Automated Planning

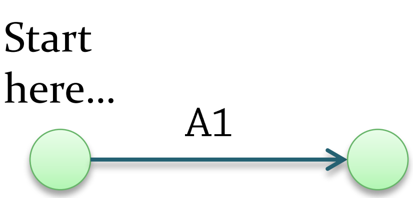
Planning under Uncertainty: An Overview

Jonas Kvarnström
Automated Planning Group
Department of Computer and Information Science
Linköping University

Planning with Complete Information

- So far, we have assumed we know in advance:
  - The state of the world when **plan execution** starts
  - The outcome of any action, given the state where it is executed
    - State + action ➔ unique resulting state
- So if there is a solution:
  - There is an **unconditional** sequential solution

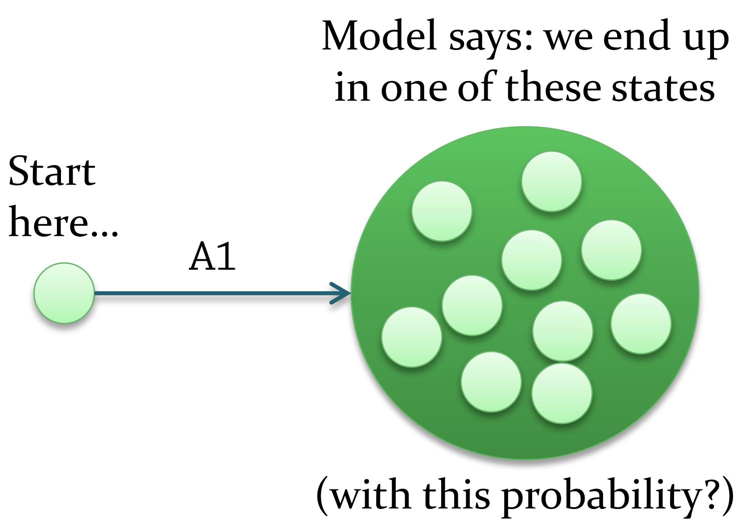
Planning	Execution
Model says: we end up in this specific state!	Just follow the unconditional plan…



Multiple Outcomes

- In reality, actions may have **multiple outcomes**
  - **Nondeterministic** planning:
    - State + action ➔ **set** of possible new states (no more info during planning)
  - **Probabilistic planning**:
    - State + action ➔ **probability distribution** over a set of possible next states
    - Can plan for all outcomes, or ignore the least probable outcomes
    - Can generate plans with **high probability** of reaching the goal

Planning	Execution
	We need something different here…



Intended Outcomes

- Sometime, specific outcomes are **intended** or **nominal**
  - **pick-up(object)**
    Intended outcome: carrying(object) is true
    Unintended outcome: carrying(object) is false
  - **move(100,100)**
    Intended outcome: xpos(robot)=100
    Unintended outcome: xpos(robot) != 100

"Intentions" are just our interpretation!

To a planner, there is generally no difference…

- Sometimes there are no intended outcomes
  - Tossing a coin: 2 different outcomes

Plan Types

- With multiple outcomes, we can generate:
  - **Strong** solutions (guaranteed to reach the goal)
  - **Weak** solutions (may reach the goal)
  - **Probabilistic** solutions (reaching the goal with probability >= limit)

Overview A

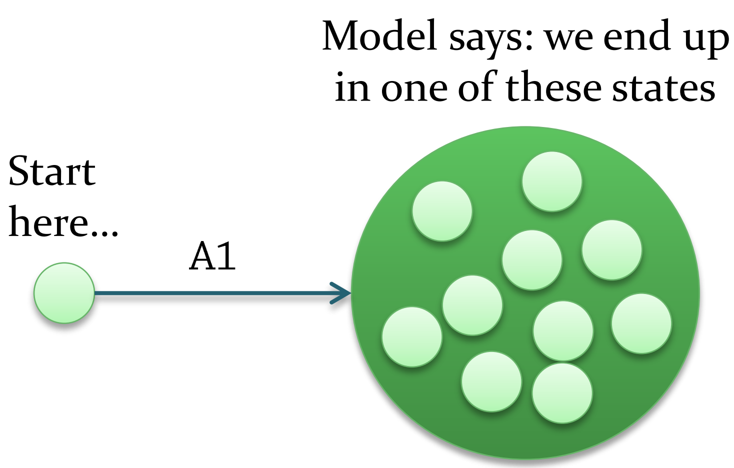
Deterministic: Exact outcome known in advance	Classical planning (possibly with extensions)
Non-deterministic: Multiple outcomes, no probabilities	?
Probabilistic: Multiple outcomes with probabilities	?

- But what about information gained **during execution**?

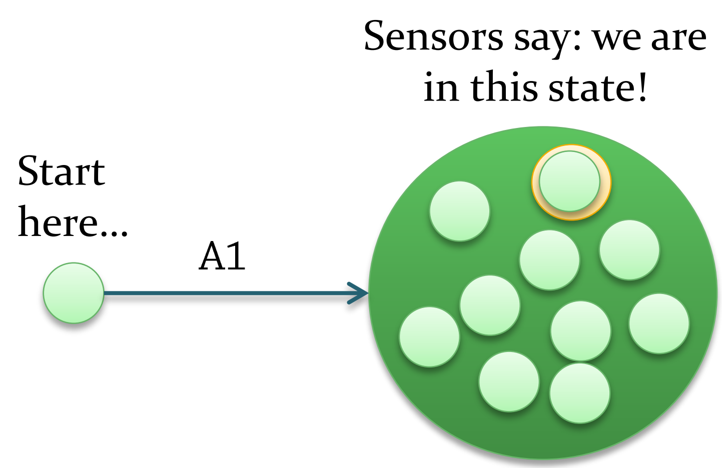
Fully Observable

	Non-deterministic or probabilistic model
Fully observable: Our sensors can determine exactly which state we are in after executing an action	A plan could: ➔ Define which action to perform depending on which **exact state** you actually ended up in

Planning



Execution

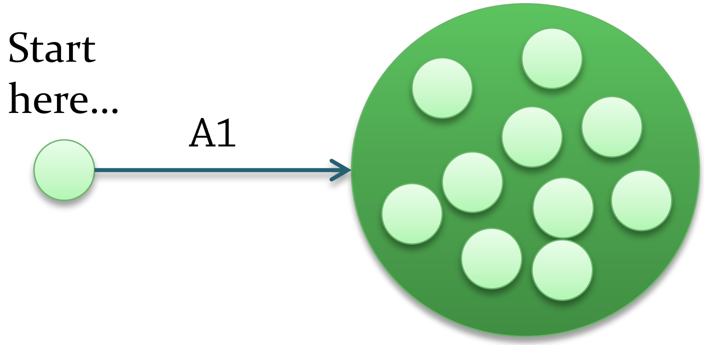


Non-Observable

	<u>**Non-deterministic or probabilistic model**</u>
<u>**Non-observable:**</u> We have *no* sensors to determine what happened *Only* predictions can guide us	A plan could: ➔ Define which action to perform depending on which <u>**set**</u> of states you <u>**might**</u> be in

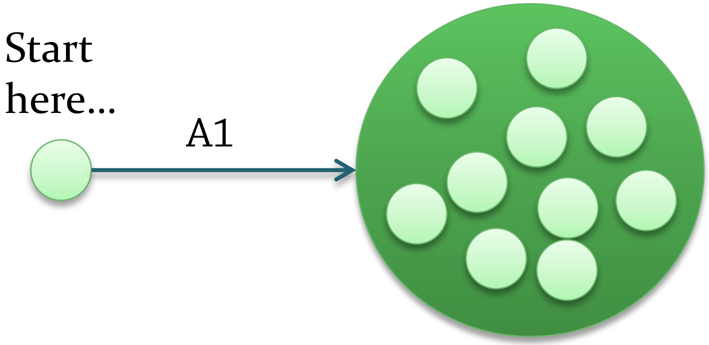
Planning

Model says: we end up
in one of these states



Execution

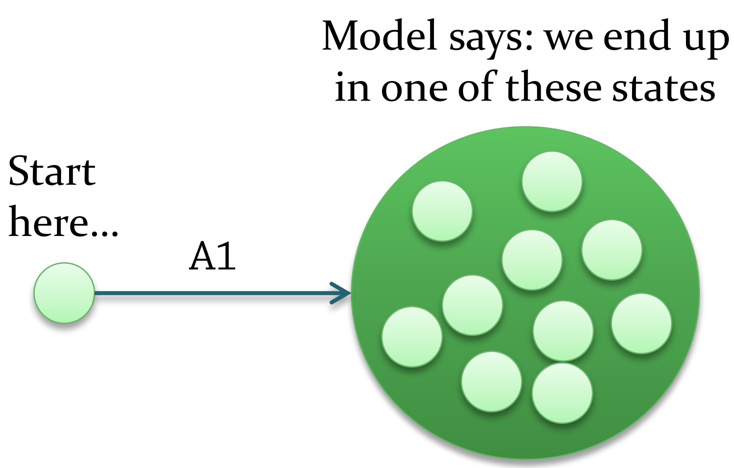
No sensors!
No new information



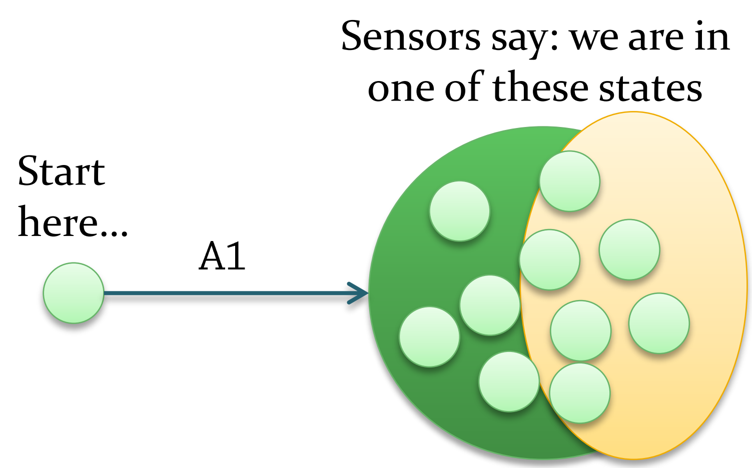
Partially Observable

	Non-deterministic or probabilistic model
Partially observable: Sensors can observe *some* properties of the world ➔ we are in a *set* of states	A plan could: ➔ Define which action to perform depending on which **set** of states you **might** be in ➔ Take into account new information after sensing

Planning



Execution



Overview B

	Non-Observable: No information gained after action	**Fully Observable**: Exact outcome known after action	**Partially Observable**: Some information gained after action
Deterministic: Exact outcome known in advance	Classical planning (possibly with extensions) (Information dimension is meaningless)		

- In general:
  - Full information is the easiest
  - Partial information is the hardest!

