Reinforcement Learning – COMS4047A

Hierarchical RL

Prof. Benjamin Rosman

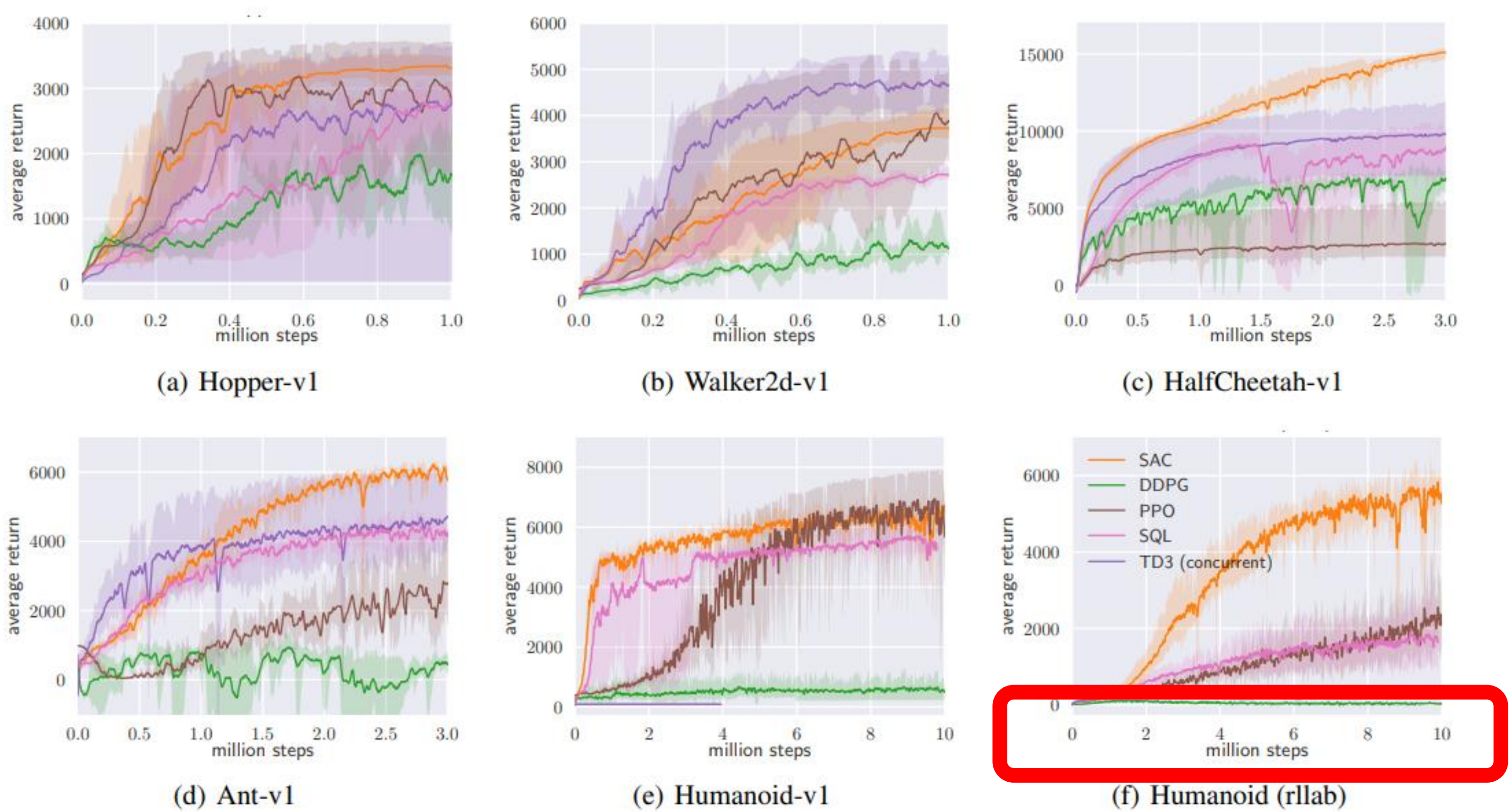
Benjamin.Rosman1@wits.ac.za / benjros@gmail.com

Based on slides by George Konidaris and Steve James

Outline so far

- We started with dynamic programming
  - But want to solve problems when we don’t know a model
- So we used Monte Carlo methods
  - But this needs full episode returns
- So we moved to tabular temporal difference methods
  - But these don’t scale to large problems
- So we introduced function approximation
  - So we’re good, right?

Some really hard problems!

