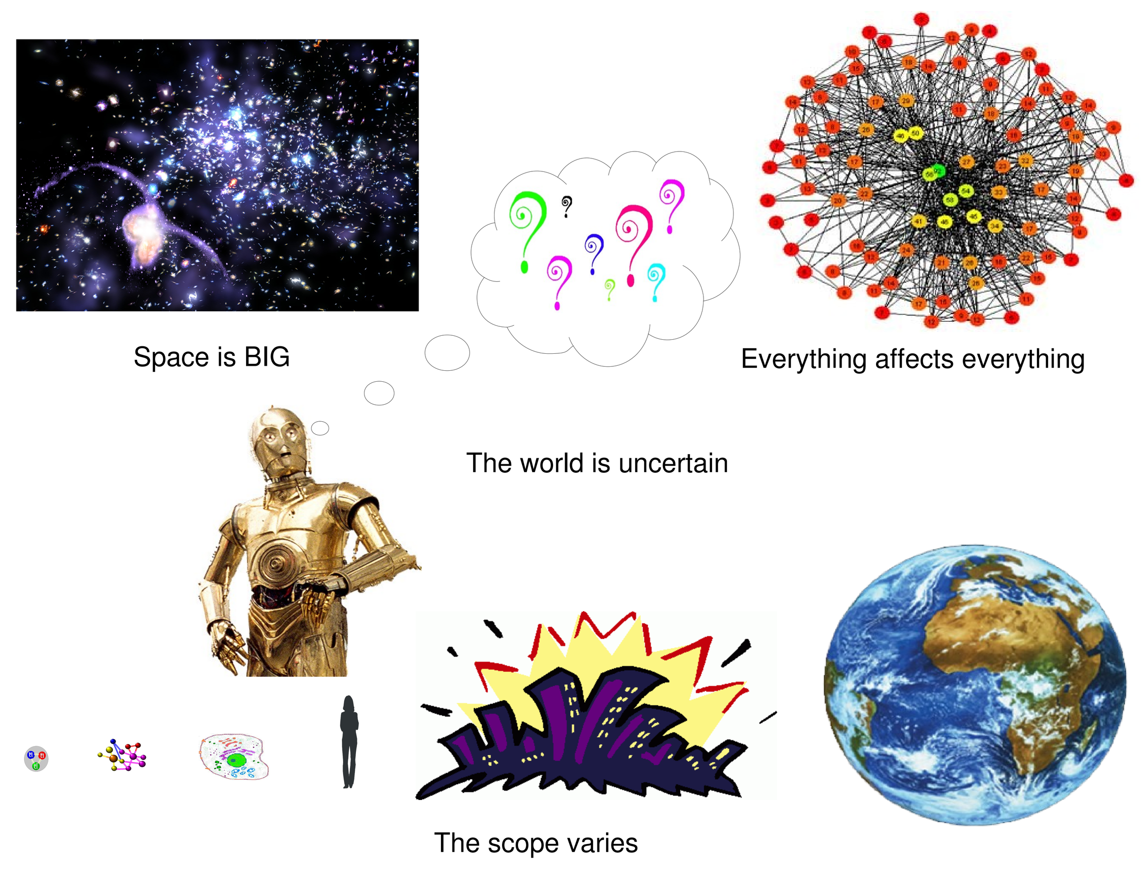
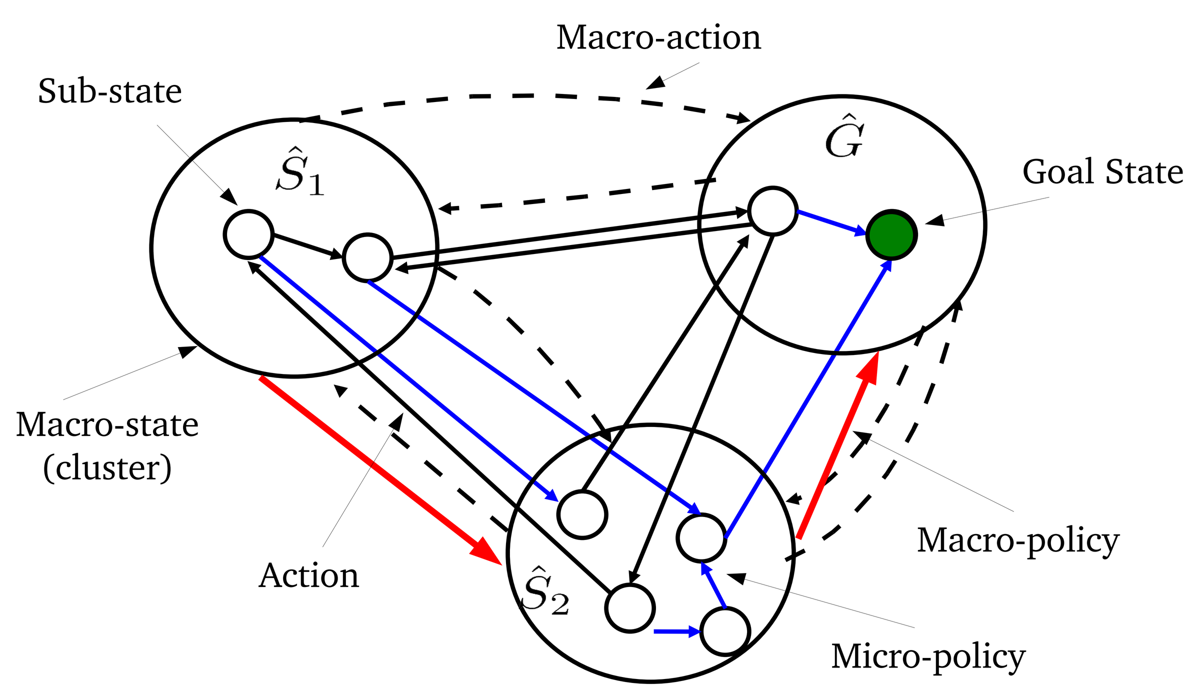
Fast Approximate Hierarchical Solutions of MDPs

