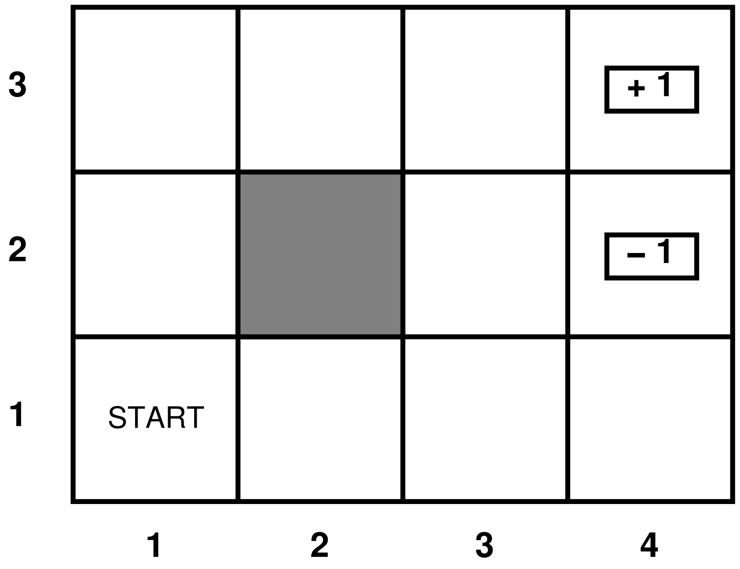
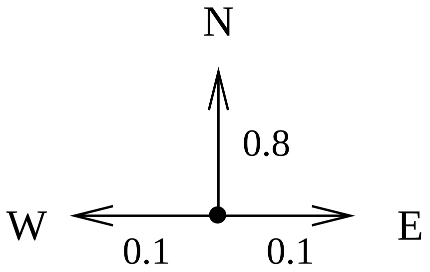
Utility-Based Agents for Multi-Step Decision-Making



Agent can move North, South, East, or West. If it reaches [4,2] or [4,3] it terminates automatically. Transition model $\mathbf{P}(S_{t+1}|S_t, A)$:



Immediate Reward and Total Utility

The utility function is specified in terms of the cumulative immediate reward:

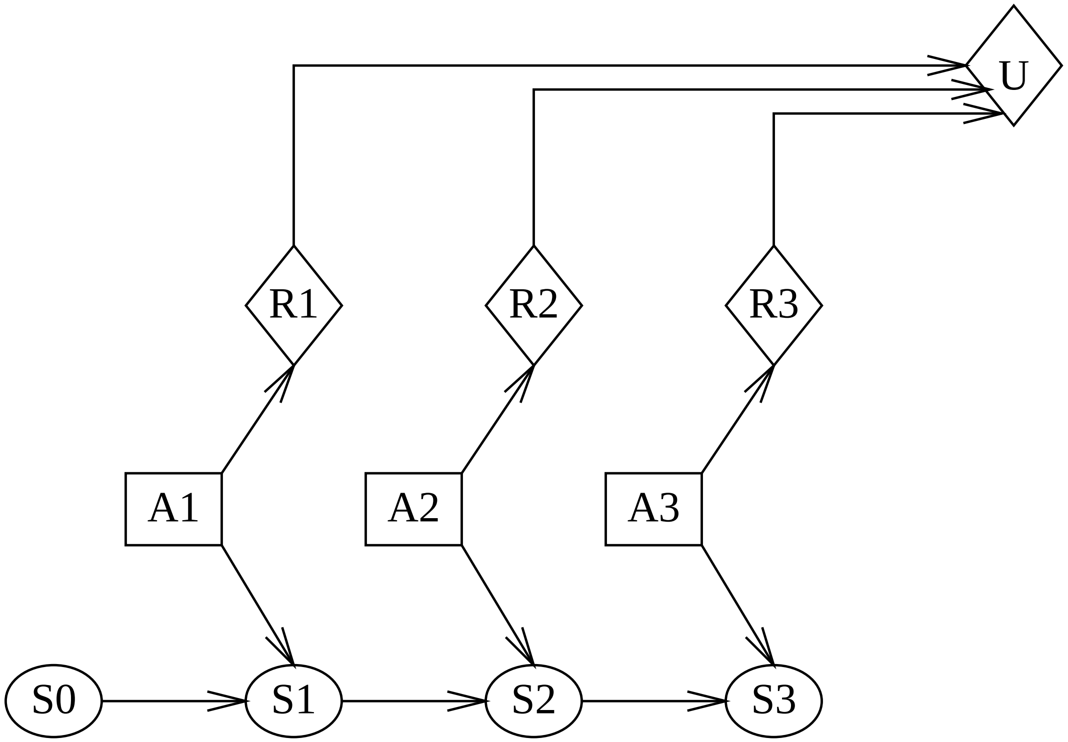
$$R(S_{t+1}|S_t, A) = \begin{cases} -\frac{1}{25} & \text{if } S_{t+1} \neq [4,2] \text{ or } [4,3] \\ 1 - \frac{1}{25} & \text{if } S_{t+1} = [4,3] \\ -1 - \frac{1}{25} & \text{if } S_{t+1} = [4,2] \end{cases}$$