Overview B

	Non-Observable: No information gained after action	**Fully Observable**: Exact outcome known after action	**Partially Observable**: Some information gained after action
Deterministic: Exact outcome known in advance	Classical planning (possibly with extensions) (Information dimension is meaningless)		
Non-deterministic: Multiple outcomes, no probabilities	Non-deterministic Conformant Planning	Conditional (Contingent) Planning	(No specific name)
Probabilistic: Multiple outcomes with probabilities	Probabilistic Conformant Planning (Special case of POMDPs)	Probabilistic Conditional Planning Markov Decision Processes (MDPs)	Partially Observable MDPs (POMDPs)

- In general:
  - Full information is the easiest
  - Partial information is the hardest!

Automated Planning

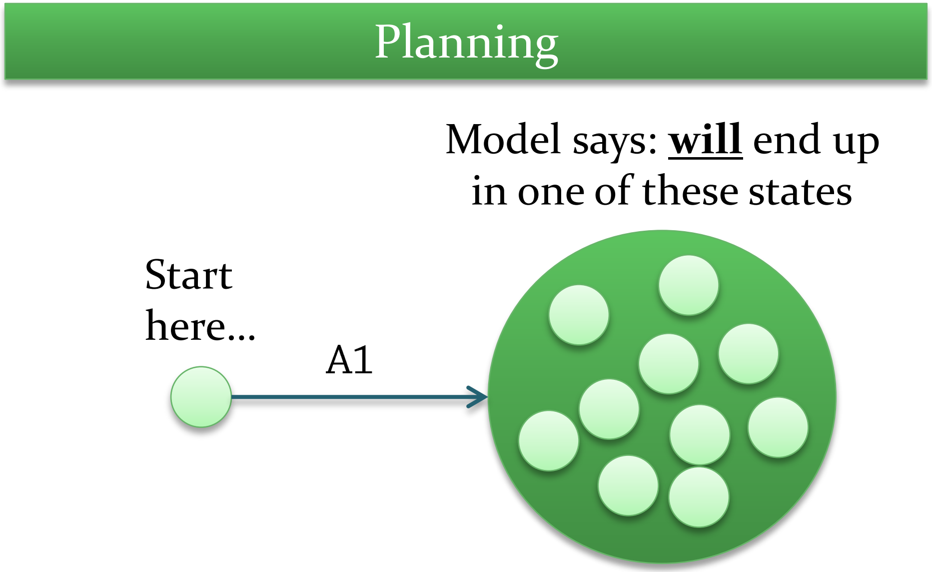
Planning Based on
Fully Observable Markov Decision Processes

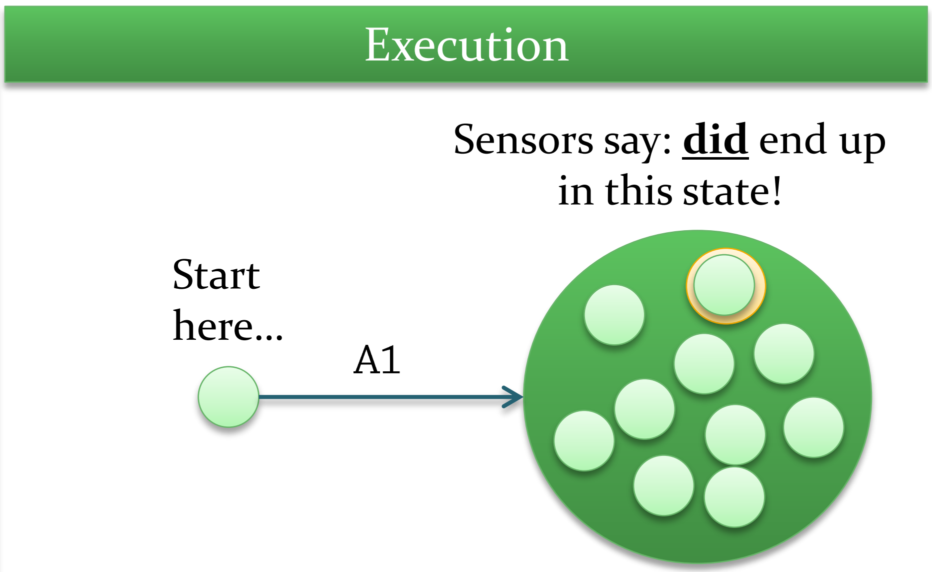
Jonas Kvarnström
Automated Planning Group
Department of Computer and Information Science
Linköping University

Some slides adapted from a presentation by Dana Nau

Fully Observable MDPs

- Fully Observable **Markov Decision Processes**:
  - Action outcomes are:
    - **Probabilistic**
    - **Fully observable**



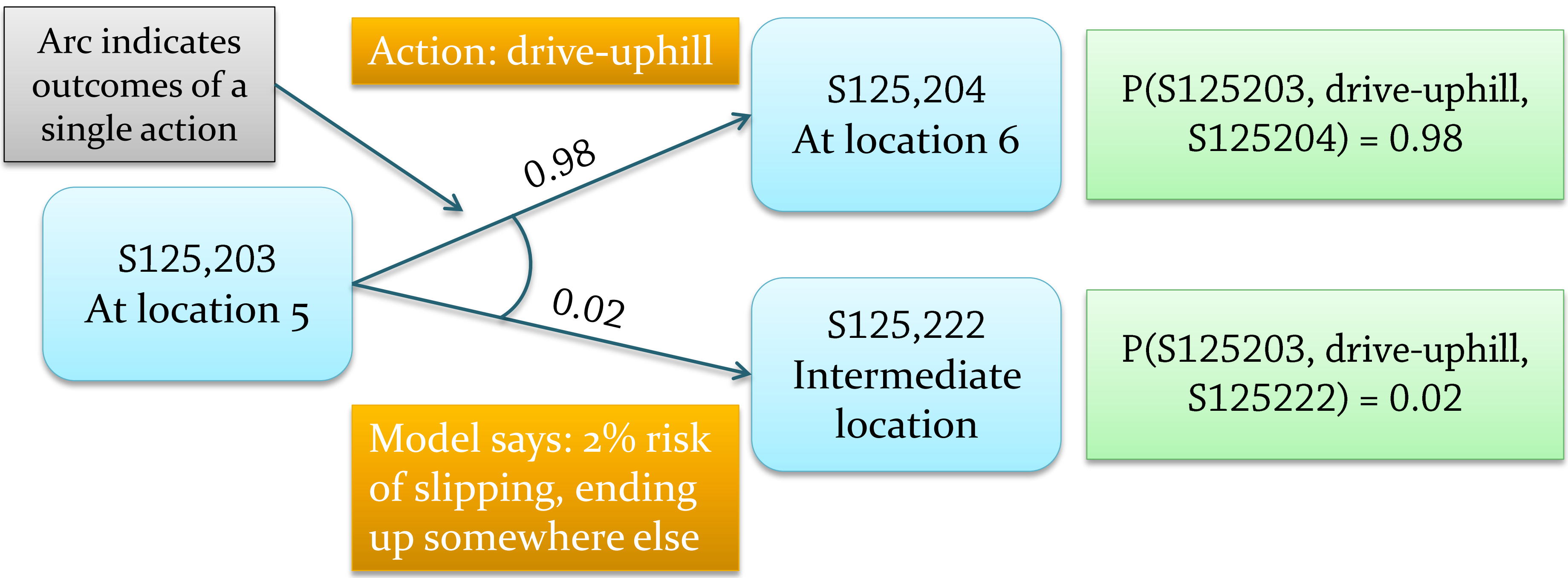


Stochastic Systems

- Formal models:
  - **Restricted state transition system** $\Sigma = (S,A,\gamma)$
    - $S = \{ s_0, s_1, \ldots \}$: Finite set of **world states**
    - $A = \{ a_0, a_1, \ldots \}$: Finite set of **actions**
    - $\gamma: S \times A \rightarrow 2^S$: **State transition function**, where $|\gamma(s,a)| \leq 1$
  - **Stochastic system** $\Sigma = (S, A, P)$
    - $P(s, a, s')$: Given that we are in s and execute a, the **probability** of ending up in s'
    - For any state s and action a, we have $\sum_{s' \in S} P(s, a, s') = 1$: Exactly 100% probability of ending up *somewhere*

