Reinforcement Learning (INF11010)

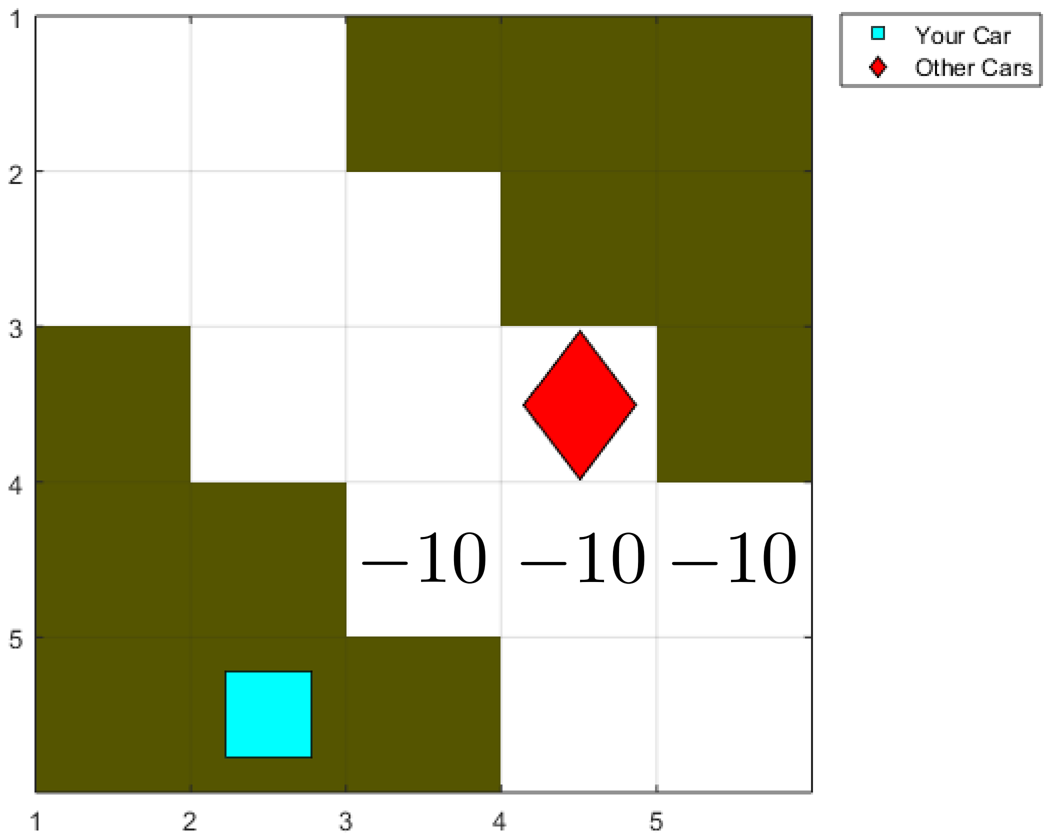
Lecture 12: Hierarchy and Abstraction

Pavlos Andreadis, March 20th 2018

Today's Content

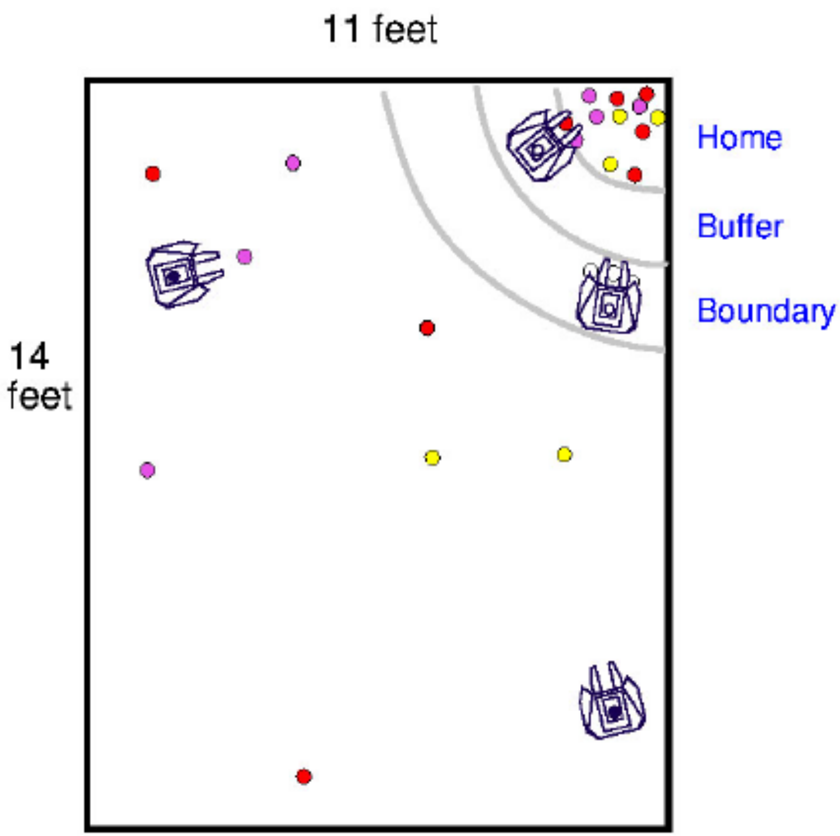
- Reward Shaping (briefly)
- Semi-Markov Decision Processes
- Options

Reward Shaping



- Speed up discovery of 'good behaviour'
- Can lead to suboptimal behaviour, in terms of original task (the one without shaping)

An Early Idea: Reward Shaping



- The robots' objective is to collectively find pucks and bring them home.
- Represent the 12-dim environment by state variables (features?):
  - have-puck?
  - at-home?
  - near-intruder?
- What should the immediate reward function be?