Jenny Barry
Advisors: Leslie Kaelbling, Tomas Lozano-Perez
MasterWorks 5/4/2009

The Problem



World Model: Hierarchical Markov Decision Processes (MDPs)



Markov Decision Process:

$M = \{S, A, T, R, G\}$
S: The world is a finite number of *states*
A: Actions allow *non-deterministic* transitions among states
T: Function giving probability action transitions between two states
R: Each state has a reward associated with it
G: We must reach some goal state(s)
Solution: *Policy* giving action for each state maximizing expected reward

Hierarchical Markov Decision Process:

$M = \{S, A, T, R\}$
S: Clusters of states (macro-states)
A: Macro actions between clusters
G: Clusters containing goal states
T: ?? R: ??
Solution: Hierarchical policy: Macro-level policy specifies next macro-state. Micro-level policy specifies how to *reach* next macro-state

Specifying MDPs:

Enumerated State MDPs
- List out all states
- Algorithm polynomial in *number of states*

Factored MDPs
- Specify boolean state variables
  - n variables = 2^n states
- Algorithm polynomial in *number of state variables*
- But! We know more about the structure

Results

Enumerated States

Domains:
- Grid world: 1040 states, 4 actions
- Factory: 1024 states, 10 actions
- Discretized Mountain Car: 1024, 3 actions

Comparison Algorithms:
- Value iteration: Optimal solution
- HVI: Alternative hierarchical method
- HDet: Our algorithm

	Grid World		Factory		Mountain Car	
Algorithm	Run Time (s)	Avg. Dev.	Run Time (s)	Avg. Dev.	Run Time (s)	Avg. Dev.
Value Iteration	20.46	0	25.22	0	83.00	0
HDet	1.41	0.48	2.58	0.49	25.79	4.14
HVI	10.66	0.84	40.72	0.62	78.94	12.94

Factored (Preliminary)

Coffee domain: 6 variables, 4 actions
- Metric is accumulated reward

Tire domain: 12 variables, 14 actions
- Metric is percent success

Domain	Without Clustering	With Clustering
Coffee	-1.71*	-1.85
Tire	35.8%	83.3%*

*Optimal solution

Objectives

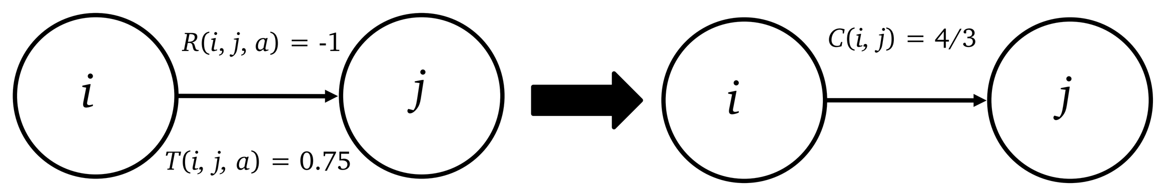
Create a solver for a hierarchical MDP
- Top-down
- Efficient method for calculating transitions/rewards
- Fast, semi-accurate solution for non-primitive levels

Clustering Algorithm: Works with solver
- Cluster "near" states
- Efficient
- Assure hierarchical policy does not strand states