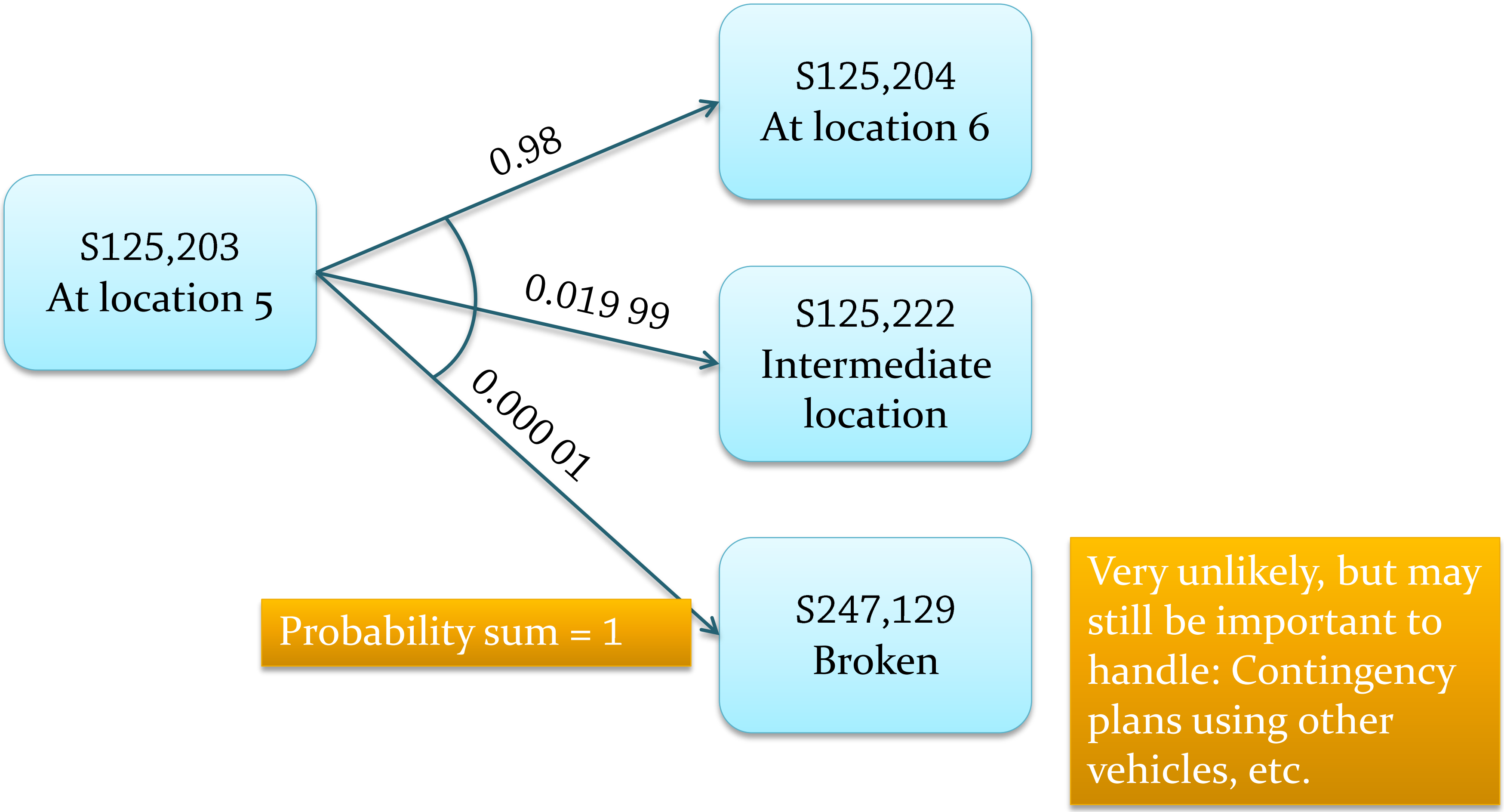
Sometimes written $P_a(s' \mid s)$

Stochastic Systems (2)



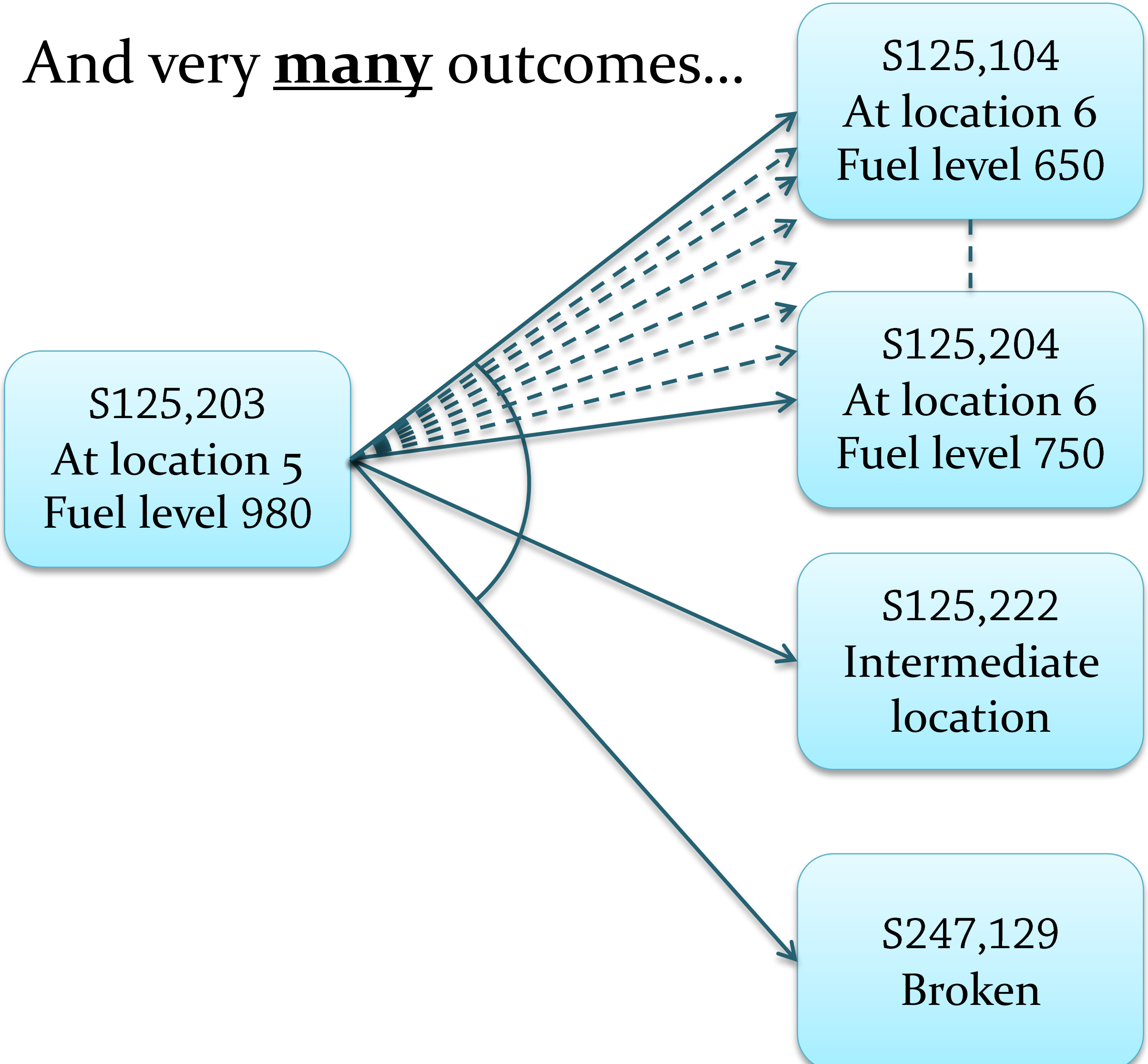
Stochastic Systems (3)

- May have very **unlikely** outcomes…



Stochastic Systems (4)

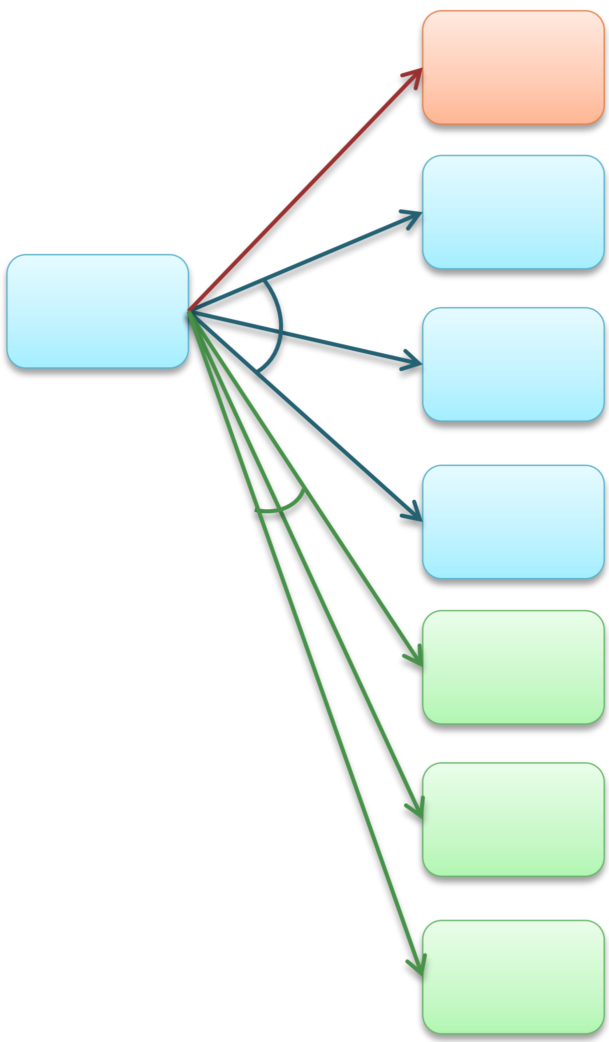
- And very **many** outcomes…



As always, one state for every **combination** of properties

Stochastic Systems (5)

- Like before, often **<u>many executable actions</u>** in every state



Probability sum = 1
(single certain outcome)

Probability sum = 1
(three possible outcomes)

Probability sum = 1
(three possible outcomes)

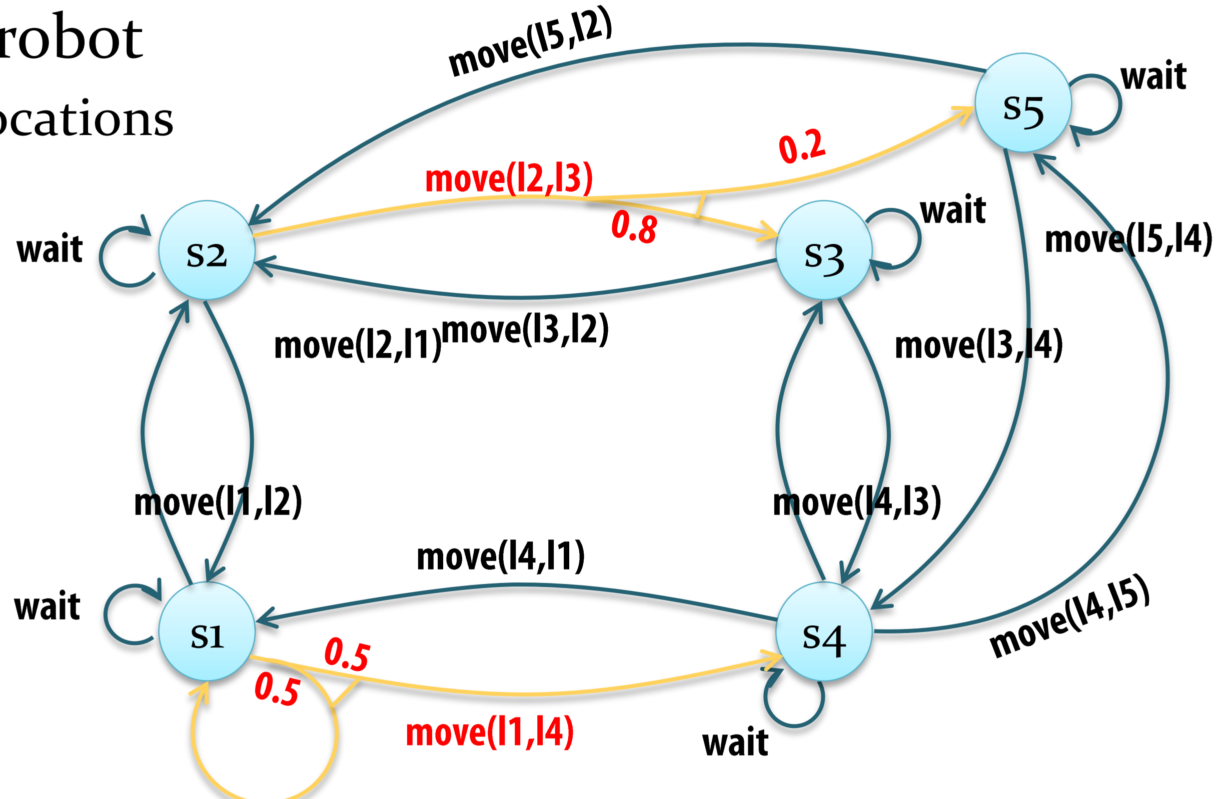
We choose
the action…

Nature chooses
the outcome!

