Hierarchical Reinforcement Learning (HRL): An Introduction

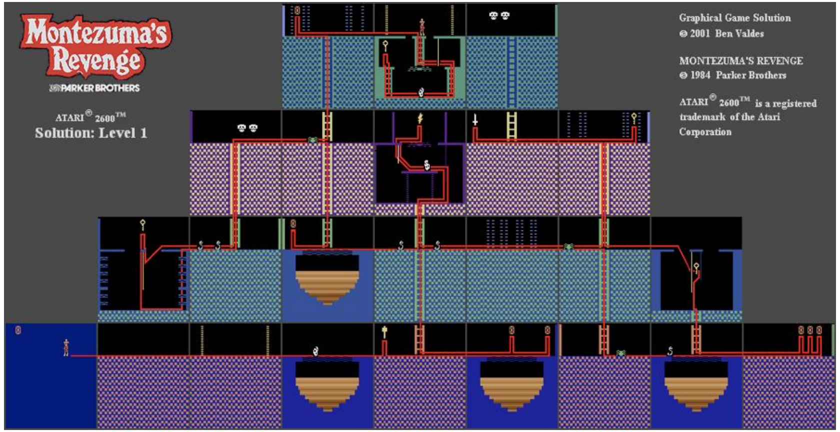
Barry Plunkett (eplu@sas.upenn.edu)
April 4, 2019

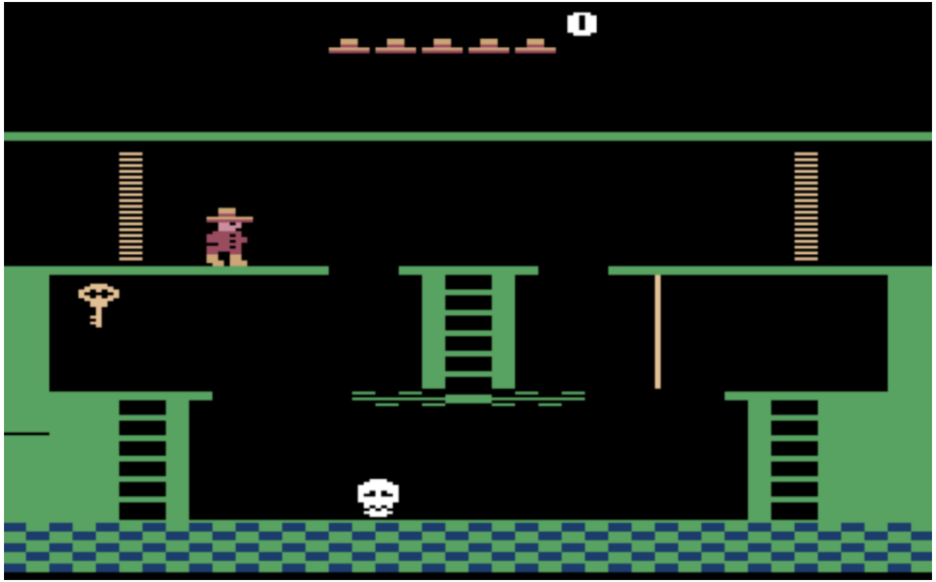
Penn Engineering

Problem & Motivation – Weaknesses of RL

- Sample inefficiency
- Sparse reward environments
- Large state, action space environments
- Unintuitive
- Generalization and abstraction
- Hold on…we solved Go!

