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

Programming

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

Decision-Theoretic Planning

Markov Decision Processes (MDPs) standard model for decision-theoretic planning problems

- Formally: $M =< S, A, T, R >$, 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!

Programming & Planning: DTGolog

- 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

Programming & Planning: DTGolog

Disadvantages:

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

READYLOG

Contributions:

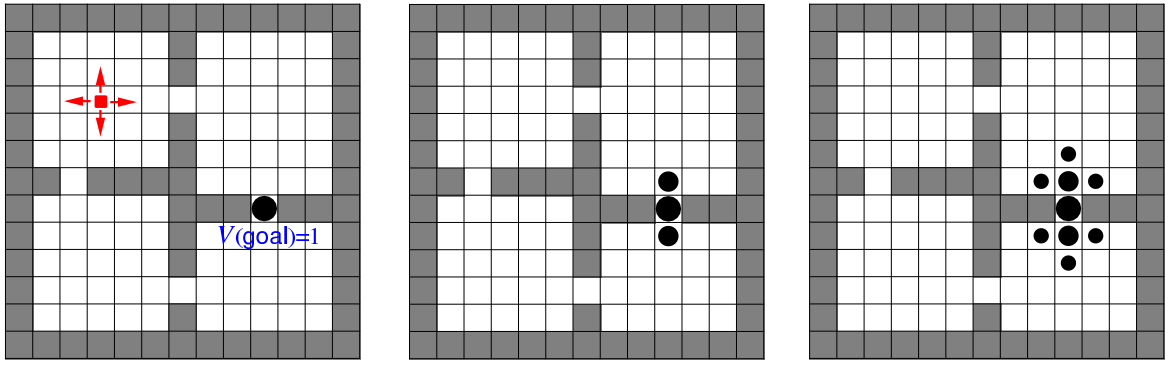
- re-added nondeterminism with decision-theoretic semantics
  $\longrightarrow$ on-line decision-theoretic Golog
- added options to speed up MDP solution
- preprocessor to minimize interpretation on-line

Part I

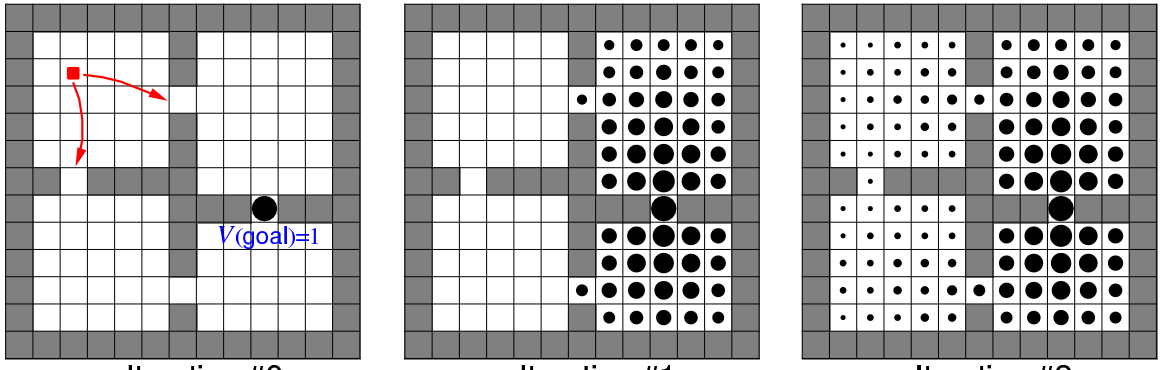
Extending DTGolog with Options

Options? what's that?

with cell-to-cell primitive actions



with room-to-room options



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

Integrating Options into Golog

how do we integrate options into DTGolog/ReadyLog?

- avoiding the inconvenience “situations = states”
- instead mappings:
  - situations $\longrightarrow$ states (when ’entering’ option)
  - states $\longrightarrow$ 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?