[Source: M. Mataric, Reward Functions for Accelerated Learning, ICML 1994]

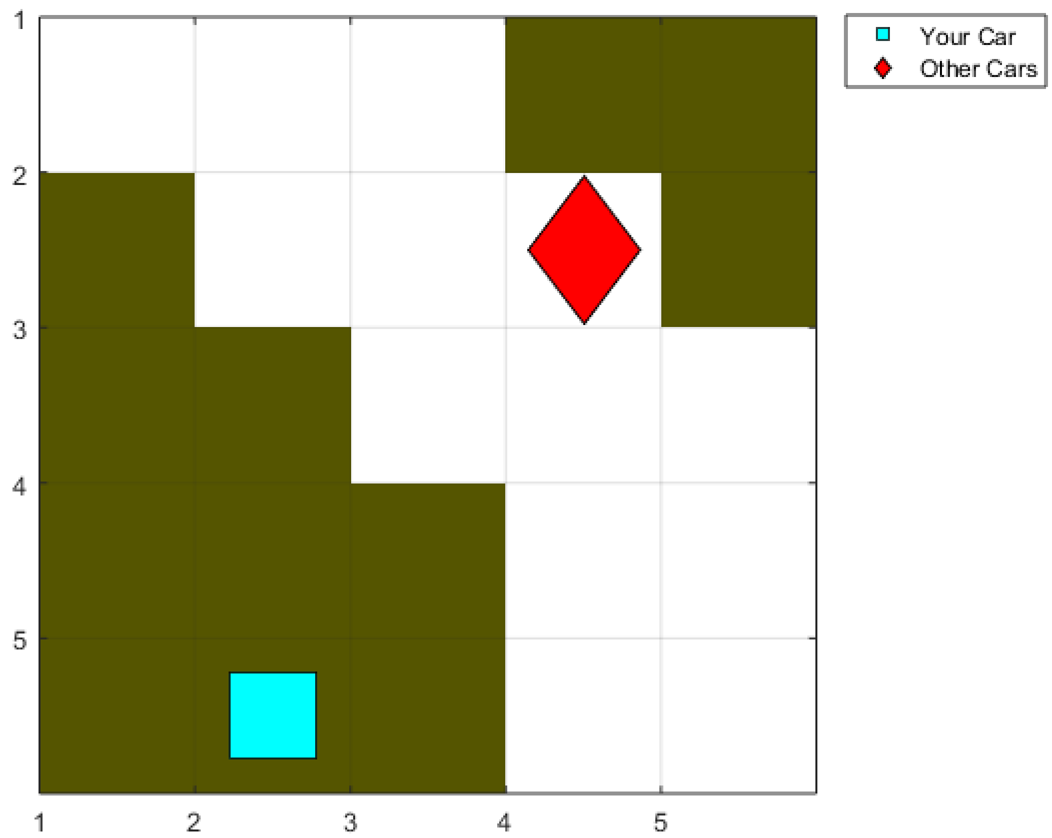
Reward Shaping

- If a reward is given only when a robot drops a puck at home, learning will be extremely difficult.
  - The delay between the action and the reward is large.

- Solution: Reward shaping (intermediate rewards).
  - Add rewards/penalties for achieving sub-goals/errors:
    - subgoal: grasped-puck
    - subgoal: dropped-puck-at-home
    - error: dropped-puck-away-from-home
- Add progress estimators:
  - Intruder-avoiding progress function
  - Homing progress function

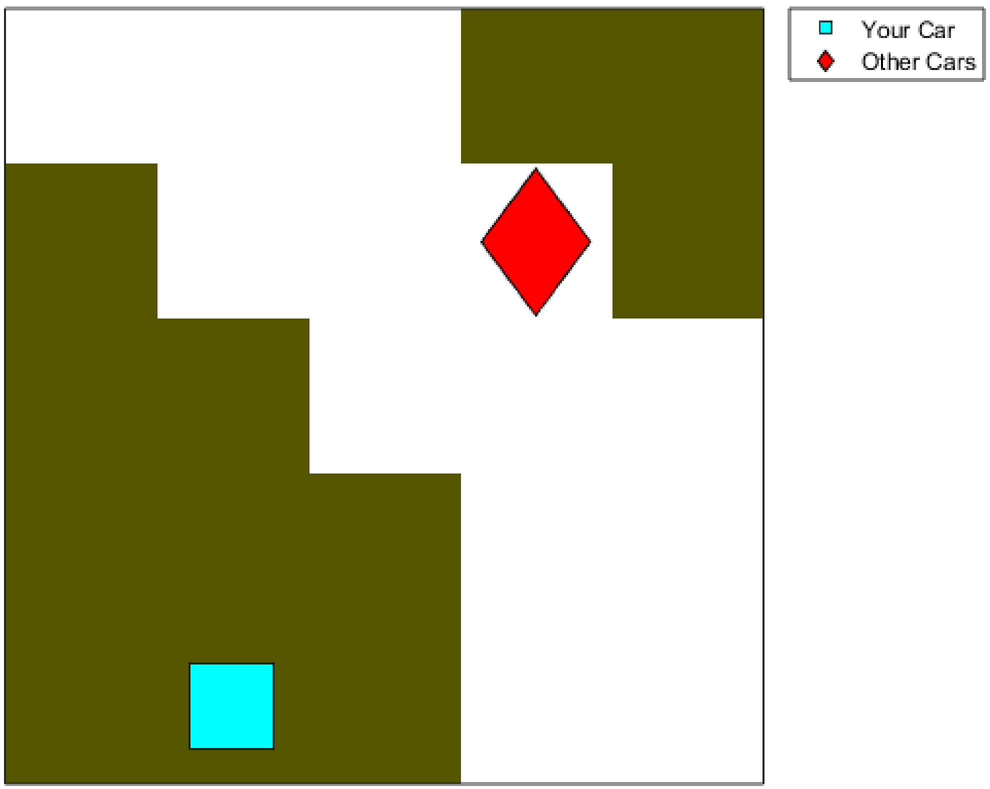
- Adding intermediate rewards will potentially allow RL to handle more complex problems.

Temporal Abstraction (discrete time)



- Take action UP until car appears directly in front of you (or termination).
- When car in front of you, initiate ‘manoeuvre’.
- In ‘manoeuvre’, take action UP_LEFT.

Temporal Abstraction (continuous time)



- Take action UP until car appears directly in front of you (or termination).
- When car in front of you, initiate ‘manoeuvre’.
- In ‘manoeuvre’, take action UP_LEFT.