Problem & Motivation – Weaknesses of RL



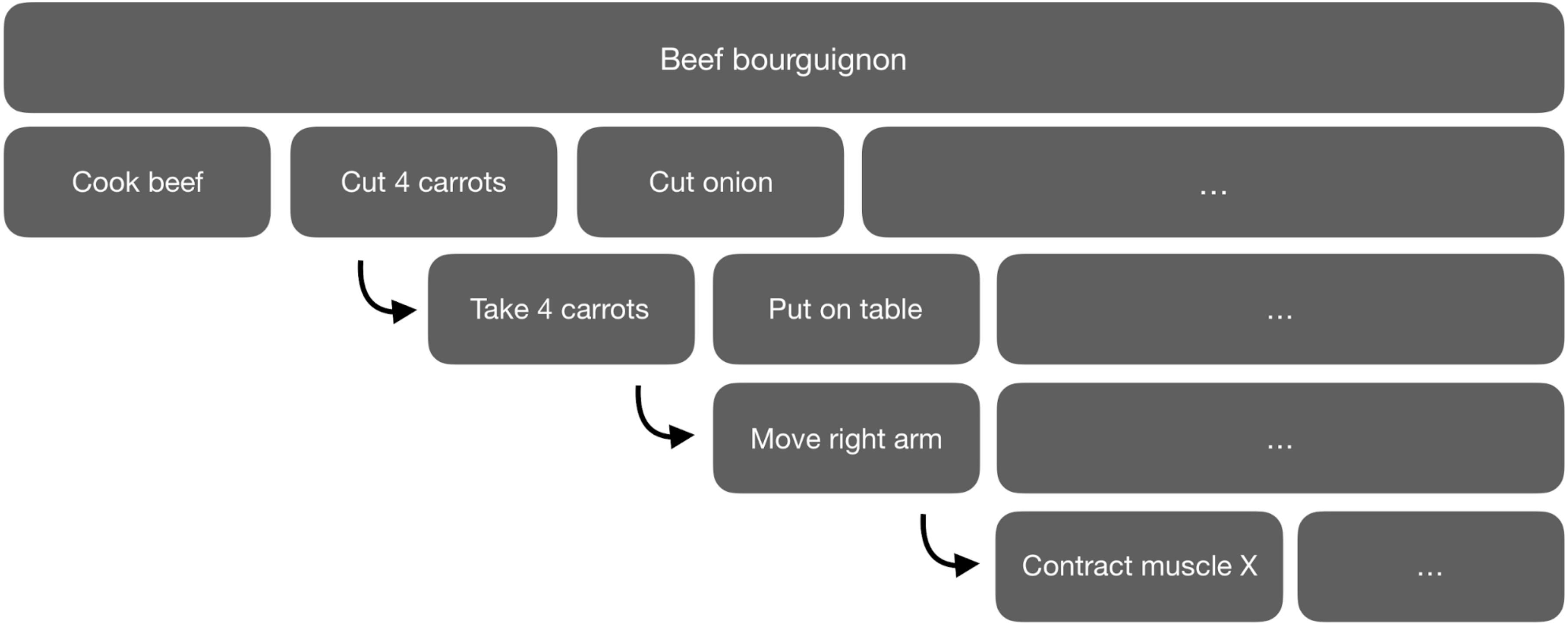


Problem & Motivation – Promise of HRL

Hierarchical Reinforcement Learning	=	Reinforcement Learning	+	Temporal Abstraction

- Decompose goal into subtasks
  - Learn low-level policies to solve small subtasks
  - Compose low-level policies into longer-term, more abstract strategies to achieve goal
- Denser rewards
- Transfer learning via subproblem re-use
- Distill state/action space into cohesive subspaces