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

Examples

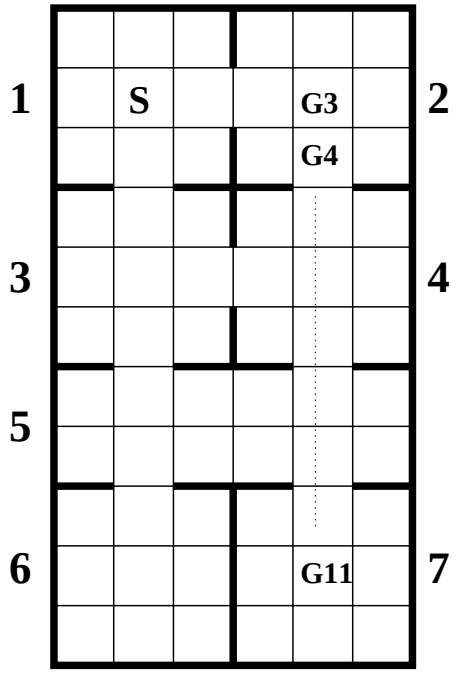
- example policy:

```
proc (room1_2 , [exogf_Update ,
                 while (is_possible (room1_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 ])]).
```

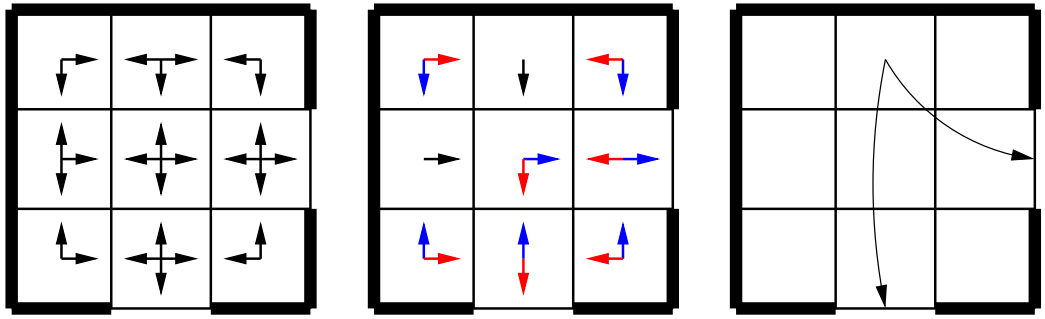
- example model (for state ’position=(0,0)’):

```
opt_costs (room1_2, [(pos, [0, 0])], 4.51650594972207).
opt_probability_list (room1_2 , [(pos , [0, 0])], [([(pos , [1, 3])], 0.00012),
                                                   ([(pos , [3, 1])], 0.99987)]).
```

