MDPs
- generate models about them:
  - when possible to execute?
  - which outcomes possible to occur?
  - which probabilities do the outcomes have?
  - expected rewards and costs? (expected value)
- these can then be used in planning
how do we integrate options into DTGolog/ReadyLog?

- avoiding the inconvenience “situations = states”
- instead mappings:
  - situations $\rightarrow$ states (when ’entering’ option)
  - states $\rightarrow$ situations (when ’leaving’ option)
- options..
  - ..are solutions to local MDPs..
  - ..encapsulated into a stochastic procedure.
- stochastic procedures..
  - ..are procedures with an explicit model (preconditions/effects/costs);
  - ..replace stochastic actions;
  - ..can model options.
Generating Options

how do we generate options?

- define:
  - $\phi$ precondition (think: states where option is applicable)
  - $\beta : \text{exitstates} \rightarrow \text{value}$ pseudo-rewards for local MDP
  - $\theta$ option-skeleton one-step program to take in each step:
    - ..usually something like $\text{nondet( [left, right, down, up] )}$;
    - ..can contain ifs;
    - ..can build on options/stochastic procedures
  - and: two mappings:
    - $\Phi : \text{situations} \rightarrow \text{states}$
    - $\Sigma : \text{states} \rightarrow \text{situations}$
    - $\text{option\_mapping}(o, \sigma, \Gamma, \varphi)$
Example policy:

```prolog
proc (room_2, [exogf_Update],
    while (is_possible (room_2),
        [ if (pos=[0, 0], go_right),
          if (pos=[0, 1], go_right),
          if (pos=[0, 2], go_up),
          if (pos=[1, 0], go_right),
          if (pos=[1, 1], go_right),
          if (pos=[1, 2], go_right),
          if (pos=[2, 0], go_down),
          if (pos=[2, 1], go_right),
          if (pos=[2, 2], go_up, []))), exogf_Update))).
```

Example model (for state 'position=(0,0)'):

```prolog
opt_costs (room_2, [(pos, [0, 0])], 4.51650594972207).
opt_probability_list (room_2, [(pos, [0, 0])], [(pos, [1, 3])], 0.00012),
                     [(pos, [3, 1]), 0.99987]).
```
Experimental Results

(a) full MDP planning (A)
(b) heuristics (B)
(c) options (C)
Experimental Results

The graph illustrates the relationship between the Manhattan distance from start to goal and the time in seconds. The data points and lines represent different scenarios labeled A to C. Each scenario shows a distinct pattern, indicating how the distance affects the time required to reach the goal.
On-line Decision-Theoretic Golog for Unpredictable Domains
READYLOG: on-line DT planning

on-line:

- incremental
  - \texttt{solve(plan-skeleton, horizon)}
  - execute returned policy

- sensing / exogenous events
  - problem:
    - dynamic environment (changes while thinking)
    - imperfect models
      \rightarrow policy can get invalid
  \Rightarrow execution monitoring:
    - program and policy coexistence
    - markers
Execution Monitoring Semantics

\[\text{Trans}(\text{solve}(p, h), s, \delta', s') \equiv \]

\[\exists \pi, v, pr. \text{BestDo}(p, s, h, \pi, v, pr) \land \delta' = \text{applyPol}(\pi) \land s' = s.\]

\[\text{BestDo}(\text{if}(\varphi, p_1, p_2); p, s, h, \pi, v, pr) \equiv \]

\[\varphi[s] \land \exists \pi_1. \text{BestDo}(p_1; p, s, h, \pi_1, pr) \land \pi = \mathcal{M}(\varphi, \text{true}); \pi_1 \lor \]

\[\neg \varphi[s] \land \exists \pi_2. \text{BestDo}(p_2; p, s, h, \pi_2, v, pr) \land \pi = \mathcal{M}(\varphi, \text{false}); \pi_2\]

\[\text{Trans}(\text{applyPol}(\mathcal{M}(\varphi, v); \pi), s, \delta', s') \equiv s = s' \land \]

\[(v = \text{true} \land \varphi[s] \land \delta' = \text{applyPol}(\pi) \lor \]

\[v = \text{false} \land \neg \varphi[s] \land \delta' = \text{applyPol}(\pi) \lor \]

\[v = \text{true} \land \neg \varphi[s] \land \delta' = \text{nil} \lor \]

\[v = \text{false} \land \varphi[s] \land \delta' = \text{nil})\]
- options (..)