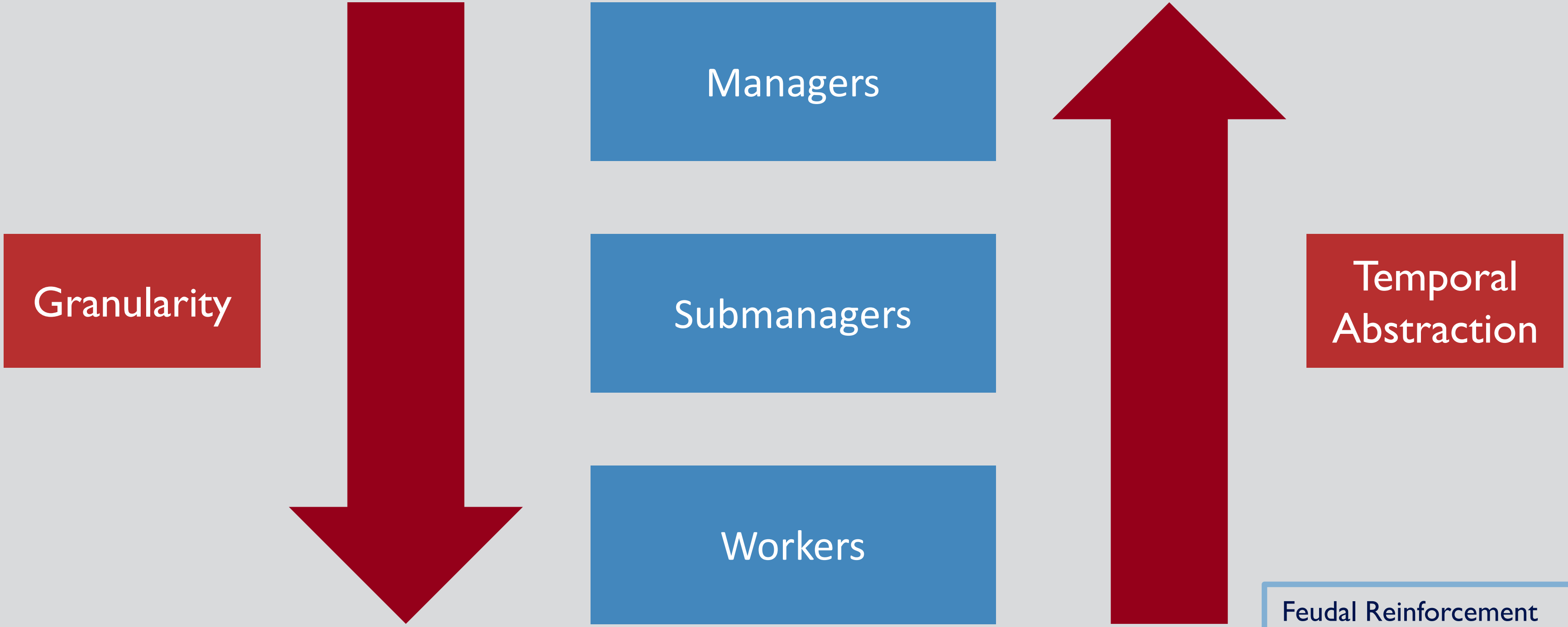
Problem & Motivation – Promise of HRL



Contents:

- Feudal Learning
- Markov Options
- HAMs
- MAXQ
- Conclusions

Feudal Learning – Introduction

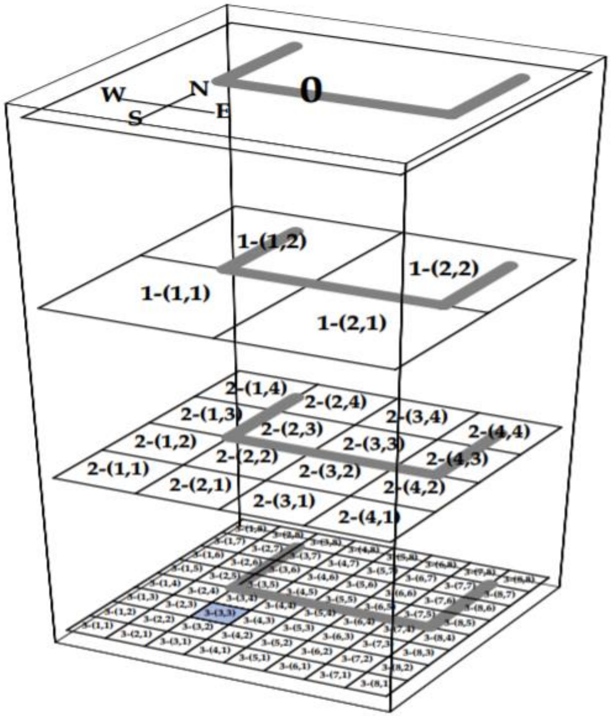


Feudal Reinforcement Learning (Dayan & Hinton)