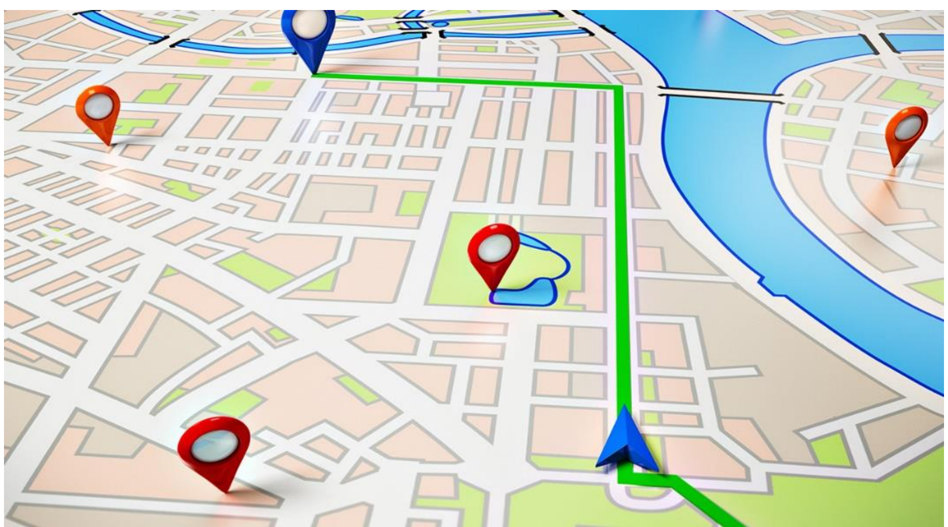


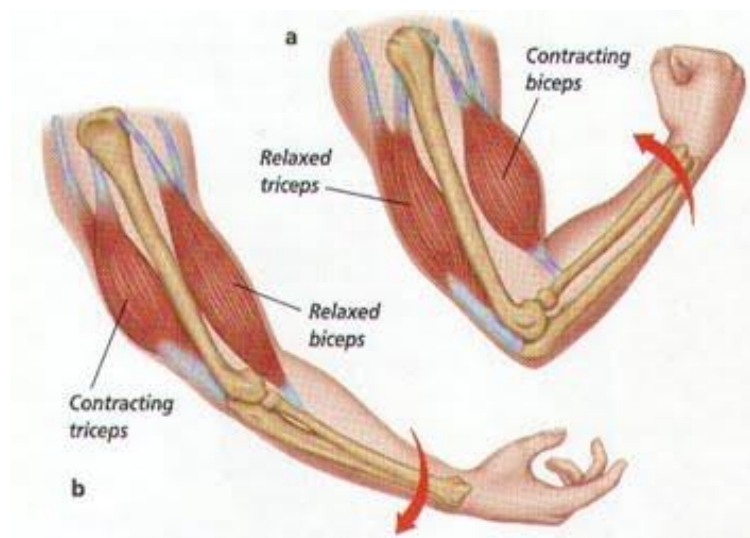
Soft actor-critic, Haaronoja et al 2018

Why is RL hard?

- Sparse
  - Most actions give no reward feedback

- Delayed
  - Rewards may come after executing whole trajectories





How does RL work?

- Solves a “flat” problem!
- Want a robot to make coffee?
- +1 for making coffee, -1 otherwise
- Find a set of actions to achieve this
  - What are those actions? Motor commands? Complex programs?

The most important task

- How do you make coffee?

What about making tea?

Boil water

Add coffee to cup

Pour water into cup

Stir liquid

Enjoy deliciousness

Fill Kettle
Plug in Kettle
...

Hierarchical decomposition

- Decompose the problem into smaller ones
- Solve each of those smaller problems
  - Maybe recursively decompose the subproblems?
- Exploits structure in problems
  - Maybe we can reuse these subproblems elsewhere?

Some questions

$$M = (S, A, T, R, \gamma)$$

- What is the best way of building a hierarchy?

- Can we learn it?

- What kind of hierarchies can we construct?
  - State and action abstraction

Hierarchical RL

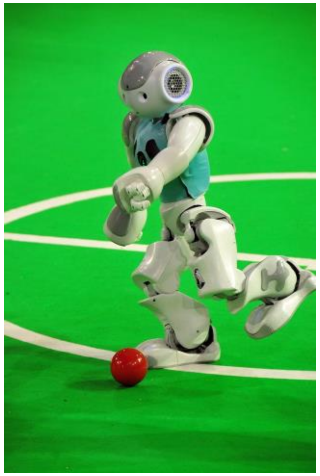
- RL typically solves a *single* problem *monolithically*
- Hierarchical RL:
  - Create and use higher-level macro-actions (skills)
  - Problem now contains subproblems
  - Each subproblem is also an RL problem
- Hierarchies of abstract machines (HAMs), MAXQ, Options

Skill hierarchies

- Hierarchical RL: base hierarchical control on skills
  - Component of behaviour
  - Performs continuous, low-level control
  - Can treat as discrete action

- Behaviour is modular and compositional
  - Like functions in a program!

```
def kick_ball(self, dest):
   # bunch of low level actions here
```



Options [Sutton, et al 2000]

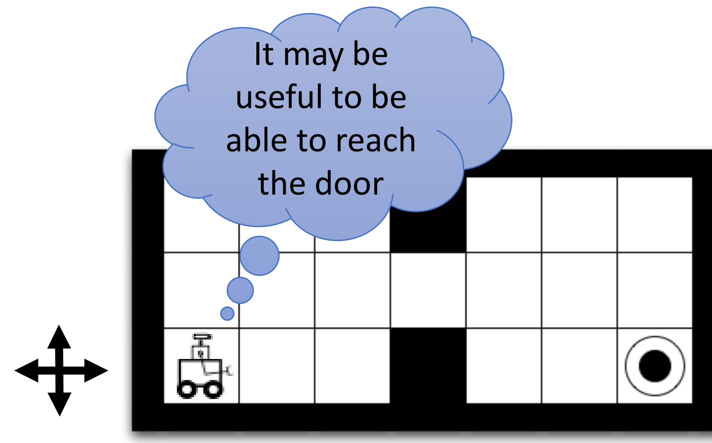
- Theoretical basis for skill acquisition,
  - learning and planning using higher-level actions (options)

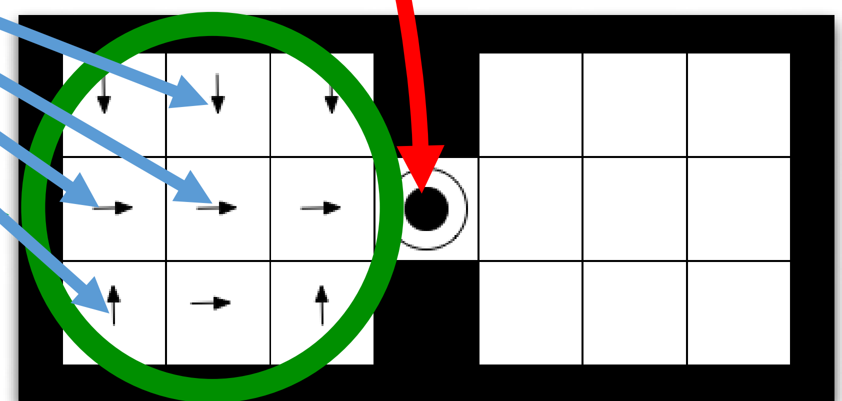
- An option is a temporally-extended action
  - An action executed over many timesteps

