RL 9: RL Algorithms and State Abstraction

Michael Herrmann

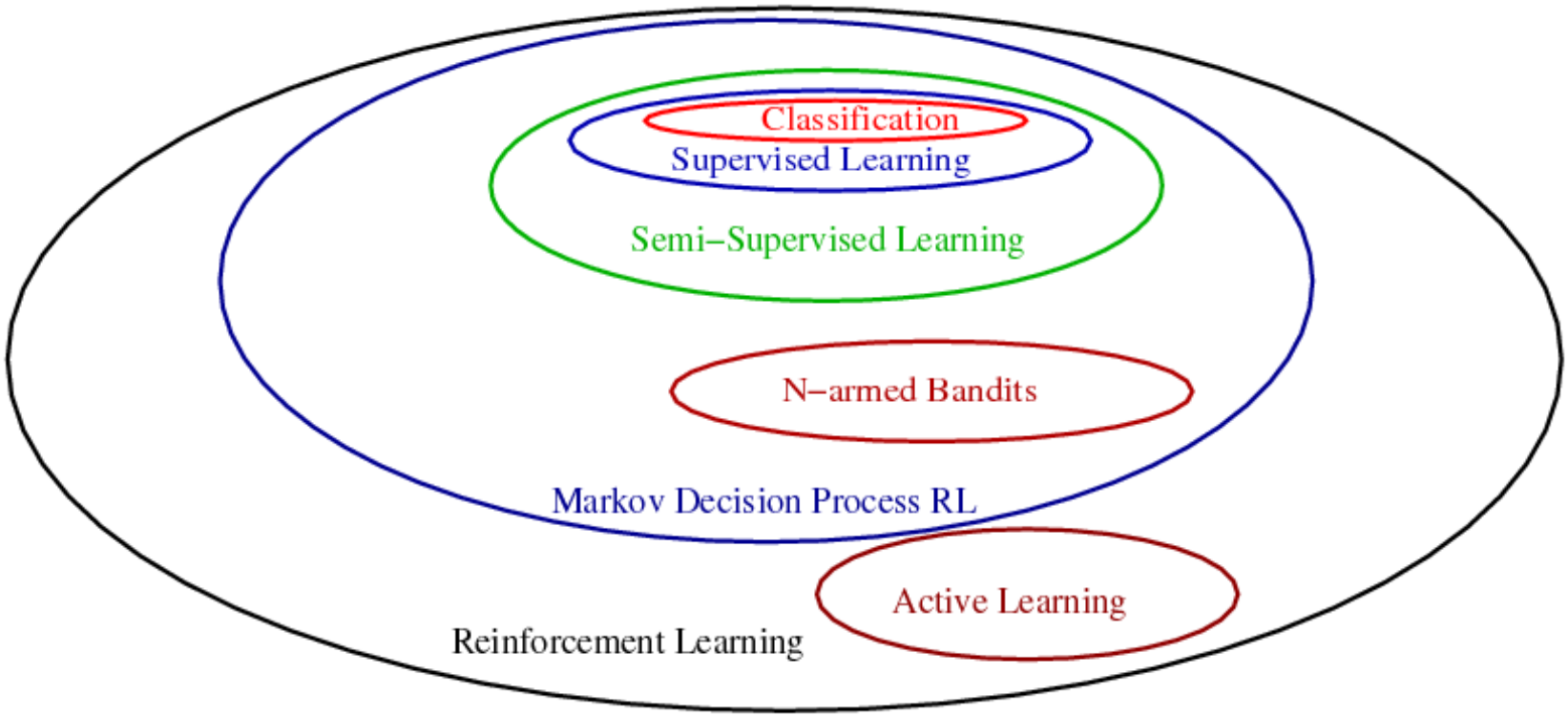
University of Edinburgh, School of Informatics

11/02/2014

Overview

- Hierarchical RL
- Temporal abstraction
- Options
- Semi Markovian Decision Problems (SMDP)
- The elevator example
- A brief look at MAXQ (Dietterich) [next time]

Reinforcement Learning



Understanding a problems as an RL problem is beginning to solve it

(John Langford)

Algorithms

- Certainty equivalence
- TD(0)
- R-learning
- $\mathcal{Q}$-learning
- SARSA
- Actor-critic

"Certainty equivalence"

- Consider trajectory: s_0, a_0, r_1, s_1, a_1, r_2, s_2, a_2, r_3 s_3, a_3, r_4,s_4, a_4, r_5, ...
- Estimate the underlying MDP
  - transition $S \times A \to S$,

$$P(s|s_t, a_t) \approx \frac{\#\{s = s_{t+1}|s_t, a_t\}}{\#\{s_t, a_t\}}$$

  - reward $S \times A \times S \to R \in \mathbb{R}$,

$$P(r|s_t, a_t, s_{t+1}) \approx \frac{\#\{r = r_{t+1}|s_t, a_t, s_{t+1}\}}{\#\{s_t, a_t, s_{t+1}\}}$$

- Assume that the MDP is reconstructed with certainty
- Compute the optimal policy for the MDP
- Critical question: How do we generate the trajectory?