Search yields an
AND/OR graph

Stochastic System Example

- **Example**: A single robot
  - Moving between 5 locations
  - For simplicity, states correspond directly to locations
    - s1: at(r1, l1)
    - s2: at(r1, l2)
    - s3: at(r1, l3)
    - s4: at(r1, l4)
    - s5: at(r1, l5)



- Some transitions are **deterministic**, some are **stochastic**
  - Trying to move from l2 to l3: You may end up at l5 instead (20% risk)
  - Trying to move from l1 to l4: You may stay where you are instead (50% risk)
- (Can't always move in both directions, e.g. due to terrain gradient)

The Markov Property

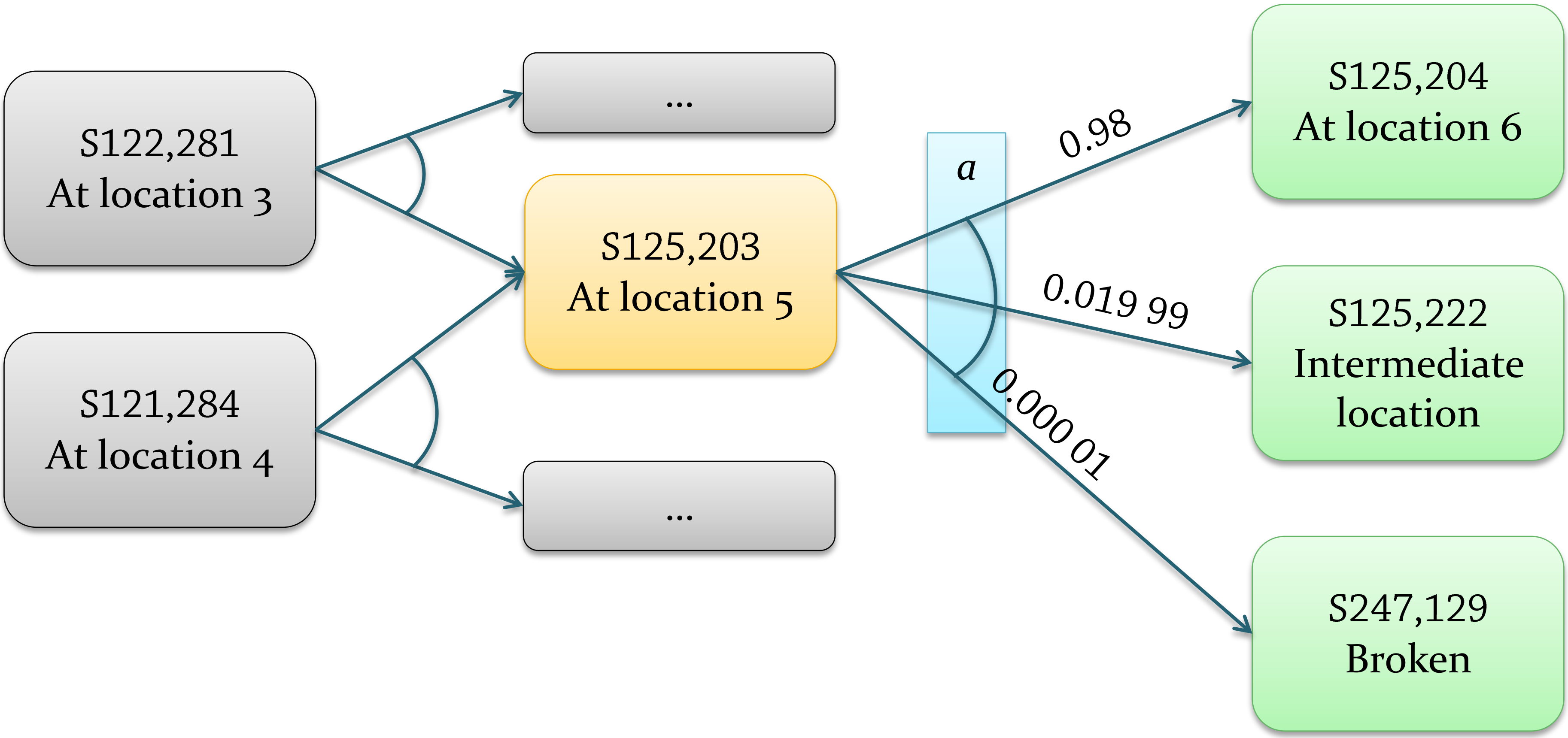
Markov Property (1)

- Recall the definition of the probability function:
  - $P(s, a, s')$ is the **probability** of ending up in *s'* given that we **are in *s*** and **execute *a***

Nothing else matters!

Markov Property (2)

- This type of system has the **Markov property**: is **memoryless**



We don't need to know the states we visited before…

Only the current state

and the action…

…To find out where we may end up, with which prob.

Remembering the Past

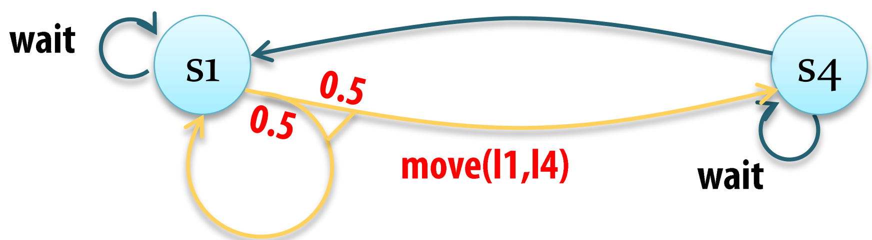
- We can still remember some things about the past!
  - Example: predicate **visited(location)**
    - Keeps track of where we have been

  - But then this information is **encoded and stored** in the **current state**
    - Which is finite, has a constant size
    - No need to query an ever-growing sequence of past states

Plans and Policies

Policies

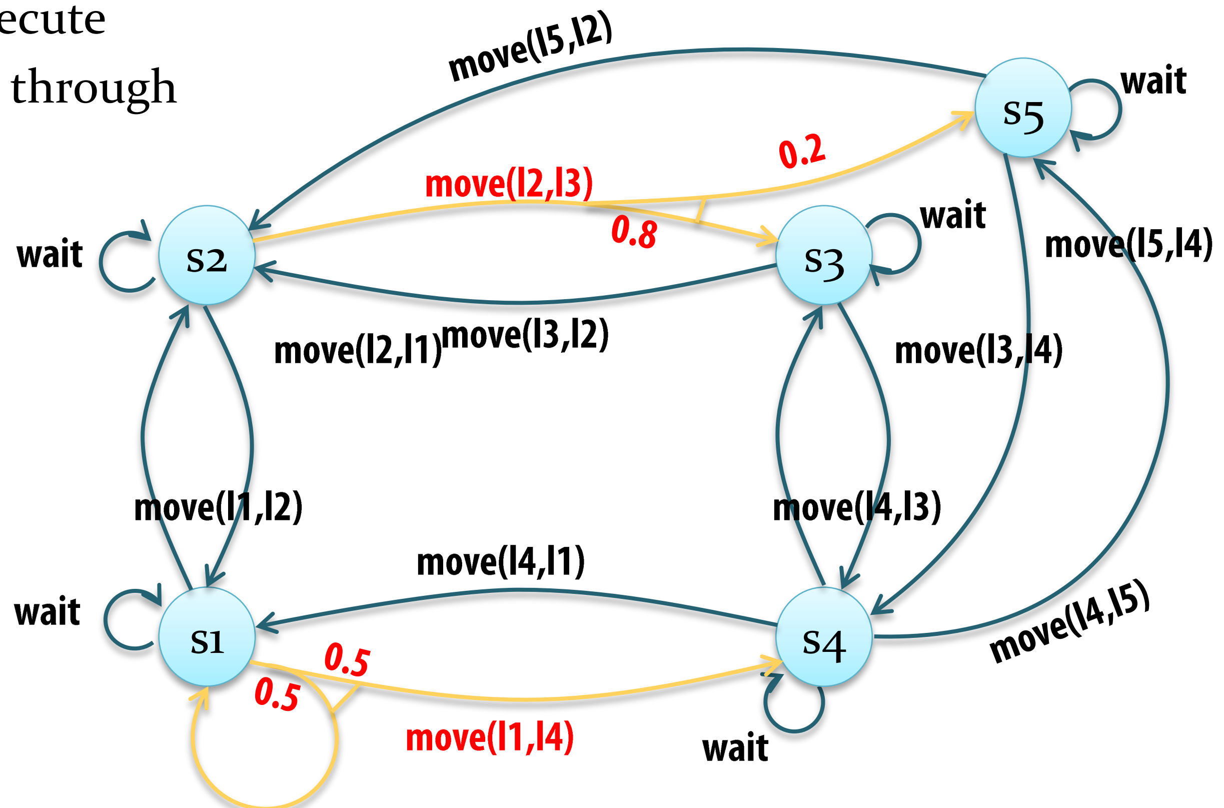
- Two important consequences for plan structures:
  - Action choice must depend on the current state
    - And thereby on earlier execution-time outcomes!
  - Cannot have a limit on the number of actions executed!