Temporal Abstraction

- What's the issue?
  - Want "macro" actions (multiple time steps)
  - Advantages:
    - Avoid dealing with (exploring/computing values for) less desirable states
    - Reuse experience across problems/regions

- What's not obvious
  - Dealing with the Markov assumption
  - Getting the calculations right (e.g., stability and convergence)

Semi-Markov Decision Processes

Semi-Markov Decision Processes

- A generalization of MDPs:

  The amount of time between one decision and the next is a random variable (either real or integer valued)
- Treat the system as remaining in each state for a random waiting time
  - after which, transition to next state is instantaneous

- Real valued case: continuous time, discrete events
- Discrete case: Decisions only made an integer multiple of an underlying time step

Semi-Markov Decision Processes

- SMDP is defined in terms of

  P(*s*',τ|*s*,*a*): Transition probability (τ is the waiting time)

  R(*s*,*a*) or just r: Reward, amount expected to accumulate during waiting time, τ, in particular state and action

- Bellman equation can then be written down as, for all s:

$$V^*(s) = \max_{a \in \mathcal{A}_s}[r + \sum_{s',\tau} \gamma^\tau P(s', \tau | s, a) V^*(s')]$$

Note the need to sum over waiting time, as well.

Semi-Markov Decision Processes

- Likewise, we can write down the Bellman equation for the state-action value function as,

$$Q^*(s,a) = r + \sum_{s',\tau} \gamma^\tau P(s',\tau|s,a) \max_{a' \in \mathcal{A}_s} Q^*(s',a')$$

$$\forall s \in \mathcal{S}, a \in \mathcal{A}_s$$

- So, Dynamic Programming algorithms can be naturally extended to the case of SMDPs as well

Q-Learning with SMDPs

- Can we also modify sampling based algorithms accordingly?
- Consider the standard Q-learning algorithm, rewritten slightly in the following form,

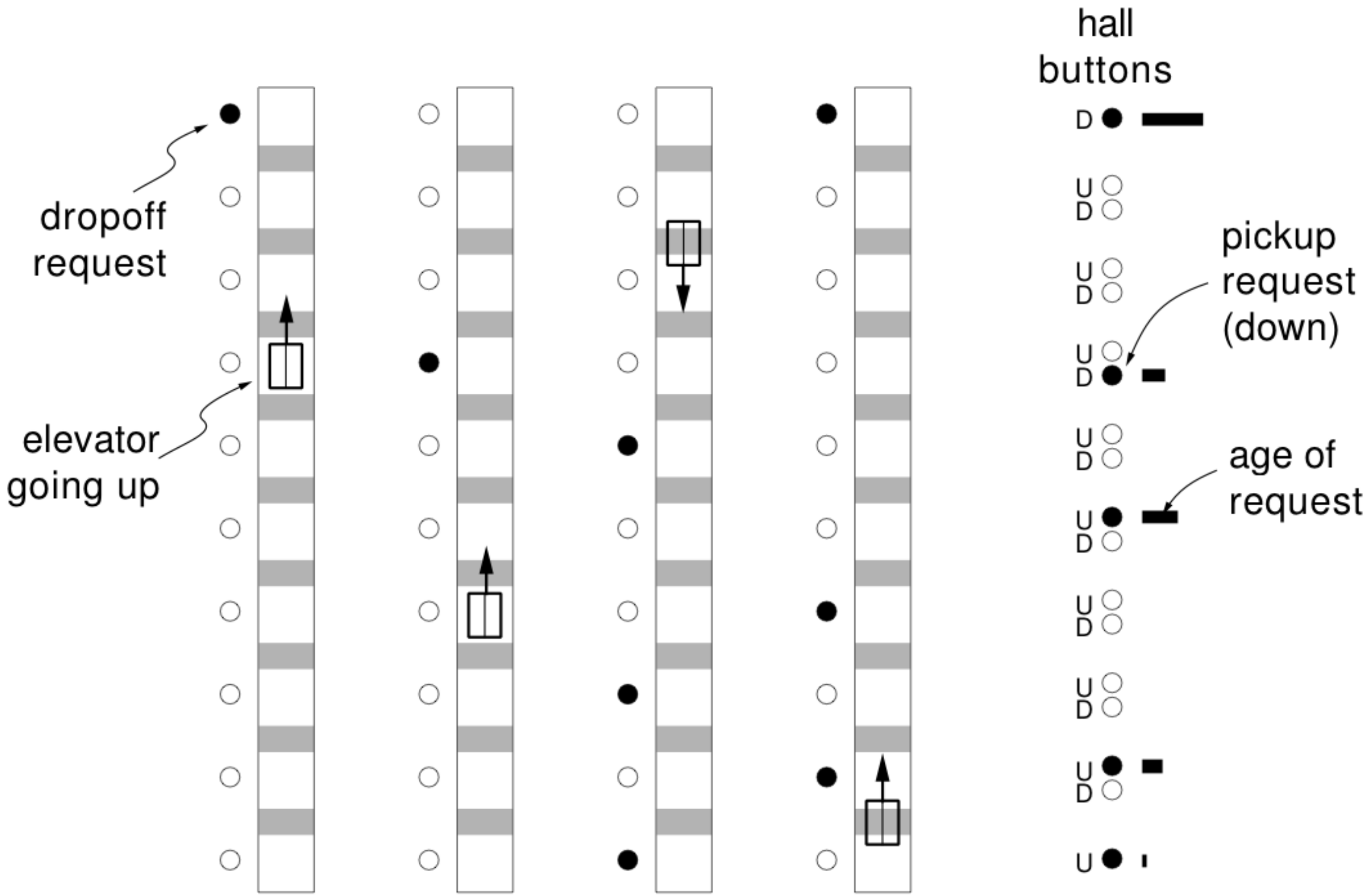
$$Q_{k+1}(s,a) = (1-\alpha_k)Q_k(s,a) + \alpha_k[r + \gamma \max_{a' \in \mathcal{A}_s} Q_k(s',a')]$$