The utility of a *sequence* of actions and states is defined as

$$\begin{aligned} U([s_0, a_1, s_1, \ldots, a_n, s_n]) &= \begin{cases} 1 - \frac{n}{25} & \text{if } s_n = [4,3] \\ -1 - \frac{n}{25} & \text{if } s_n = [4,2] \end{cases} \\ &= \sum_t R(s_{t+1}|s_t, a_t) \end{aligned}$$

where $R(S_{t+1}|S_t, A_t)$ is the *immediate reward function*.

Designing Optimal Multistep Agents



Three General Cases

- **Unobservable environment** (no sensors).
- **Fully observable environment**. Markov Decision Problem (MDP).
- **Partially observable environment**. Partially-observable Markov Decision Problem (POMDP).

Unobservable Environments

Choose a sequence of actions $[a_1, \ldots, a_n]$ that maximizes the expected utility:

$$\underset{[a_1,\ldots,a_n]}{\text{argmax}} \sum_{[s_0,\ldots,s_n]} P(s_0, \ldots, s_n | a_1, \ldots, a_n) \cdot U([s_0, a_1, \ldots, a_n, s_n])$$

For each possible sequence of actions $[a_1, \ldots, a_n]$

compute $P(S_0, S_1, \ldots, S_n | a_1, \ldots, a_n)$

compute $U([s_0, a_1, \ldots, a_n, s_n])$

compute $score = \sum_{[s_0,\ldots,s_n]} P(s_0, \ldots, s_n | a_1, \ldots, a_n) \cdot U([s_0, a_1, \ldots, a_n, s_n])$

remember the sequence that gives the maximum *score*

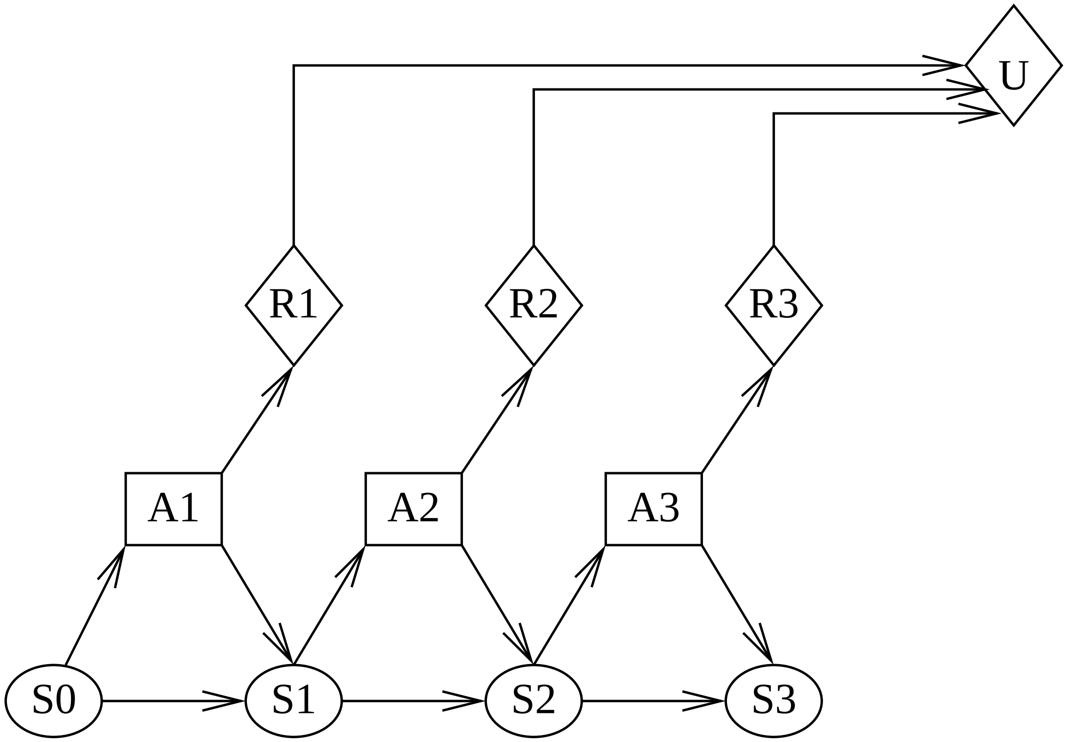
Note:

