2534 Lecture 4: Sequential Decisions and Markov Decision Processes

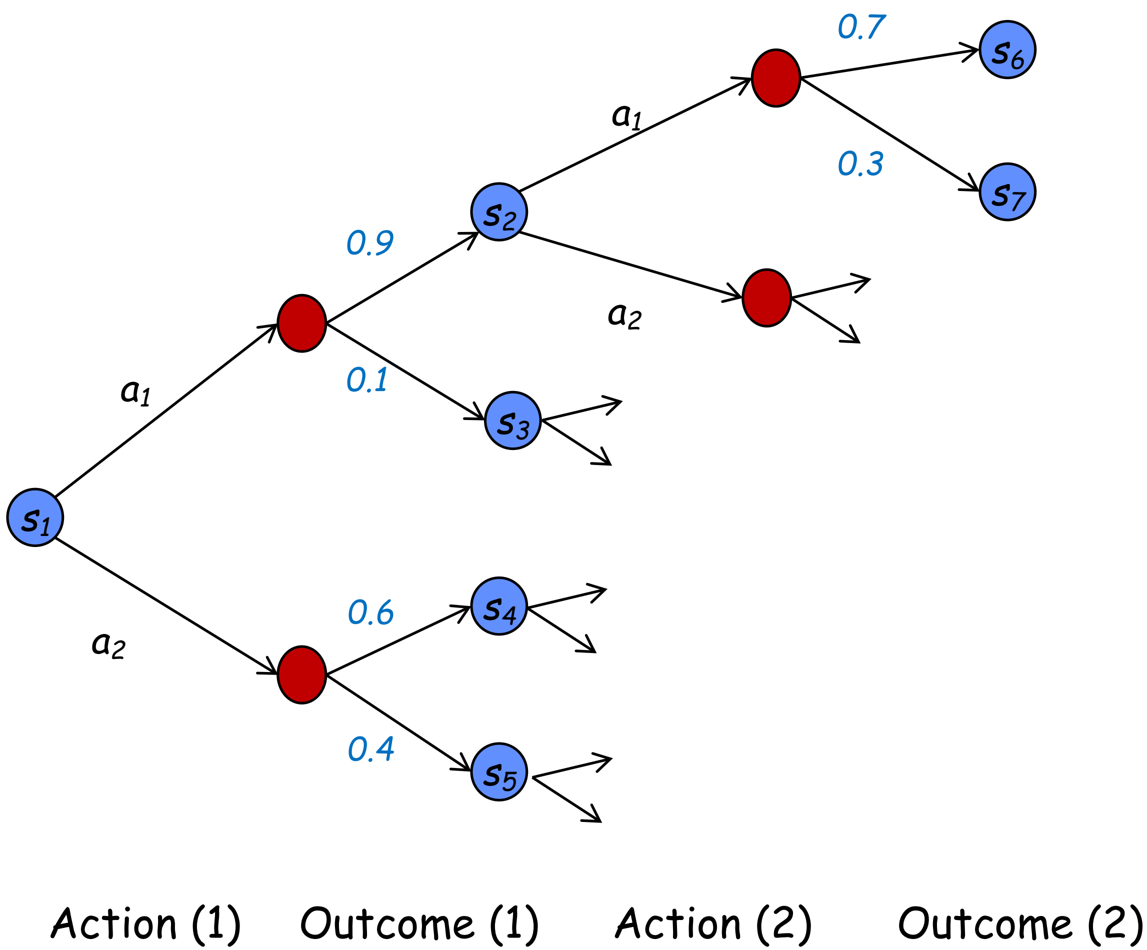
- Briefly: preference elicitation (last week's readings)
  - *Utility Elicitation as a Classification Problem.* Chajewska, U., L. Getoor, J. Norman,Y. Shahar. In Uncertainty in AI 14 (UAI '98), pp. 79-88, 1998.
  - *Constraint-based Optimization and Utility Elicitation using the Minimax Decision Criterion.* C. Boutilier, R. Patrascu, P. Poupart, and D. Schuurmans. Artificial Intelligence 170:686-713, 2006.
- Sequences of Decisions
  - Basic considerations
  - Quick discussion of decision trees
- Basics of Markov Decision Processes (MDPs)
- Announcements
  - Asst.1 posted yesterday, due in two weeks (Oct.13)
  - See web page for handout on course projects

Sequential Decision Problems

- Few decisions in life can be treated in isolation
- Sequences of decision are much more common
  - think of Robbie's plans for maintaining the lab, etc.
- We take actions not just for their *immediate benefit*, but:
  - because they *lead to opportunities* to take other actions
    - Robbie risks getting crushed in the street to buy coffee
  - because they *provide information* that can inform future decisions
    - Doctor takes MRI before deciding on course of treatment
  - and a *combination* of all three (benefits, opportunities, info)

  - We'll set aside information gathering until next time…

A Simple Perspective



- To compute best action sequence

1. Assign utility to each trajectory
   - e.g., u(s1 → s2 → s6)
2. For each sequence of actions compute prob of any trajectory
   - e.g., Pr(s1 → s2 → s6| [a1,a1]) = 0.9*0.7 = 0.63
3. Compute EU of each action sequence:
   - EU of [a1,a1], [a1,a2], [a2, a1], [a2,a2]
   - Choose the best

What's wrong with this perspective?

- **Practical:** easier to think of utility of individual states (and action costs) then utility of entire trajectories

- **Computational:** k actions, t stages: k^t action sequences to evaluate; and if n outcomes per action, $k^t n^t$ trajectories!

- **Conceptual:** sequences of actions are often not the right form of behavior:
  - After doing a_1, I go to s_2 or s_3. It may be better to do a_1 again if I end up to s_2, but best to do a_2 if I end up at s_3.

Policies

- Can only be captured with *policies*
  - assume observable outcomes
  - Takes form: *Do* a_1; *if* s_2, *do* a_1, *if* s_3, *do* a_2; …
- Policies make more state trajectories possible
  - Hence they (weakly) increase EU of best behavior, since they includes sequences as a special case
- Difficulty: far more policies than sequences
  - computation problem seemingly harder
  - *dynamic programming* comes to the rescue
- First *decision trees* (briefly)
- Then (our focus): *Markov decision processes (MDPs)*

Decision Trees

- Simple way to structure sequences of decisions
- Consists of:
  - *decision nodes:* representing actions available to decision maker
  - *chance nodes:* representing uncertain outcomes of decisions; must be labeled with *observable events*
  - sequencing of decisions based on observed
- A simple form of *dynamic programming* allows one to compute optimal course of action, or *policy*
  - choices at each stage can depend on observed outcomes at any previous stages
  - same principle as backward induction in extensive form games