- In MDP planning, we generate **<u>policies</u>**
  - Usually denoted by π
  - Defines, **<u>for each state</u>**,
    which action to execute **<u>whenever</u>** we end up in that state
    - $\pi 1$ = { (s1, move(l1,l2)), (s2, move(l2,l3)), (s3, move(l3,l4)), (s4, wait), (s5, wait) }

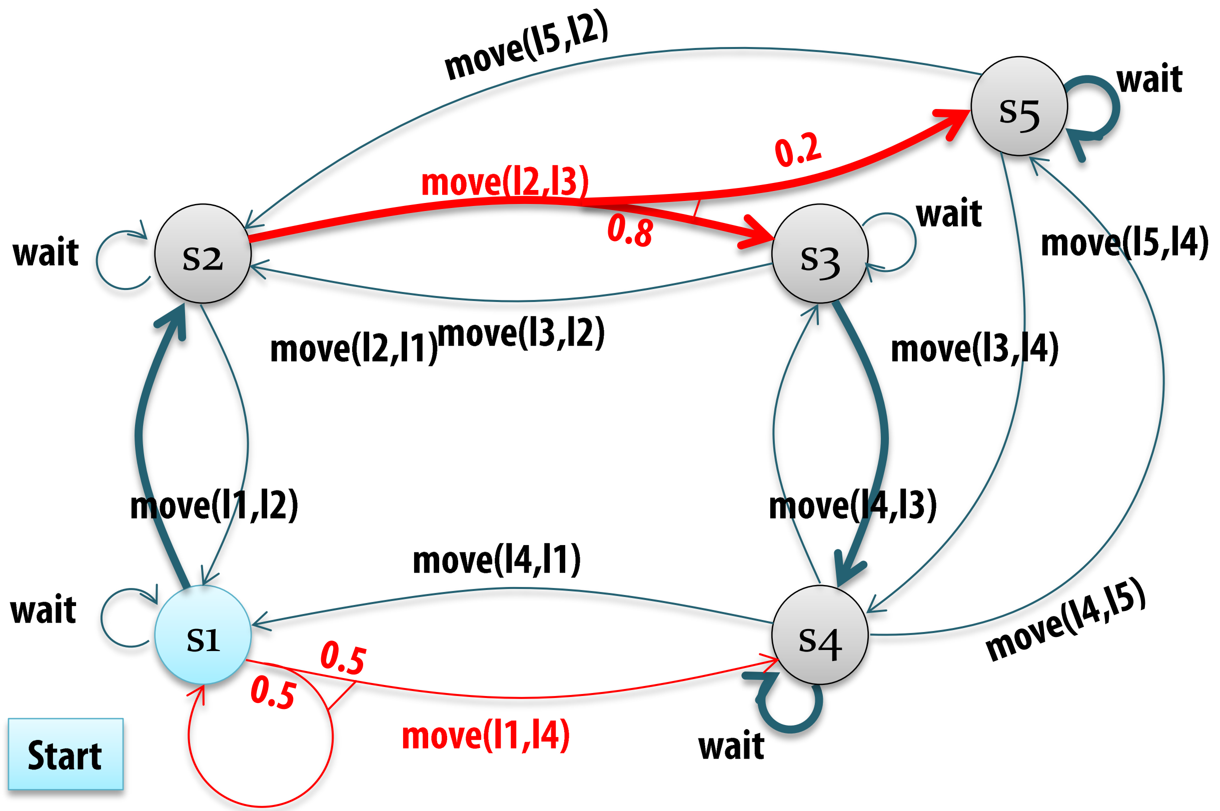
Termination?

- Since a policy defines an action for every state:
  - We **<u>could</u>** define a set of goal states where **<u>execution can end</u>**
    - Similar to classical planning
  - **<u>Usually</u>** one assumes a policy **<u>never terminates</u>**!
    - The policy **<u>always</u>** specifies another action to execute
    - Objectives specified through costs and rewards (later!)



Policy Example 1

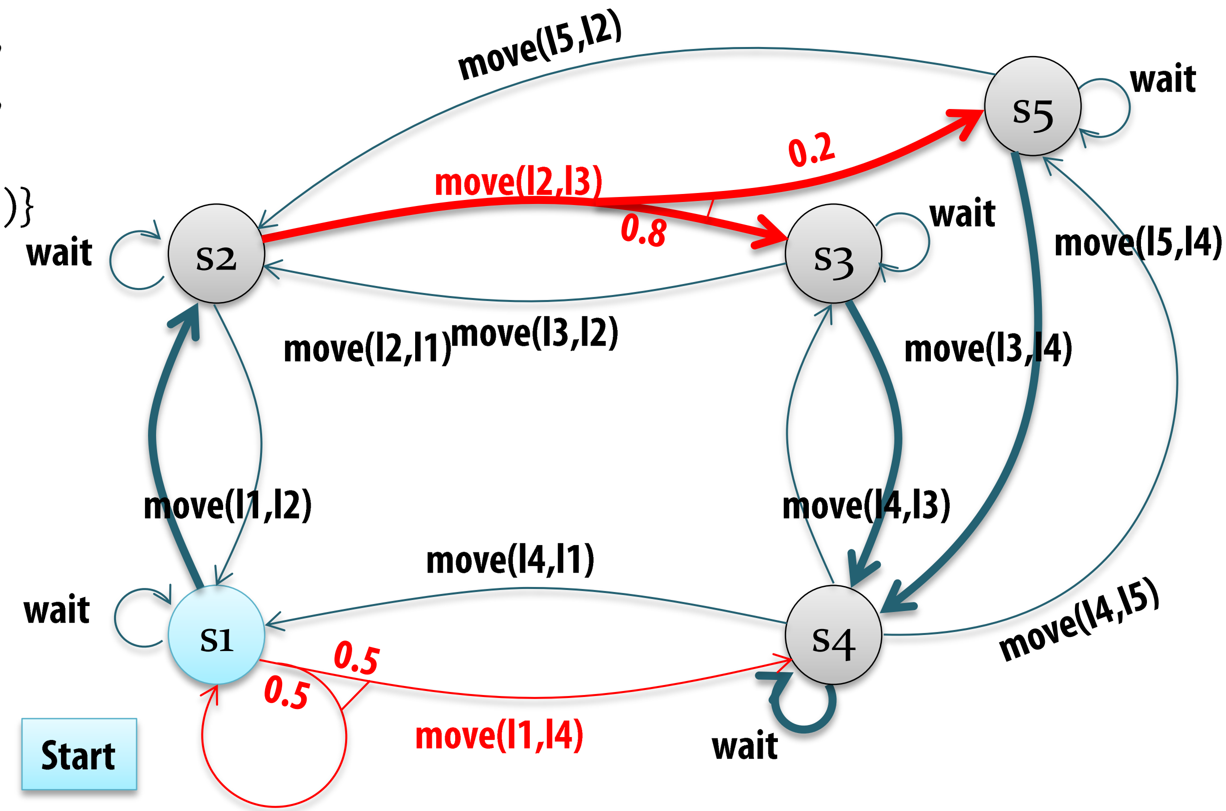
- Example 1
  - π1 = { (s1, move(l1,l2)),
    (s2, move(l2,l3)),
    (s3, move(l3,l4)),
    (s4, wait),
    (s5, wait)}



May end up in s4 or s5, wait there infinitely many times

Policy Example 2

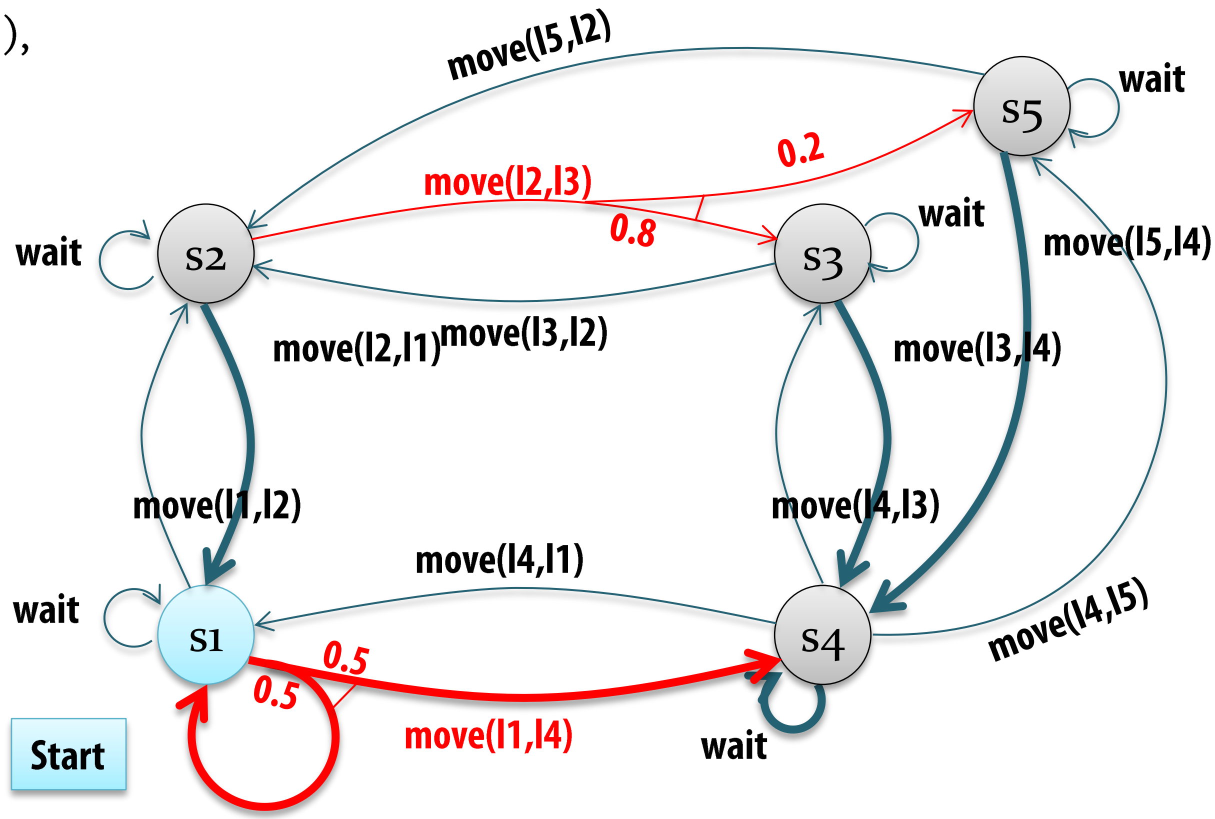
- Example 2
  - π2 = { (s1, move(l1,l2)),
    (s2, move(l2,l3)),
    (s3, move(l3,l4)),
    (s4, wait),
    (s5, **move(l5,l4)**)}



Always reaches the state s4, waits there infinitely many times

Policy Example 3

- Example 3
  - π3 = { **(s1, move(l1,l4))**,
    (s2, move(l2,l1)),
    (s3, move(l3,l4)),
    (s4, wait),
    (s5, move(l5,l4)}



Reaches state s4 with 100% probability "in the limit"