$$P(S_0, S_1, \ldots, S_n | a_1, \ldots, a_n) = P(S_1 | S_0, a_1) \cdot P(S_2 | S_1, a_2) \cdots P(S_n | S_{n-1}, a_n)$$

$$U([s_0, a_1, \ldots, a_n, s_n]) = \sum_t R(S_{t+1} | S_t, a_{t+1})$$

Therefore, for each sequence of actions, this requires a search of all possible sequences of resulting states. If $P(S_{t+1} | S_t, A_{t+1})$ is deterministic, this can be solved using search.

Observable (Accessible) Environments
Markov Decision Problems



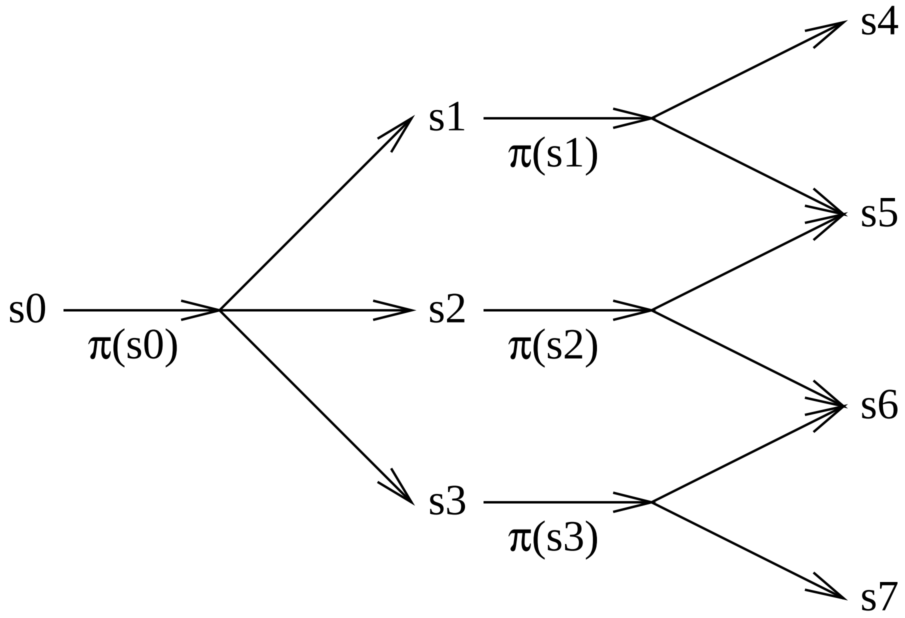
After each action, the agent can observe the resulting state S_{t+1}. This is indicated by an arrow from each state to the next action.

Observable (Accessible) Environments
Markov Decision Problems

In a fully accessible environment, the optimal agent can be a reflex agent, because the current state contains the same information as the percept history (this is called the Markov Property: The past is conditionally independent of the future given the current state.)

An optimal agent is therefore a mapping from states to actions. This is called a *policy*:

$$\pi(s) = a$$



Defining a Utility Function Over States

$U(S|\pi)$ = expected cumulative reward of executing policy π starting in state S. (also known as a *value function*)

$$U(S_t|\pi) = \sum_{S_{t+1}} P(S_{t+1}|S_t, \pi(S_t)) \cdot [R(S_{t+1}|S_t, \pi(S_t)) + U(S_{t+1}|\pi)]$$

Key Point: We can use $U(S_{t_1}|\pi)$ because we can observe S_{t+1} after taking the action $\pi(S_t)$.

