RL 9: State Abstraction

Michael Herrmann

University of Edinburgh, School of Informatics

09/02/2016

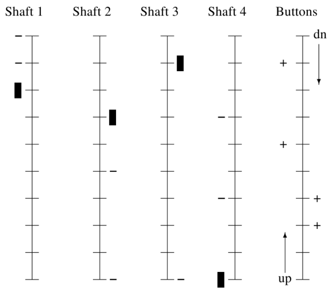
Overview

- The elevator example
- Temporal abstraction
- Options
- Semi Markovian Decision Problems (SMDP)
- Hierarchical RL

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

Example: 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

2^18 · 2^40 · 18^4 $\approx 10^{22}$ 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)

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

where $\kappa \gtrsim 0$ is a time scale corresponding to $(1-\gamma)^{-1}$.
Now, for events at t_x and t_y the learning rule becomes

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

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)

Remarks

- Process is non-stationary: Waiting time is reduced if (in a business building) in the morning unused elevators move autonomously to ground floor and, in the evening, to a higher floor. [I.e. if the process is stationary we can prove optimality of the algorithm, if it is not then further improvements may be possible.]
- Additional constraints may help to avoid overly complex reward functions,
  - e.g. while the elevator leaves from 8th floor downward, it is called from 9th floor also for a downward ride
  - going up before proceeding downward reduces total waiting time, but annoys the person at floor 8, while the person at floor 9 does not know
  - if non-empty cars are not allowed to change direction, the search space becomes smaller and some of the unwanted optima are removed
- $\mathcal{Q}$-function represented by a neural network (s. next lectures)

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

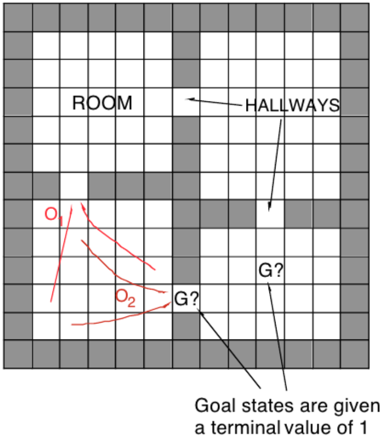
- "One of the greatest difficulties in applying RL to the elevator control problem was finding the correct temperature and step-size parameters"
- "The importance of focusing the experience of the learner onto the most appropriate areas of the state space cannot be overstressed" [represented functions by a neural network]
- Time intervals where nothing happens are tied together by calculating the accumulated reward. This provides an efficient temporal structure without having to choose a step size.

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)

Options

- Can we combine reoccurring (non-trivial) sequences of actions into “macro”-actions?
- Seems reasonable, but are there any side effects?
- How to identify useful action sequences?

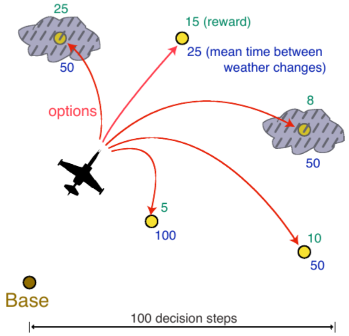
Rooms example: Exploiting the structure of the environment



- 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
- Cost per step as "reward"
- $\gamma = 0.9$

Example: Reconnaissance Mission Planning

- Mission: Fly over (observe) most valuable sites, return to base
- Stochastic weather affects observability (cloudy/clear) of sites
- Intractable with classical optimal control methods



- Limited fuel
- Temporal scales:
    - Actions: which direction to fly now
    - Options: which site to head for
    - Options compress space and time
    - Reduce steps from ≈ 600 to ≈ 6
    - Reduce states from $\approx 10^{11}$ to $\approx 10^{6}$

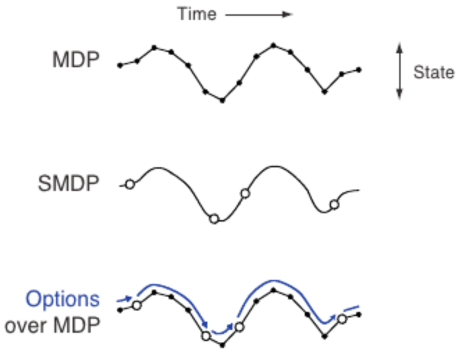
$$\mathcal{Q}^*_{\mathcal{O}}(s,o) = r^o_s + \sum_{s'} p^o_{ss'} V^*_{\mathcal{O}}(s'),\ s \text{ all states } (10^6),\ s' \text{ sites only } (6)$$

Options

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

- Instead of memorising action-sequences, use partial policies
- $o = \langle \mathcal{I}, \pi, \beta \rangle$ 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

Options and Semi-Markov Decision Processes (SMDP)



- 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 Bellman criterion for $\mathcal{O}$ (set of avaiable options).