Simple Example

- ABC Computer needs to decide if (and how) to bid on a government contract for 10,000 special purpose computers
- One other potential bidder (Complex Inc.), low bidder wins
- New manufacturing process being developed, uncertain of true costs!
  - under current process: cost is $8000/unit
  - under new process?   0.25 $5000;   0.50 $7500; $0.25 $8500

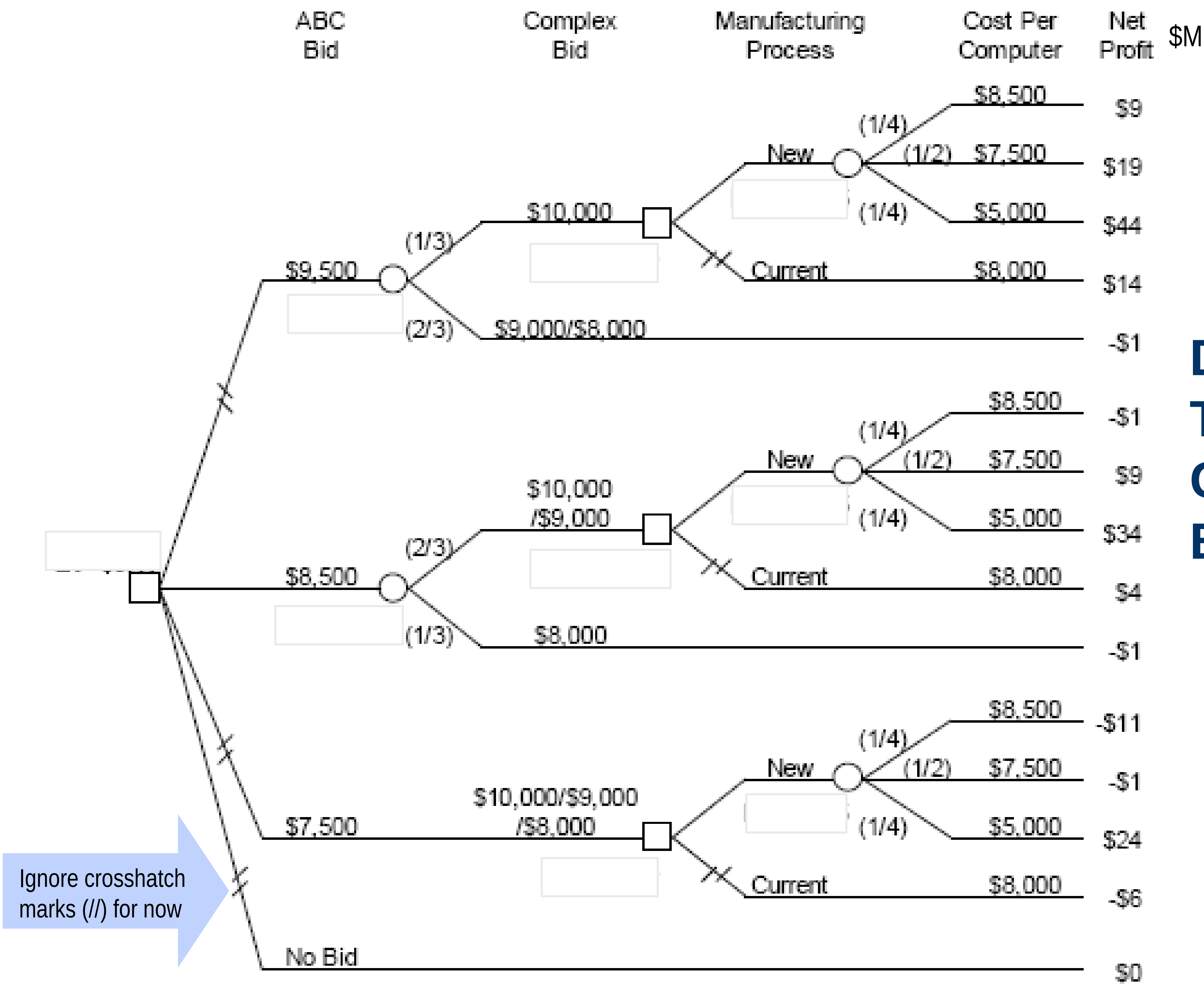
- Three bids for ABC to consider: $9500 per unit, $8500, or $7500
- Prepping bid will cost $1M
- Complex will bid $10,000 per unit, $9000 or $8000 *(Pr = 1/3 each)*

- Should ABC bid? If so, should it bid $7500, $8500, or $9500?

Decision Sequencing

- First decision:
  - whether to bid (and what)
- Second decision:
  - if it wins: attempt new process or use old process
  - predicting outcome of this impacts bidding decision
- Structure decisions in *decision tree*
  - *Decision nodes* (square): emerging edges labeled with actions, point to (i) next decision nodes or (ii) chance nodes if stochastic
  - *Chance nodes* (circles): emerging edges indicate possible outcomes and their probabilities; must be *observable*
  - *Terminal nodes:* final outcome of trajectory (labeled with utilities)

Decision Tree for Contract Bidding



From Craig Kirkword: A Primer on Decision Trees

Backward Induction (Rollback, DP)