Bellman's Equation

Define the *optimal policy* to be

$$\pi^* = \operatorname*{argmax}_{\pi} U(S_0|\pi)$$

Define the *optimal state utility function* to be

$$U^*(S) = U^*(S|\pi^*)$$

Theorem (Bellman and Dreyfus)

$$\begin{aligned} U^*(S_t) &= \max_a \sum_{S_{t+1}} P(S_{t+1}|S_t, a) \cdot [R(S_{t+1}|S_t, a) + U^*(S_{t+1})] \\ \pi^* &= \operatorname*{argmax}_a \sum_{S_{t+1}} P(S_{t+1}|S_t, a) \cdot [R(S_{t+1}|S_t, a) + U^*(S_{t+1})] \end{aligned}$$

Value Iteration: An algorithm for computing U^*

$U(S) = 0$ for all S

while U is changing **do**

  for each state S **do**

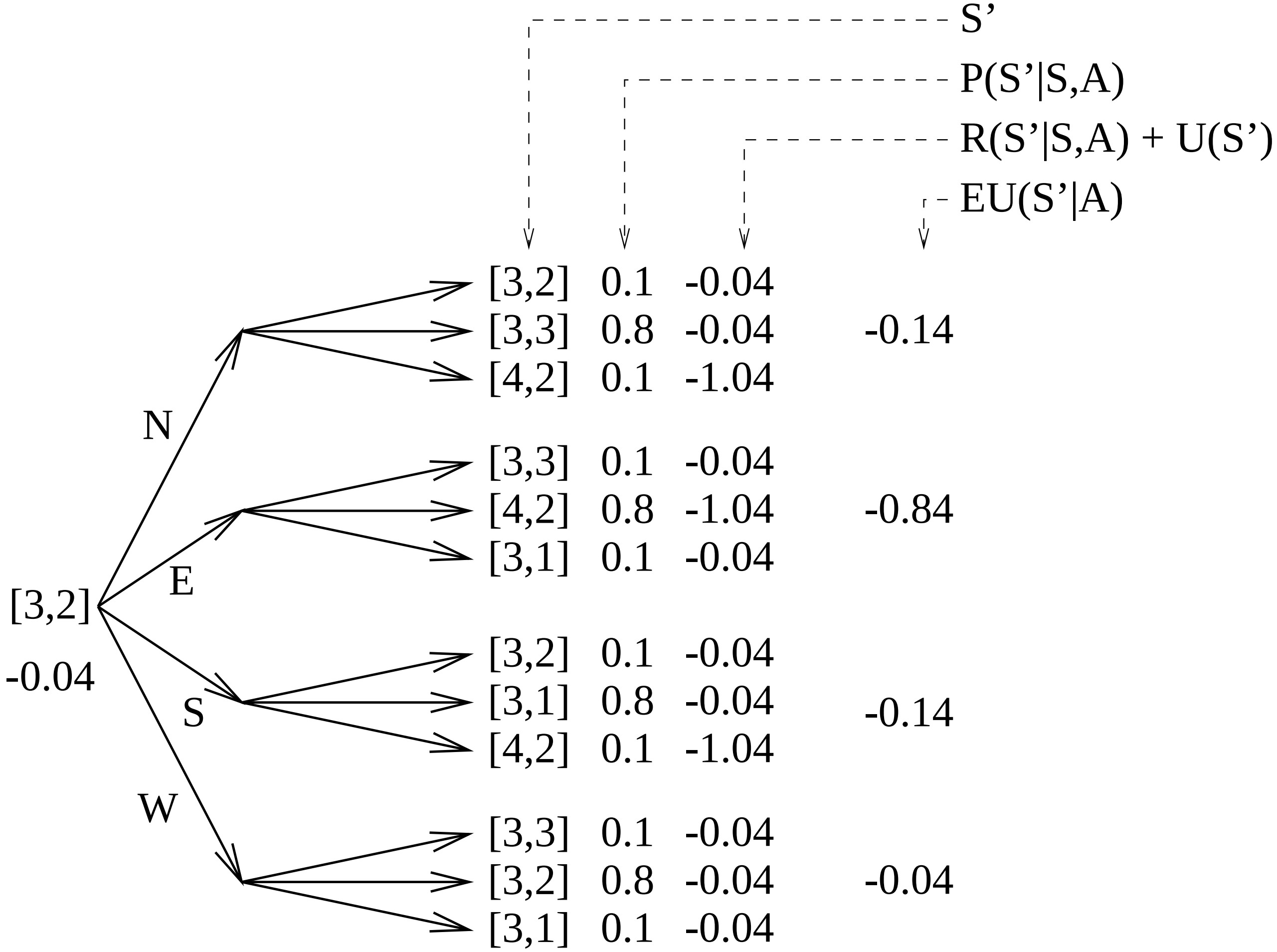
    $U(S) := \max_a \sum_{S'} P(S'|S, a) \cdot [R(S'|S, a) + U(S')]$

    end // for

  end // while

The update of U is called a *Bellman backup*.