In general, $\mathcal{O}$-optimality does not imply $\mathcal{A}$-optimality.

An option can also be a single action. If $\mathcal{A} \subseteq \mathcal{O}$ then the option-based algorithm can find the same solution as the standard algorithm (but may be slower).

Consequence of choosing an option

- Reward:

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

- Next state:

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

Synchronous Value Iteration Generalised to Options

- Initialise :

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

- Iterate :

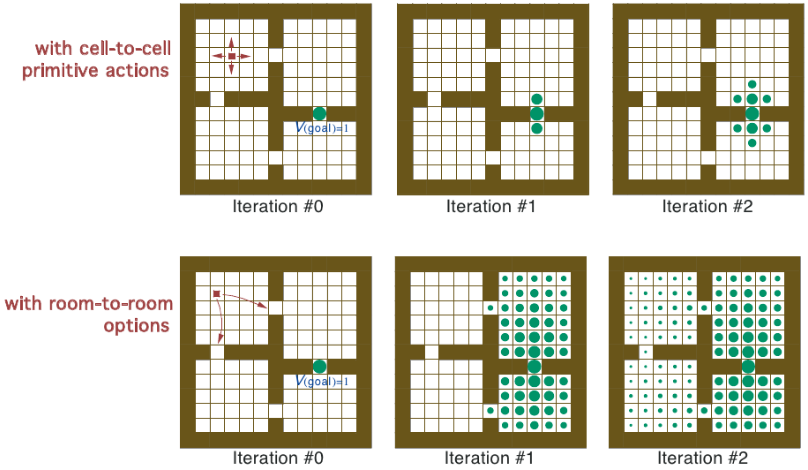
$$V_{k+1}(s) \leftarrow \max_{o \in \mathcal{O}} \left(r_s^o + \sum_{s' \in \mathcal{S}} p_{ss'}^o V_k(s') \right) \quad \forall s \in \mathcal{S}$$

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

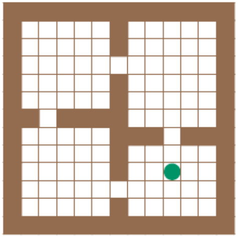
$$\lim_{k \to \infty} V_k = V_{\mathcal{O}}^*$$

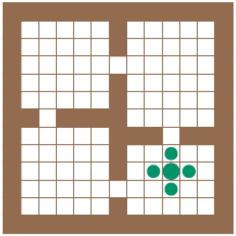
- Once $V_{\mathcal{O}}^*$ is computed, $\mu_{\mathcal{O}}^*$ can be determined.
- If $\mathcal{O} = \mathcal{A}$, we are back at the conventional value iteration
- If $\mathcal{A} \subseteq \mathcal{O}$, then $V_{\mathcal{O}}^* = V^*$

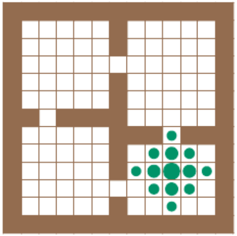
Rooms Example

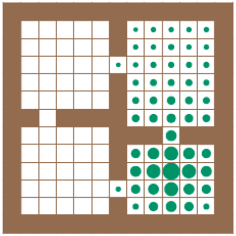


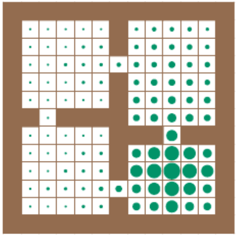
If Goal≠Subgoal: Both primitive actions and options

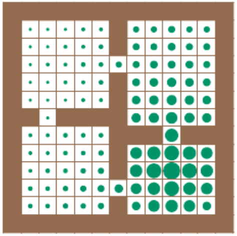

Initial values


Iteration #1


Iteration #2


Iteration #3


Iteration #4

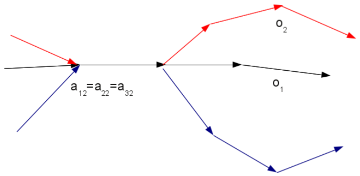

Iteration #5

Advantages of Dual MDP/SMDP View

- At the SMDP level
    - Compute value functions and policies over options with the benefit of increased speed / flexibility
- At the MDP level
    - Learn how to execute an option for achieving a given goal
- Between the MDP and SMDP level
    - Termination Improvement: Improving the value function by changing the termination conditions of options
    - Intra-Option Learning: Learning the values of options in parallel, without executing them to termination
    - Learning the models of options in parallel, without executing them to termination
    - Tasks and Subgoals: Learning the policies inside the options

Intra-option learning

- SMDP learning: Update only w.r.t. the option that was executed
- Intra-option learning: Update all options that coincide in the current elementary action



Update value function for o_1 (option taken) and for a_1 and coinciding elementary actions in other options (here a_{22} and a_{32})

Can be proven to converge to the same result as standard $\mathcal{Q}$-learning.