- Value of a terminal node T: $EU(T) = U(T)$

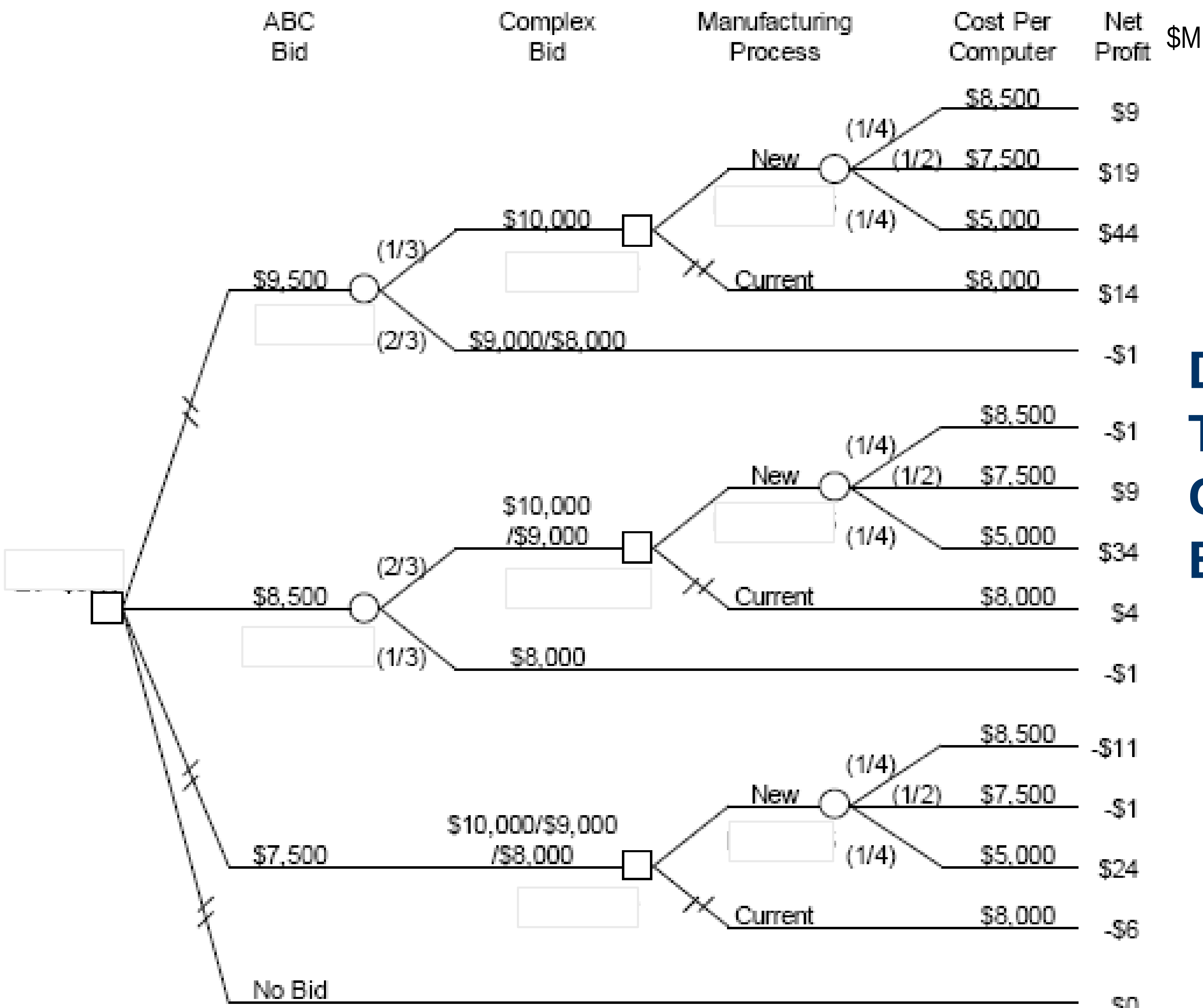
  i.e., utility given in problem spec.

- Value of chance node *C*: $$EU(C) = \sum_{n \in Child(C)} \Pr(D) EU(n)$$

- Value of decision node *D:* $$EU(D) = \max_{n \in Child(D)} EU(n)$$

- Policy π: maximize decision *d* at each decision node D
  - Recall edge to each child labeled with a decision *d*

$$\pi(D) = \arg\max_{C \in Child(D)} EU(C)$$



Decision Tree for Contract Bidding

From Craig Kirkword: A Primer on Decision Trees

Decision Trees: Wrap

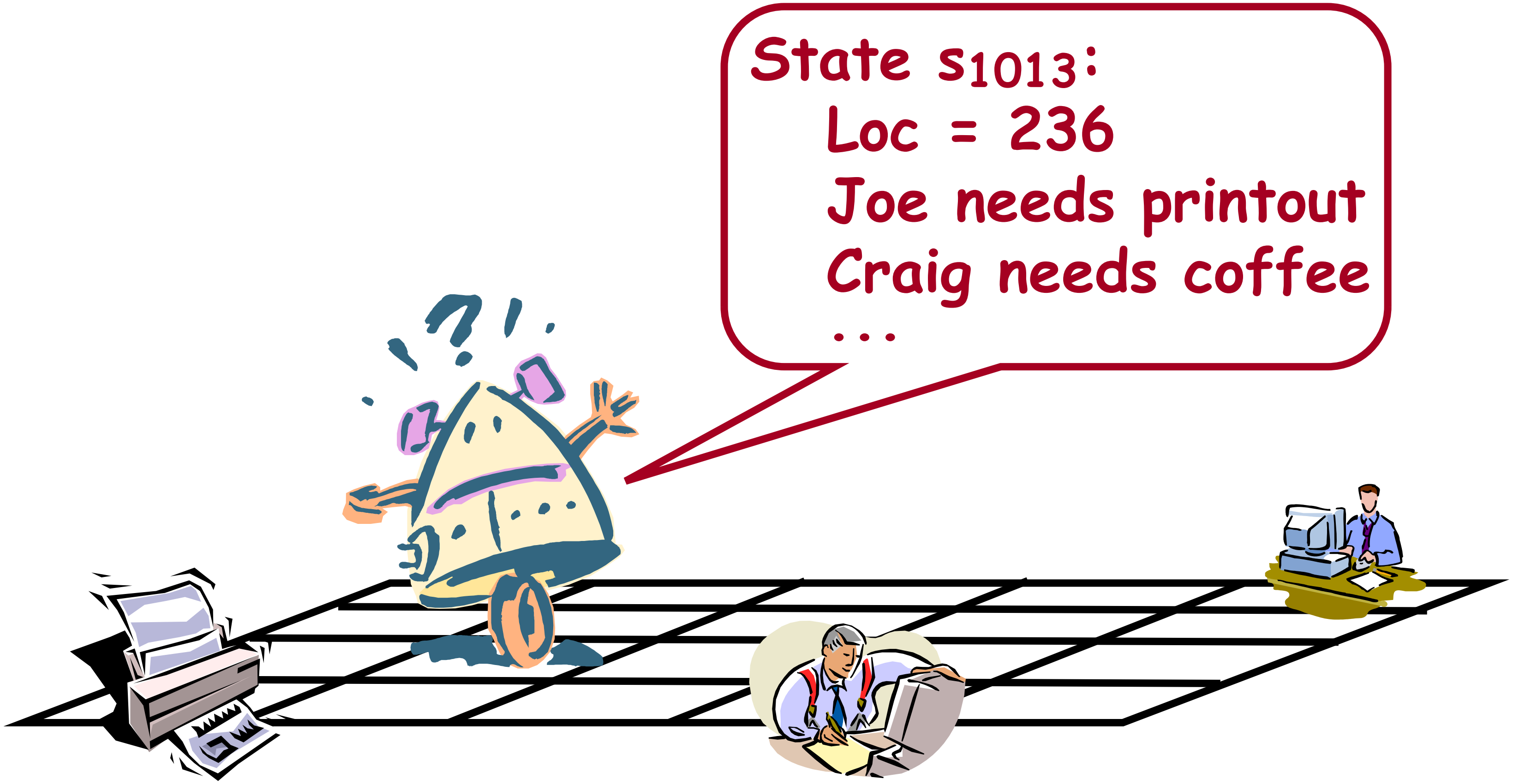
- A lot more worth looking at, but we'll move into a more general (less structured) formalism: MDPs
- An important aspect of decision trees is the fact that *information-gathering actions* are important (and easily modeled)
  - hence they are important decision-analytic tools for understanding *value of information* (e.g., pay for tests, studies, trials, consultants to determine more precise likelihood of the outcomes of certain actions)
  - require direct use of Bayes rule in evaluating trees
  - will discuss this briefly when we get to POMDPs