Like a policy

Intuitively

- An option *o* is a policy unit:
  - Initiation set
  - Termination condition
  - Option policy





Formally

- A (Markov) option o is defined by
  - Initiation set: $I_o \subseteq S$
  - Policy: $\pi_o : S \times A \rightarrow [0, 1]$
  - Termination condition: $\beta_o : S \rightarrow [0, 1]$

- Can have non-Markov options
  - Functions not solely of state, but also execution history
  - Run for at most n steps, repeat n times, etc
  - Not often used, but can be useful

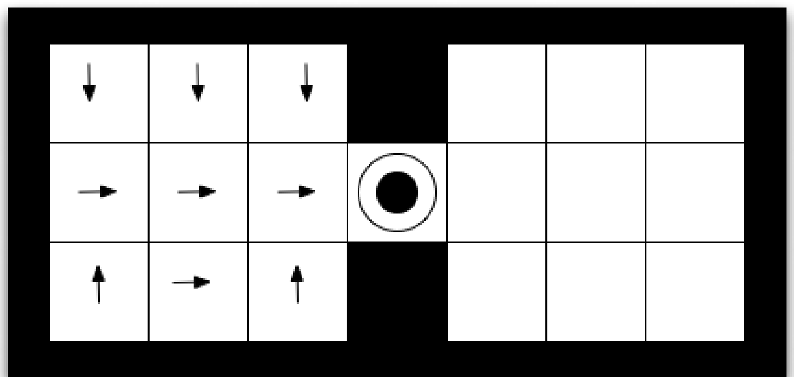
What about actions?

- Primitive action a can be represented by an option:

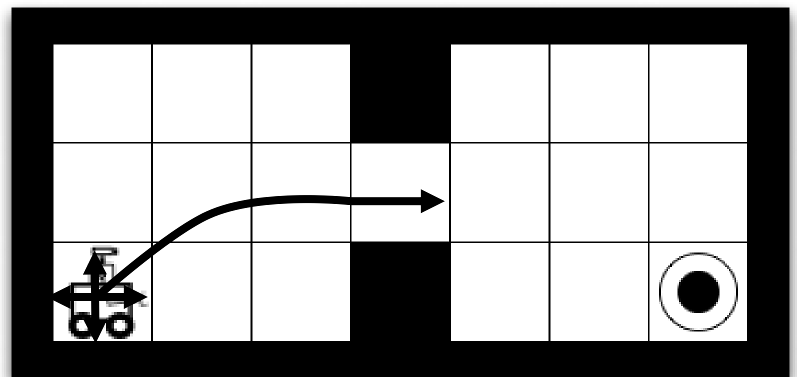
- $I_a = S$
- $\pi_a(s, b) = \begin{cases} 1, a = b \\ 0, a \neq b \end{cases}$
- $\beta_a(s) = 1 \; \forall s \in S$

- A primitive action can be executed anywhere, lasts exactly one timestep, and always chooses action a

Options as actions



Option



Set of decisions available

Questions…

- Given an MDP (S, A, R, T, γ)
  - Replace A with set of options O (some may be primitive actions)

- How do we characterise the resulting problem?
- How do we plan with options?
- How do we learn with options?
- How do we characterise resulting policies?

SMDPs

- Resulting problem is Semi-Markov Decision Process
  - (S, O, T, R, γ)

- S is the set of states
- O is the set of options
- $T = \Pr(s', t \mid o, s)$ is the transition model
- $R(s, o, s', t)$ is the reward function
- γ is the per-step discount factor

SMDPs

- Note
  - All times are integers
  - “Semi” means transitions can last $t > 1$ timesteps
  - Transition and reward functions involve time taken for option to execute

Planning with options

- Regular Bellman equation:

$$Q_\pi(s,a) = \sum_{s'} p(s'|s,a)[r(s,a,s') + \gamma Q_\pi(s',\pi(s'))]$$

- Bellman equation with options:

Immediate reward

$$\boxed{Q_\pi(s,o)} = \sum_{s',t} p(s',t|s,o)\boxed{[r(s,o,s',t)} + \boxed{\gamma^t Q_\pi(s',\pi(s'))]}$$