Histories

Policies and Histories

- Executing a policy results in a **state sequence**: A **history**
  - Infinite, since policies do not terminate
  - $h = \langle s_0, s_1, s_2, s_3, s_4, \ldots \rangle$

s_0 (index zero): Variable used in histories, etc

s0: concrete state name used in diagrams

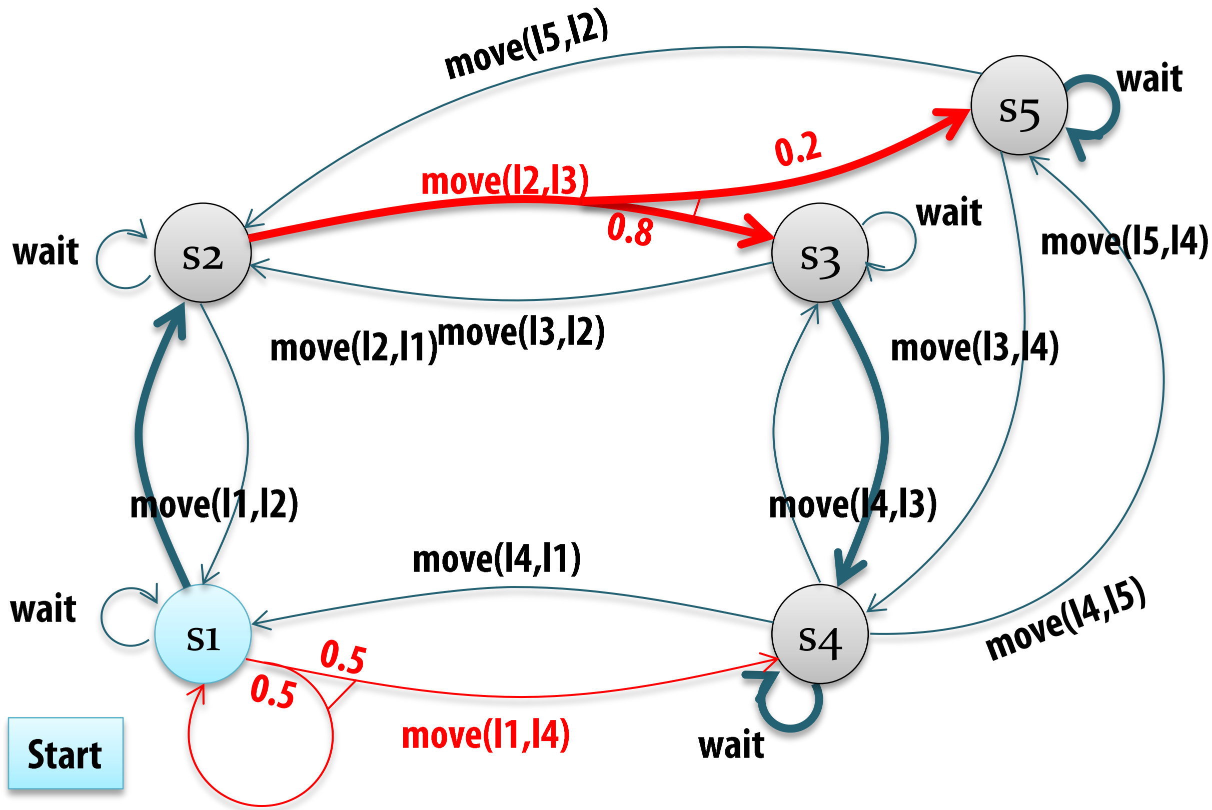
We may have s_0 = s27

- For classical planning:
  - We know the initial state
  - Actions are deterministic
  - ➔ A plan yields a **single** history (last state repeated infinitely)

- For probabilistic planning:
  - Initial states can be probabilistic
    - For every state *s*, there will be a **probability** $P(s)$ that we **begin** in the state *s*
  - Actions can have multiple outcomes
  - ➔ A policy can yield **many** different histories
    - Which one? Only known at execution time!

History Example 1

- Example 1
  - $\pi 1 = \{$ (s1, move(l1,l2)),
    (s2, move(l2,l3)),
    (s3, move(l3,l4)),
    (s4, wait),
    (s5, wait)$\}$

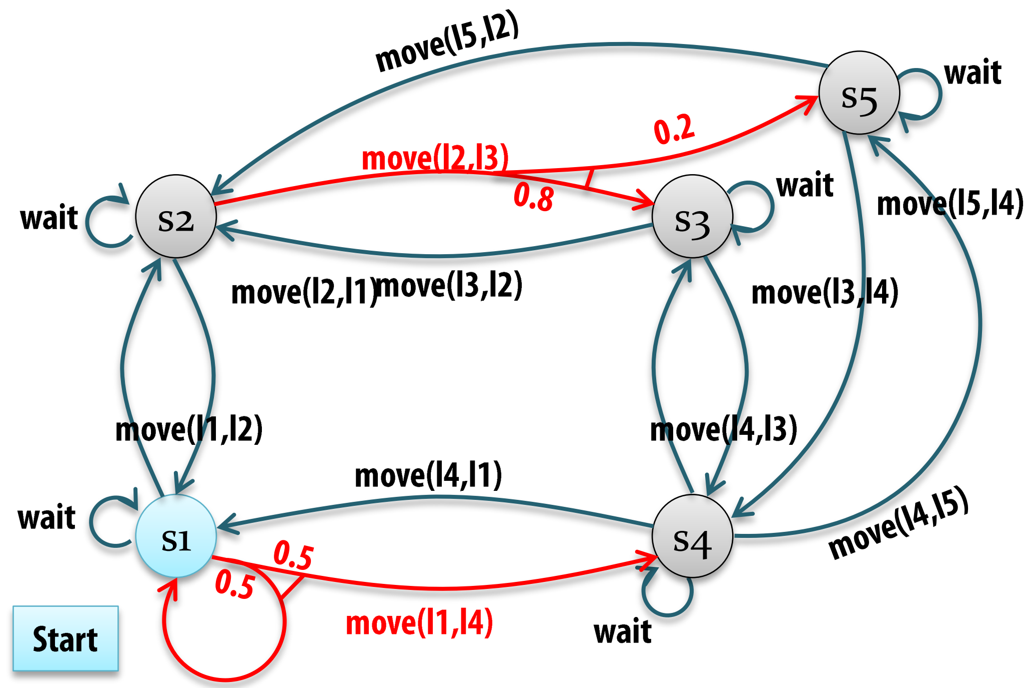


- Even if we only consider starting in s1: Two possible histories
  - $h_1 = \langle$s1, s2, s3, s4, s4, … $\rangle$ – Reached s4, waits indefinitely
    $h_2 = \langle$s1, s2, s5, s5 … $\rangle$ – Reached s5, waits indefinitely

How likely are these histories?

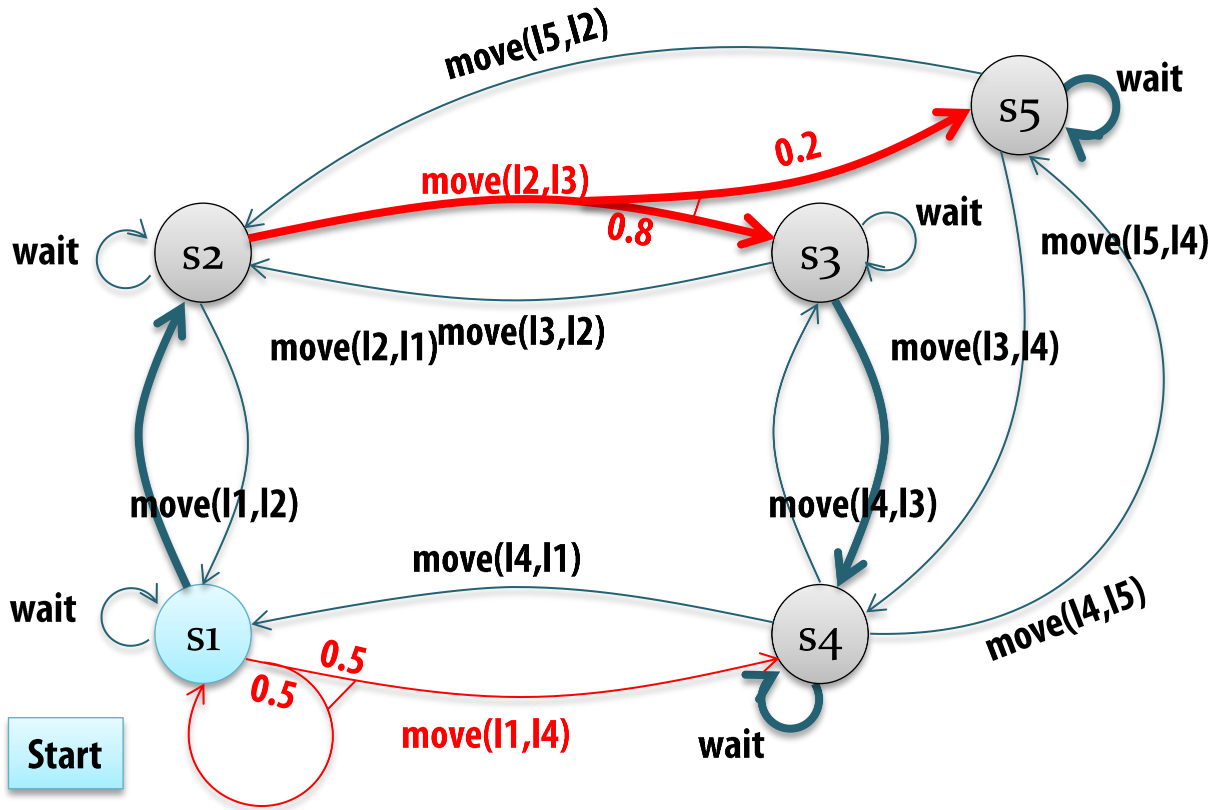
Probabilities: Initial States, Transitions

- Each policy induces a **probability distribution over histories**
  - Let $h = \langle s_0, s_1, s_2, s_3, \ldots \rangle$
  - With unknown initial state:
    - $P(\langle s_0, s_1, s_2, s_3, \ldots \rangle \mid \pi) = P(s_0) \prod_{i \geq 0} P(s_i, \pi(s_i), s_{i+1})$
  - The book:
    - Assumes you start in a known state s_0
    - So all histories start with the same state
    - $P(\langle s_0, s_1, s_2, s_3, \ldots \rangle \mid \pi) = \prod_{i \geq 0} P(s_i, \pi(s_i), s_{i+1})$