Feudal Learning – Key Features

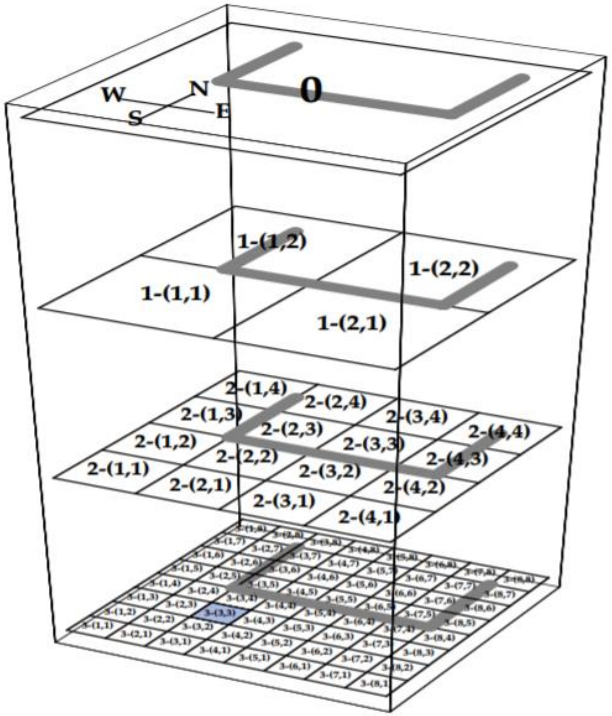
Reward Hiding	Information Hiding
• A submanager receives reward if and only if it achieves the goal set for it by its manager • No reward if manager goal attained but not submanager goal attained • Receives reward if goal attained but manager goal not attained	• Information hidden downwards – submanagers do not know their manager's task • Information hidden upwards – managers do not know how their sub-managers have assigned workers to complete task

Feudal Learning – Maze Task



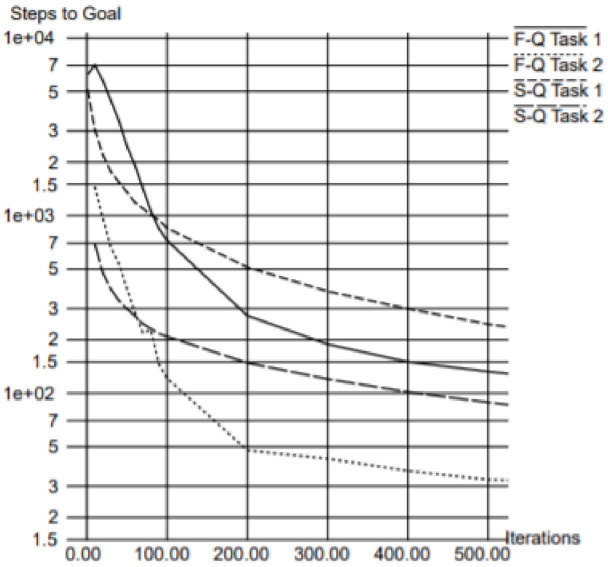
- N, S, E, W: Move to region in given cardinal direction at current level
- *: pass control to sub-managers to search for goal within current region at finer grain
- A_1: {*}
- $A_{1\text{-}(n-1)}$: {N, S, E, W, *}
- A_n: {N, S, E, W}
- State:
  - Action selected by manager above
  - Location of agent on the board in the granularity below

Feudal Learning – Maze Task



Tabular Q-values updated at all levels where a transition occurred, if the transition was ordered at all lower levels