Test Setting



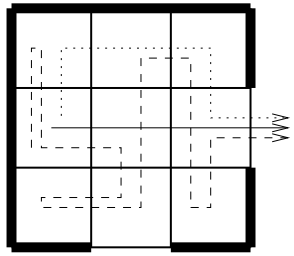
Experimental Results



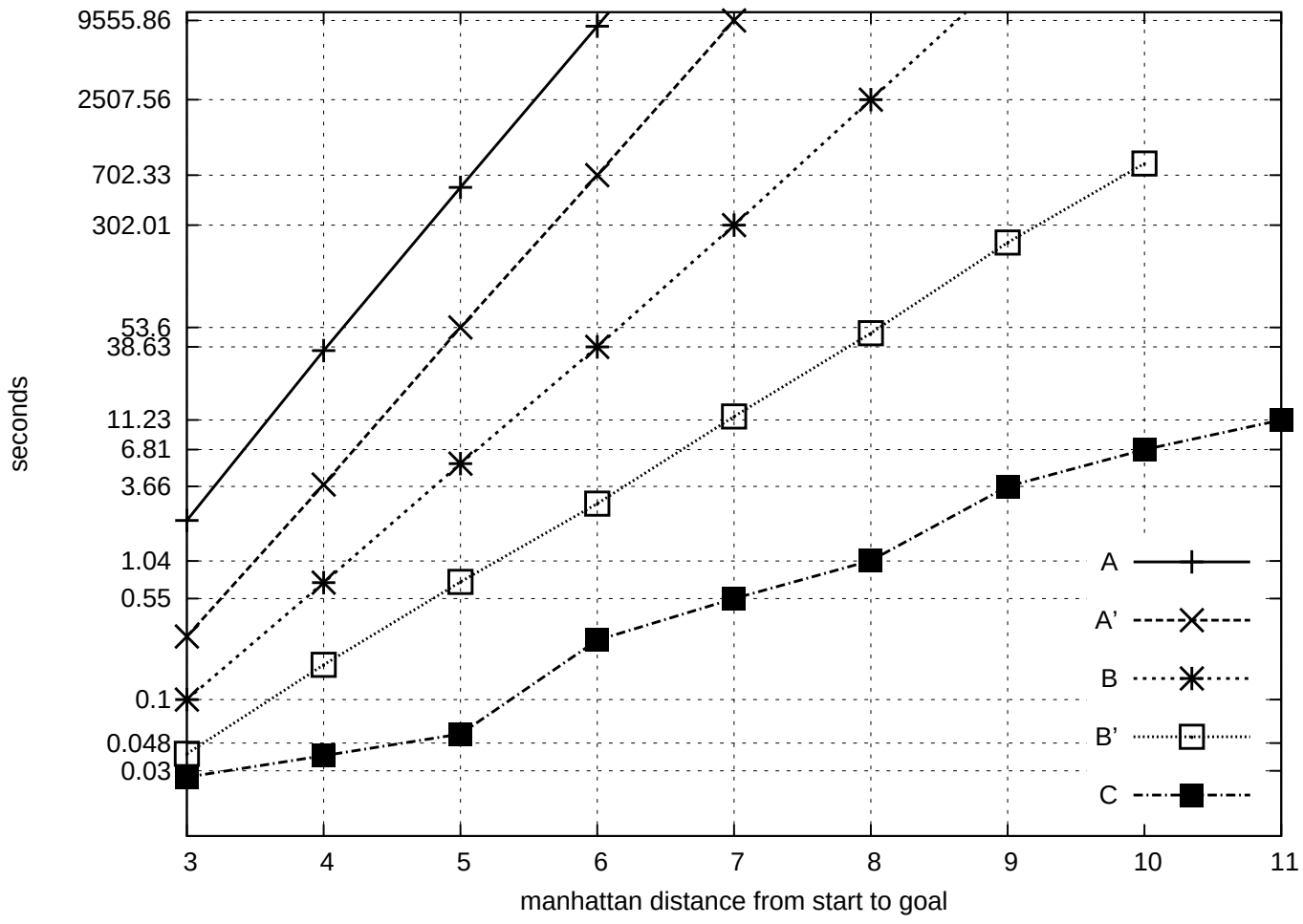
(a) full MDP planning (A)

(b) heuristics (B)

(c) options (C)



Experimental Results



Part II

On-line Decision-Theoretic Golog for Unpredictable Domains

READYLOG: *on-line DT planning*

on-line:

- incremental
  - solve(*plan-skeleton* , horizon)
  - execute returned policy
- sensing / exogenous events
  - problem:
    - dynamic environment (changes while thinking)
    - imperfect models

    $\longrightarrow$ **policy can get invalid**

  $\Rightarrow$ *execution monitoring*:
    - program and policy coexistence
    - markers

Execution Monitoring Semantics

$$Trans(solve(p,h),s,\delta',s') \equiv$$

$$\exists \pi, v, pr.BestDo(p,s,h,\pi,v,pr) \wedge \ \delta' = applyPol(\pi) \wedge s' = s.$$

$$BestDo(if(\varphi,p_1,p_2);p,s,h,\pi,v,pr) \doteq$$

$$\varphi[s] \wedge \exists \pi_1.BestDo(p_1;p,s,h,\pi_1,pr) \wedge \pi = \mathfrak{M}(\varphi,true);\pi_1 \vee$$