On-policy vs off-policy

On-policy

- Start with a simple policy
- Sample state space with this policy
- Improve policy
- May approach local minima

Off-policy

- Gather information from (partially) random moves
- Can incorporate exploration
- Slowly reduce randomness (practically)

TD

- Does not learn the underlying MDP, but learns the value function directly
- TD is on-policy, i.e. the resulting value function depends on policy
- Information from sampling the value function is not used immediately to improve the policy
- Can be used with policy or value iteration.

R-learning

Off-policy algorithm, in particular for non-discounted, non-episodic problems

Consider average reward $\rho = \lim_{n\to\infty} \frac{1}{n} \sum_{t=1}^{n} E\left[r_t\right]$

Value is define here as "above average":

$$\tilde{V}\left(s_t\right) = \sum_{k=1}^{\infty} E\left[r_{t+k} - \rho\right]$$

$$\tilde{\mathcal{Q}}\left(s_t, a_t\right) = \sum_{k=1}^{\infty} E\left[r_{t+k} - \rho | s_t = s, a_t = a\right]$$

Relative value function (relative to the average)

ρ is adapted and measures (average) success

Implies a different concept of optimality in non-episodic tasks

R-learning: Algorithm

1. Initialise ρ and $\mathcal{Q}(s,a)$
2. Observe s_t and choose a_t (e.g. ε-greedy), execute a_t
3. Observe r_{t+1} and s_{t+1}
4. Update

$$\mathcal{Q}_{t+1}(s_t,a_t) = (1-\eta)\,\mathcal{Q}_t(s_t,a_t) + \eta\left(r_{t+1} - \rho_t + \max_a \mathcal{Q}_t(s_{t+1},a)\right)$$

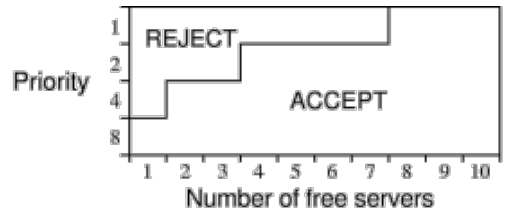
5. If $\mathcal{Q}(s_t,a_t) = \max_a \mathcal{Q}(s_t,a)$ then

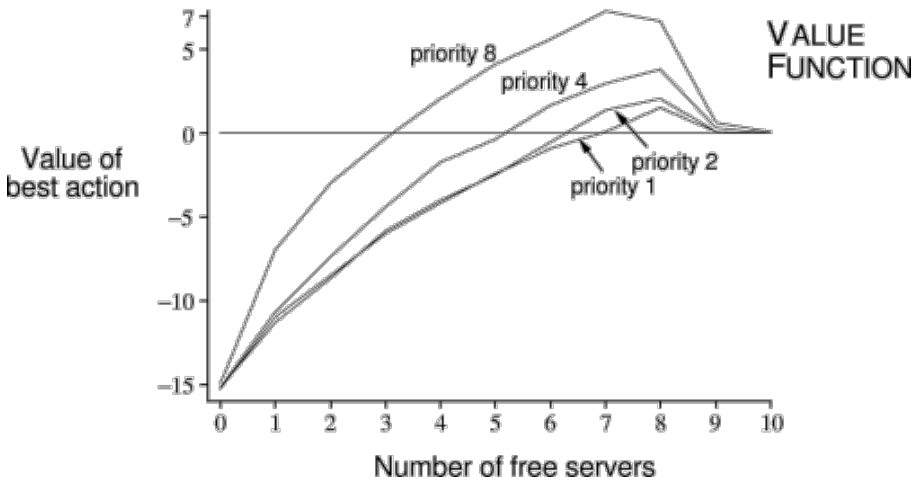
$$\rho_{t+1} = (1-\alpha)\,\rho_t + \alpha\left(r_{t+1} + \max_a \mathcal{Q}_t(s_{t+1},a_{t+1}) - \max_a \mathcal{Q}_{t+1}(s,a)\right)$$