Feudal Learning – Maze Task



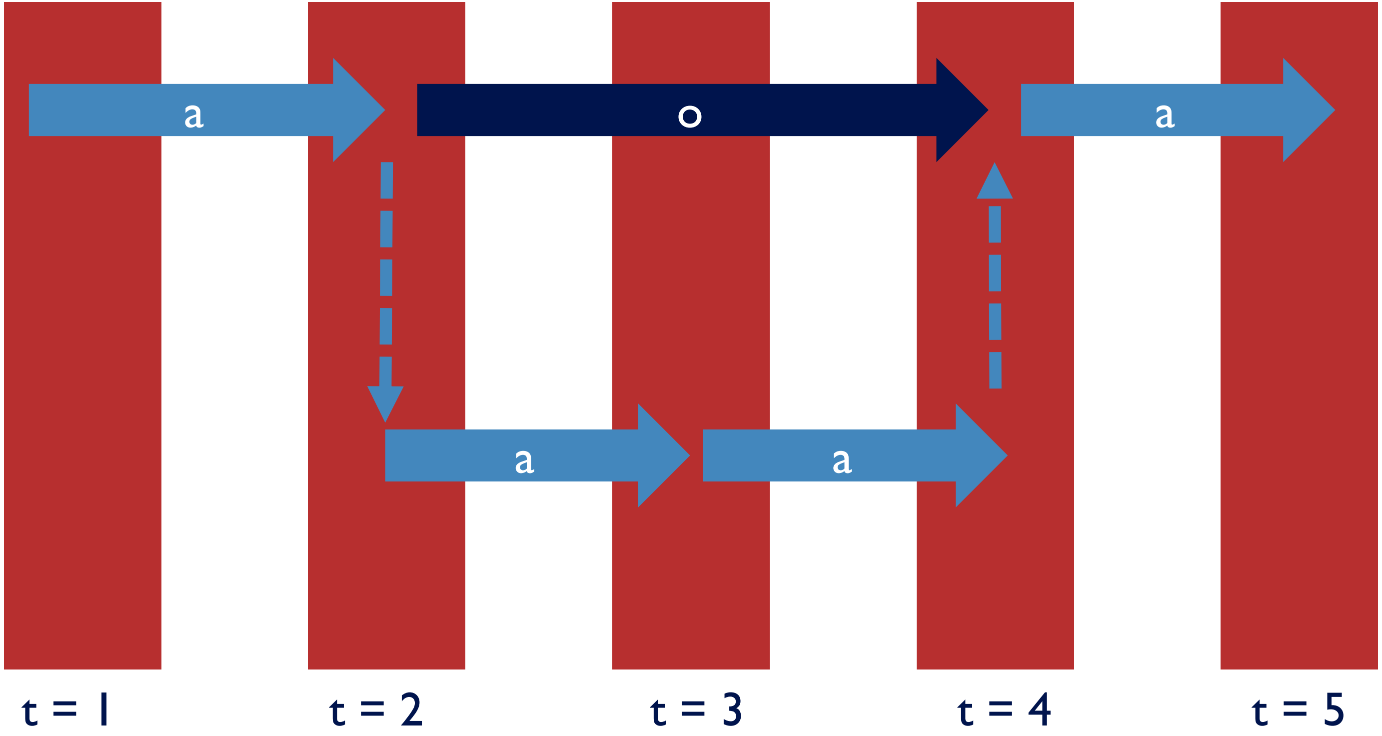
- F-Q: Feudal system
- S-Q: Standard tabular Q

- Feudal slower initially
- Faster later

Feudal Learning – Conclusions

Advantages	Costs
• Learns more about environment than standard Q-Learning approach • Structured exploration	• Information hiding may introduce inefficiencies • Submanagers learn solutions to subproblems, even if these are not relevant to goal • Task may appear non-markovian at high level abstraction

Markov Options – Introduction



Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning Sutton et al (1999)

Penn Engineering

Markov Options – Semi-Markov Decision Processes (SMDP)

- MDP : Amount of time between decisions fixed
- SMDP: Amount of time between decisions is random variable (τ)
  - Continuous
  - Discrete
- Treat system as "waiting" for τ periods
- Instantaneous state transition afterward

Markov Options – Semi-Markov Decision Processes (SMDP)

MDP	SMDP
$P(s' \| s, a)$	$P(s', \tau \| s, a)$
$r(s, a)$	$R(s, a)$
$V^*(s) = \max_a [R(s, a) + \gamma(\Sigma_{s'} P(s' \| a, s) V^*(s)]$	$V^*(s) = \max_a [R(s, a) + (\Sigma_{s' \tau} \gamma^\tau P(s', \tau \| a, s) V^*(s')]$
$Q^*(s,a) = R(s, a) + \gamma \Sigma_{s'} P(s', \| a, s) \max_{a'} Q^*(s', a')$	$Q^*(s,a) = R(s, a) + \Sigma_{s' \tau} \gamma^\tau P(s', \tau \| a, s) \max_{a'} Q^*(s', a')$

Markov Options - SMDP – Q-Learning

- Control via V, Q-learning again

$$Q^{\pi}(s,a) = Q^{\pi}(s,a) + \alpha(r_{t+1} + \gamma r_{t+2} + \ldots + \gamma^{\tau} r_{t+\tau} + \max_{a'} Q^{\pi}(s', a') - Q^{\pi}(s,a))$$

- Converges under same guarantees as MDP Q-Learning
  - Linear function approximator
- Symmetric update for new value function too