Example Bellman Backup

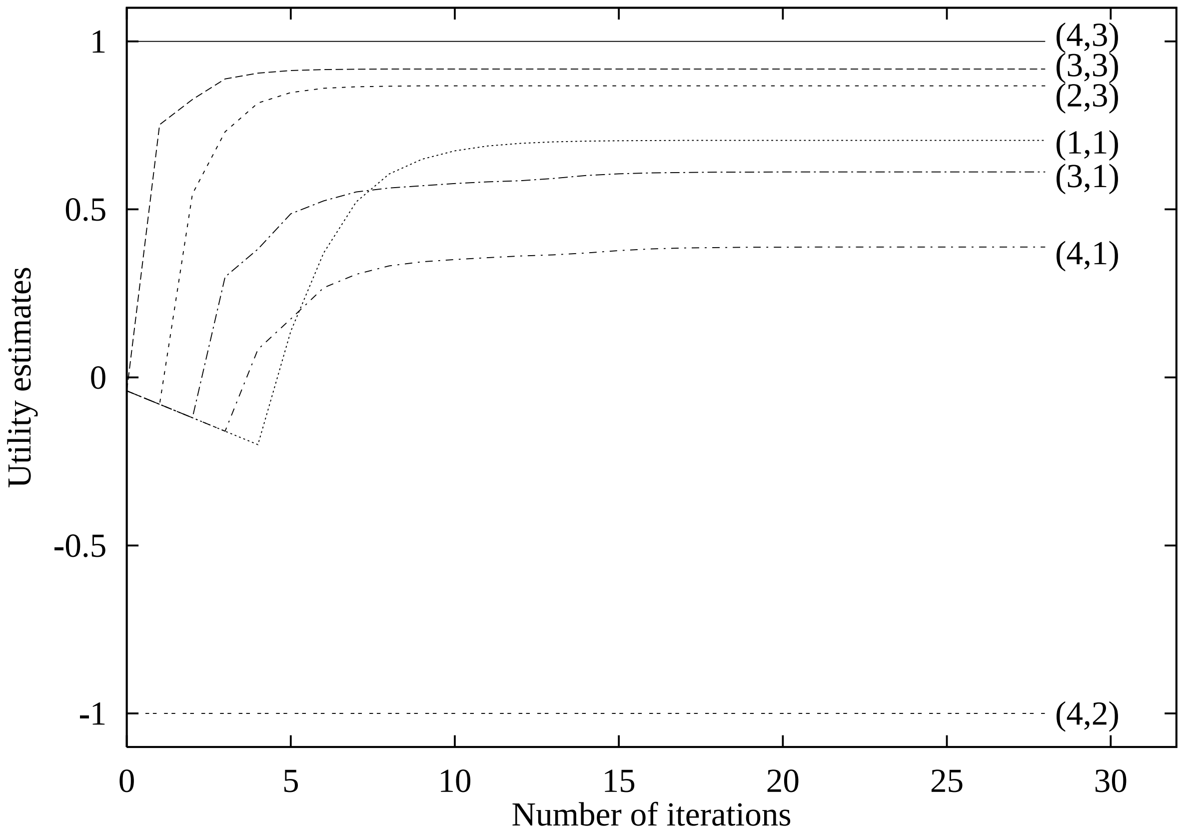


Optimal Value Function and Policy

3	→	→	→	+1
2	↑		↑	−1
1	↑	←	←	←
	1	2	3	4

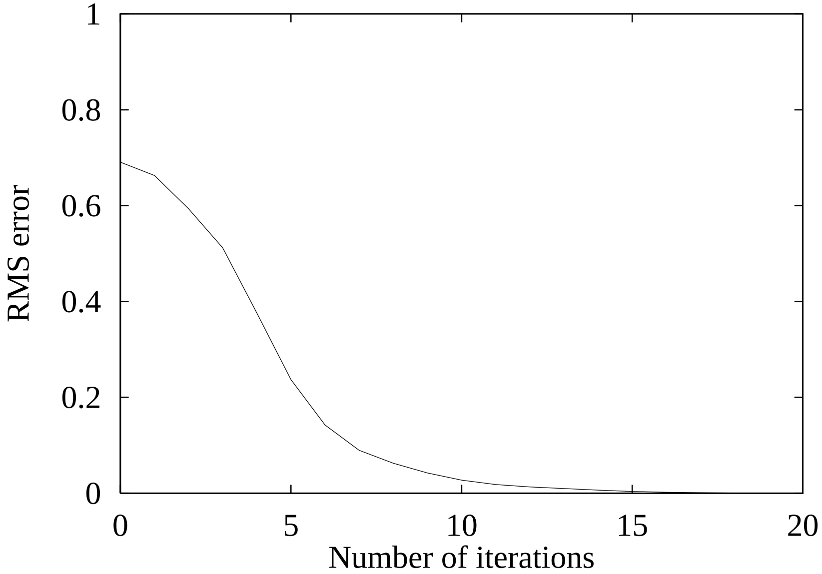
3	0.812	0.868	0.912	+1
2	0.762		0.660	−1
1	0.705	0.655	0.611	0.388
	1	2	3	4