Hint: Choose $\eta \gg \alpha$

R-learning example: Access-control queuing task

- Customers pay 1, 2, 4, or 8 (this is a reward) of four different priorities to be served
- States are the number of for free servers
- Actions: customer at the head of the queue is either served or rejected (and removed from the queue)
- Proportion of high priority customers in the queue is $h = 0.5$





- Busy server becomes free with prob. $p = 0.06$ (p and h are not known to the algorithm) on each time step

http://webdocs.cs.ualberta.ca/~sutton/book/ebook/node67.html

SARSA and $\mathcal{Q}$-learning

- SARSA and $\mathcal{Q}$-learning can be represented as look-up tables
- Policy is derived form current state-action value estimates

$\mathcal{Q}$-learning:

$$\mathcal{Q}_{t+1}(s_t, a_t) = (1-\eta)\,\mathcal{Q}_t(s_t, a_t) + \eta\left(r_{t+1} + \gamma \max_a \mathcal{Q}_t(s_{t+1}, a)\right)$$

SARSA:

$$\mathcal{Q}_{t+1}(s_t, a_t) = (1-\eta)\,\mathcal{Q}_t(s_t, a_t) + \eta\left(r_{t+1} + \gamma \mathcal{Q}_t(s_{t+1}, a_{t+1})\right)$$

- SARSA: $a_t = \pi(s_t)$, π is not necessary the $\arg\max_a \mathcal{Q}(s_t)$
- $\mathcal{Q}$-learning: $V(s_{t+1}) = \max_a \mathcal{Q}(s_{t+1}, a)$, a_t can be anything
- SARSA is on-policy, $\mathcal{Q}$-learning is off-policy
- SARSA like TD but for state-action pairs, i.e. can learn policy and value function simultaneously

Actor-Critic Methods

- Policy (actor) is represented independently of the (state) value function (critic)
- A number of variants exist, in particular among the early reinforcement learning algorithms

Advantages[1]

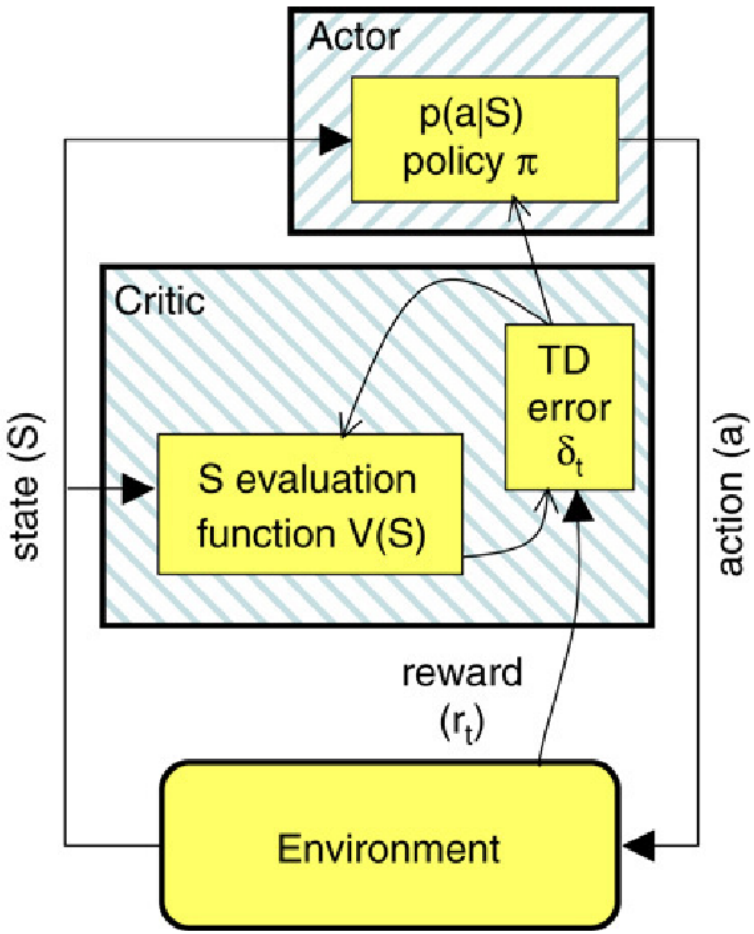
- AC methods require minimal computation in order to select actions which is beneficial in continuous cases, where search becomes a problem.
- They can learn an explicitly stochastic policy, i.e. learn the optimal action probabilities. Useful in competitive and non-Markov cases[2].