Value of o in s

Expected future value

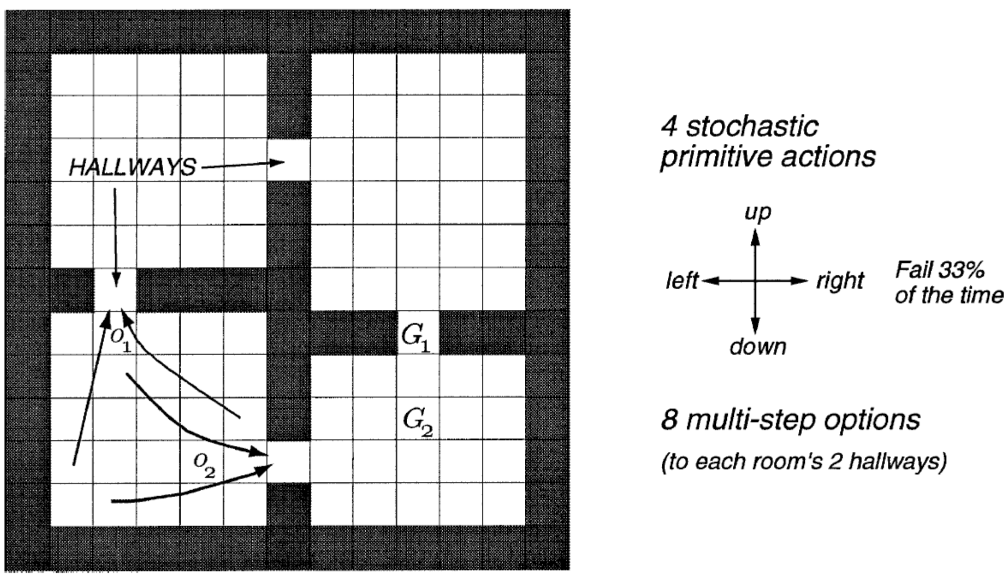
Learning and Planning

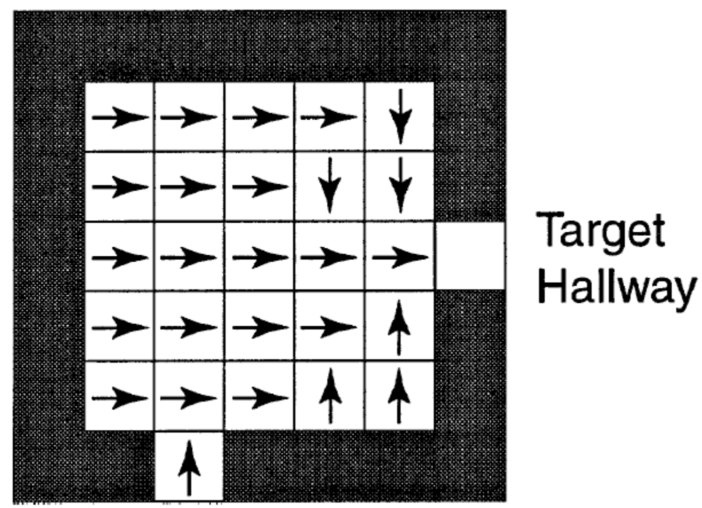
- For learning:
  - Collect stochastic samples
  - Use SMDP Bellman equation
- For planning:
  - Synchronous Value Iteration
  - Value Iteration using the SMDP Bellman Equation

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)]$$

$$Q(S_t, O_t) = Q(S_t, O_t) + \alpha[\sum_{i=1}^{k} \gamma^{i-1} R_{t+i} + \gamma^k \max_o Q(S_{t+k}, o) - Q(S_t, O_t)]$$

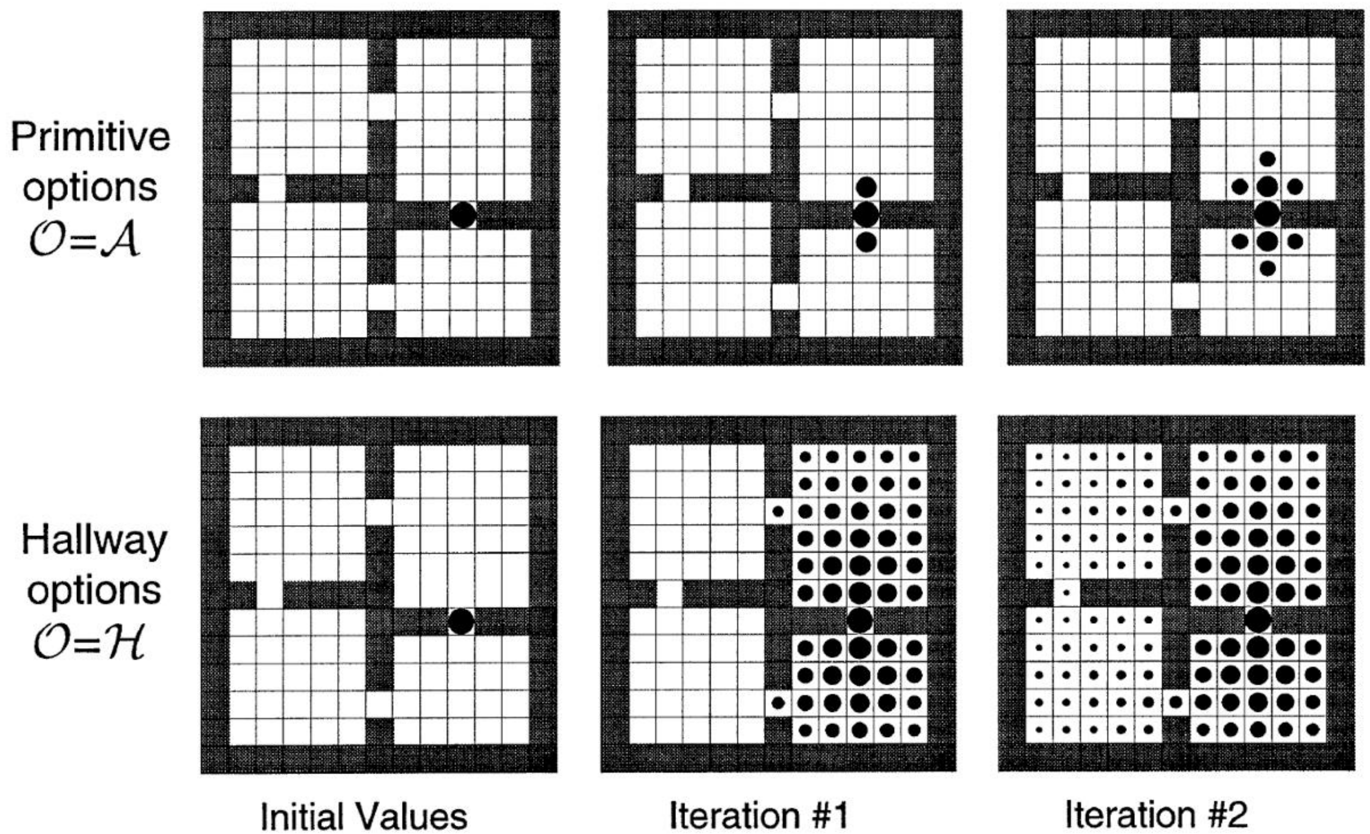
Example





(Sutton, Precup and Singh, AIJ 1999)

