Hierarchical Reinforcement Learning

Ron Parr
CompSci 590.2
Duke University

Some material adapted from Jervis Pinto's slides (web.engr.oregonstate.edu/~doppa/pubs/hrl.pptx), which were adapted from my slides. ☺

Overview

- Hierarchical (classical) planning
- Issues in hierarchical RL
- Methods
  - SMDPs (options)
  - HAMs
  - MAX-Q

Motivation for Hierarchical Planning

- Planning is hard
- Plans to achieve important/useful things can involve long sequences of actions
  - Finding these plans can be hard
  - Can be exponential in plan length
- Exhaustive search seems unnecessary
  - We don’t do that
  - Not all sequences of plans make sense

Hierarchical Trip Planning

- Plan to get from here to the west coast
- Don’t plan at the level of individual steps
- Experience tells us that most reasonable plans follow a template:
  - Walk to your car
  - Drive to the airport
  - Fly to the west coast
  - Take a taxi to your hotel
  - Walk to your hotel room

Hierarchy Restricts our Search

- We may find suboptimal plans because we fail to consider certain options, e.g., driving to a more distant airport to take a non-stop flight
- Depending upon how persistent we are, we may fail to find plans, e.g., if all flights from your favored airport are booked, do you consider other options?
- Hierarchical optimality finds the best plan consistent with our goal hierarchy, though not necessarily best overall

Hierarchy Encourages Plan Reuse

- Some plan subgoals may ignore irrelevant parts of the state
    - Passing somebody on I-40 is roughly the same every time you do it
    - Shouldn't need to replay from scratch every time you do this
    - Same plan/subgoal pair can be reused multiple times within big plan
    - Could be cached and reused
- Questions
    - How do we appropriately factor the state to capture this?
    - How do we provide clear semantics for this?

One View of Human Planning

- Humans never solve for very complicated plans
- They always solve for fairly small plans that result from decomposing problems into smaller problems
- Such decompositions may be learned or developed on our own
- Note: This is an *introspective* approach to gaining insight into intelligent behavior – not always a good idea

Views of Hierarchical Planning

- Use: We can provide the planner with a hierarchical decomposition of the task to guide the planner
- Discovery: We could ask the planner to discover such a task decomposition on its own – very hard
- Most work in planning focuses on use because:
  - Use is a prerequisite (no point in discovering things you can't use)
  - Discovery is very challenging (though also very interesting!)

Hierarchical action semantics

- Challenge in hierarchical planning: Dealing with how abstract actions/plans get refined
- Suppose you have the abstract plan of:
  - Drive to your airport
  - Fly to your destination city
  - Take a cab to your hotel
- Traditionally, in classical planning, plans are guaranteed to succeed
- What if your hotel is in Venice?

Dealing with Plan Failure in Hierarchical Classical Planning

- At the planner level:
  - Fully refine plans before declaring success
  - Add “ground level” actions so that planner can fall back on planning the hard way if hierarchy fails
  - Add **LOTS OF** abstract actions
  - Downside: Successful use of hierarchical planning is often very much like programming, requires lots of effort an anticipation of possible refinement failures
- At run time:
  - Could decide to refine the plan on the fly
  - Risky, but that different from what people do?
  - Acceptable as a guarantee?

What can hierarchical RL learn from hierarchical classical planning?

Good news/bad news

- Good news:
  - We never guaranteed success, so no need to worry about giving that up!
  - Probabilistic outcomes are part of our language, so perhaps we can take advantage of that to deal with uncertainty in plan refinement (not really)
- Bad news:
  - Opportunities for violating the Markov property abound
  - Example: Planning to pass another car
    - Superficially seems like this is a subtask that can be reused
    - Lots of subtle violations of Markov property can sneak in
  - Probabilistic refinement guarantees tend to be meaningless because probabilities are defined over a family of MDPs, but you are in only one at a time (discussion)