[1] Mark Lee following Sutton&Barto

[2] see, e.g., Singh, Jaakkola, and Jordan, 1994

Actor-Critic Methods

- Actor aims at improving policy (adaptive search element)
- Critic evaluates the current policy (adaptive critic element)
- Learning is based on the TD error δ_t
- Reward only known to the critic
- Critic should improve as well



Example: Policies for the inverted pendulum

- Exploitation (**actor**): Escape from low-reward regions as fast as possible
- aim at max. r
- e.g. Inverted pendulum task: Wants to stay near the upright position
- preferentially greedy and deterministic

- Exploration (**critic**): Find examples where learning is optimal
- aim at max. δ
- e.g. Inverted pendulum task: Wants to move away from the upright position
- preferentially non-deterministic

State Abstraction

Abstraction in Learning and Planning

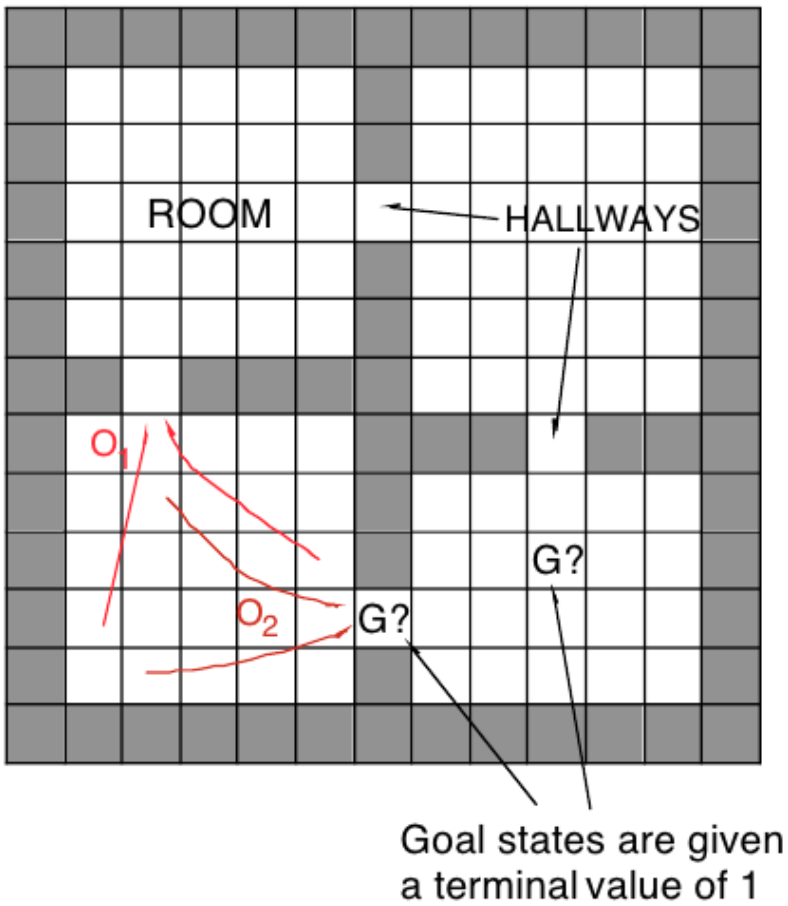
- General problem in AI: Semantics of abstract knowledge
- Reduction of complexity by exploiting task structure
- How can different levels of abstraction be related?
  - spatial: states
  - temporal: time scales
- Environmentally implied or repeated action trajectories
- Options are also called: Skills, macros, temporally abstract actions (Sutton, McGovern, Dietterich, Barto, Precup, Singh, Parr, ...)
- In other contexts also: Behavioural primitives, (elementary) behaviours, schemata