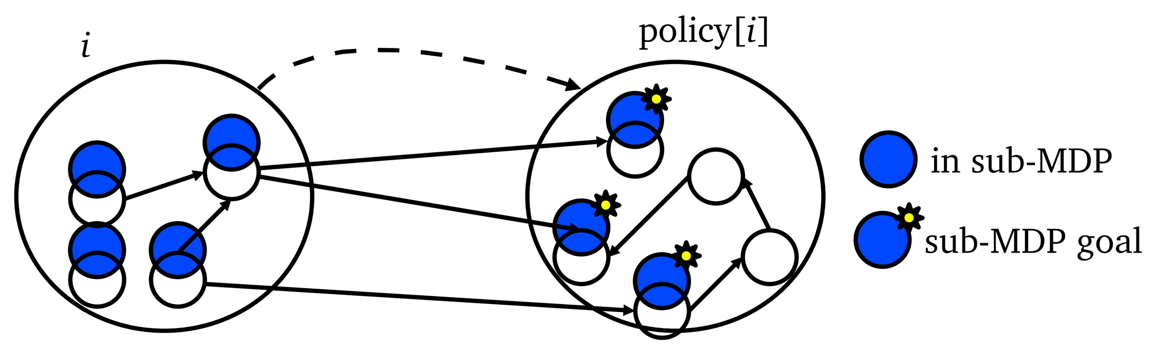
Solver

Fast Upward Pass

Assume deterministic transitions
Calculate "cost" of moving between each state
$C(i, j, a) = -R(i, j, a)/T(i, j, a)$
Solve using shortest path algorithm
Cost between clusters = average cost



Downward Pass

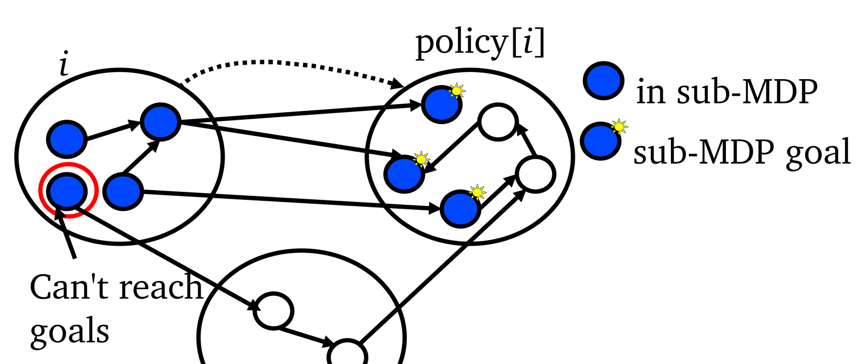


Given policy, create "sub-MDP" from i to policy[i]
States in sub-MDP
- All sub-states of i
- All sub-states of policy[i] reachable in one transition from i

Goal states are states in policy[i]
Upper level: solve for policy using shortest path among clusters
Primitive level: solve for policy using value iteration

Clustering

No "stranded" states
- Bad Clustering:

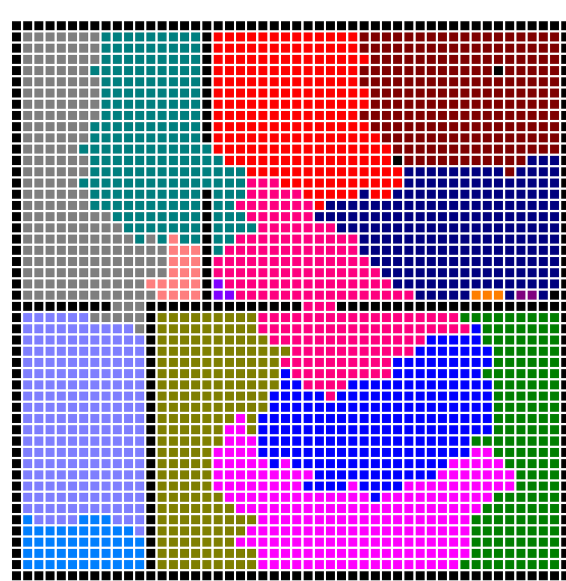


Guarantee: If any state in cluster i can reach a state in cluster j, all states in cluster i can reach a state in cluster j

Enumerated States

Agglomerative: state begins as its own cluster
Pre-process:
- For each cluster add any states that
  - Transition into cluster, but not out
  - Are adjacent to the cluster in both directions

Until done (maximum cluster size or number):
- Compute cluster adjacency matrix
- Find circuits in adjacency matrix
- Cluster circuits



Example: Grid world

Factored

Divisive: All states start in same cluster
Cluster *sets* of states (fStates)
Rule: f_1 and f_2 can be clustered together if all states in f_1 can reach some state in f_2 and vice-versa
Operations:
- **Split** cluster C on cluster S creates
  - $C_S = C \cap S$
  - $C_{\setminus S} = C \setminus S$
- **Insert** cluster N into cluster C
  - For set S let $R_{\to S}$ be all states that can reach some state in S and $R_{\leftarrow S}$ be all states that can be reached by some state in S
  - $C \leftarrow N = (C \cap R_{\to N} \cap R_{\leftarrow N}) \cup (N \cap R_{\to C} \cap R_{\leftarrow C})$
- **Prune**: Removes all empty and duplicate clusters

Until no more starting fStates exist:
- Start fState s = new set of states with same transition probability to previous start state or goal
- Find circuit N including S
- Split each cluster C on $C \leftarrow S$
- Prune