Markov Decision Processes

- An MDP has four components, *S, A, R, Pr*:
  - (finite) state set *S*   ($|S| = n$)
  - (finite) action set *A*   ($|A| = m$)
  - transition function *Pr(s,a,t)*
    - each *Pr(s,a,●)* is a distribution over *S*
    - represented by set of *n x n* stochastic matrices
  - bounded, real-valued reward function *R(s)*
    - represented by an *n*-vector
    - can be generalized to include action costs: *R(s,a)*
    - can be stochastic (but replaceable by expectation)
- Model easily generalizable to countable or continuous state and action spaces

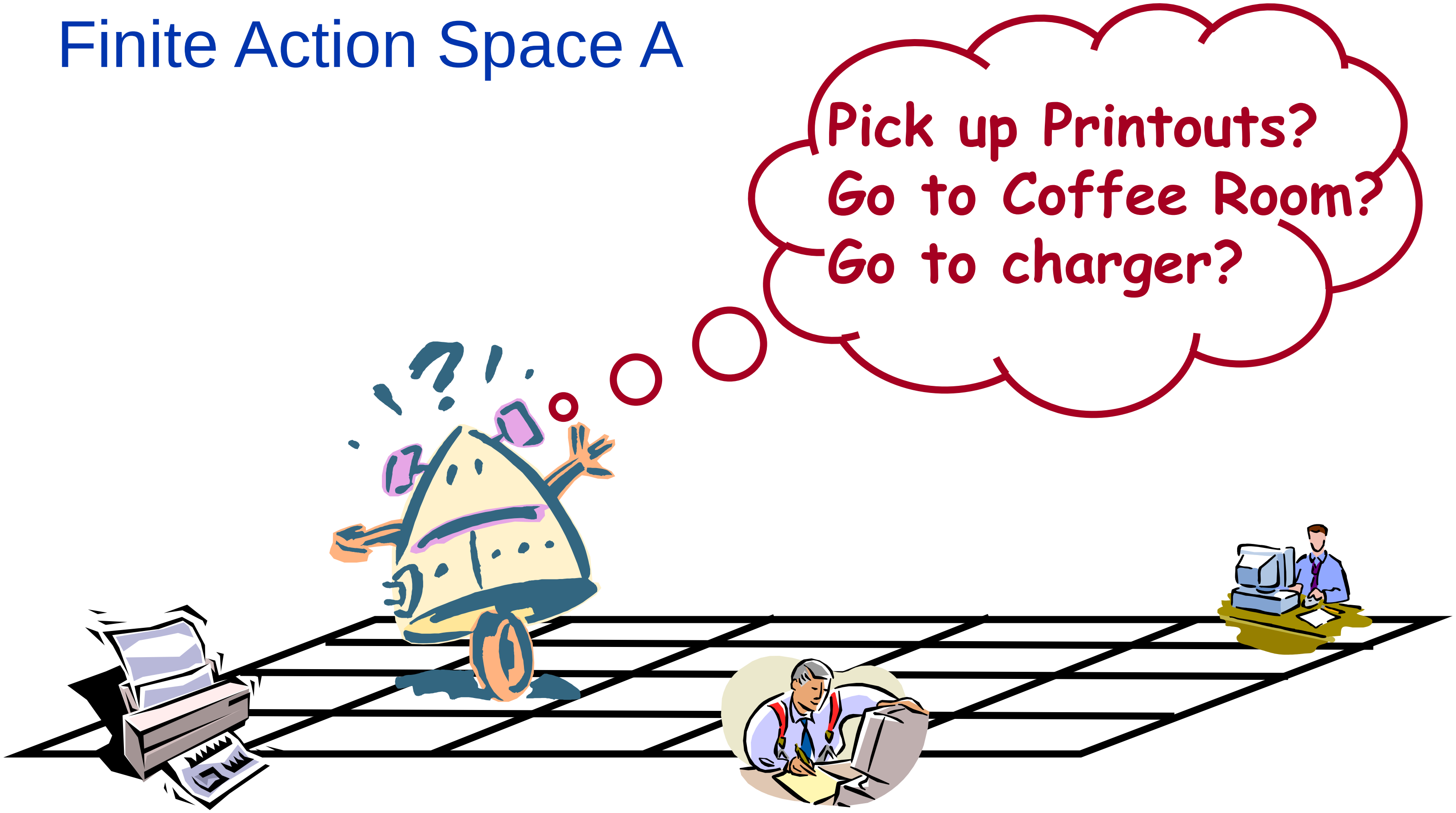
System Dynamics

Finite State Space S



System Dynamics

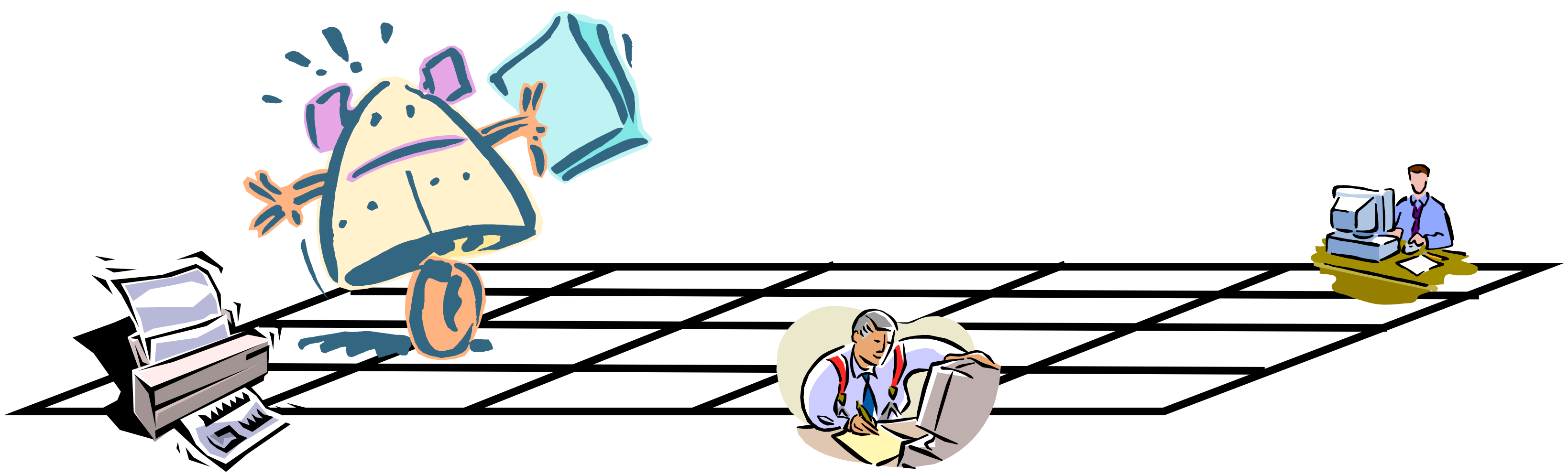
Finite Action Space A



System Dynamics

Transition Probabilities: $Pr(s_i, a, s_j)$

Prob. = 0.95

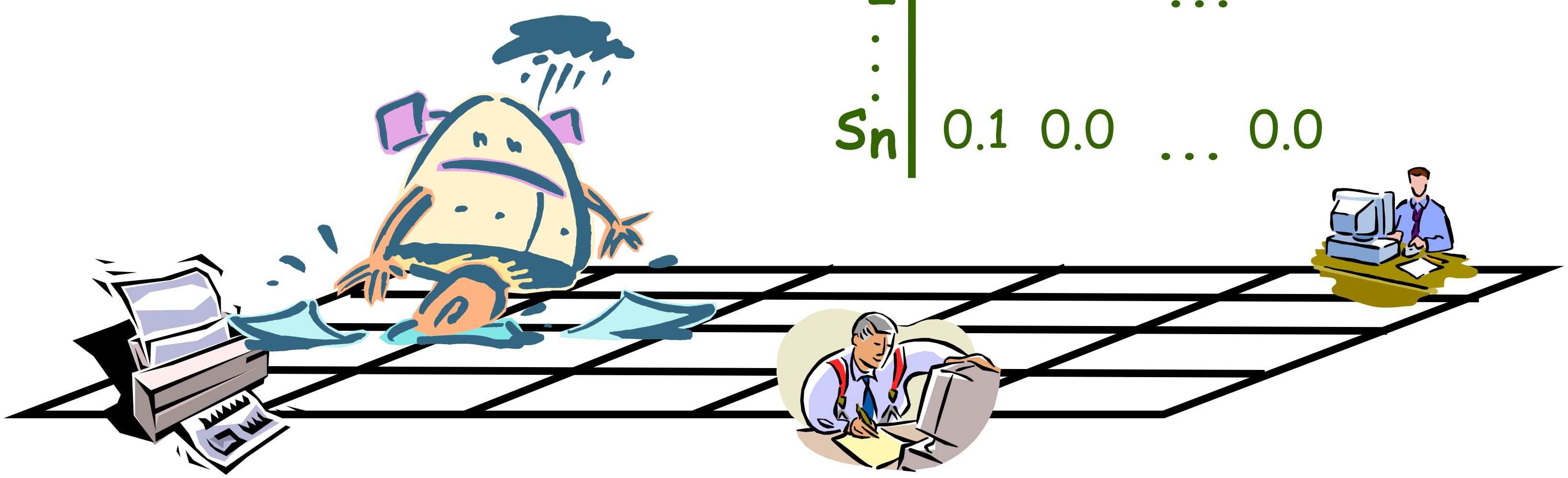


System Dynamics

Transition Probabilities: $Pr(s_i, a, s_j)$

Prob. = 0.05

	s_1	s_2	...	s_n
s_1	0.9	0.05	...	0.0
s_2	0.0	0.20	...	0.1
$\vdots$				
s_n	0.1	0.0	...	0.0