Options

Sequences of actions that follow a common theme but are not of fixed lengths

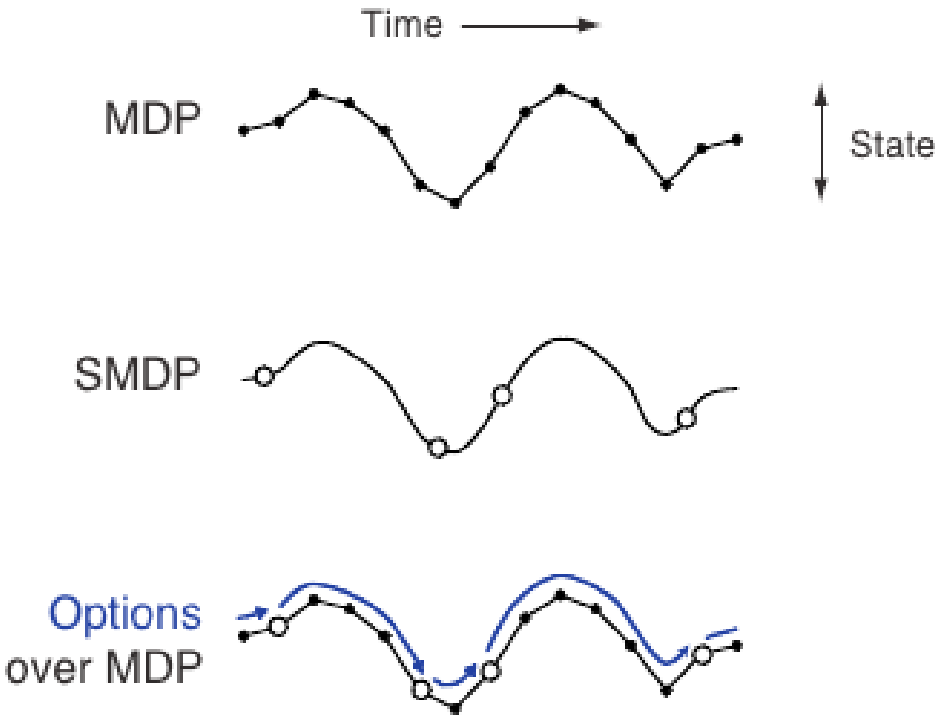
- $o = \langle \mathcal{I}, \pi, \beta \rangle$ call-and-return option
  - $\mathcal{I} \subseteq \mathcal{S}$ set of starting states
  - $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$ probabilistic policy to be follow during o
  - $\beta : \mathcal{S} \to [0, 1]$ probability to terminate in each state
- Example: Docking of a robot
  - $\mathcal{I}$: all states in which charges is in sight
  - π: approach charger if recharging-necessary-bit is set
  - β: terminate when docked or charger not visible

Exploiting the structure of the environment: Rooms example



- 4 rooms
- 4 hallways
- 4 unreliable primitive actions: up, left, right, down; fail 33% of the time
- 8 multi-step options (to each room's 2 hallways)
- Given goal location, quickly plan shortest route
- All rewards zero
- $\gamma = 0.9$

Options and Semi-Markov Decision Processes (SMDP)



- State Discrete time
  Homogeneous discount
- Continuous time
  Discrete events
  Interval-dependent discount
- Discrete time
  Overlaid discrete events
  Interval-dependent discount

Discrete-time SMDP overlaid on MDP (Can be analysed at either level)

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

Value Functions for Options

Value functions for options can be defined similar to the MDP case

$$V^{\mu}(s) = \mathbb{E}\{r_{t+1} + \gamma r_{t+2} + \ldots | \mu, s, t\}$$

$$\mathcal{Q}^{\mu}(s, o) = \mathbb{E}\{r_{t+1} + \gamma r_{t+2} + \ldots | o, \mu, s, t\}$$

Consider policies $\mu \in \Pi(\mathcal{O})$ that can choose only among options

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

$$\mathcal{Q}^{*}_{\mathcal{O}}(s, o) = \max_{\mu \in \Pi(\mathcal{O})} \mathcal{Q}^{\mu}(s, o)$$

Optimal w.r.t. to the Bellman criterion for $\mathcal{O}$.

Consequence of choosing an option

- Reward part

$$r_s^o = \mathbb{E}\left\{ r_1 + \gamma r_2 + \cdots + \gamma^{k-1} r_k | s_0 = s, o \text{ taken in } s_0, \text{ for } k \text{ steps} \right\}$$

- Next state part

$$p_{ss'}^o = \mathbb{E}\left\{ \gamma^k \delta_{s_k s'} | s_0 = s, o \text{ taken in } s_0, \text{ for } k \text{ steps} \right\}$$

Synchronous Value Iteration Generalised to Options

- Initialise :

$$V_0(s) \leftarrow 0 \quad \forall s \in \mathcal{S}$$

- Iterate :

$$V_{k+1}(s) \leftarrow \max_{o \in \mathcal{O}} \left(r_s^o + \sum_{s' \in \mathcal{S}} p_{ss'}^o V_k(s') \right) \quad \forall s \in \mathcal{S}$$