- If we write down the reward sum, in brackets, for the entire waiting time duration, then we will have

$$\begin{aligned} Q_{k+1}(s,a) = (1-\alpha_k)Q_k(s,a) + \alpha_k[r_{t+1} + \gamma r_{t+2} + \ldots \\ + \gamma^{\tau-1} r_{t+\tau} + \gamma^{\tau} \max_{a' \in \mathcal{A}_s} Q_k(s',a')] \end{aligned}$$

Case Study: Elevator Dispatching

[Crites and Barto, 1996]



Semi-Markov Q-Learning

Continuous-time problem but decisions in discrete jumps.
For this SMDP, the expression for returns can be written as,

$$R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \quad \text{or} \quad R_t = \int_0^{\infty} e^{-\beta\tau} r_{t+\tau}\, d\tau$$

Note that the meaning of quantity r differs in the two expressions:

- reward at a discrete time step in discrete case
- reward "rate" in continuous case

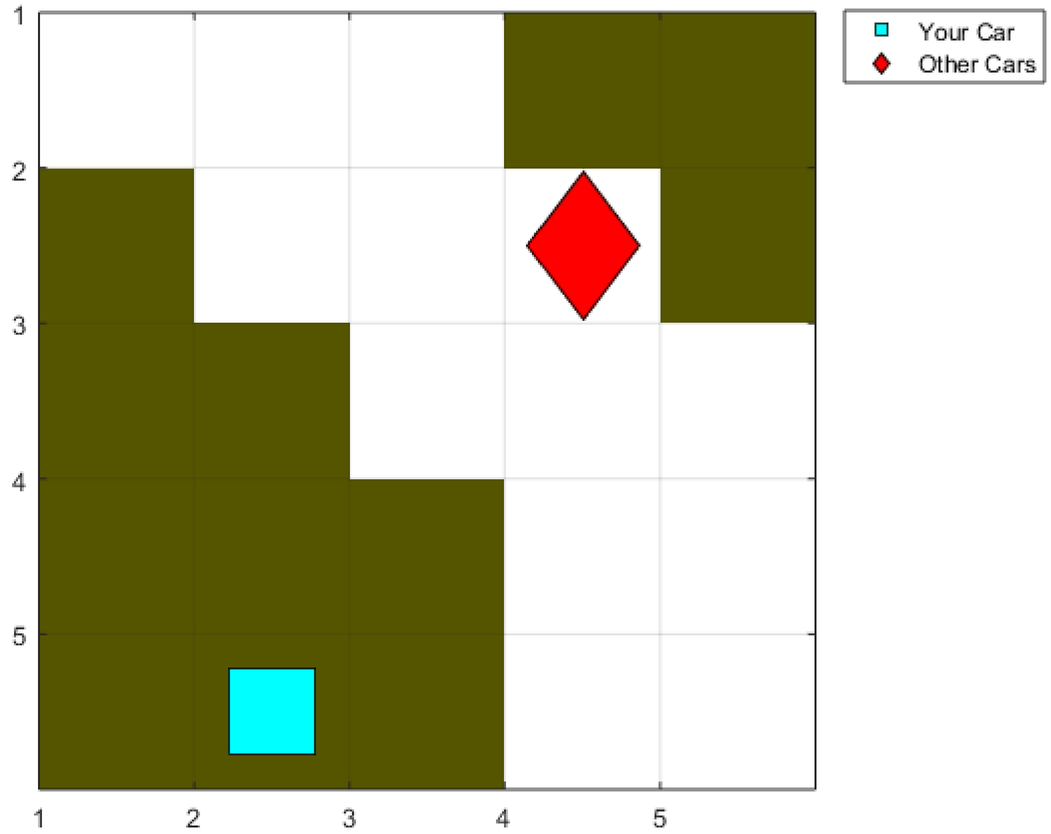
The negative exponential has a similar role as the discount factor as we have been using it so far.

Semi-Markov Q-Learning

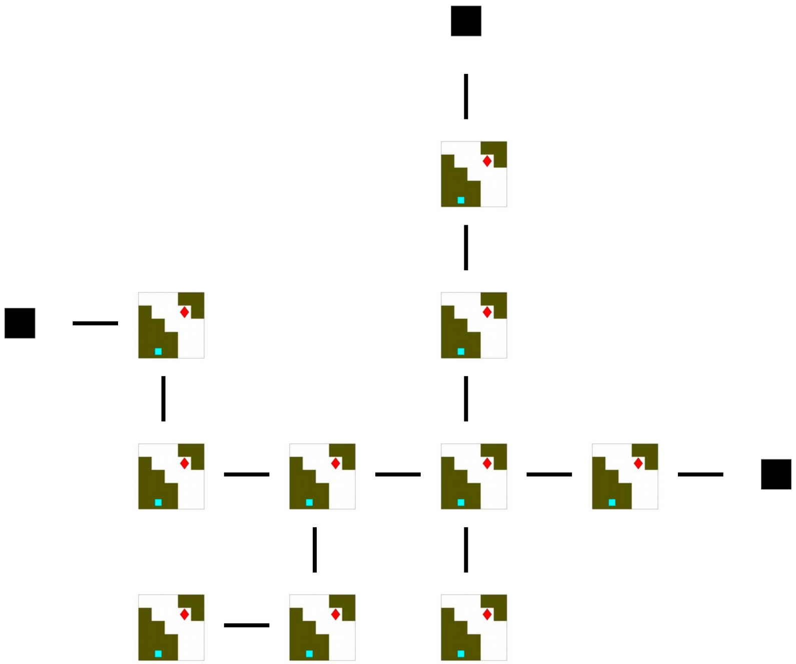
Suppose system takes action a from state s at time t_1, and next decision is needed at time t_2 in state s':

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha\left[\int_{t_1}^{t_2} e^{-\beta(\tau - t_1)} r_\tau \, d\boldsymbol{\tau} + e^{-\beta(t_2 - t_1)} \max_{a'} Q(s',a')\right]$$