Taste in Hierarchical RL

- There are lots of ways we could aggregate together states and define high level actions on top of these aggregated states, but doing so haphazardly would produce value functions and policies that we do not trust
- Typical "taste" in hierarchical RL methods
  - Value function should still reflect actual value
  - Policy should be executable without need for replanning at run time

Multi-step Actions

- Underlying machinery behind most hierarchical RL is that of semi Markov decision processes (SMDPs)
- SMDPs include a distribution over the number of time steps and action takes
- Complicated sequences of actions or entire policies can be "compiled" into a single SMDP action
- "Compiling" process is just Gaussian elimination

State Transitions → Macro Transitions

- F plays the role of generalized transition function

$$T' : V^{i+1}(s) = \max_a \sum_{s'} F(s' \mid s, a)\left(R'(s, a, s') + V^i(s')\right)$$

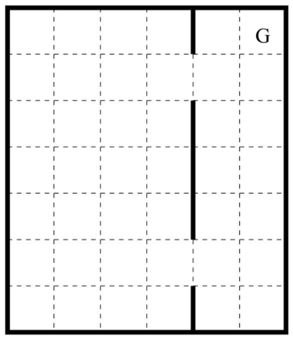
- More general:
  - Need not be a probability
  - Coefficient for value of one state in terms of others
  - May be:
    - P (special case)
    - Arbitrary SMDP (discount varies w/state, etc.)
    - Discounted probability of following a policy/running program

Where do high level actions come from?

- May be reused from a similar MDP that was solved in the past
- May be provided by hand by a human
- May be learned by creating subproblems with "pseudo-rewards" (there are many issues in doing this)

Example problem

- **actions:** North, South, East, West
- **rewards:** Each action costs −1. Goal gives reward of 0.



Slide from Pinto

Adding macro (temporally extended) actions

- Suppose we add a new action: go to the top door
- This is actually policy
  - Work in any state
  - Terminates when the top door is reached
- How do we get the transition probabilities for this policy?
  - Could solve MDP where this state has positive reward (+1), all other states have reward 0
  - Solve for expected, discounted probability of reaching the goal
    - Discounted to take into account (variable) number of steps
    - Still probabilistic because could get stuck along the way

How do we use this? Idea 1

- Add this new temporally extended action back into the mix of actions
- Advantages:
  - Can’t result in suboptimal policies (we still have the original actions)
  - May speed up convergence and exploration
- Disadvantages:
  - May increase computational cost
  - Increase in action space
  - No reduction in state space

How do we use this? Idea 2

- Eliminate some base level actions while also introducing the temporally extended actions
- Advantages:
  - Reduces the size of the action space
  - May effectively reduce the size of the state space if we only pass through some states and never need to plan actions for them
- Disadvantages:
  - May sacrifice optimality
  - Can achieve “hierarchical” optimality – best policy consistent with restrictions imposed by hierarchy

Options

- Options were a re-invention of SMDPs as macro actions
- Defined starting conditions and termination conditions for policies that were then compiled into SMDP actions

HAMs

- HAMs – a more flexible way of partially specifying a hierarchical action
- A HAM is a finite state controller with two extensions:
  - HAMs can have actions that invoke other HAMs as subroutines
  - HAMs can specify: choose{a1, a2,…} rather than requiring a specific action
- Working with HAMs
  - Compute cross product of HAM state space and MDP states given a start state distribution (done “anytime” in RL applications)
  - Convert to SMDP by removing states where action is predetermined by HAM structure
  - Solve the resulting SMDP

Properties of HAMs

- Can reduce size of state space in best case
- Can blow up size of state space in worst case
- (Can be tricky to know in advance which will happen)
- Solution to MDP will have "hierarchical" optimality but may not be optimal overall