History Example 1

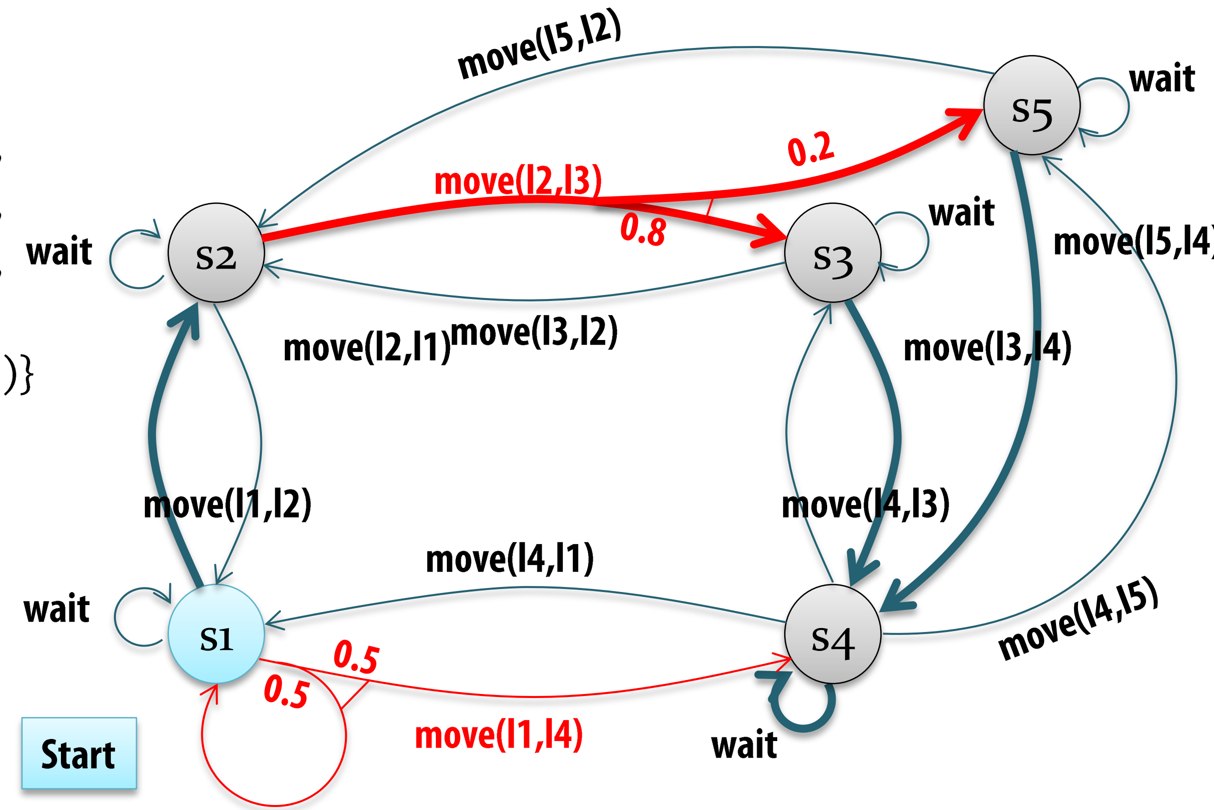
- Example 1
  - $\pi 1$ = { (s1, move(l1,l2)),
    (s2, move(l2,l3)),
    (s3, move(l3,l4)),
    (s4, wait),
    (s5, wait)}



- Two possible histories, if we always start in s1
  - $h_1 = \langle s1, s2, s3, s4, s4, \ldots \rangle$ – $P(h_1 \mid \pi_1) = 1 \times 1 \times 0.8 \times 1 \times \ldots = 0.8$
    $h_2 = \langle s1, s2, s5, s5 \ldots \rangle$ – $P(h_2 \mid \pi_1) = 1 \times 1 \times 0.2 \times 1 \times \ldots = 0.2$
    – $P(h \mid \pi_1) = 1 \times 0$ for all other h

History Example 2

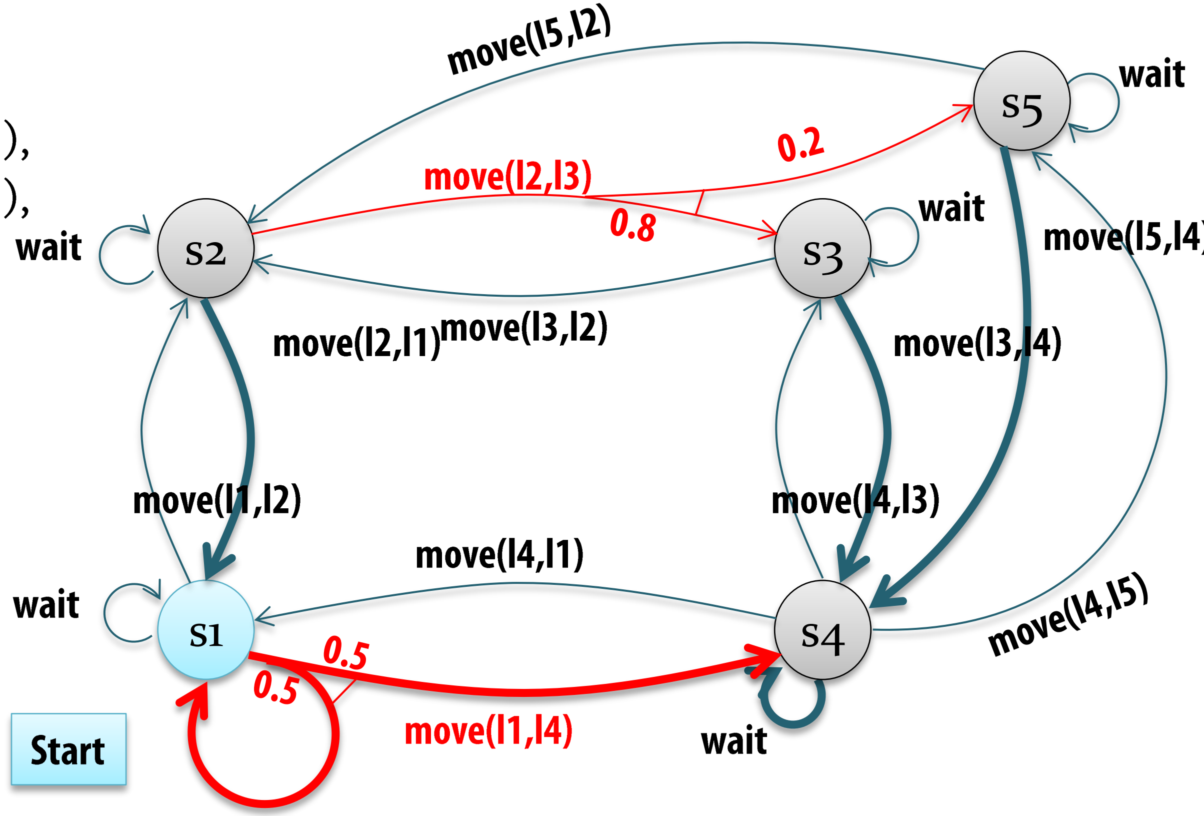
- Example 2
  - $\pi 2$ = { (s1, move(l1,l2)),
    (s2, move(l2,l3)),
    (s3, move(l3,l4)),
    (s4, wait),
    (s5, **move(l5,l4)**)}



- $h_1 = \langle s1, s2, s3, \boxed{s4, s4, \ldots} \rangle$ $\quad P(h_1 \mid \pi_2) = 1 \times 1 \times 0.8 \times 1 \times \ldots = 0.8$
  $h_3 = \langle s1, s2, s5, \boxed{s4, s4, \ldots} \rangle$ $\quad P(h_3 \mid \pi_2) = 1 \times 1 \times 0.2 \times 1 \times \ldots = 0.2$
  $P(h \mid \pi_2) = 1 \times 0$ for all other h

History Example 3

- Example 3
  - π3 = { **(s1, move(l1,l4))**, **(s2, move(l2,l1))**, (s3, move(l3,l4)), (s4, wait), (s5, move(l5,l4)}



  - $h_4 = \langle s1, s4, s4, \ldots \rangle$ $\quad P(h_4 \mid \pi_3) = 0.5 \times 1 \times 1 \times 1 \times 1 \times \ldots = 0.5$

    $h_5 = \langle s1, s1, s4, s4, \ldots \rangle$ $\quad P(h_5 \mid \pi_3) = 0.5 \times 0.5 \times 1 \times 1 \times 1 \times \ldots = 0.25$

    $h_6 = \langle s1, s1, s1, s4, s4, \ldots \rangle$ $\quad P(h_6 \mid \pi_3) = 0.5 \times 0.5 \times 0.5 \times 1 \times 1 \times \ldots = 0.125$

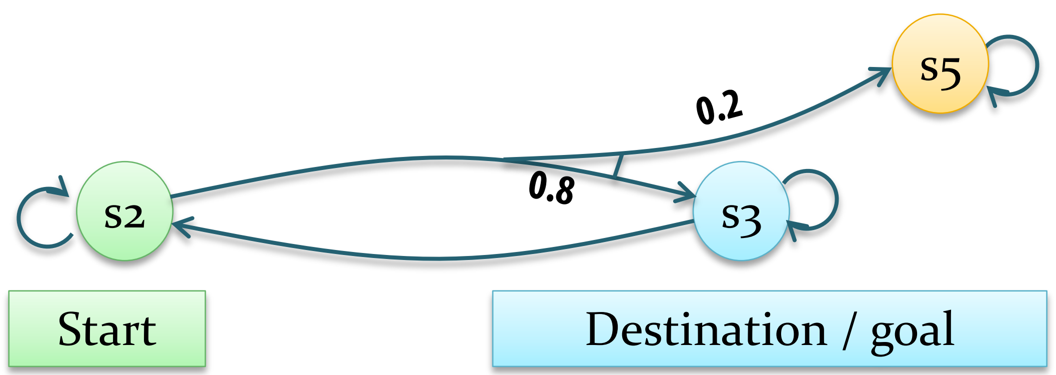
    …

    $h_\infty = \langle s1, s1, s1, s1, s1, s1, \ldots \rangle$ $\quad P(h_n \mid \pi_3) = 0.5 \times 0.5 \times 0.5 \times 0.5 \times 0.5 \times \ldots = 0$

