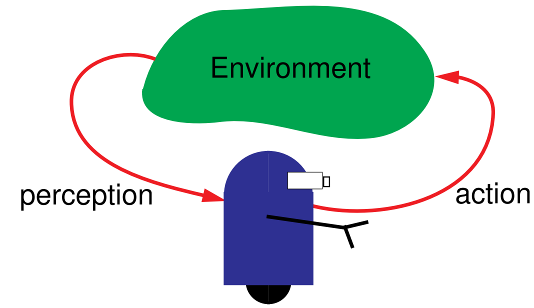
Markov Decision Processes:

A Foundation for Intelligent Agents

Leslie Pack Kaelbling
Computer Science Department
Brown University

Agents Interact with an Environment



Traditional Action Framework

How should an agent choose its actions?

- treat world as having deterministic dynamics
- take actions to optimize outcome
- if actions do not have expected results, make a new plan of action

The world model is often an approximation in which actions are assumed to have

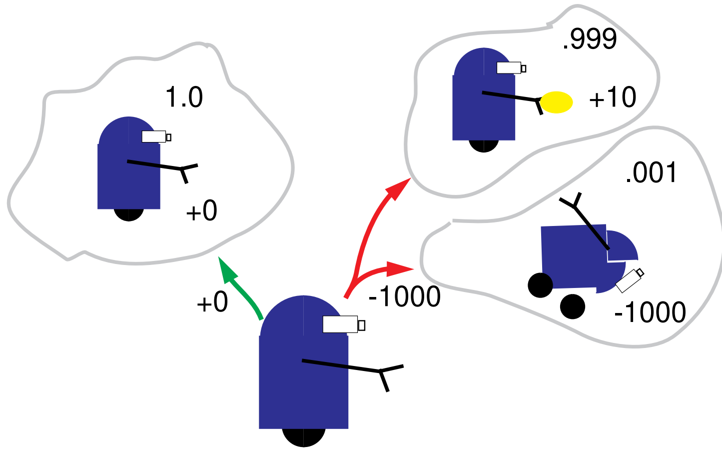
- worst-case results
- most likely results

Sometimes these approximations are inappropriate.

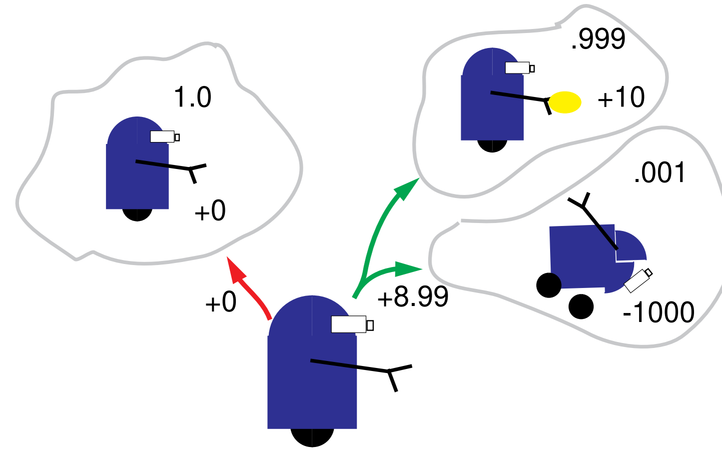
Nondeterministic Domains

- active medical diagnosis
- air traffic control
- target tracking
- robot navigation

Worst Case World View



Expected Value World View



Markov Decision Processes

- finite set of states S
- finite set of actions A
- discrete time
- actions result in immediate rewards

$$R: S \times A \to \Re$$

- actions result in state transitions

$$T: S \times A \times S \to \Re$$

- transitions depend only on the last state and action

Bellman, 1957; Howard, 1960

Models of Optimal Behavior

- finite horizon

$$E\left(\sum_{t=0}^{k} r_t\right)$$

- infinite horizon discounted

$$E\left(\sum_{t=0}^{\infty} \gamma^t r_t\right)$$

- average case

$$\lim_{k \to \infty} E\left(\frac{1}{k}\sum_{t=0}^{k} r_t\right)$$

Reward Functions and Goals

Can encode most goals as reward functions:

$\mathrm{asap}(Q)$: enter a state satisfying Q as soon as possible

- if $Q(s)$ then $R(s) = 0$; otherwise $R(s) = -1$
- make all states such that $Q(s)$ absorbing

$\mathrm{alap}(Q)$: stay in states satisfying Q as long as possible

- if $Q(s)$ then $R(s) = 0$; otherwise $R(s) = -1$
- make all states such that $\neg Q(s)$ absorbing

Use infinite horizon discounted model.