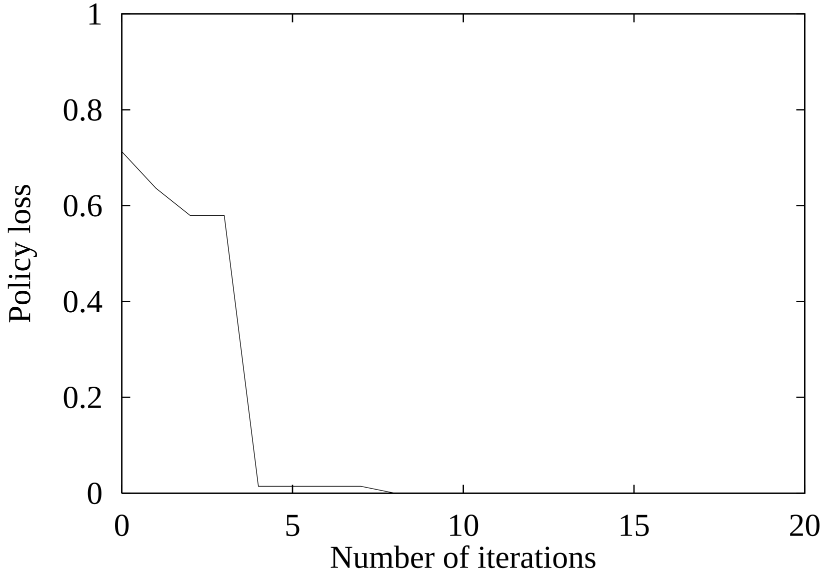
Behavior of Value Iteration



The utility value for selected states at each iteration step in VALUEITERATION.

Convergence of Value Iteration (2)





Policy Iteration

Choose a policy
Repeat
    Compute value of the policy (a *linear* problem)
    Compute a new policy by taking argmax action in each state
until convergence

procedure PolicyIteration
$\pi(S)$ = initial policy for all states S
repeat
    // value determination
    U := ValueDetermination(π)
    // policy improvement
    for each state S
        $\pi(S) := \operatorname*{argmax}_a \sum_{S'} P(S'|S,a)[R(S'|S,a) + U(S'|\pi)]$
    end // for
until π converges
end // PolicyIteration

Iterative Implementation of ValueDetermination

ValueDetermination can be implemented using Gaussian elimination and many other methods for solving a linear system. The following method is basically value iteration without the "max":

function ValueDetermination(π)
$U(S|\pi) := 0$ **for** all states S
repeat
    for each state S
        $U(S|\pi) := \sum_{S'} P(S'|S, \pi(S))[R(S'|S, \pi(S)) + U(S')]$
    end // for
until U converges
return U
end // ValueDetermination