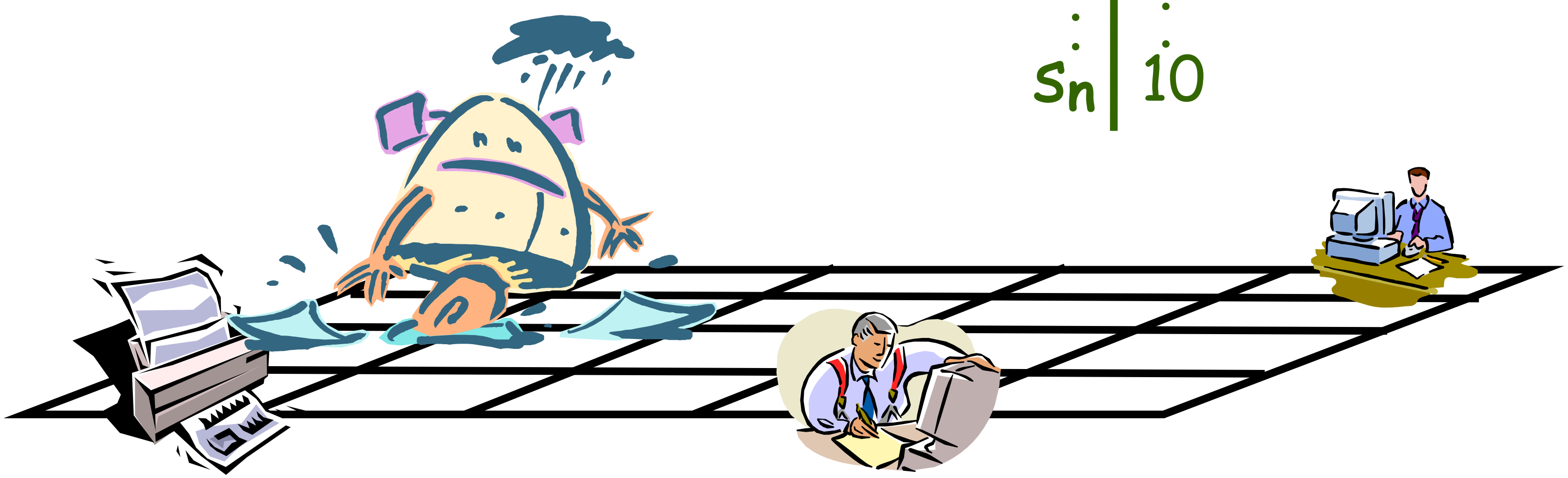
Reward Process

Reward Function: $R(s_i)$

- action costs possible

	R
s_1	12
s_2	0.5
⋮	⋮
s_n	10

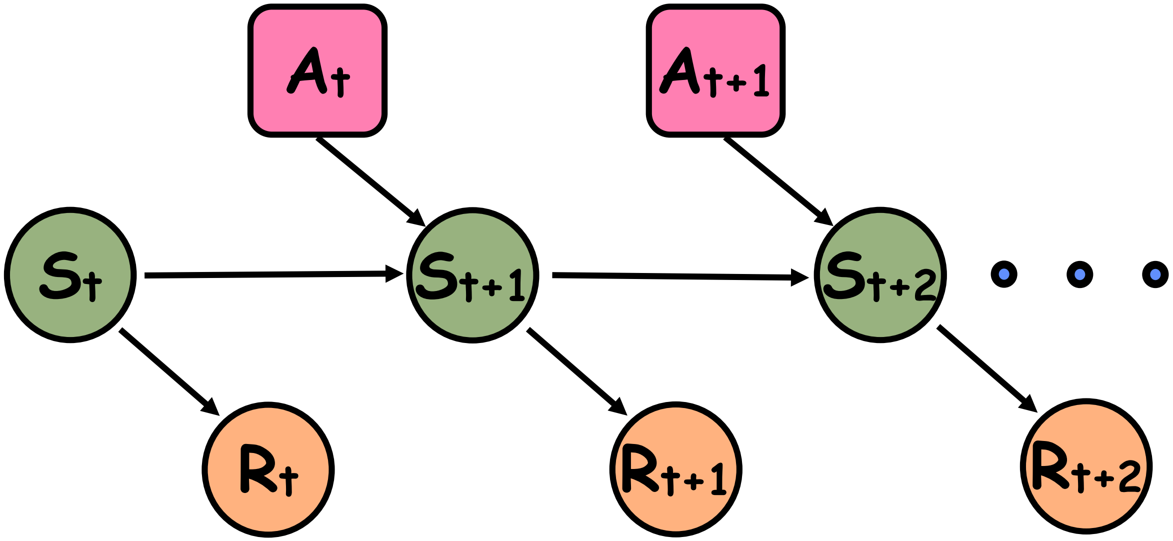
Reward = -10



Assumptions

- *Markovian dynamics* (history independence)
  - $Pr(S^{t+1} \mid A^t, S^t, A^{t-1}, S^{t-1}, \ldots, S^0) = Pr(S^{t+1} \mid A^t, S^t)$
- *Markovian reward process*
  - $Pr(R^t \mid A^t, S^t, A^{t-1}, S^{t-1}, \ldots, S^0) = Pr(R^t \mid A^t, S^t)$
- *Stationary dynamics and reward*
  - $Pr(S^{t+1} \mid A^t, S^t) = Pr(S^{t'+1} \mid A^{t'}, S^{t'})$ for all t, t'
- *Full observability*
  - though we can't predict what state we will reach when we execute an action, once it is realized, we know what it is

Graphical View of MDP



Markov Decision Processes

- Recall components of a fully observable MDP
  - states *S* (|*S*| = *n*)
  - actions *A*
  - transition function *Pr(s,a,t)*
    - represented by set of *n x n* stochastic matrices
  - reward function *R(s)*
    - represented by *n*-vector

	s_1	s_2	…	s_n
s_1	0.9	0.05	…	0.0
s_2	0.0	0.20	…	0.1
⋮				
s_n	0.1	0.0	…	0.0

	R
s_1	12
s_2	0.5
⋮	⋮
s_n	10

Policies

- *Nonstationary policy*
  - $\pi: S \times T \rightarrow A$
  - $\pi(s,t)$ is action to do at state *s* with *t*-stages-to-go
- *Stationary policy*
  - $\pi: S \rightarrow A$
  - $\pi(s)$ is action to do at state *s* (regardless of time)
  - analogous to reactive or universal plan
- These assume or have these properties:
  - full observability
  - history-independent
  - deterministic action choice

Value of a Policy

- How good is a policy π? How do we measure "accumulated" reward?
- *Value function* $V: S \rightarrow \mathbb{R}$
  - associates value with each state (sometimes *S x T*)
- $V_\pi(s)$ denotes *value* of policy at state *s*
  - expected accumulated reward over horizon of interest
  - note $V_\pi(s) \neq R(s)$; it measures utility
- Common formulations of value:
  - *Finite horizon n:* total expected reward given π
  - *Infinite horizon discounted:* discounting keeps total bounded
  - *Infinite horizon, average reward per time step*

Finite Horizon Problems

- Utility (value) depends on stage-to-go
  - hence so should policy: nonstationary $\pi(s, k)$

Tiger trap with juicy piece of meat:

- How to act if world about to end?
- How to act otherwise?

Finite Horizon Problems

- Utility (value) depends on stage-to-go
  - hence so should policy: nonstationary $\pi(s,k)$
- $V_\pi^k(s)$ is *k-stage-to-go value function* for π

$$V_\pi^k(s) = E\,[\sum_{t=0}^{k} R^t \mid \pi, s\,]$$

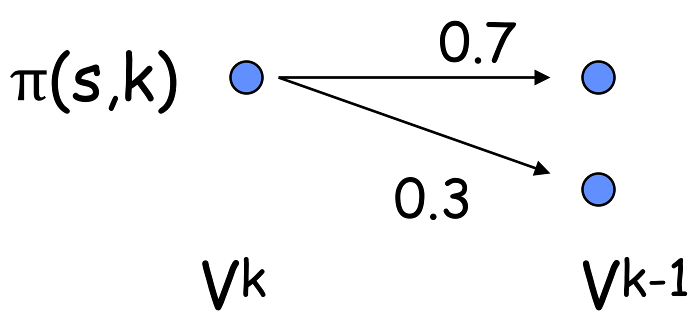
- Here R^t is a random variable denoting reward received at stage t

Successive Approximation

- Successive approximation algorithm used to compute $V_\pi^k(s)$ (akin to dynamic programming)

(a) $V_\pi^0(s) = R(s), \quad \forall s$

(b) $V_\pi^k(s) = R(s) + \sum_{s'} \Pr(s, \pi(s,k), s') \cdot V_\pi^{k-1}(s')$



Successive Approximation

- Let $P^{\pi,k}$ be matrix constructed from rows of action chosen by policy

- In matrix form: $V_\pi^k = R + P^{\pi,k} V_\pi^{k-1}$

- Notes:
  - π requires *T* *n*-vectors for policy representation
  - V_π^k requires an *n*-vector for representation
  - *Markov property* is critical in this formulation since value at *s* is defined independent of how *s* was reached

Value Iteration

- Markov property allows exploitation of dynamic programming (DP) principle for optimal policy construction
  - no need to enumerate $|A|^{Tn}$ possible policies
- Value Iteration