Many other goals possible in this rich language.

Using the MDP Framework

- planning: find a policy given a model
- learning: find a policy without a model
- two extensions to the framework

MDPs as a basis for planning

What is a Plan?

- a *single trajectory*: unlikely to be successful in a stochastic domain
- a *universal plan*: called a *policy* in the language of MDPs; often too expensive to compute
- a *partial policy*: a set of reaction rules covering a subset of the world states

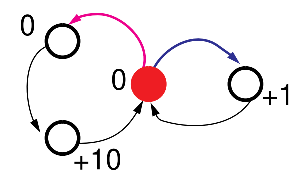
Finite Horizon

Act to gain as much reward as possible in k steps

- optimal policy is non-stationary; depends on the number of steps remaining
- choose action to perform on last step
- work backwards from there
- computation time depends on horizon

Finite Horizon Example

- one step to go: right
- otherwise: left



Infinite Horizon Discounted

- a policy is a mapping from states to actions: $\pi\colon S \to A$
- optimal value of a state $V^*(s) = \max_\pi E_\pi\left(\sum_{t=0}^{\infty} \gamma^t r_t\right)$
- the optimal value function V^* is unique
  $$V^*(s) = \max_a\left[R(s,a) + \gamma \sum_{s'} T(s,a,s') V^*(s')\right]$$
- a stationary optimal policy exists
  $$\pi^*(s) = \operatorname{argmax}_a\left[R(s,a) + \gamma \sum_{s'} T(s,a,s') V^*(s')\right]$$

Find an Optimal Policy: Value Iteration

Value iteration algorithm:

initialize $V(s)$ **arbitrarily**
loop
  loop for $s \in S$
    $$V(s) := \max_a\left[R(s,a) + \gamma \sum_{s' \in S} T(s,a,s') V(s')\right]$$
  end loop
end loop

- assignments to V can be done asynchronously and/or in parallel
- knowing V gives us π

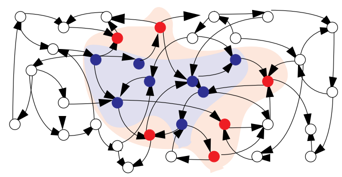
Bellman, 1960; Bertsekas & Tsitsiklis, 1989

Generating Partial Policies

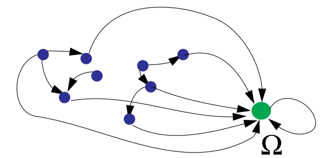
A *partial policy*, π, is a mapping from a subset of S into A.

The *envelope*, E, of a partial policy is the set of states over which the policy is defined.

The *fringe* of a partial policy is the set of states that are not in the envelope of the policy, but that can be reached in one step from some state in the envelope.



Restricted MDP



Choose $R(\Omega)$ to be negative, with magnitude reflecting the consequences of reaching a state on which the policy is undefined (falling out)

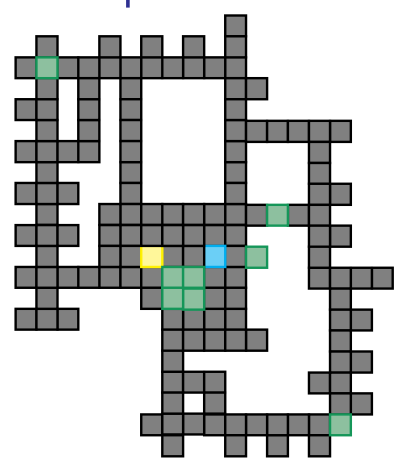
Planning Algorithm

Generate a very tenuous strategy; keep adding states to the envelope, making it more robust.