Intro-option learning is faster than SMDP value learning

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

Disadvantages

- 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

Hierarchical RL: Biological evidence

- Experiment: Participants chose repeatedly between two casinos, and then chose, within each casino, among a set of slot machines.
- Behavioural analysis indicated that participants learned at both levels of the task, discovering both which casino and which slots were most lucrative.
- fMRI indicated that outcomes at both levels generated independent reward prediction-error signals, detectable within the ventral striatum.
- Strikingly, prediction errors at the slot-machine and casino levels could be separately identified even when task events triggered them concurrently

Diuk C, Schapiro A, Cordova N, Ribas-Fernandes JJF, Niv Y, Botvinick M: Divide and conquer: Task decomposition and hierarchical reinforcement learning in humans. In: Computational and Robotic Models of the Hierarchical Organization of Behavior. Ed. by Baldassare G, Mirolli M: Springer Verlag, 2012. (following Botvinick)

Hierarchical RL algorithms

Learning Problems (T. G. Dietterich, 2006)

- Given a set of options, learn a policy over those options [Precup, Sutton, and Singh; Kalmar, Szepesvari, and Lörincz]
- Given a hierarchy of partial policies, learn policy for the entire problem [Parr and Russell]
- Given a set of subtasks, learn policies for each subtask [Mahadevan and Connell; Sutton, Precup and Singh; Ryan and Pendrith]
- Given a set of subtasks, learn policies for the entire problem [Kaelbling (HDG), Singh (Compositional Tasks), Dayan and Hinton (Feudal Q), Dietterich (MAXQ); Dean and Lin]

NB: Samuel's Checkers Player (Samuel, 1959, 1967) used hierarchical look-up tables called signature tables (Griffith, 1966), see S&B: 15:2

Feudal RL (Dayan and Hinton, 1995)
Quotes

Reward Hiding:

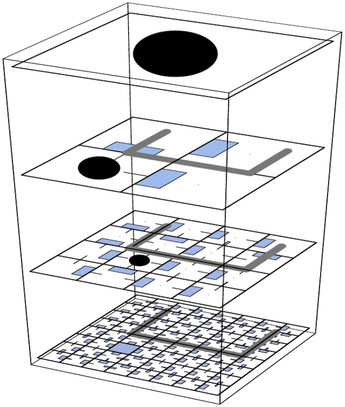
- If a sub-manager fails to achieve the sub-goal set by its manager it is not rewarded
- Conversely, if a sub-manager achieves the sub-goal it is given it is rewarded, even if this does not lead to satisfaction of the manager's own goal.
- In the early stages of learning, low-level managers can become quite competent at achieving low-level goals even if the highest level goal has never been satisfied.

Information Hiding:

- Managers only need to know the state of the system at the granularity of their own choices of tasks.
- A super-manager does not know what choices its manager has made to satisfy its command.

Feudal RL (Dayan and Hinton, 1995)

- Lords dictate subgoals to serfs
- Subgoals = reward functions?
- Demonstrated on a navigation task
- Markov property problem
    - Stability?
    - Optimality?



P. Dayan and G. E. Hinton. "Feudal reinforcement learning. NIPS (1993): 271-271.

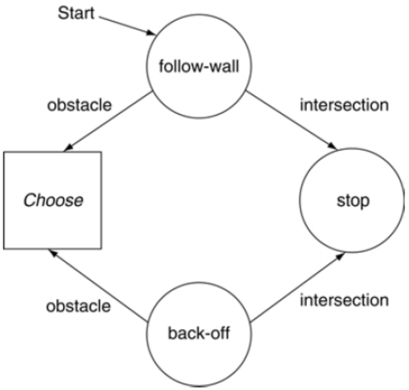
Hierarchical Abstract Machines (HAM), (R. Parr, 1998)

- Policies of a core MDP ($\mathcal{M}$) are defined as programs which execute based on own state and current state of core MDP
- HAM policy $\approx$ collection of FSMs: $\{\mathcal{H}_i\}$
- Four types of states of $\mathcal{H}_i$
  - **Action**: Generate action (for $\mathcal{M}$) based on state of $\mathcal{M}$ and currently executing $\mathcal{H}_i$

    $$a_t = \pi\left(m_t^i, s_t\right) \in \mathcal{A}_{s_t}$$

    where m_t^i is the current state of $\mathcal{H}_i$ and s_t is the current state of $\mathcal{M}$; resulting in a state change in $\mathcal{M}$
  - **Call**: Suspend $\mathcal{H}_i$ and start $\mathcal{H}_j$ setting its state based on s_t
  - **Choose**: (Non-deterministically) pick the next state of $\mathcal{H}_i$
  - **Stop**: return control to calling machine (and continue there)