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

$$Trans(applyPol(\mathfrak{M}(\varphi,v);\pi),s,\delta',s') \equiv s = s' \wedge$$

$$(v = true \wedge \varphi[s] \wedge \delta' = applyPol(\pi) \vee$$

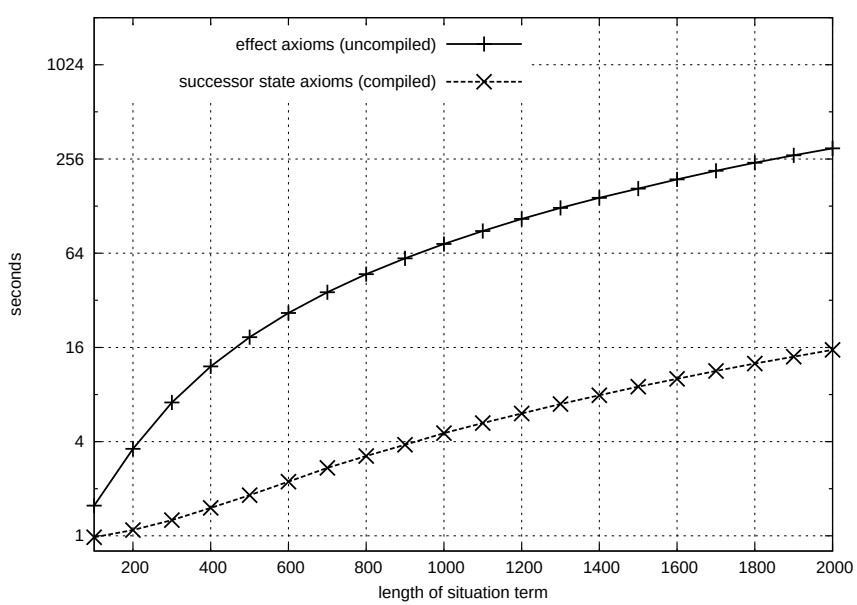
$$v = false \wedge \neg\varphi[s] \wedge \delta' = applyPol(\pi) \vee$$

$$v = true \wedge \neg\varphi[s] \wedge \delta' = nil \vee$$

$$v = false \wedge \varphi[s] \wedge \delta' = nil)$$

READYLOG II

- options (..)
- preprocessor:
  - translates READYLOG functions, conditions, defi nitions.. 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

	ICPGOLOG	READYLOG
goal shot	0.35	0.01
direct pass	0.25	0.01

- speed-up due to preprocessor

Example where these are combined (demo):