Markov Options Formalization

Option (O)		
Input set	Policy	Termination Condition
$I \subseteq S$ Set of states where option O is available	$\pi: (S, A) \rightarrow [0, 1]$ Distribution of actions taken by options	$\beta: (S) \rightarrow [0, 1]$ Probability option ends in a given state

Markov Options – Assumptions

- Actions of core MDP
  - "Primitive actions" or one-step options

$$\beta(s) = 1 \;\forall\; s \in S$$

- At least one option available in all states
- Option available in all states where it may continue

$$\{s : \beta(s) < 1\} \subseteq I$$

Open-the-door

Markov Options – Semi-Markov Options

- Semi-Markov options: Options where actions may depend on entire history of observations, since beginning of option

$$\mu : S \times \cup_{s \in S} O_s \rightarrow [0, 1]$$

- Allow options that terminate after fixed number of steps
- Allow policies over options
- Flat policies
  - Policy over primitive actions of core MDP
  - All μ correspond to some flat policy *flat(μ)*
  - Non-Markovian even when all policies are Markovian

Markov Options – SMDP Q-learning

Reward	$R(s, o) = E[r_{t+1} + \gamma\, r_{t+2} + \ldots + \gamma^{\tau} r_{t+\tau}]$
Transition Function	$P(s'\|s, o) = \Sigma_{t=1} p(s', t)\, \gamma^{t}$

$$V_O^*(s) = \max_{o \in O_s} \left[R(s,o) + \sum_{s'} P(s'|s,o) V_O^*(s') \right] \qquad Q_O^*(s,o) = R(s,o) + \sum_{s'} P(s'|s,o) \max_{o' \in O_{s'}} Q_O^*(s',o')$$

$$Q_{k+1}(s,o) = (1-\alpha_k) Q_k(s,o) + \alpha_k \left[r + \gamma^{\tau} \max_{o' \in O_{s'}} Q_k(s',o') \right]$$

Penn Engineering

Markov Options – Intra-option learning

- Q-Learning Drawbacks
  - Updates 1 option at a time
  - Must wait until option completes to update
- Intra-option learning methods
  - Learn online while option executes
- One-step intra-option Q-Learning
  - Suppose primitive action *a* taken, then for every option whose policy could have selected *a* with the same distribution π(s, *):

$$Q_{k+1}(s_t, o) = (1 - \alpha_k) Q_k(s_t, o) + \alpha_k [r_{t+1} + \gamma U_k(s_t, o)]$$

$$U_k(s, o) = (1 - \beta(s)) Q_k(s, o) + \beta(s) \max_{o' \in O} Q_k(s, o')$$

Markov Options – Conclusions

- Add temporally-extended activities without precluding fine-grained control
- Exclude some primitives
  - Restrict set of learnable policies
  - Increase efficiency, prevent “flailing”
- Utilize options to achieve subgoals
  - Define subgoal-specific reward functions and use for option policy
  - Set options to terminate upon subgoal completion

Hierarchies of Abstract Machines (HAMs)

- Apply temporal abstraction to **SIMPLIFY** rather than augment
- Well-known set of optimal (or good enough) policies for long-time horizon actions
  - Robot navigation

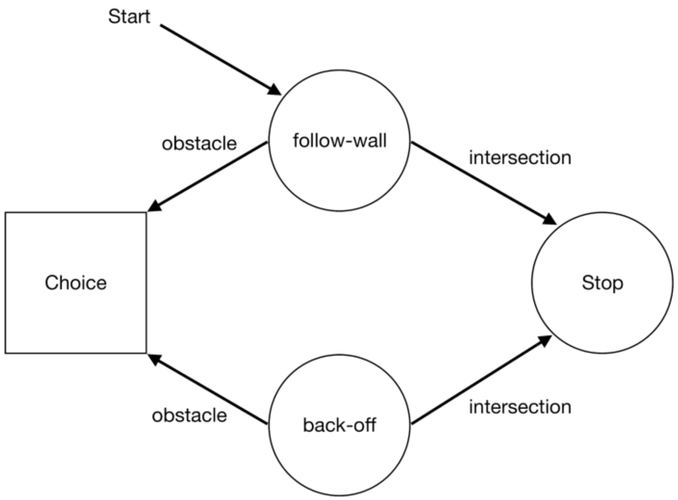
HAMs – Formalization

- Collection of finite state machines $\{H_i\}$
- Core environment MDP (M)
- Machine state initialization function I_i: $S_M \rightarrow S$
- Stochastic state transition function: δ_i $S \rightarrow S$