State Transition Structure in HAM



Upon hitting a obstacle the choose action starts (recursively) either follow-wall or back-off

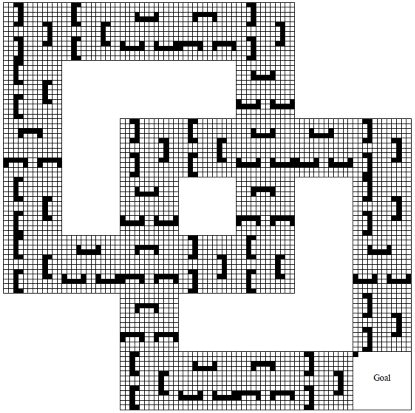
HAM $\mathcal{H}$: Initial machine + closure of all machine states in all machines reachable from possible initial states of the initial machine

HAM $\circ$ MDP = SMDP

- Composition of HAM and core MDP defines an SMDP
- Only actions of the composite machine = choice points
  - These actions only affect HAM part
- As an SMDP, things run autonomously between choice points
  - What are the rewards?
- Moreover, optimal policy of the composite machine only depend on the choice points!
- There is a reduce $(\mathcal{H} \circ \mathcal{M})$ whose states are just choice points of the original $\mathcal{H} \circ \mathcal{M}$ (i.e. Action, Call, Stop are contracted into neighbouring choices).
- We can apply SMDP $\mathcal{Q}$-learning to keep updates manageable

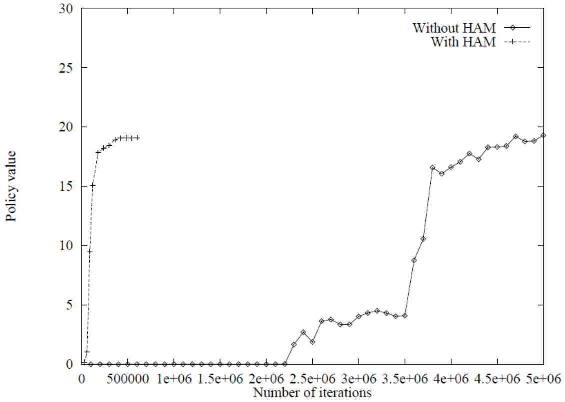
Partial Policies

Example: Parr's Maze Problem



- TraverseHallway(d) calls ToWallBouncing and BackOut.
- ToWallBouncing(d_1, d_2) calls ToWall, FollowWall
- FollowWall(d)
- ToWall(d)
- BackOut(d_1, d_2) calls BackOne, PerpFive
- BackOne(d)
- PerpFive(d_1, d_2)

Partial Policies
Results on Parr's Maze Problem



Shown is value of starting state. "Flat" Q is ultimately better.

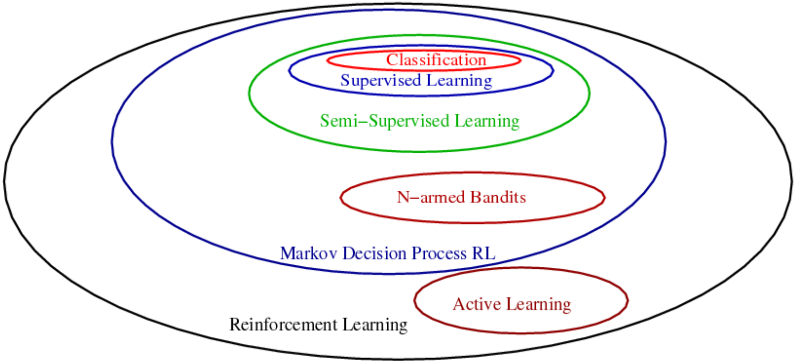
Acknowledgements

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

Recommended literature:

- A. G. Barto and S. Mahadevan (2003) Recent advances in hierarchical reinforcement learning. *Discrete Event Dynamic Systems* **13**:4, 341-379.
- T. G. Dietterich (1999) Hierarchical Reinforcement Learning (tutorial)
- T. G. Dietterich (2000) Hierarchical reinforcement learning with the MAXQ value function decomposition. *J. Artif. Intell. Res.* (*JAIR*), **13**, 227-303.

Reinforcement Learning



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

(John Langford)