Options



Option Hierarchies



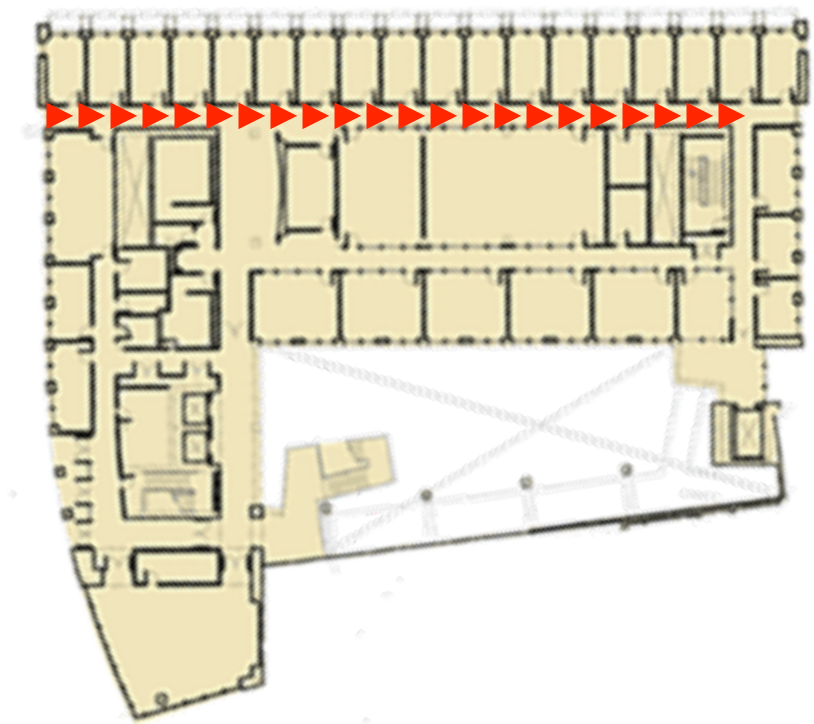
Options Framework

Options example: Move until end of hallway

Start : Any state in the hallway.

Execute : policy π as shown.

Terminate : when state s is the end of hallway.



Options can take variable number of steps

[Reference: R.S. Sutton, D. Precup, S. Singh, Between MDPs and Semi-MDPs: A framework for temporal Abstraction in reinforcement learning, Artificial Intelligence Journal 112:181-211, 1999.

Options [Sutton, Precup, Singh '99]

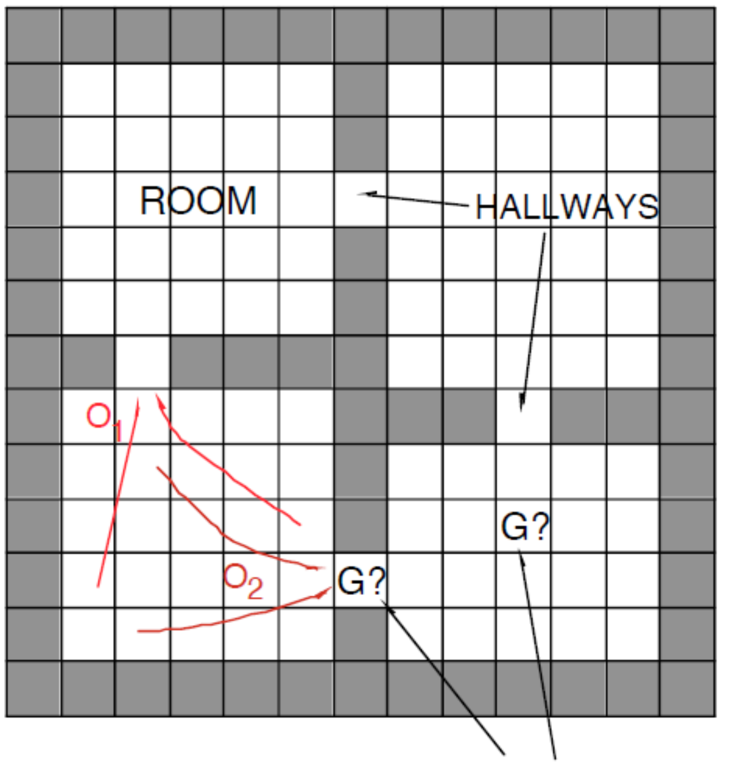
- An option is a *behaviour* defined in terms of:

  $o = \{ I_o, \pi_o, \beta_o \}$

- I_o : Set of states in which o can be initiated.
- $\pi_o(s)$: Policy (mapping S to A)[§] when o is executing.
- $\beta_o(s)$: Probability that o terminates in s.

[§]Can be a policy over lower level options.