Bellman backup

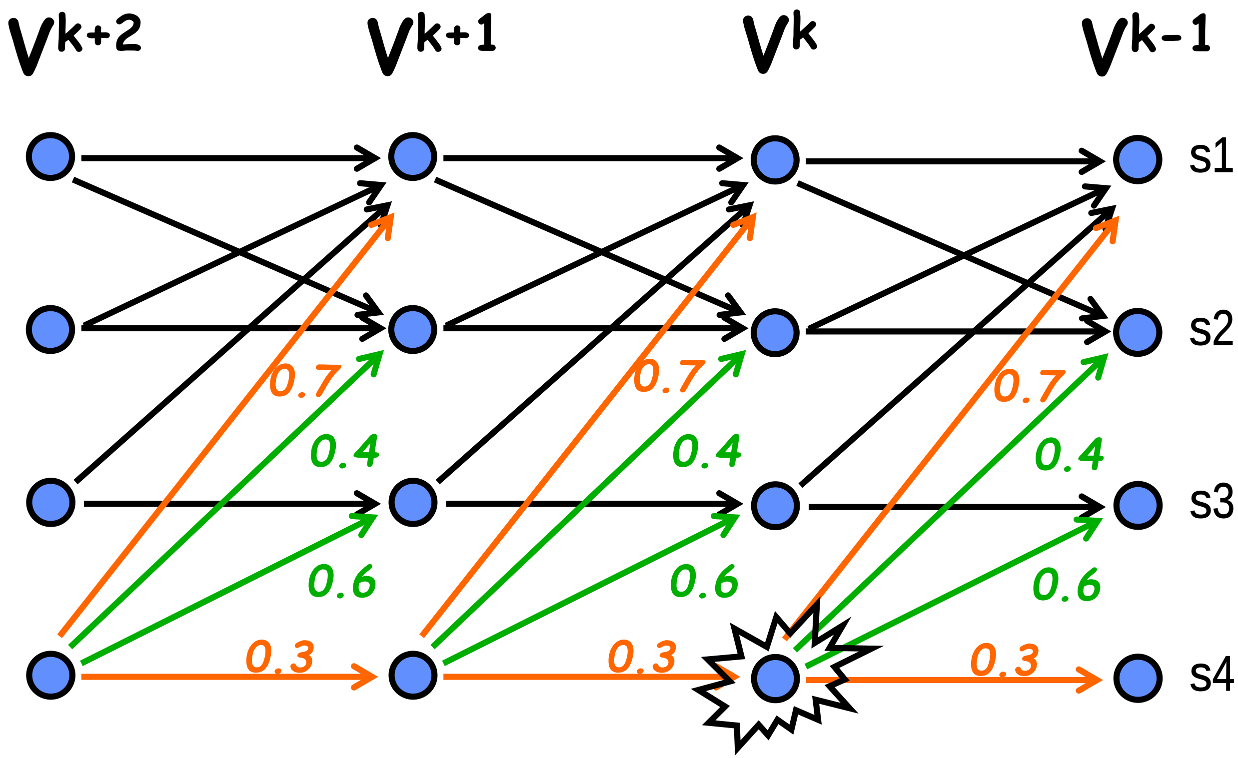
$$V^0(s) = R(s), \quad \forall s$$

$$\boxed{V^k(s) = R(s) + \max_a \sum_{s'} \Pr(s,a,s') \cdot V^{k-1}(s')}$$

$$\pi^*(s,k) = \arg\max_a \sum_{s'} \Pr(s,a,s') \cdot V^{k-1}(s')$$

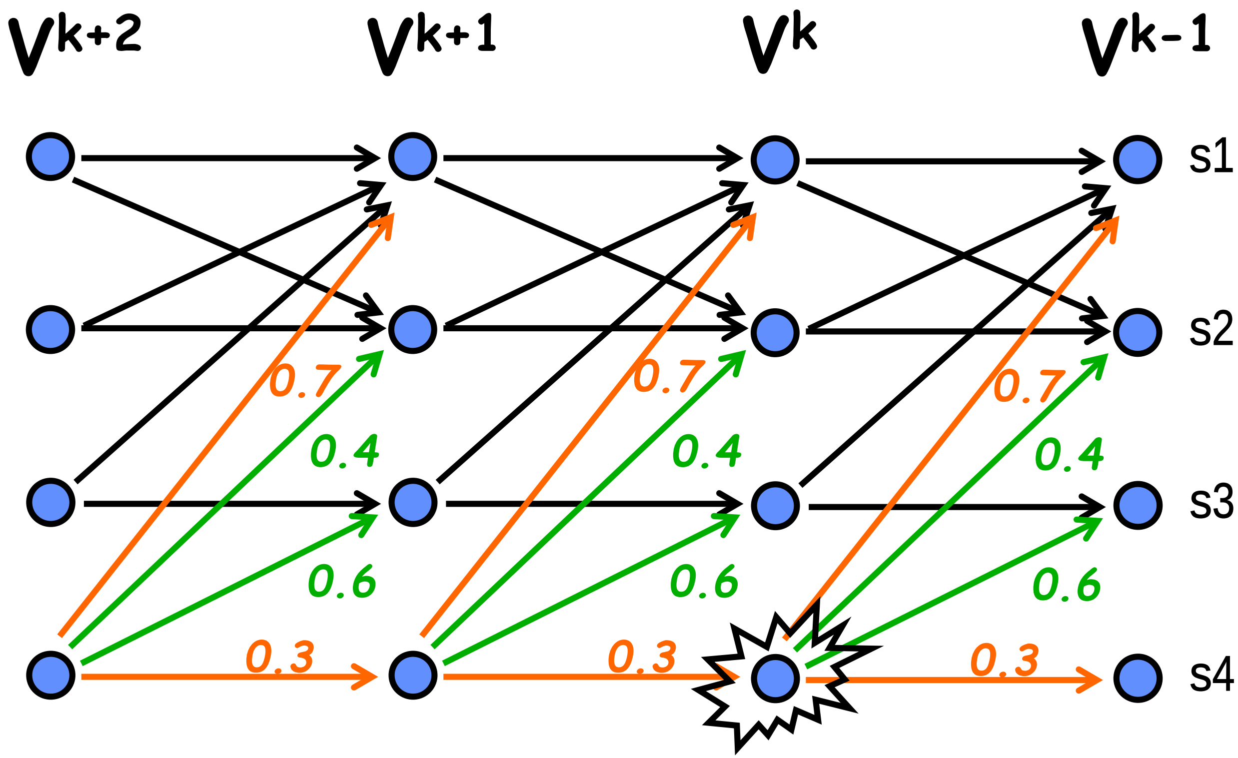
V^k is optimal k-stage-to-go value function

Value Iteration



$$V^k(s4) = R(s4) + \max\{0.7\ V^{k-1}(s1) + 0.3\ V^{k-1}(s4)$$
$$0.4\ V^{k-1}(s2) + 0.6\ V^{k-1}(s3)\}$$

Value Iteration



$\Pi^k(s4) = \max \{ \square \ \square \}$

Value Iteration

- Note how DP is used
  - optimal solution to *k-1* stage problem can be used without modification as part of optimal solution to *k*-stage problem
- Because of finite horizon, policy is nonstationary
- In practice, Bellman backup computed using:

$$Q^k(a,s) = R(s) + \sum_{s'} \Pr(s,a,s') \cdot V^{k-1}(s'), \quad \forall a$$

$$V^k(s) = \max_a Q^k(a,s)$$

Complexity of Value Iteration

- T iterations
- At each iteration $|A|$ computations of n x n matrix times n-vector: $O(|A|n^2)$
- Total $O(T\,|A|n^2)$
- Can exploit sparsity of matrix: $O(T\,|A|n)$