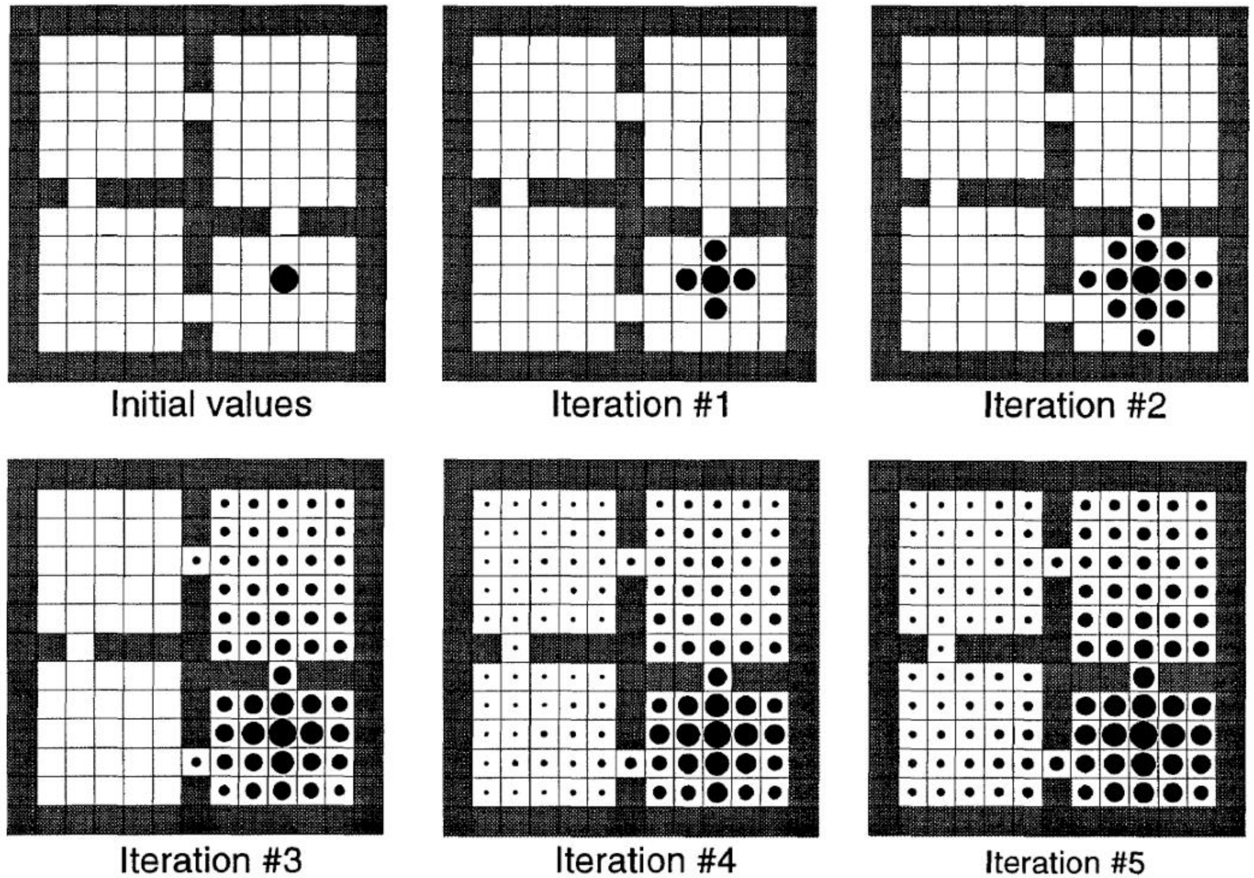
Example



(Sutton, Precup and Singh, AIJ 1999)

Example

Primitive and hallway options $\mathcal{O}=\mathcal{A}\cup\mathcal{H}$



(Sutton, Precup and Singh, AIJ 1999)

A note on policies

- A policy over an MDP with primitive actions is a *Markov policy*:

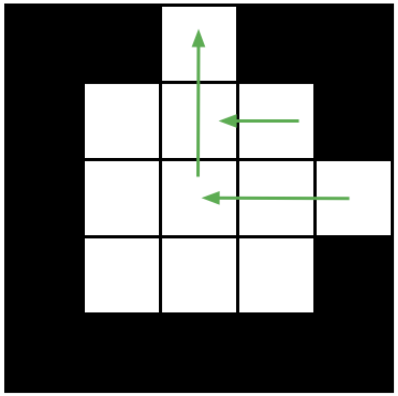
$$\pi: S \times A \rightarrow [0, 1]$$

- A policy over an SMDP with options could also be Markov:

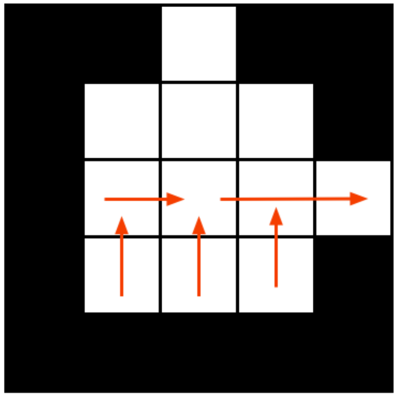
$$\pi: S \times O \rightarrow [0, 1]$$

- But the policy in the original MDP may not be
  - The probability of taking an action at a state depends on the option currently running.

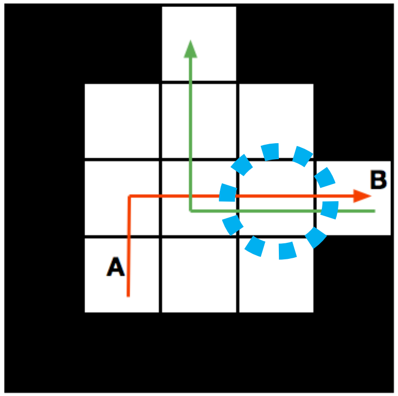
Example



Option A



Option B



Policy

Semi-Markov policies

- A Markov policy for an SMDP may result in a semi-Markov policy for the underlying MDP

- (Even if the options are Markov options!)