Rooms Example

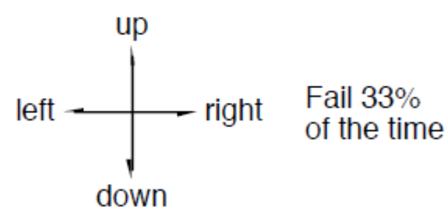


Goal states are given a terminal value of 1

4 rooms

4 hallways

4 unreliable primitive actions



8 multi-step options

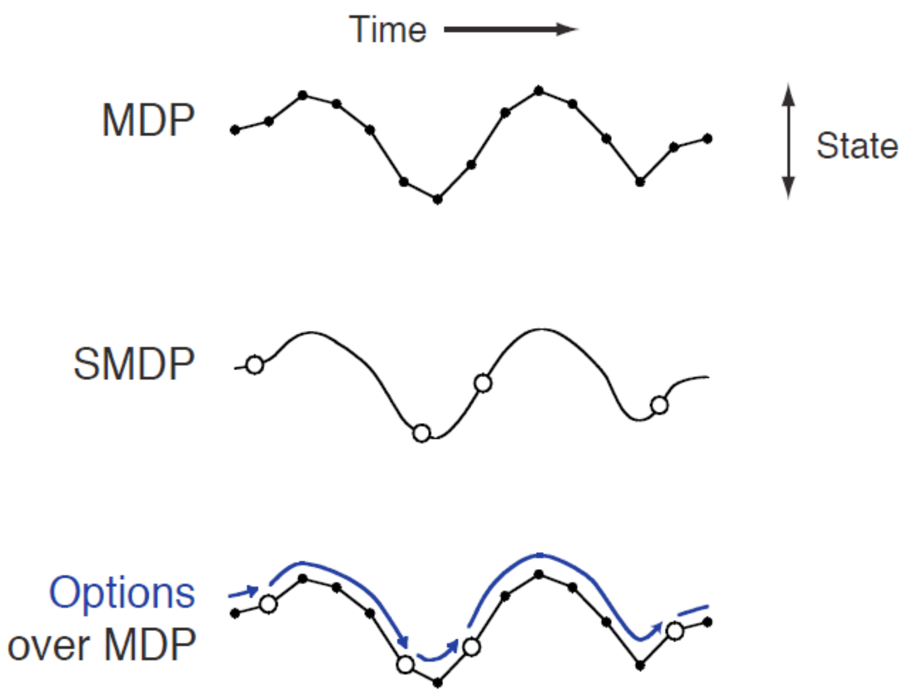
(to each room's 2 hallways)

Given goal location, quickly plan shortest route

All rewards zero

$\gamma = .9$

Options Define a Semi-MDP



Discrete time
Homogeneous discount

Continuous time
Discrete events
Interval-dependent discount

Discrete time
Overlaid discrete events
Interval-dependent discount

A discrete-time SMDP overlaid on an MDP
Can be analyzed at either level

MDP + Options = SMDP

Theorem:

For any MDP,
and any set of options,
the decision process that chooses among the options,
executing each to termination,
is an SMDP.

Thus all Bellman equations and DP results extend for value functions over options and models of options (cf. SMDP theory).

Why is this Useful?

- We can now define policy over options as well:

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

- And redefine all value functions appropriately:

$$V^{\mu}(s), Q^{\mu}(s, o), V_{O}^{*}(s), Q_{O}^{*}(s, o)$$

- All policy learning methods discussed so far, e.g., Value and Policy Iteration, can be defined over S and O
- Coherent theory of learning and planning, with courses of action at variable time scales, yet at the same level

Value Functions Over Options

We can write the expression for optimal value as,

$$V^*_{\mathcal{O}}(s) = \max_{\mu \in \prod(\mathcal{O})} V^{\mu}(s)$$

$$V^*_{\mathcal{O}}(s) = \max_{o \in \mathcal{O}_s} E\{r_{t+1} + ... + \gamma^{k-1} r_{t+k} + \gamma^k V^*_{\mathcal{O}}(s_{t+k}) | \mathcal{E}(o, s, t)\}$$

$$V^*_{\mathcal{O}}(s) = \max_{o \in \mathcal{O}_s} E[r + \gamma^k V^*_{\mathcal{O}}(s') | \mathcal{E}(o, s)]$$

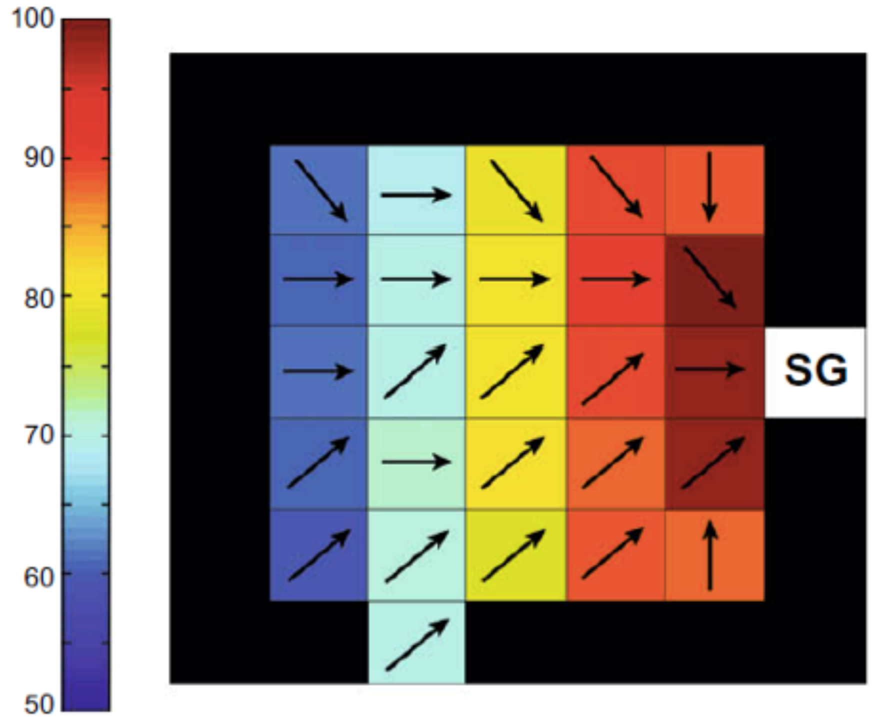
k being the duration of o when taken in s; conditioning is over the event that the option is initiated at that state and time.

Motivations for Options Framework

- Add temporally extended activities to choices available to RL agent, without precluding planning and learning at finer grained MDP level
- Optimal policies over primitives are not compromised due to addition of options
- However, if an option is useful, learning will quickly find this out – prevent prolonged and useless 'flailing about'