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

Experimental Results: MidSize

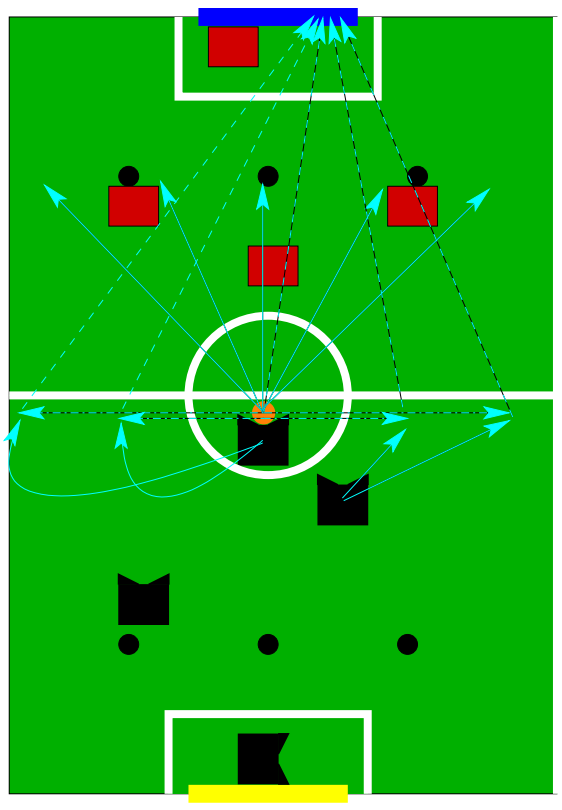
Experimental Results: MidSize, Code

```
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



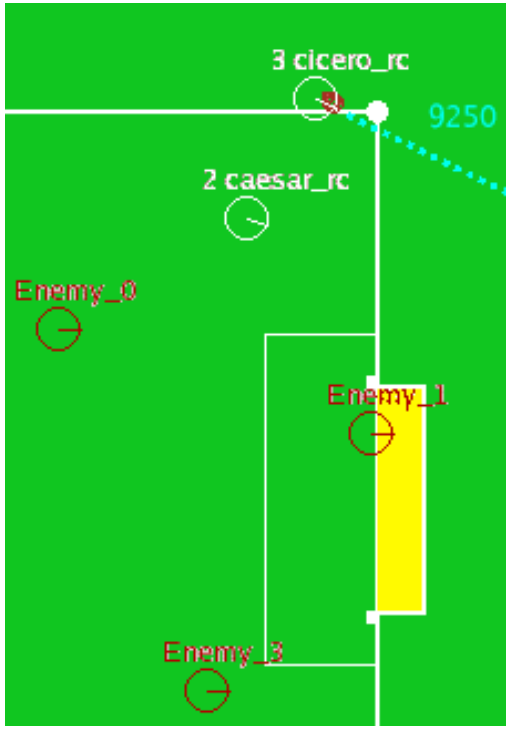
move/dribble/intercept

move_kick/dribble

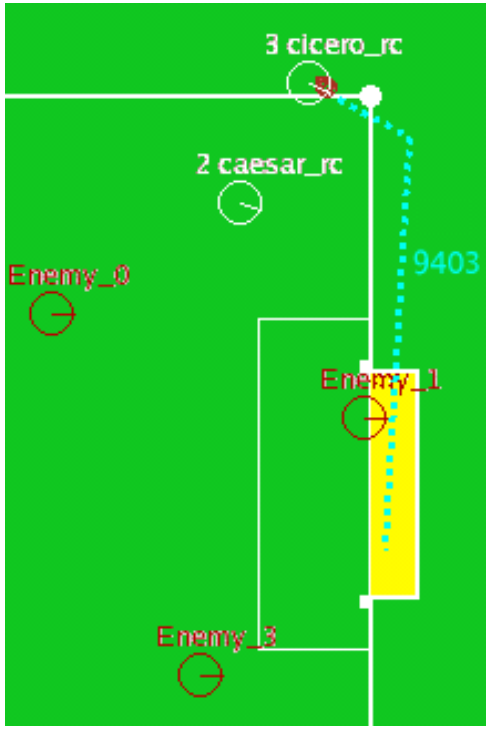
ball behavior when turning with ball