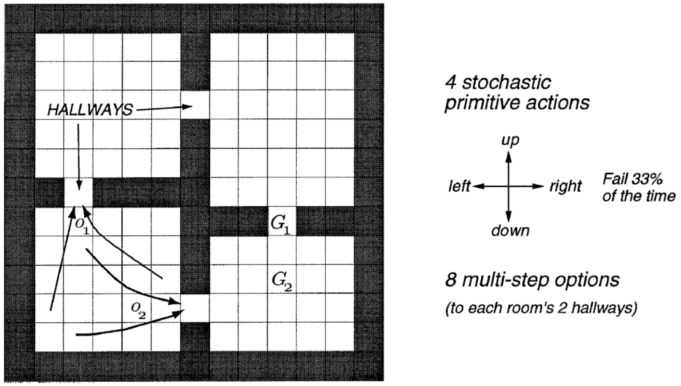
- *Here, semi-Markov means that the probability of taking a primitive action at each step depends on more than the current state*

Summary

- Original problem: MDP
- MDP + Options = SMDP
- Options framework allows us to both *express a* low-level policy, and plan and learn using the higher-level SMDP
- Additionally, the ability to:
  - Create new options
  - Update option policies
  - Learn with options
  - Interrupt them ...

What are skills for?

- Adding an option changes the connectivity of the MDP
- This affects:
  - Learning and planning
  - Exploration
  - State-visit distribution
  - Branching factor
  - Diameter!



(Sutton, Precup and Singh, AIJ 1999)

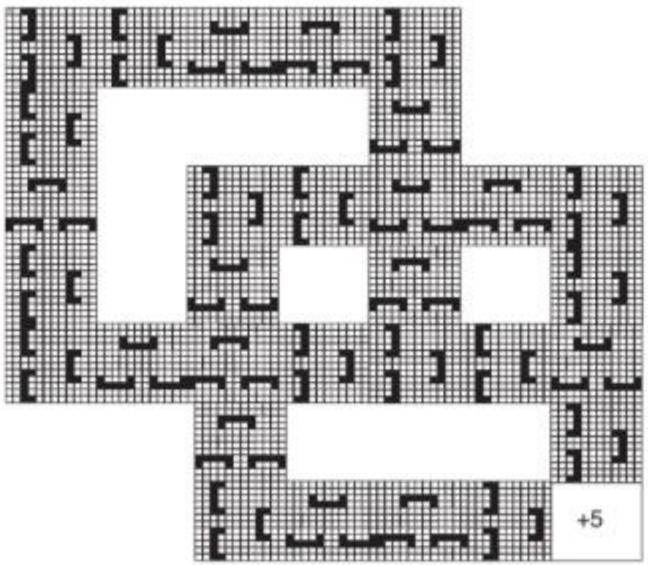
Where do options come from?

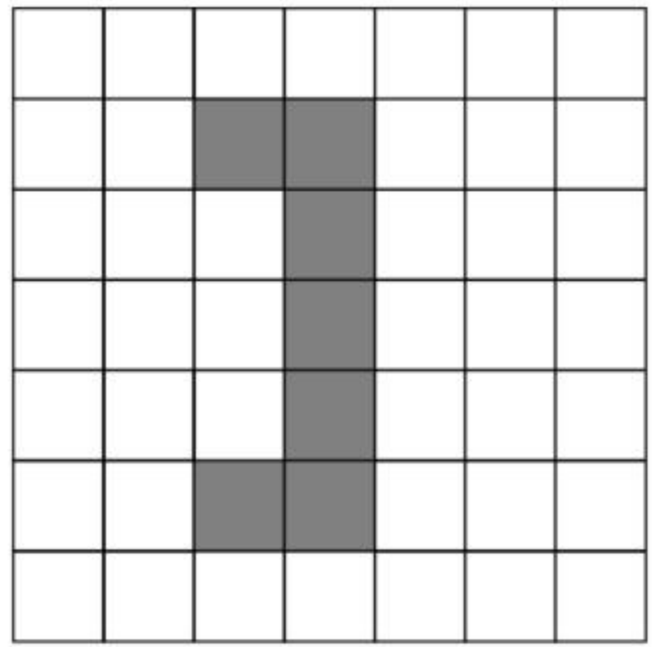
- Good question!
- Locate bottleneck states
  - Learn options to reach these subgoals [Simsek and Barto, 2008]
- Could be extracted from solution to existing tasks
  - NPBRS [Ranchod, Rosman, Konidaris, 2015]
- Options that minimise planning are NP-hard [Jinnai, et al, 2019]

Hierarchies of Abstract Machines (Parr, 1998)

- Policies of core MDP are defined as programs
- These execute based on state of MDP, as well as own internal state
- Programs modelled as finite state machines (FSMs)
- There are four machine states:
  - Action states: execute action in environment
  - Call states: execute another FSM as a subroutine
  - Choice states: stochastically select a next machine state
  - Stop states: halt execution and return control to previous machine