The Curse of Dimensionality

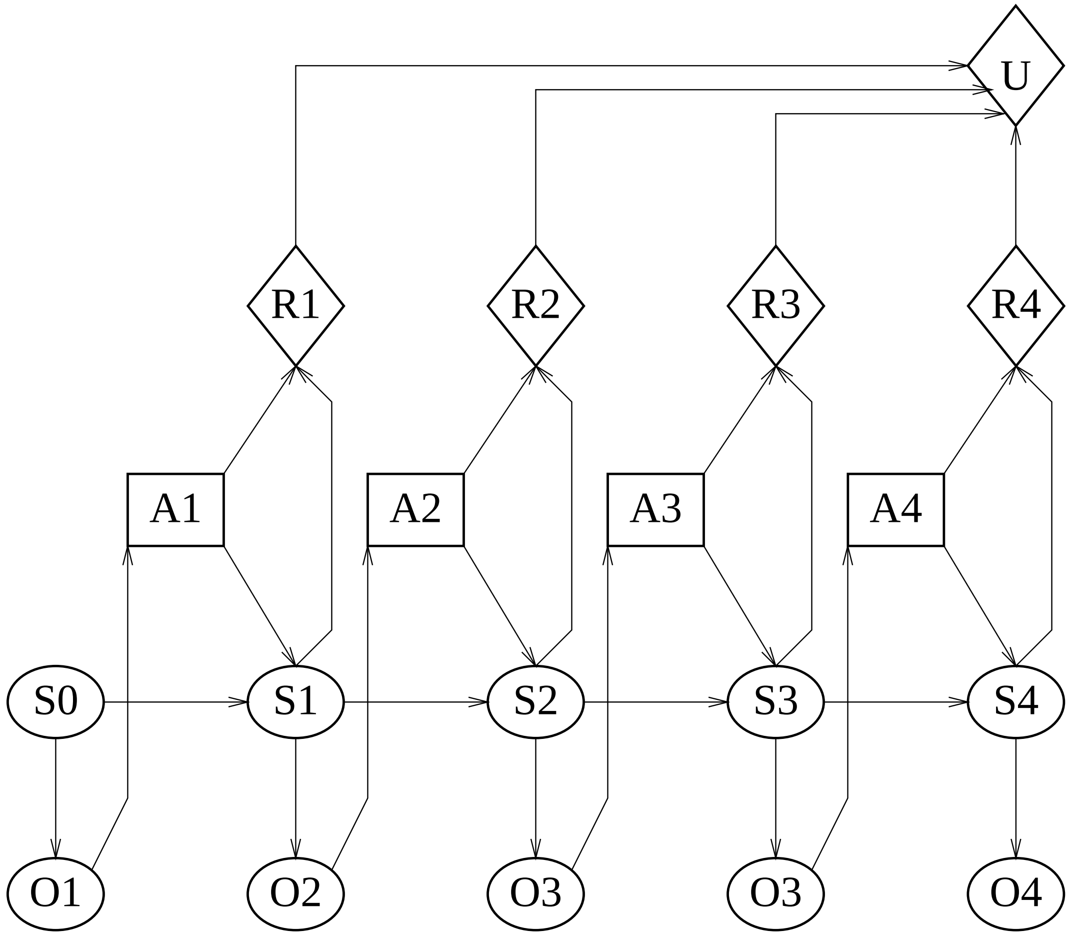
Computational cost scales with the number of states $|S|$.

The number of states is exponential in the number of "dimensions" (sensors).

Bellman called this the "Curse of Dimensionality"

- **Value Iteration**: $O(n|S||A|B)$
- **Value Determination**: $O(n'|S|B)$
- **Policy Iteration**: $O(n''|S||A|)$

Partially Observable Markov Decision Problems
Dynamic Decision Networks



Approximate Method for Solving POMDP's

Two Key Ideas:

- **Compute optimal value function** $U(S)$ **assuming complete observability**
- **Maintain** $Bel(S_t) = P(S_t|O_t, A_t, S_{t-1}, \ldots, S_0, O_0)$

At each time t

- **Observe current percept** O_t
- **Update** $Bel(S_t) = \mathbf{P}(S_t|O_t, A_t)$