Experimental Results: MidSize, Example: Situation

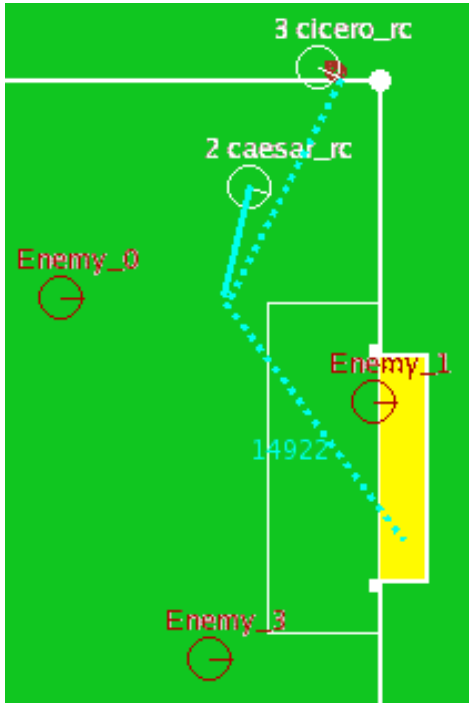
Experimental Results: MidSize, Example: Plans



(d)

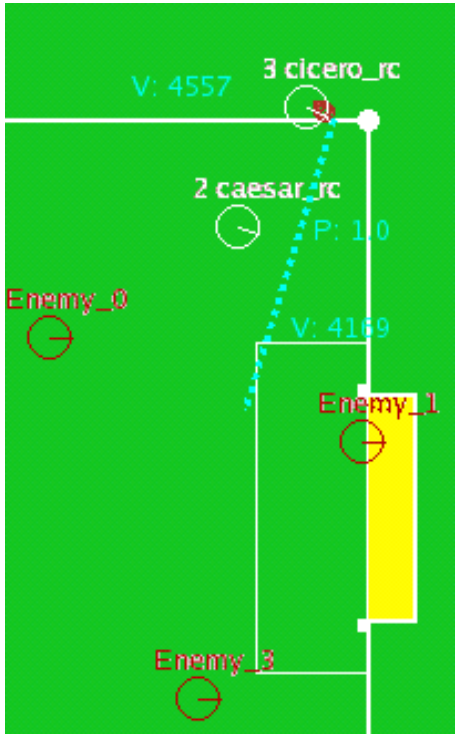


(e)

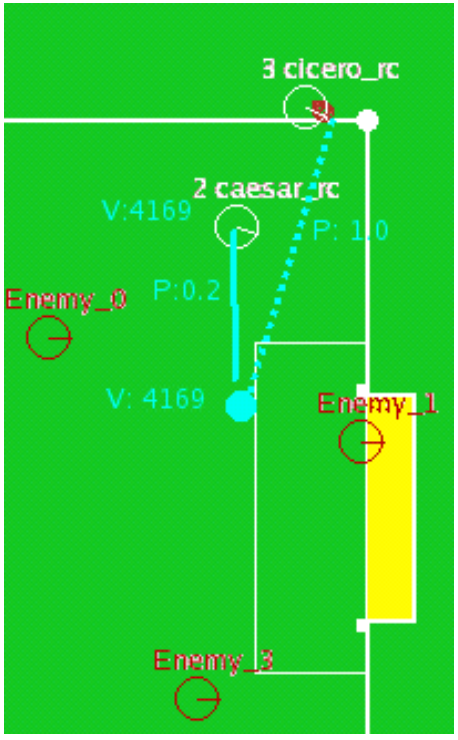


(f)

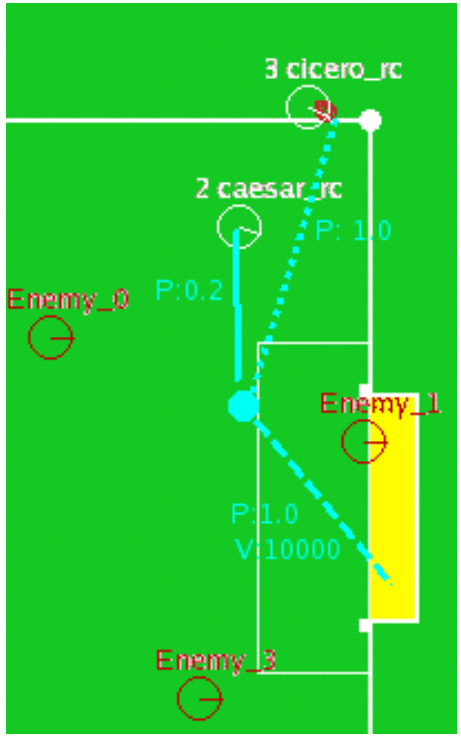
Experimental Results: MidSize, Example: Teamplay



(g)

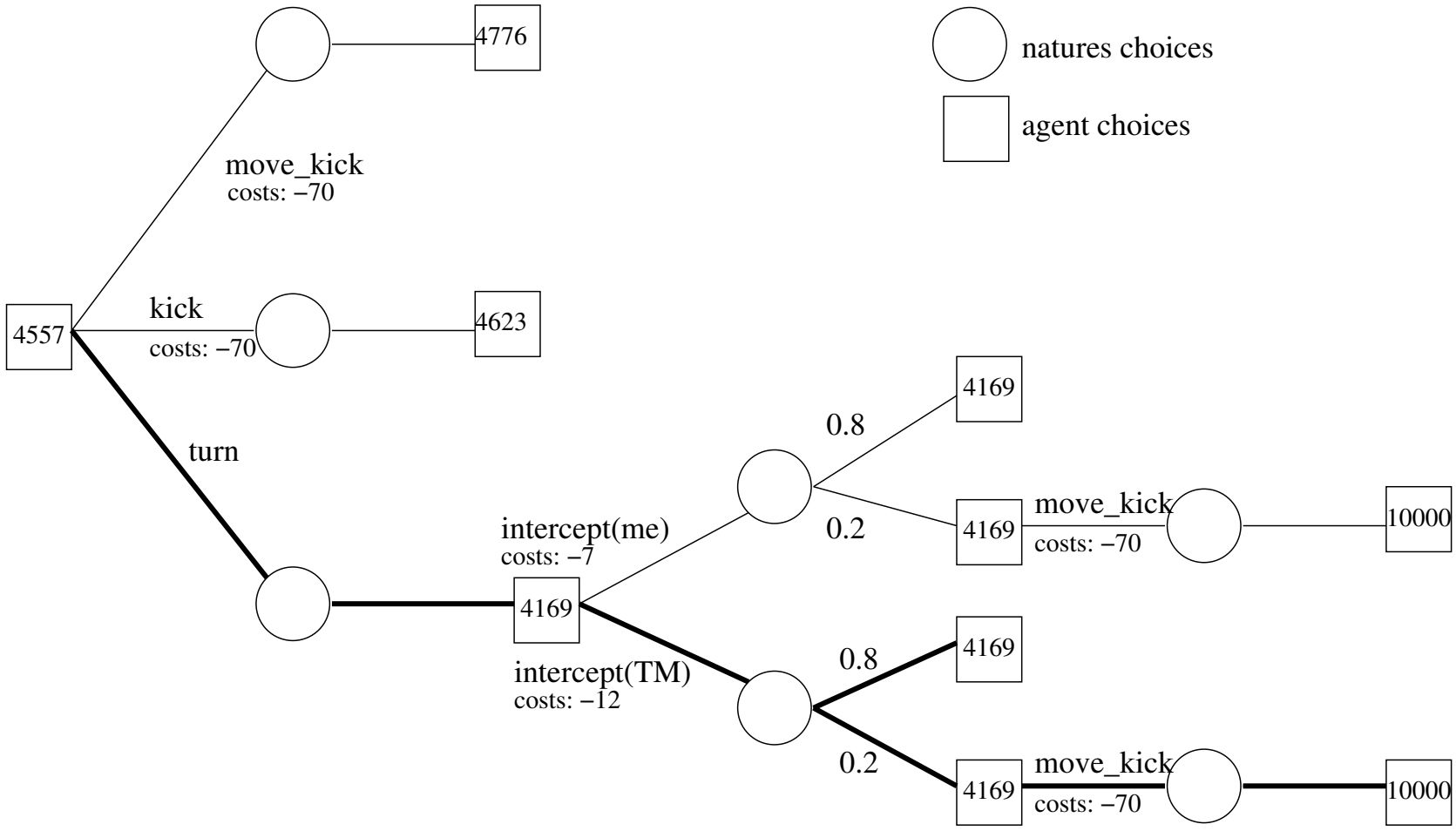


(h)



(i)

Experimental Results: MidSize, Example: Decision Tree



Experimental Results: MidSize, Computation

planning time in seconds

	examples	min	avg	max
without ball	698	0.0	0.094	0.450
with ball	117	0.170	0.536	2.110

- variance due to processor load on robots
- qualitatively: enough for rudimentarily playing soccer

Conclusion

- 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(!).