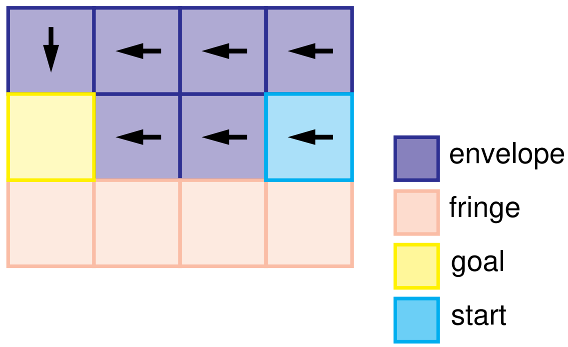
- Generate an initial envelope E
- While $E \neq S$ and (not deadline) do
  - Extend the envelope E
  - Generate an optimal policy π for the restricted MDP with state set $E \cup \{\Omega\}$
- Return π

Dean, Kaelbling, Kirman, Nicholson, 1993

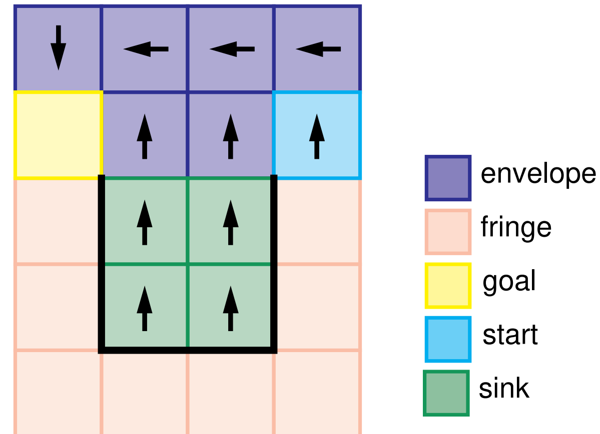
Example: Floor Plan



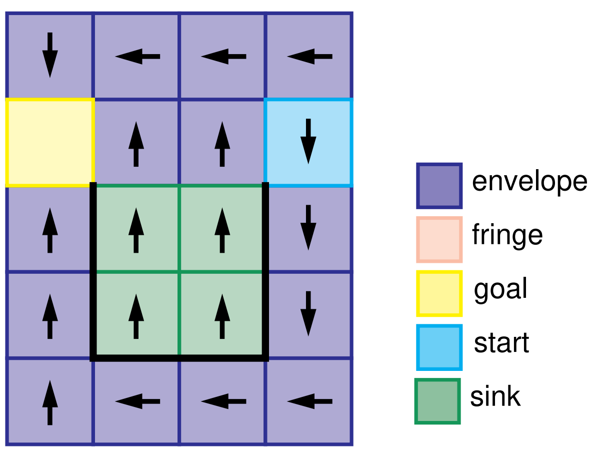
Planning Example: Initial Path



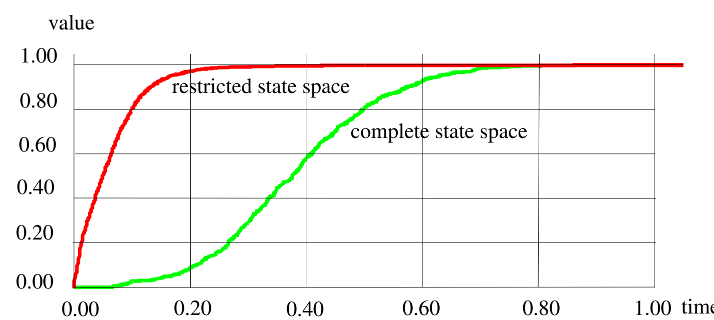
Planning Example: Finding Danger



Planning Example: Avoiding Danger



Experimental Results



MDPs as a basis for learning

Learning an Optimal Policy

- first learn a model (the T and R functions)
- then find the optimal policy using value iteration
- finally, execute that policy for ever more

Not very satisfactory for an agent in a potentially changing world.

Instead, let's try to learn a policy without first learning a model.

Temporal Credit Assignment

How do we know whether the action we chose was a good one, when it might have far reaching effects?

- we could wait until the "end" and reward the actions if the result was good and punish them, if bad
- instead, we'll use insights from value iteration to adjust the value of a state based on the immediate reward and the value of the next state
- Q learning is an instance of this *temporal difference* learning method

Sutton, 1988

Q Values

Let $Q^*(s, a)$ be the expected discounted reinforcement of taking action a in state s, then continuing with the policy of choosing actions to maximize $Q*$

$$Q^*(s, a) = R(s, a) + \gamma \sum_{s'} T(s, a, s') \max_{a'} Q^*(s', a')$$

Theorem:

$\pi^*(s) = \operatorname{argmax}_a Q^*(s, a)$ is an optimal policy

Watkins, 1990

Q-Learning

Estimate the Q values on-line, finding the policy and the value function together:

$$Q(s,a) \leftarrow (1-\alpha)\,Q(s,a) + \alpha\,(r + \gamma \max_{a'} Q(s',a'))$$