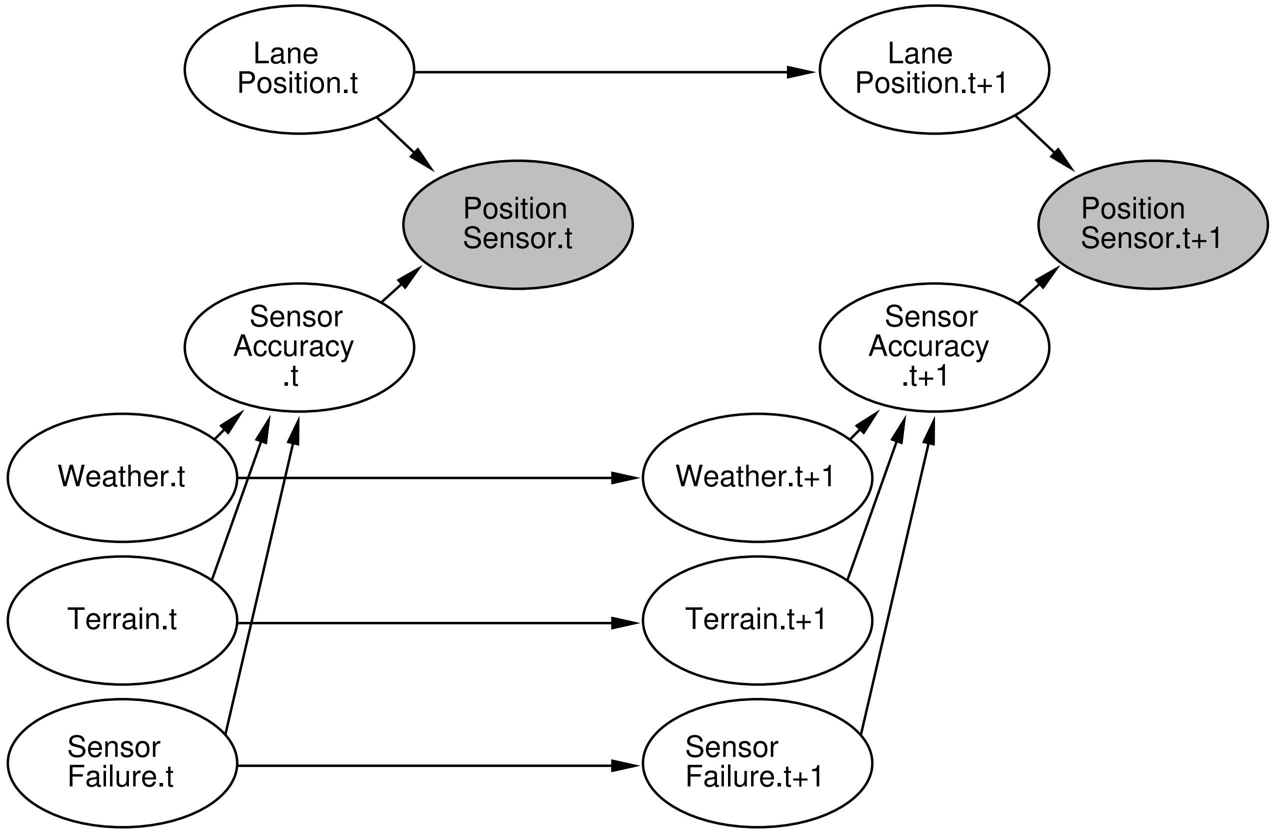
$$\mathbf{P}(S_t|O_t A_t) = \alpha[\sum_{S_t} Bel(S_t) \cdot \mathbf{P}(S_t|S_{t-1}, A_{t-1})] \cdot \mathbf{P}(O_t|S_t)$$

- **Choose the next** k **optimal actions** $[a_{t+1}, \ldots, a_{t+n}]$ **to maximize**

$$\sum_{S_{t+1},\ldots,S_{t+k}} \sum_{O_{t+1},\ldots,O_{t+k}} \alpha\mathbf{P}(S_{t+1}|S_t, a_{t+1})]\mathbf{P}(O_{t+1}|S_{t+1}) \cdots \mathbf{P}(S_{t+k}|S_{t+k-1}, a_{t+k})\mathbf{P}(O_{t+k}|S_{t+k}) \cdot \left[\sum_{i=1}^{k} R(S_{t+i}|S_{t+i-1}, a_{t+i}) + U(S_{t+k})\right]$$

- **Perform action** a_{t+1}**.**

Representing the State as a Set of Random Variables



This reduces the complexity of representing $\mathbf{P}(S'|S, A)$ and $\mathbf{P}(O|S)$.

Dynamic Decision Networks

- Handle uncertainty correctly and sometimes efficiently.
- Deal with continuous streams of sensor input.
- Handle unexpected events, because they have no fixed "plan".
- Handle noisy sensors and sensor failure.
- Can act in order to obtain information as well as to receive rewards.
- Can handle relatively large state spaces because they decompose the state into a set of state variables with sparse connections.
- Exhibit graceful degradation under time pressure and in complex environments using various approximation techniques.

Probabilistic Agents Summary

- **Three key components:**
  - $P(S'|S, A)$ (action model)
  - $P(O|S)$ (sensor model)
  - $R(S'|S, A)$ (reward function)
- **Value iteration and policy iteration work in accessible environments**.

  They compute a local (state) utility function and an optimal policy.
- **Lookahead search gives approximate solutions in unobservable or partially-observable environments**

  Updating the current beliefs in a DDN is easy.

  Lookahead search is hard.

Open Problems in Probabilistic Agents

- **First-order probabilistic representations**
- **More efficient algorithms for POMDP's**
- **Methods for scaling up MDP's**