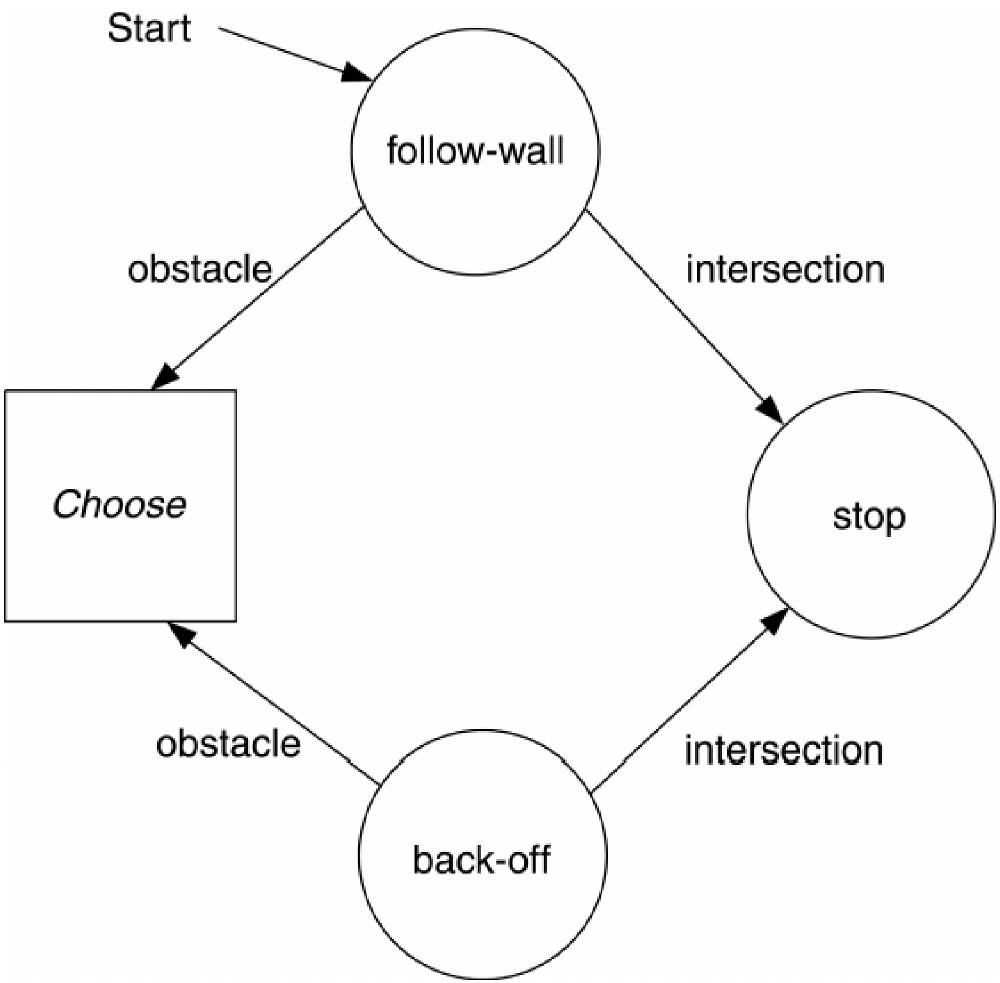
Maze Navigation

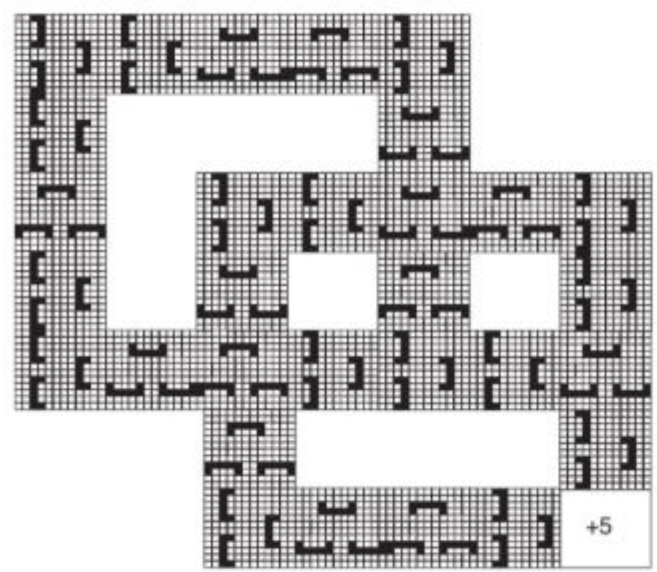




Environment

Example of a HAM





HAMs

- Action, call, choice, stop states
- New state space is HAM states × MDP state space
  - $\mathcal{H} \times S$ state space

- Action, call and stop states are predefined!
  - So learning happens only for the choice states!

- Can apply Q-learning to learn choice states for FSMs

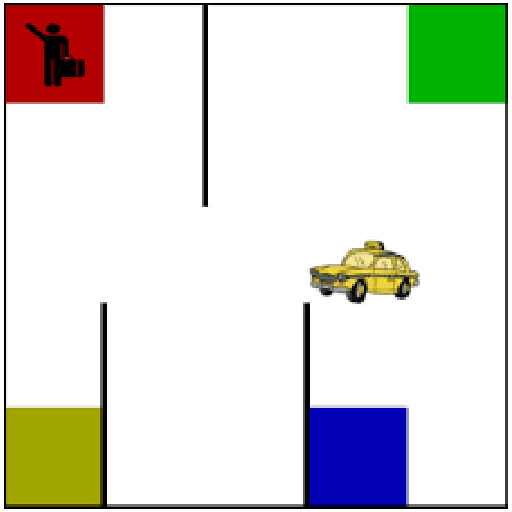
Discussion

- Restrict the set of possible policies
- Good way of injecting prior knowledge
- HAMs + MDP = SMDP
- Link between programming and control (Programmable HAMS, Stuart and Russell 2001)