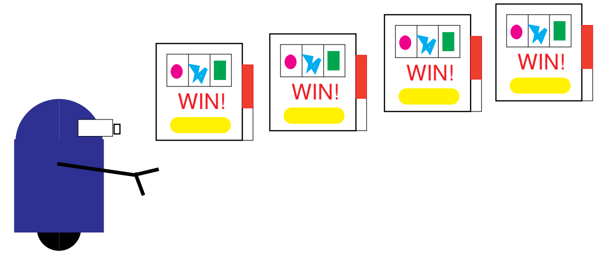
Update whenever we take action a in state s

Theorem: If each action is executed in each state an infinite number of times on an infinite run, the Q values will converge to Q^*, yielding an optimal policy.

Watkins & Dayan, 1992

Exploration vs Exploitation

To satisfy the theorem, we must take exploratory actions; to gain reward, we must take exploitationary actions.



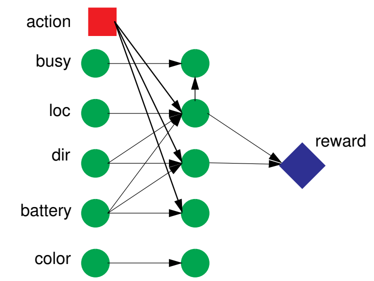
Open question: how to explore an MDP optimally

Representation

- enumeration is an inefficient representation strategy
- function approximation methods allow transfer in learning
- abstraction and hierarchy increase efficiency, perhaps at some expense in correctness

Compositional Models

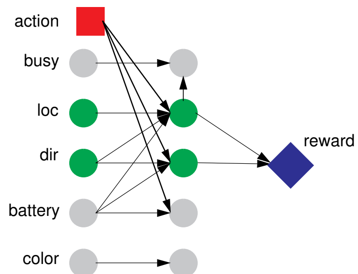
Use a belief network with two time slices and a value node to characterize the effects of actions:



Multi-stage belief network can be used for finite-horizon planning. *[Nicholson, 1992; Dean & Kanazawa, 1989]*

Construct Approximate Models

Given a belief network, use sensitivity analysis to choose relevant variables



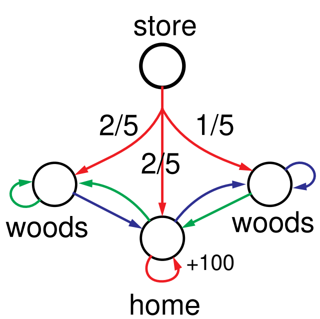
- marginalize out all but the most relevant variables
- add others back in if time remains

Hidden State

- agent gets observations only
- doesn't know current state
- may have to act to gain information

Treating Observations as States

- Q-learning and value iteration can be led astray
- finding the best mapping is NP-hard



Chrisman, 1992; Littman, 1994

Probabilistic Policies

- acting randomly can be better
- Q-learning-like algorithm converges to locally optimal probabilistic policy

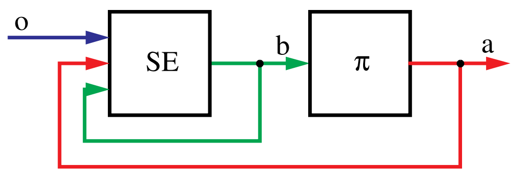
When in the woods, it is best to

- go right with probability $2 - \sqrt{2} \approx 0.6$
- go left with probability $\sqrt{2} - 1 \approx 0.4$

Jaakkola, Jordan, & Singh, 1994

Perfect Memory

- makes the problem completely observable
- probability distribution over states (belief states)
- given a model, building state estimator (SE) is easy



Partially Observable MDPs

- use belief states as states
- good: belief state is sufficient for acting optimally
- bad: V is from n-dimensional vectors to values
- ok: value function is approximately piecewise-linear and convex
- model can be learned using HMM techniques

Lovejoy, 1991; Cassandra, Kaelbling & Littman, 1994

Conclusion

Markov Decision Processes provide a solid foundation for planning and learning in intelligent agents; they

- allow trade-offs based on utility and probability
- provide an expanded vocabulary for expressing goals
- engender more flexible and robust notions of “plan”
- present interesting research problems in hidden state, abstraction, and risk