Example of HAMs

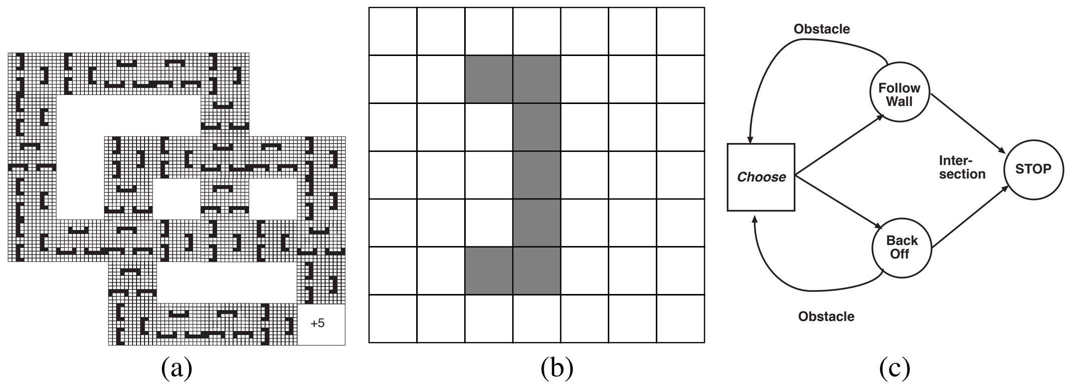


Figure 1: (a) An MDP with ≈ 3600 states. The initial state is in the top left. (b) Close-up showing a typical obstacle. (c) Nondeterministic finite-state controller for negotiating obstacles.

Example of benefit

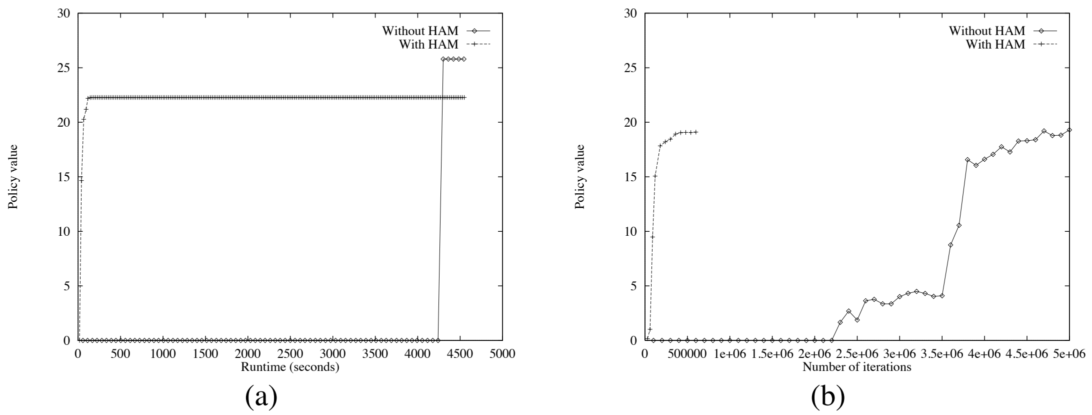


Figure 2: Experimental results showing policy value (at the initial state) as a function of runtime on the domain shown in Figure 1. (a) Policy iteration with and without the HAM. (b) Q-learning with and without the HAM (averaged over 10 runs).

Options as HAMs

- For each option:
  - Create one HAM state for every base MDP action
  - Create transitions between HAM states according to policy specified in the option

- Create one high level state that chooses between options

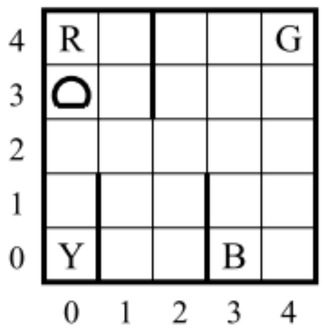
MAXQ

- Break original MDP into multiple sub-MDP's
- Each sub-MDP is treated as a temporally extended action
- Define a hierarchy of sub-MDP's (sub-tasks)

- Each sub-task M_i defined by:
  - T = Set of terminal states
  - A_i = Set of child actions (may be other sub-tasks)
  - R'_i = Local reward function