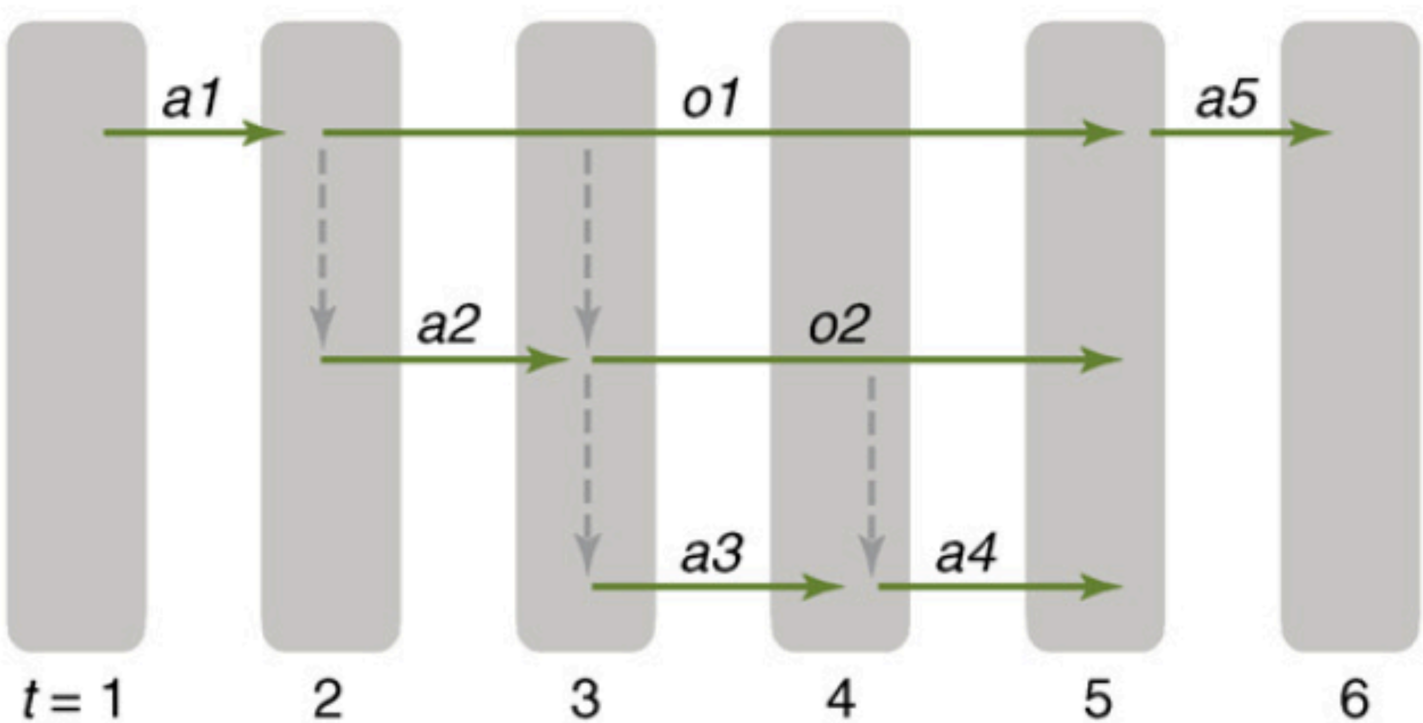
PS: If all options are 1-step, you recover the core MDP

Rooms Example –
Policy within One Room



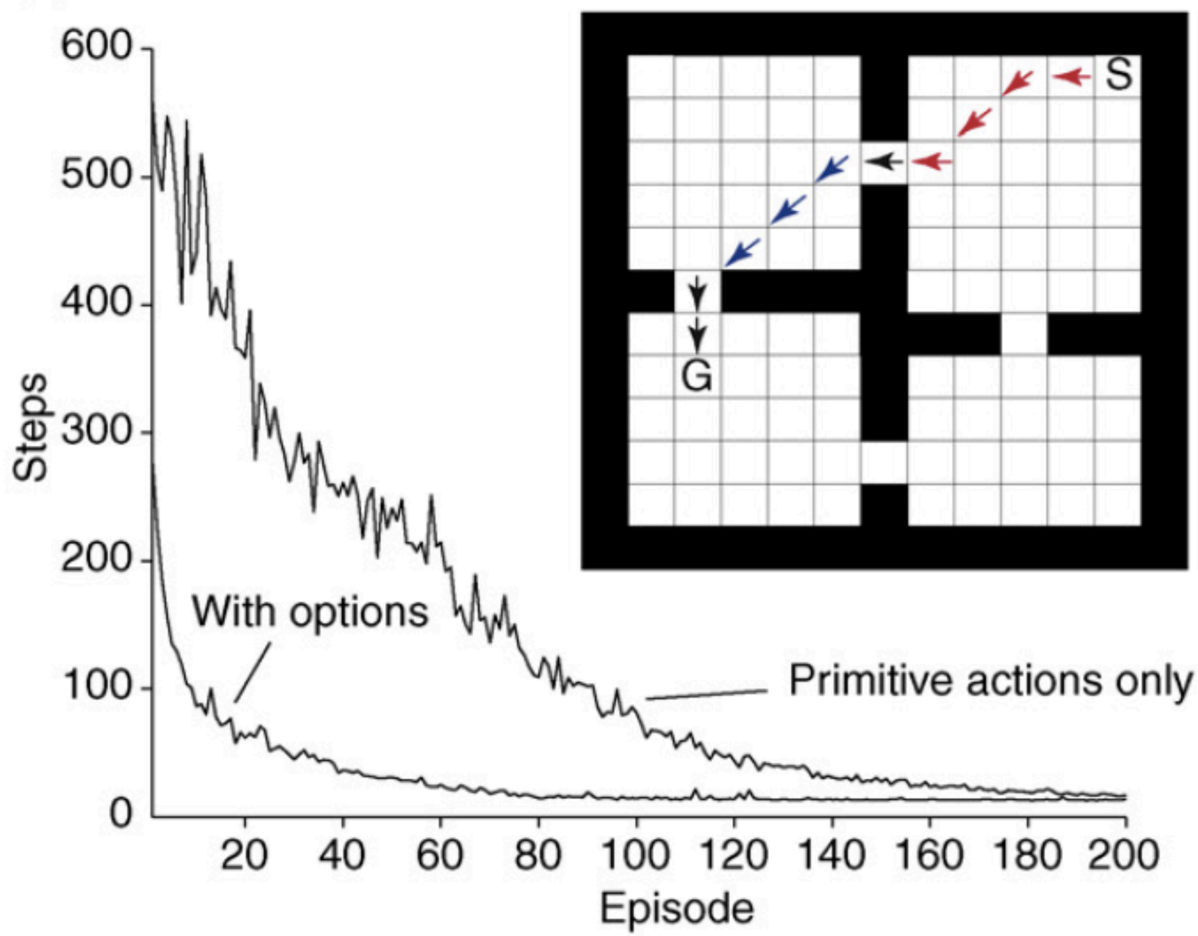
Colour = option specific value

Time Course of Use of Action/Option



[Source: M. Botvinick, Hierarchical models of behavior and prefrontal function, Trends in Cognitive Science 12(5), 2008]

Performance Improvement with Options



[Source: M. Botvinick, Hierarchical models of behavior and prefrontal function, Trends in Cognitive Science 12(5), 2008]

Summary

- Semi-MDPs for decisions at some points in time (with, discrete or continuous, time intervals)
  - All techniques learnt (DP, MC, TD) applicable with slight modifications to Bellman and Backup.
- Options for a hierarchical representation of available activities
  - Can use Semi-MDP theory to solve problems.