Summary

- Resulting policy is *optimal*

$$V_{\pi^*}^k(s) \geq V_\pi^k(s), \quad \forall \pi, s, k$$

  - convince yourself of this
  - convince yourself that non-Markovian, randomized policies are not necessary
- Notes:
  - optimal value function is unique…
  - but optimal policy need not be unique

Discounted Infinite Horizon MDPs

- Total reward problematic (usually)
  - many or all policies have infinite expected reward
  - some MDPs (e.g., zero-cost absorbing states) OK
- "Trick": introduce discount factor $0 \leq \beta < 1$
  - future rewards discounted by β per time step

$$V_\pi^k(s) = E\left[\sum_{t=0}^{\infty} \beta^t R^t \mid \pi, s\right]$$

- Note: $$V_\pi(s) \leq E\left[\sum_{t=0}^{\infty} \beta^t R^{\max}\right] = \frac{1}{1-\beta} R^{\max}$$
- Motivation: economic? failure prob? convenience?

Some Notes

- Optimal policy maximizes value at each state
- Optimal policies guaranteed to exist *(Howard 1960)*
- Can restrict attention to stationary policies
  - why change action at state *s* at new time *t*?
- We define $V^*(s) = V_\pi(s)$ for some optimal π

Value Equations

- Value equation for fixed policy value

$$V_\pi(s) = R(s) + \beta \sum_{s'} \Pr(s, \pi(s), s') \cdot V_\pi(s')$$

- Bellman equation for optimal value function

$$V^*(s) = R(s) + \beta \max_a \sum_{s'} \Pr(s, a, s') \cdot V^*(s')$$

Backup Operators

- We can think of the fixed policy equation and the Bellman equation as operators in a vector space
  - e.g., $L_\pi(V) = V' = R + \beta P_\pi V$
  - V_π is unique fixed point of policy backup operator L_π
  - *V** is unique fixed point of Bellman backup L
- We can compute V_π easily: *policy evaluation*
  - simple linear system with *n* variables, *n* equalities
  - solve $V = R + \beta P_\pi V$
- Cannot do this for optimal policy
  - max operator makes things nonlinear

Value Iteration

- Can compute optimal policy using value iteration, just like FH problems (just include discount term)

$$V^k(s) = R(s) + \beta \max_a \sum_{s'} \Pr(s,a,s') \cdot V^{k-1}(s')$$

  - no need to store argmax at each stage (stationary)

Convergence

- $L(V)$ is a *contraction mapping* in R^n (so is L_π)
  - $||LV - LV'|| \leq \beta\, ||V - V'||$ (we're using max-norm)
- When to stop value iteration? when $||V^k - V^{k-1}|| \leq \varepsilon$
  - $||V^{k+1} - V^k|| \leq \beta\, ||V^k - V^{k-1}||$
  - this ensures $||V^k - V^*|| \leq \varepsilon\beta/(1-\beta)$
- Convergence is assured
  - any guess *V*: $||V^* - LV|| = ||LV^* - LV|| \leq \beta\, ||V^* - V||$
  - so fixed point theorems ensure eventual convergence

How to Act

- Given *V** (or approximation), use *greedy* policy:

$$\pi^*(s) = \arg\max_a \sum_{s'} \Pr(s,a,s') \cdot V^*(s')$$

  - if *V* within ε of *V**, then $V(\pi)$ within 2ε of *V**
- There exists an ε s.t. optimal policy is returned
  - even if value estimate is off, greedy policy is optimal
  - *proving* a policy is optimal can be difficult (methods like *action elimination* can be used)

Complexity of VI

- *Unknown number of* iterations: assume stopping at time *T*
  - Convergence rate: linear
  - Expected number of iterations grows as $1/(1-\beta)$

- At each iteration,we have $|A|$ matrix-vector multiplications: n x n matrix, n-vector so: $O(|A|n^2)$
- Total $O(T|A|n^2)$

- Can exploit sparsity of matrix: $O(T\,|A|n)$

Policy Iteration

- Given fixed policy, can compute its value exactly:

$$V_\pi(s) = R(s) + \beta \sum_{s'} \Pr(s, \pi(s), s') \cdot V_\pi(s')$$

* This is a linear system with *n* vars ($V_\pi(s)$ for each *s*)

- Policy iteration exploits this

1. Choose a random policy π
2. Loop:
   (a) Evaluate V_π
   (b) For each s in S, set $\pi'(s) = \arg\max_a \sum_{s'} \Pr(s, a, s') \cdot V_\pi(s')$
   (c) Replace π with π'

Until no improving action possible at any state

Policy Iteration Notes

- Convergence assured (Howard 1960)
  - intuitively: no local maxima in value space, and each policy must improve value; since finite number of policies, will converge to optimal policy
- Very flexible algorithm
  - need only improve policy at one state (not each state)
- Gives exact value of optimal policy
- Generally converges much faster than VI
  - each iteration more complex $O(n^3)$, but fewer iterations
  - quadratic rather than linear rate of convergence (sometimes)
  - known to be pseudo-polynomial for fixed β

Modified Policy Iteration

- Modified policy iteration (MPI):flexible alternative to VI, PI
- Run PI, but don't solve linear system to evaluate policy:
  - instead do several iterations of successive approximation (SA) to evaluate policy
- You can run SA until near convergence
  - but in practice, you often only need *a few backups* to get an estimate of $V(\pi)$ that allows improvement in π
  - quite efficient in practice
  - choosing number of SA steps an important practical issue

Asynchronous Value Iteration (AVI)

- Needn't do full backups of VF when running VI
- *Gauss-Siedel:* Start with V^k .Once you compute $V^{k+1}(s)$, you replace $V^k(s)$ before proceeding to the next state (assume some ordering of states)
  - tends to converge much more quickly
  - note: V^k no longer k-stage-to-go VF
- *Asynchronous VI:* set some V^0; Choose random state s and do a Bellman backup at that state alone to produce V^1; Choose random state s…
  - if each state backed up frequently enough, convergence assured
  - useful for online algorithms (reinforcement learning)

Some Remarks on Search Trees

- Analogy of Value Iteration to decision trees
  - decision tree *(expecti-max search)* is really value iteration with computation focused on reachable states
- *Real-time Dynamic Programming (RTDP)*
  - simply real-time search applied to MDPs
  - can exploit heuristic estimates of value function
  - can bound search depth using discount factor
  - can cache/learn values
  - can use pruning techniques