Slide from Pinto

Taxi

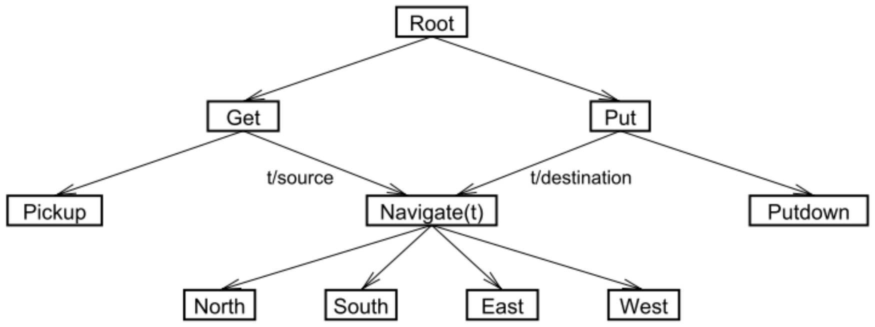


- Passengers appear at one of 4 special location
- -1 reward at every timestep
- +20 for delivering passenger to destination
- -10 for illegal actions

•500 states, 6 primitive actions

Subtasks: Navigate, Get, Put, Root

Sub-task hierarchy



Value function decomposition

$$V^\pi(i,s) = E\left\{ \sum_{u=0}^{N-1} \gamma^u r_{t+u} \; + \; \sum_{u=N}^{\infty} \gamma^u r_{t+u} \middle| s_t = s, \pi \right\}$$

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

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

“completion function”
Must be learned!

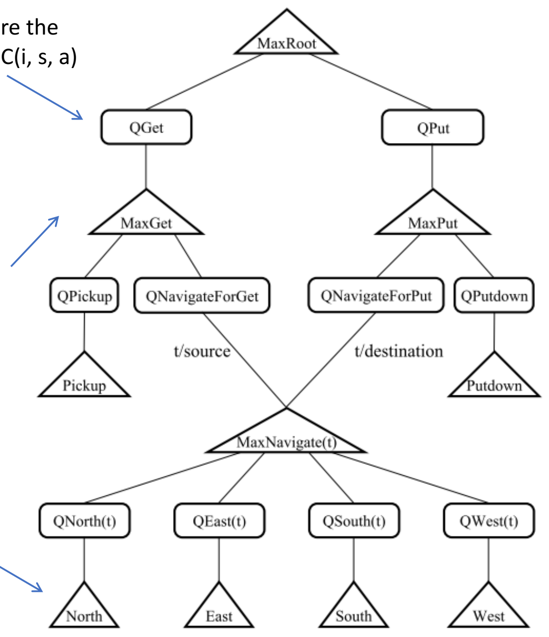
$$V^\pi(i,s) = \begin{cases} Q^\pi(i,s,\pi_i(s)) & \text{if } i \text{ is composite} \\ \sum_{s'} P(s'|s,i)R(s'|s,i) & \text{if } i \text{ is primitive} \end{cases}$$

MAXQ hierarchy

Q nodes store the completion C(i, s, a)

Composite Max nodes compute values V(i,s) using recursive equation.

Primitive Max nodes estimate value directly V(i,s)



Key differences between MAXQ and others

- Subtasks defined in a way that ignores some parts of state
- Subtask completion functions and policies reused across multiple parts of the state space, e.g., navigating from one area to another

- Advantages:
  - Faster learning due to subtask reuse

- Disadvantages
  - May violate the Markov property

What's Missing/Why hasn't this solved Everything?

- Gap between theoretical guarantees and practical use – most methods still don't scale well due to large number of states in typical MDPs
- No function approximation (MAXQ comes closest to this with shared subtasks)
- Hard to make good hierarchies
- Methods for automatically generating hierarchies, transferring hierarchical knowledge across problems remain primitive