State set (S)			
Action	Call	Choice	Stop
Executes action of M $a = \pi(m_t, s_t)$	Suspends execution of current machine, calls another machine H_j, function of m_t. $I_j(s_t)$ sets initial state	Non-deterministically chooses next state of H_i	Suspends execution of current machine. Resumes execution of calling machine

HAMs

- If no action is generated at step t, M remains in current state
- H : Initial machine
  - Assume no stop state
  - Assume no probability 1 loops
  - Ensure MDP will continue to receive primitive actions



HAMs – SMDP view

- Equivalent to SMPD : H ○ M
- State : S x S_M
- Actions: Choice points of H
  - Runs autonomously via action states until next choice point reached
  - Do not learn within machine policies – program these
- Reward: Discounted awards accumulated during timesteps between choice states
  - Reward of 0 for timesteps where M does not change

HAMs – Q-learning

- *Reduce(H ○ M)*
  - SMDP equivalent to H ○ M with states defined as only choice points of H ○ M
  - Optimal policy for *Reduce(H ○ M)* same as H ○ M
- Apply standard SMDP q-learning update to Reduce(*H ○ M*)

$$
\begin{aligned}
Q_{k+1}([s_c, m_c], a_c) = {} & (1 - \alpha_k) Q_k([s_c, m_c], a_c) \\
& + \alpha_k [r_{t+1} + \gamma r_{t+2} + \cdots + \gamma^{\tau-1} r_{t+\tau} + \gamma^{\tau} \max_{a'} Q_k([s'_c, m'_c], a')]
\end{aligned}
$$

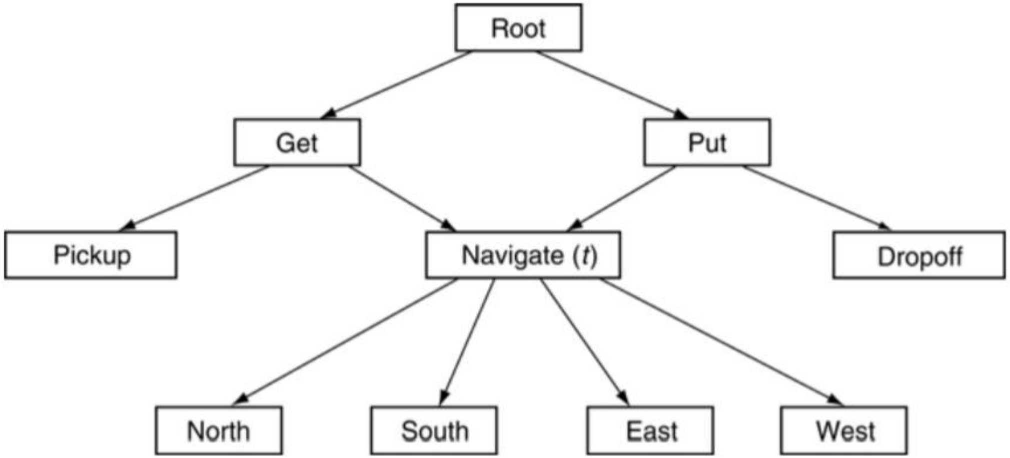
Penn Engineering

HAMs - Conclusions

- Success depends on quality of programmed policies and state transition functions for each machine
- Not used in any large-scale applications
- Allows integration of multiple controls problems, whose solutions are well known, but whose relationships are not

MAXQ

Decompose into hierarchy of SMDPs (rather than 1) and solve simultaneously



MAXQ – Formalization

- Decompose MDP M into subtasks $M_0 \ldots M_n$
  - M_0 corresponds to original task
- Each subtask similar to an option

Subtask anatomy (M_i)		
Policy	Active States	Pseudo-Reward
$a = \pi(s_t, k)$ • Assume deterministic • K denotes subtask call stack	• S_i: Set of states where M_i can execute • T_i: $(S \setminus S_i)$ states where subtask terminates	• Task-specific reward function that assigns reward to each state in Ti