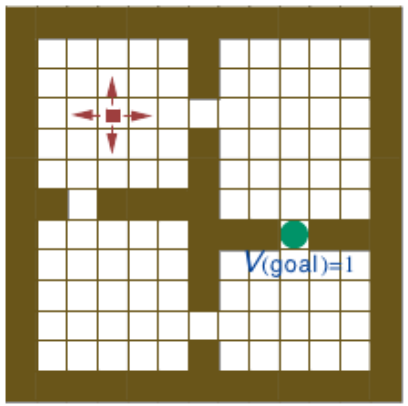
- Converges to the optimal value function, given the options:

$$\lim_{k \to \infty} V_k = V_{\mathcal{O}}^*$$

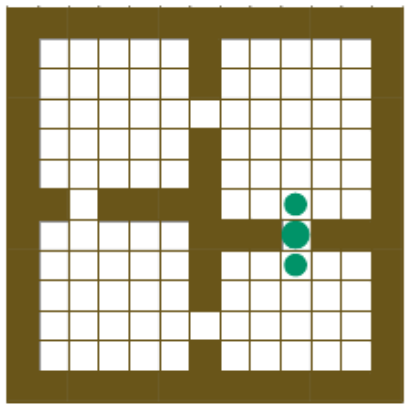
- Once $V_{\mathcal{O}}^*$ is computed, $\mu_{\mathcal{O}}^*$ can be determined.
- If $\mathcal{O} = \mathcal{A}$, we are back at the conventional value iteration
- If $\mathcal{A} \subseteq \mathcal{O}$, then $V_{\mathcal{O}}^* = V^*$