- preprocessor:
  - translates READYLOG functions, conditions, definitions.. to Prolog code
  - creates successor state axioms from effect axioms
  - speed-up of about factor 16
Experimental Results
Experimental Results:

SimLeague

compared with ICPGOLOG (Normans results)
planning time in seconds

<table>
<thead>
<tr>
<th></th>
<th>ICPGOLOG</th>
<th>READYLOG</th>
</tr>
</thead>
<tbody>
<tr>
<td>goal shot</td>
<td>0.35</td>
<td>0.01</td>
</tr>
<tr>
<td>direct pass</td>
<td>0.25</td>
<td>0.01</td>
</tr>
</tbody>
</table>

speed-up due to preprocessor

Example where these are combined (demo):

```prolog
solve (nondet ([goalKick (OwnNumber ),
    [pickBest (bestP , [2..11],
        [directPass (OwnNumber , bestP, pass_NORMAL ),
            goalKick (bestP )]]),
    [goalKick (OwnNumber , bestP, pass_NORMAL )]
    ]), Horizon )
```

Integrating decision-theoretic planning and programming for robot control in highly dynamic domains – p.22/32
Experimental Results: MidSize
solve (  
    nondet ([ kick(ownNumber , 40),  
            dribble_or_move_kick (ownNumber ),  
            dribble_to_points (ownNumber )],  
    if (isKickable (ownNumber ),  
        pickBest (var_turnAngle , [-3.1, -2.3, 2.3, 3.1],  
            [ turn_relative (ownNumber , var_turnAngle , 2),  
                nondet ([] [intercept_ball (ownNumber , 1),  
                           dribble_or_move_kick (ownNumber )],  
                           [intercept_ball (numberByRole (supporter ), 1),  
                            dribble_or_move_kick (numberByRole (supporter ))]  
                    ) ) )],  
    nondet ([] [intercept_ball (ownNumber , 1),  
                dribble_or_move_kick (ownNumber )],  
    intercept_ball (ownNumber , 0.0, 1)]) )); 4)  
  
proc (dribble_or_move_kick (Own ),  
    nondet ([] [dribble_to (Own , oppGoalBestCorner , 1)],  
            [move_kick (Own , oppGoalBestCorner , 1)]));)  
  
proc (dribble_to_points (Own),  
    pickBest (var_pos , [2.5, -1.25], [2.5, -2.5], [2.5, 0.0], [2.5, 2.5], [2.5, 1.25],  
            dribble_to (Own , var_pos , 1)));)
Experimental Results: MidSize, Behavior

move/dribble/intercept
move_kick/dribble
ball behavior when turning with ball
Experimental Results: MidSize, Example: Situation
Experimental Results: MidSize, Example: Plans

(d) 
(e) 
(f)
Experimental Results: MidSize, Example: Teamplay

(g) 

(h) 

(i) 

Integrating decision-theoretic planning and programming for robot control in highly dynamic domains – p.28/32
Experimental Results: MidSize, Example: Decision Tree
Experimental Results: MidSize, Computation

<table>
<thead>
<tr>
<th></th>
<th>examples</th>
<th>min</th>
<th>avg</th>
<th>max</th>
</tr>
</thead>
<tbody>
<tr>
<td>without ball</td>
<td>698</td>
<td>0.0</td>
<td>0.094</td>
<td>0.450</td>
</tr>
<tr>
<td>with ball</td>
<td>117</td>
<td>0.170</td>
<td>0.536</td>
<td>2.110</td>
</tr>
</tbody>
</table>

- variance due to processor load on robots
- qualitatively: enough for rudimentarily playing soccer
On-line decision theoretic Golog:
- can be applied to highly dynamic domains with infinite/continuous state spaces,
- can coexist with passive sensing,
- motivates more sophisticated execution monitoring.

Options:
- can be added to decision theoretic Golog,
- provide good speed-ups,
- rely on finite state spaces(!).