Penn Engineering

MAXQ – SMPDs

- Representation of top-level policy

$$\pi = \{\pi_0, \ldots, \pi_n\}$$

- Transition probabilities

$$P_i(s', \tau | s, a)$$

- Value of completing ith subtask task from state s and following π

$$V^{\pi}(i, s)$$

MAXQ – State Value Function

- Reward of selecting subtask a from subtask i:

$$R_i(s,a) = V^\pi(a,s)$$

- Corresponding Bellman state value equation

$$V^\pi(i,s) = V^\pi(\pi_i(s),s) + \sum_{s',\tau} P_i^\pi(s',\tau|s,\pi_i(s))\gamma^\tau V^\pi(i,s')$$

Penn Engineering

MAXQ – Action-Task Value Function

Q-Function

$$Q^{\pi}(i, s, a) = V^{\pi}(a, s) + \sum_{s', \tau} P_i^{\pi}(s', \tau | s, a)\gamma^{\tau} Q^{\pi}(i, s', \pi(s'))$$

$$C^{\pi}(i, s, a) = \sum_{s', \tau} P_i^{\pi}(s', \tau | s, a)\gamma^{\tau} Q^{\pi}(i, s', \pi(s'))$$

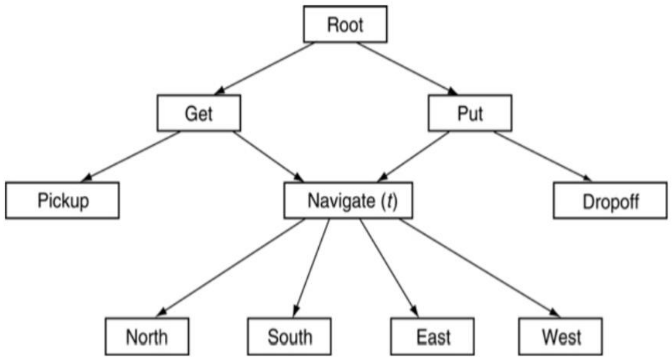
$$Q^{\pi}(i, s, a) = V^{\pi}(a, s) + C^{\pi}(i, s, a)$$

$$V^{\pi}(0, s) = V^{\pi}(a_n, s) + C^{\pi}(a_{n-1}, s, a_n) + \cdots + C^{\pi}(a_1, s, a_2) + C^{\pi}(0, s, a_1)$$

$$V^{\pi}(a_n, s) = \sum_{s'} P(s' | s, a_n) R(s' | s, a_n)$$

MAXQ Learning

- High-level overview of complex algorithm
  - Similar to Monte Carlo Q-learning with completion function
- Estimate completion C(i, s, a) for each i->a edge in the tree
- Beginning at 0, recursively execute actions to descend to primitive MDP (choosing subtask with highest completion estimate)
- After each subtask a concludes, use reward accumulated at leaf below to update C(i, s, a)



MAXQ Conclusions

- Recursively optimal policy
  - Optimal for a given subtask SMDP, given SMDPs of children
- Weaker than hierarchically optimal policy
  - Optimal policy among all policies that can be expressed within constraints of hierarchy
- Why? Hierarchically optimal policy may need to exploit context of calling subtask
  - E.g. Optimal way to travel to a destination may depend on what you're doing at the destination

Conclusions

- HRL allows temporal abstraction for long-term and short-term planning –maps onto human decision making
- Potentially valuable avenue for improving transfer learning of tasks and strategies
- Unsolvable problems today have characteristics HRL seems well suited to address
  - Sparse rewards
  - Large action and state spaces
  - Slow data generation

Penn Engineering

Conclusions

- Fundamental Problem: All frameworks we've seen today require hand-generated decompositions
  - Deal-breaker for complex tasks
  - …and that's the whole point
- Learning decompositions automatically is an active area of research
  - Heuristics & approximations
  - MetaRL (approaches reminiscent of AutoML)
- Marriages with deep RL increasingly common
  - FUN : Deep feudal learning
  - Option-Critic: Actor-critic policy gradient setup meets options framework
  - HIRO, h-DQN, and many more