Reading +

- Case study, Sec 11.4 (Elevator Dispatching) in print version of S+B book

- Up to and including Sec 4.1 from A.G. Barto, S. Mahadevan, Recent Advances in Hierarchical Reinforcement Learning http://www.springerlink.com/content/tl1n705w7q452066/

Optional:

- R.S. Sutton, D. Precup, S. Singh, Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning, http://dx.doi.org/10.1016/S0004-3702(99)00052-1

Appendix

Elevator Dispatch

Problem Setup: Passenger Arrival Patterns

Up-peak and Down-peak traffic

- Not equivalent: down-peak handling capacity is much greater than up-peak handling capacity; so up-peak capacity is limiting factor.
- Up-peak easiest to analyse: once everyone is onboard at lobby, rest of trip is determined. The only decision is when to open and close doors at lobby. Optimal policy for pure case is: close doors when threshold number on; threshold depends on traffic intensity.
- More policies to consider for two-way and down-peak traffic.
- We focus on down-peak traffic pattern.

Various Extant Control Strategies

- Zoning: divide building into zones; park in zone when idle. Robust in heavy traffic.
- Search-based methods: greedy or non-greedy. Receding Horizon control.
- Rule-based methods: expert systems/fuzzy logic; from human "experts"
- Other heuristic methods: Longest Queue First (LQF), Highest Unanswered Floor First (HUFF), Dynamic Load Balancing (DLB)
- Adaptive/Learning methods: NNs for prediction, parameter space search using simulation, DP on simplified model, non-sequential RL

The Elevator Model (Lewis, 1991)

Discrete Event System: continuous time, asynchronous elevator operation

Parameters:

- Floor Time (time to move one floor at max speed): 1.45 secs.
- Stop Time (time to decelerate, open and close doors, and accelerate again): 7.19 secs.
- TurnTime (time needed by a stopped car to change directions): 1 sec.
- Load Time (the time for one passenger to enter or exit a car): a random variable with range from 0.6 to 6.0 secs, mean of 1 sec.
- Car Capacity: 20 passengers

Traffic Profile:

- Poisson arrivals with rates changing every 5 minutes; down-peak

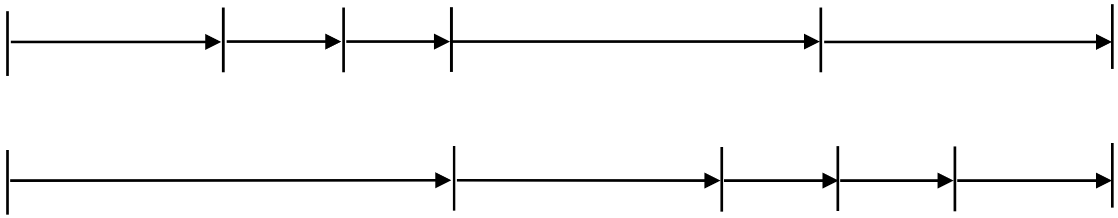
State Space

- 18 hall call buttons: 2^{18} combinations
- positions and directions of cars: 18^4 (rounding to nearest floor)
- motion states of cars (accelerating, moving, decelerating, stopped, loading, turning): 6
- 40 car buttons: 2^{40}
- Set of passengers waiting at each floor, each passenger's arrival time and destination: unobservable. However, 18 real numbers are available giving elapsed time since hall buttons pushed; we discretize these.
- Set of passengers riding each car and their destinations: observable only through the car buttons

Conservatively about 10^{22} sates

Actions

- When moving (halfway between floors):
  - stop at next floor
  - continue past next floor
- When stopped at a floor:
  - go up
  - go down
- Asynchronous



Constraints

standard

- A car cannot pass a floor if a passenger wants to get off there
- A car cannot change direction until it has serviced all onboard passengers traveling in the current direction
- Don't stop at a floor if another car is already stopping, or is stopped, there

special heuristic

- Don't stop at a floor unless someone wants to get off there
- Given a choice, always move up

→ Stop and Continue

Performance Criteria

Minimize:

- Average wait time
- Average system time (wait + travel time)
- % waiting > T seconds (e.g., T = 60)
- Average squared wait time (to encourage fast and fair service)

Average Squared Wait Time

Instantaneous cost, p individuals**:**

$$r_\tau = \sum_p \left(\text{wait}_p(\tau)\right)^2$$

Define return as an integral rather than a sum (Bradtke and Duff, 1994):

$$\int_0^\infty e^{-\beta\tau} r_\tau \, d\tau$$

Computing Rewards

Must calculate

$$\int_0^\infty e^{-\beta(\tau - t_s)} r_\tau \, d\tau$$

- "Omniscient Rewards": the simulator knows how long each passenger has been waiting.
- "On-Line Rewards": Assumes only arrival time of first passenger in each queue is known (elapsed hall button time); estimate arrival times

Neural Networks

47 inputs, 20 sigmoid hidden units, 1 or 2 output units

Inputs:

- 9 binary: state of each hall down button
- 9 real: elapsed time of hall down button if pushed
- 16 binary: one on at a time: position and direction of car making decision
- 10 real: location/direction of other cars: "footprint"
- 1 binary: at highest floor with waiting passenger?
- 1 binary: at floor with longest waiting passenger?
- 1 bias unit ≡ 1

Elevator Results