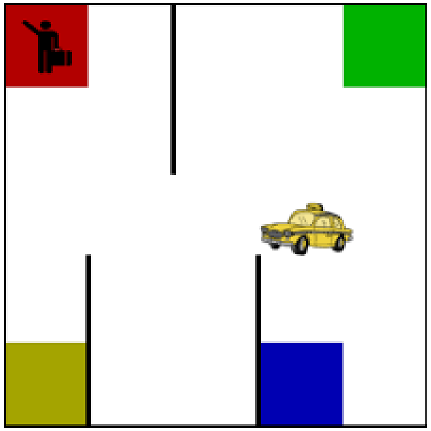
- No large scale applications to date

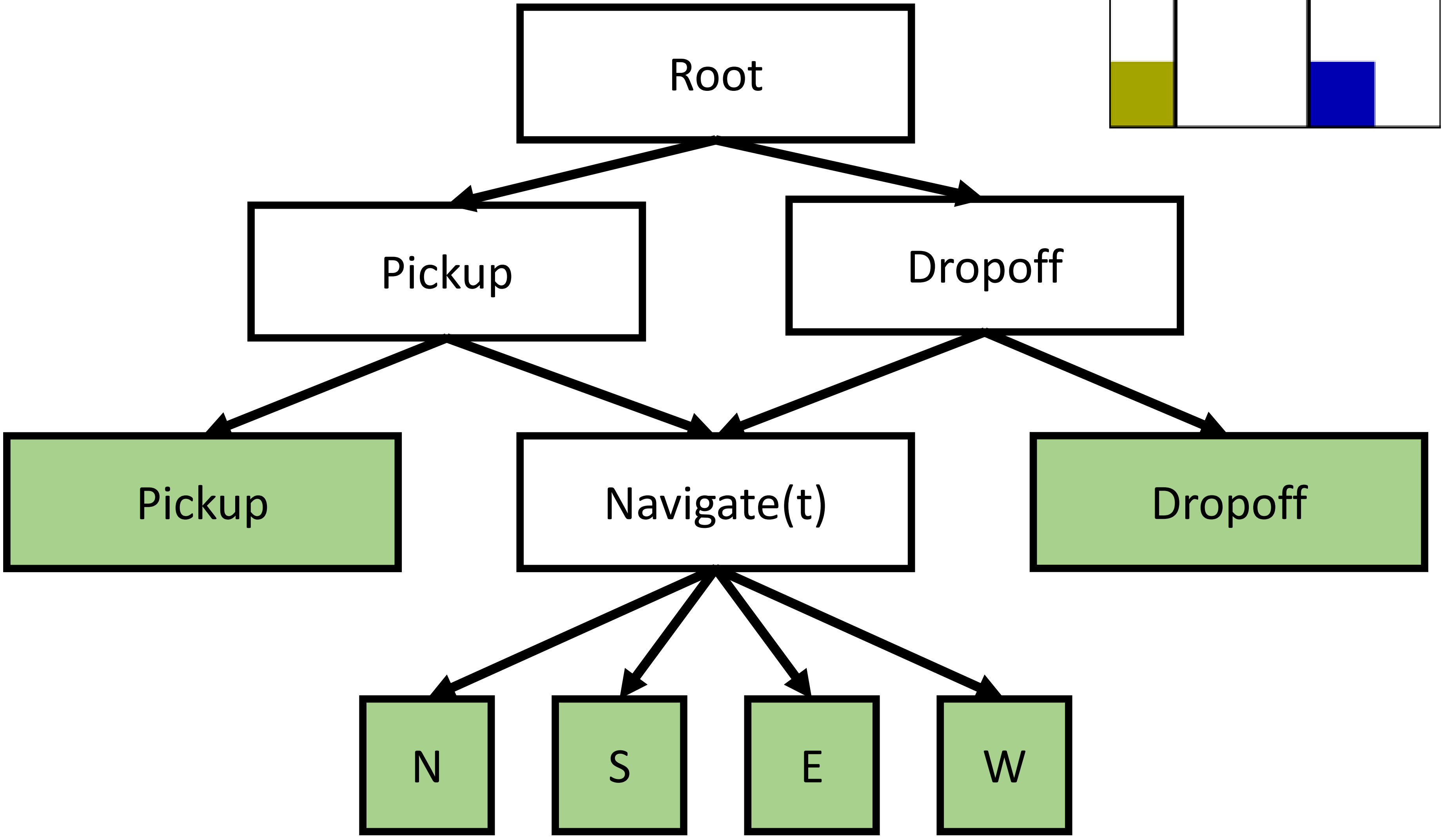
MAXQ value function decomposition [Dietterich, 2000]

- Taxi domain
  - 5x5 grid. Taxi can move N/S/E/W
  - Collect passenger at one of the colours
  - Drop off at one of the colours
  - -1 for each step, +10 for correct drop-off, -2 for invalid pickup, -5 for wrong dropoff

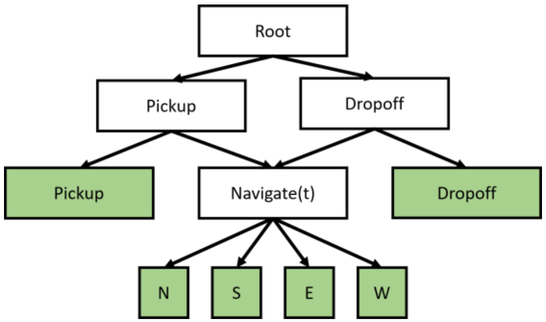


Decomposition





Formalising



- Let original base MDP be $M = (S, A, T, R)$
- Break down into subtasks: $M_0, M_1, \ldots, M_k$
- For each subtask, define $M_i = \{S_i, A_i, \bar{R}_i\}$
  - S_i: set of states where subtask is active
    - Terminal states are all states not in S_i
  - A_i: set of actions available
    - Primitive actions, and subtasks specified by DAG
  - $\bar{R}_i$: pseudo-reward associated with subtask
    - E.g. for Navigate(G), get reward for reaching G

    Even if the optimal policy was not to go to G!

Subtasks are SMDPs

- Each M_i is an SMDP with
  - State space S_i
  - Action space A_i
- Augment state space with K, a stack containing names and params of calling subtasks
  - Like call stack in programming languages
- Subtask's policy is Markov w.r.t. augmented state
- Transition probabilities are well-defined given policies of lower-level subtasks

MAXQ Discussion

- Real hierarchical decomposition of a task
- Easy reuse of sub-policies
- Very complex structure
- Learns recursively optimal policy
  - Policy for a parent task is optimal given the learnt policies of its children
  - Context-free!
- Recursively optimal policies may be highly suboptimal policies

Summary

- RL suffers from curse of dimensionality
- HRL seeks to decompose problems
- Can abstract states/actions
- Transfer between tasks
- Focus on general problem solving
  - Long lived agents

- Strong AI?!?!?!?!

Homework

- Complete assignments
- Get 100% on everything
- Apply for MSc and/or PhD
- Publish paper on RL topic ☺
- Build all the AIs!