Rooms Example

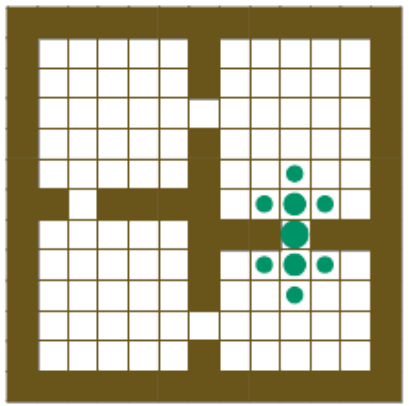
with cell-to-cell primitive actions



Iteration #0

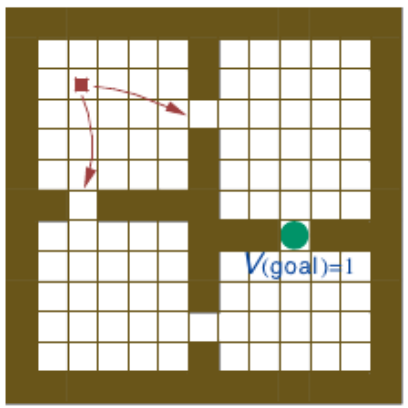


Iteration #1

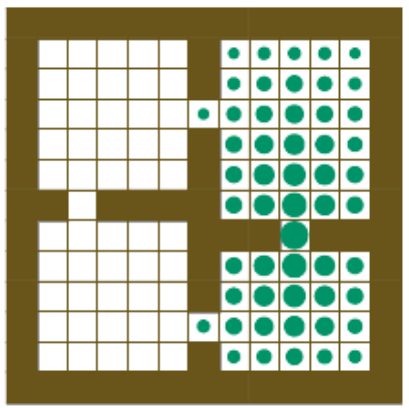


Iteration #2

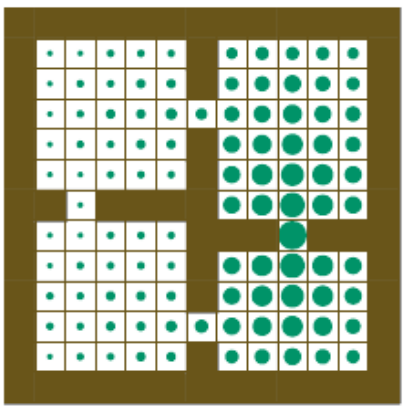
with room-to-room options



Iteration #0

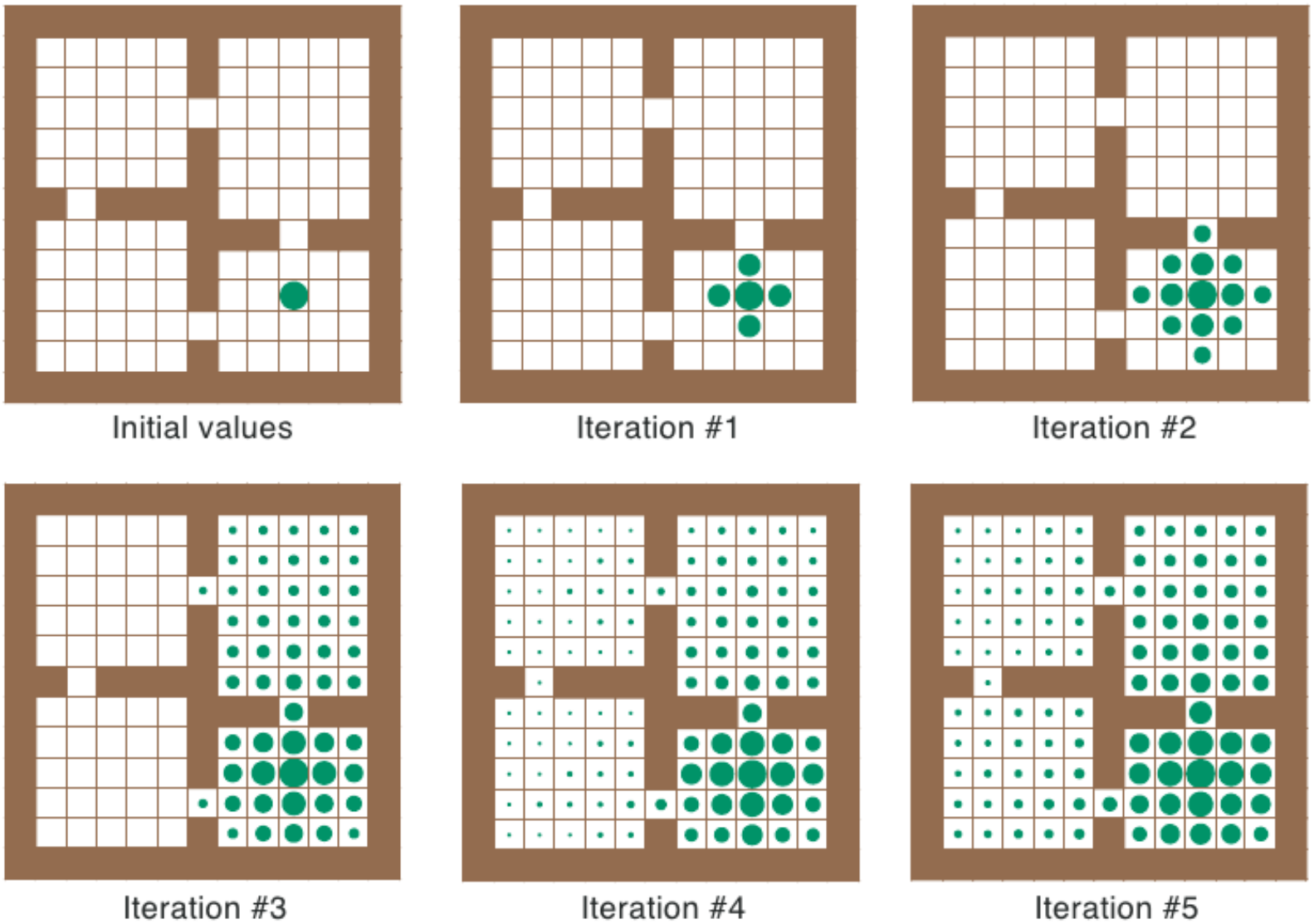


Iteration #1



Iteration #2

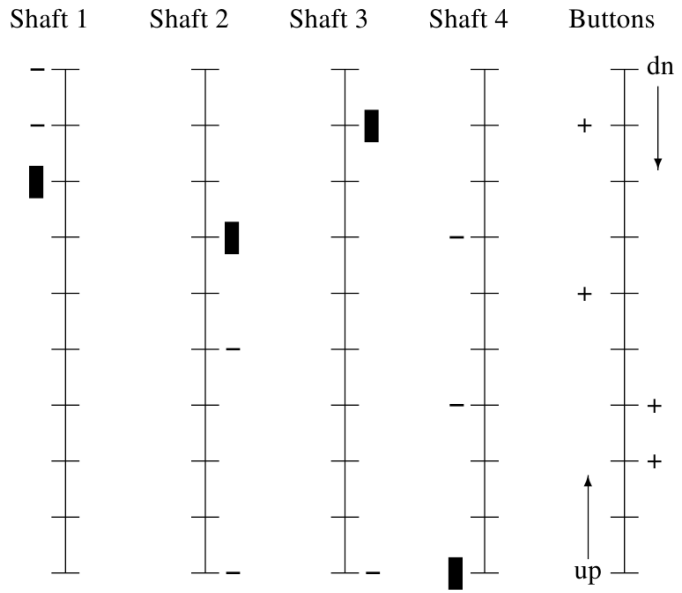
If Goal≠Subgoal: Both primitive actions and options



Benefits of Options

- Transfer
  - Solutions to sub-tasks can be saved and reused
  - Domain knowledge can be provided as options and subgoals
- Potentially much faster learning and planning
  - By representing action at an appropriate temporal scale
- Models of options are a form of knowledge representation
  - Expressive
  - Clear
  - Suitable for learning and planning
- Much more to learn than just one policy, one set of values
  - framework for "constructivism"
  - for finding models of the world that are useful for rapid planning and learning

Example V: Elevator control: State space and reward signal



- 2^{18} possible combinations of the 18 hall call buttons (only one at top and bottom)
- 2^{40} possible combinations of the 40 car buttons
- 18^4 possible combinations of positions and directions of cars: up/down to target floor

$2\hat{}18 \cdot 2\hat{}40 \cdot 18\hat{}4 \approx 10^{22}$ states

Formulation of goal:

- Minimise the average wait time
- Minimise the average system time (wait time plus travel time)
- Minimise the percentage of passengers that wait longer than some dissatisfaction threshold (usually 60 seconds)
- Minimise squared wait time (combination of first and third)

A. Barto, R. H. Crites. Improving elevator performance using RL. NIPS 8 (1996) 1017-1023.

Elevator control: Continuous time problem

Modelled as discrete event systems, but the amount of time between events is a real-valued variable. A constant discount factor γ is thus inadequate.

Use variable discount factor for cost c_τ (here minimised, instead of maximised reward)

$$\int_0^\infty e^{-\rho\tau} c_\tau d\tau \text{ instead of discrete version } \sum_{t=0}^{\infty} \gamma^t c_t$$

where $\rho \gtrsim 0$ is an inverse time scale corresponding to $1-\gamma$.
Now, for events at t_x and t_y the learning rule becomes

$$\Delta\hat{\mathcal{Q}}(s,a) = \eta\left(\int_{t_x}^{t_y} e^{-\rho(\tau-t_x)} c_\tau d\tau + e^{-\rho(t_y-t_x)} \min_b \hat{\mathcal{Q}}(u,b) - \hat{\mathcal{Q}}(s,a)\right)$$

Learning time 60,000 h of simulated elevator time (4 d @ 100 MIPS)

R.H. Crites and A.G. Barto, Elevator group control using multiple RL agents, Machine Learning 33 (1998) 235–262. (covered here only in part)

Elevator control: Evaluation and practical issues

Results

- shown to "*outperform all of the elevator algorithms with which we compared them*"
- performance is restricted by certain rules (e.g. no reversals during tours)
- today's hybrid algorithms perform better

Conclusions

- "One of the greatest difficulties in applying RL to the elevator control problem was finding the correct temperature and step-size parameters"
- "The importance of focusing the experience of the learner onto the most appropriate areas of the state space cannot be overstressed" [represented functions by a neural network]
- RL algorithms can learn from actual (or simulated) experience and solve realistic stochastic dynamic optimisation problems

R.H. Crites and A.G. Barto. Elevator group control using multiple RL agents. Machine Learning 33 (1998) 235-262. (covered here only in part)

Disadvantages of hierarchical RL (and a positive outlook)

- Choice of options is difficult: Suboptimal choice of options implies suboptimal behaviour
- Option typically become rigid when high-level planning sets in (Habits)
- Negative transfer: Options learnt for one task may be inappropriate for already for a relatively similar task
- Algorithm's complexity increases

$\Longrightarrow$ no free lunch

- Combine option learning and intra-option learning
- Define subgoals: Refine and simplify policy learning within options

Acknowledgements

Some material was adapted from web resources associated with Sutton and Barto's Reinforcement Learning book

. . . before being used by Dr. Subramanian Ramamoorthy in this course in the previous years.

Bryan Pardo, Northwestern University, EECS 349

http://webdocs.cs.ualberta.ca/~sutton/book/ebook/node66.html

A. G. Barto and S. Mahadevan (2003) Recent advances in hierarchical reinforcement learning. *Discrete Event Dynamic Systems* **13**:4, 341-379.

Some slides are adapted from: S. Singh, Reinforcement Learning: A Tutorial. Computer Science & Engineering, U. Michigan, Ann Arbor. www.eecs.umich.edu/~baveja/ICML06Tutorial/