Goals and Utility Functions

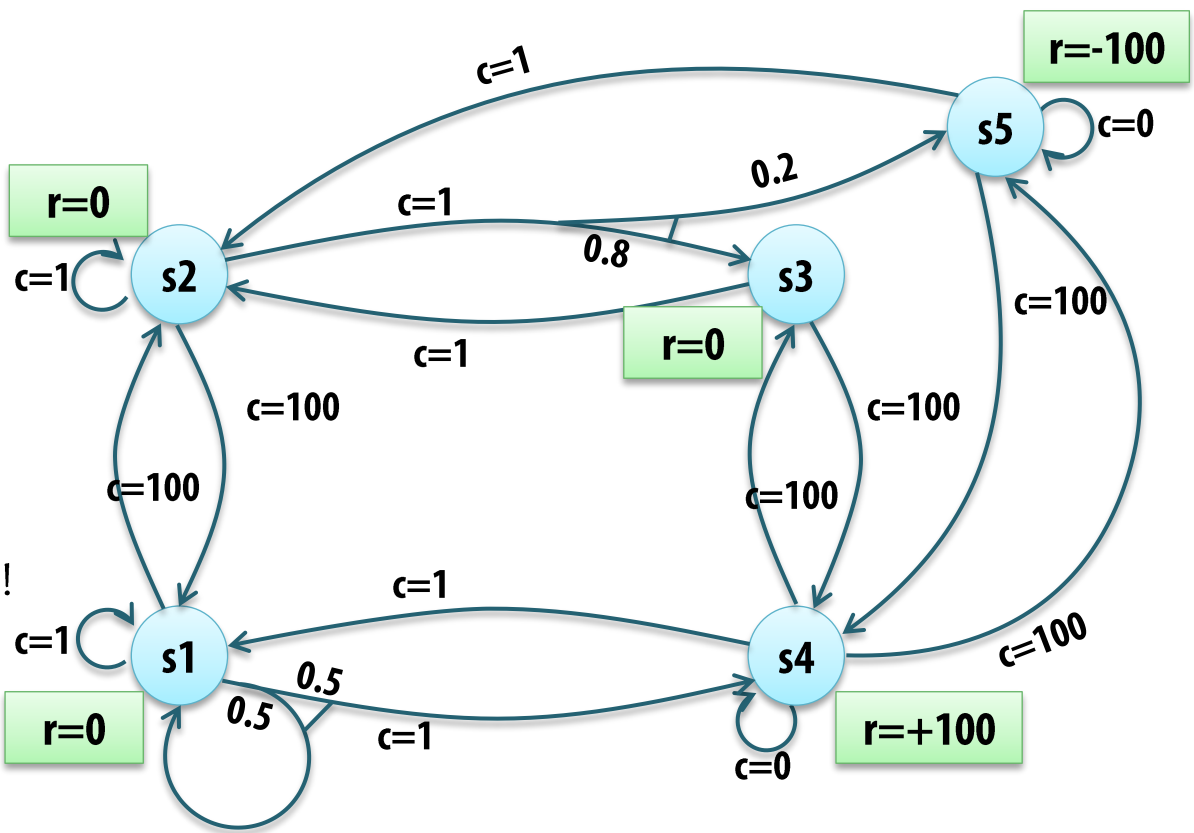
What is the Objective?

- What is the **objective**?
  - In classical planning: Want a plan resulting in a **goal state**
    - Natural formulation, since a plan always ends up in the same state
  - In probabilistic planning: This is still possible
    - A **weak solution** *may* reach a goal state in a finite number of steps
    - A **strong solution** *will* reach a goal state in a finite number of steps
    - A **strong cyclic solution** *will* reach a goal state in a finite number of steps given a fairness assumption:
      Informally, ”if we can exit a loop, we eventually will”



Costs and Rewards

- Alternative model, often used in MDP planning:
  - Numeric **cost** $C(s,a)$ for each state s and action a
  - Numeric **reward** $R(s)$ for each state s
- Example:
  - $C(s,a) = 1$ for each "horizontal" action
  - $C(s,a) = 100$ for each "vertical" action
  - $C(\text{s1,wait}) = 1$
    $C(\text{s2,wait}) = 1$
    $C(\text{s4,wait}) = 0$
    $C(\text{s5,wait}) = 0$
  - $R(\text{s5}) = -100$: Don't want to be there!
  - $R(\text{s4}) = 100$: This is a state that we want to reach



Utility Functions

- **<u>Utility functions</u>**
  - Suppose a policy leads us to go through a certain history (state sequence)
  - How ”useful / valuable" is this history to us?

<u>Utility</u> of history h
given policy π

- First attempt:
  - $h = \langle s_0, s_1, \ldots \rangle \rightarrow \boxed{V(h \mid \pi)} = \sum_{i \geq 0} \left(\boxed{R(s_i)} - \boxed{C(s_i, \pi(s_i))}\right)$

Add the **<u>reward</u>** for
being in state s_i

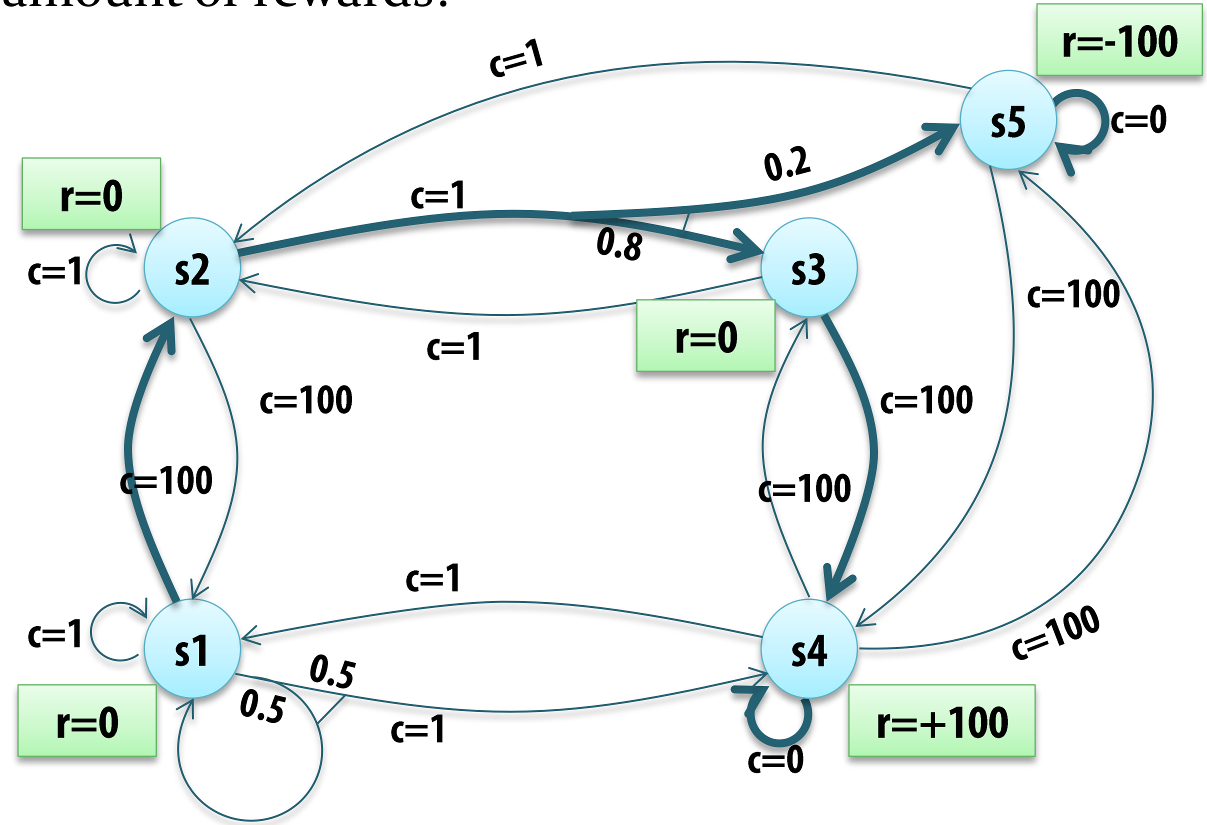
Subtract the **<u>cost</u>** of
the action chosen in s_i

Utility Functions

- Example:
  - Suppose π_1 happens to result in $h_1 = \langle s1, s2, s3, s4, s4, \ldots \rangle$
  - $V(h_1 \mid \pi_1) = (0 - 100) + (0 - 1) + (0 - 100) + 100 + 100 + \ldots$
  - We stay at s4 forever, executing "wait",
    so we get an **infinite** amount of rewards!

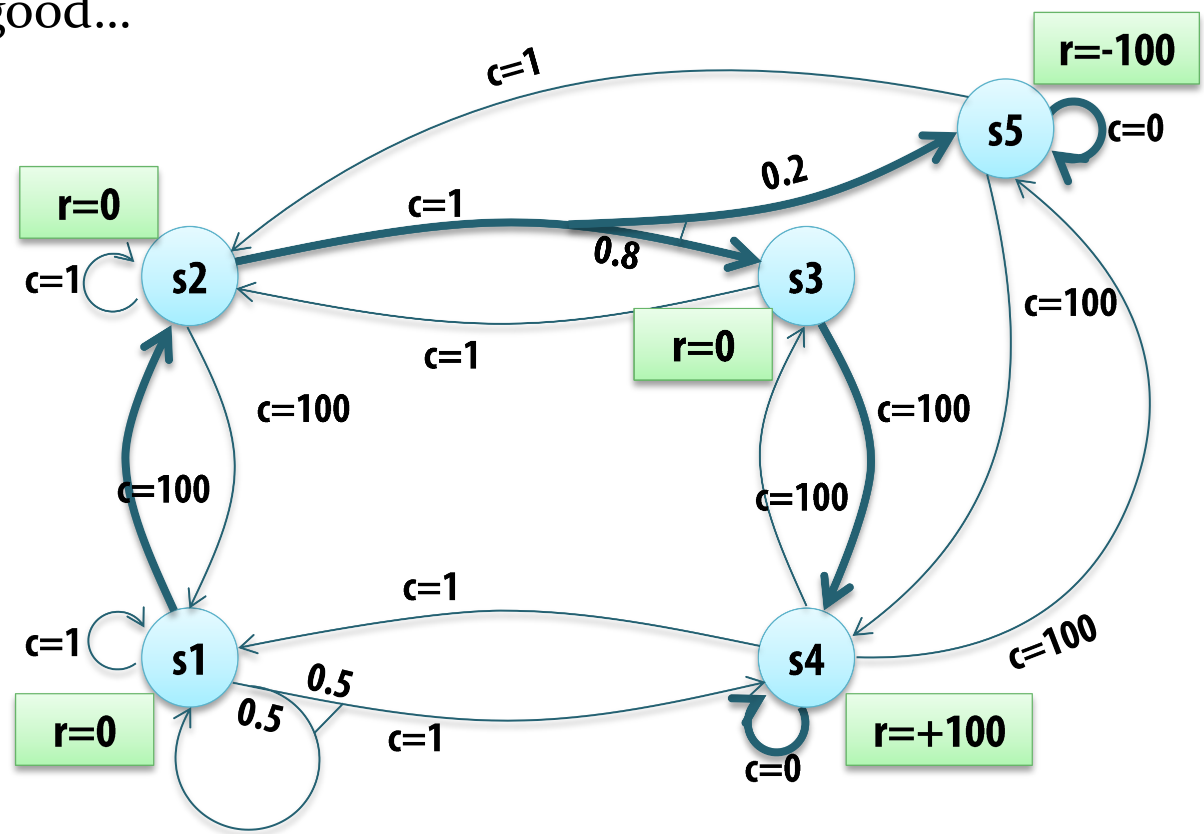
This is not the only history that could result from the policy!

That's why we specify the policy **and** the history to calculate a utility…



Utility Functions

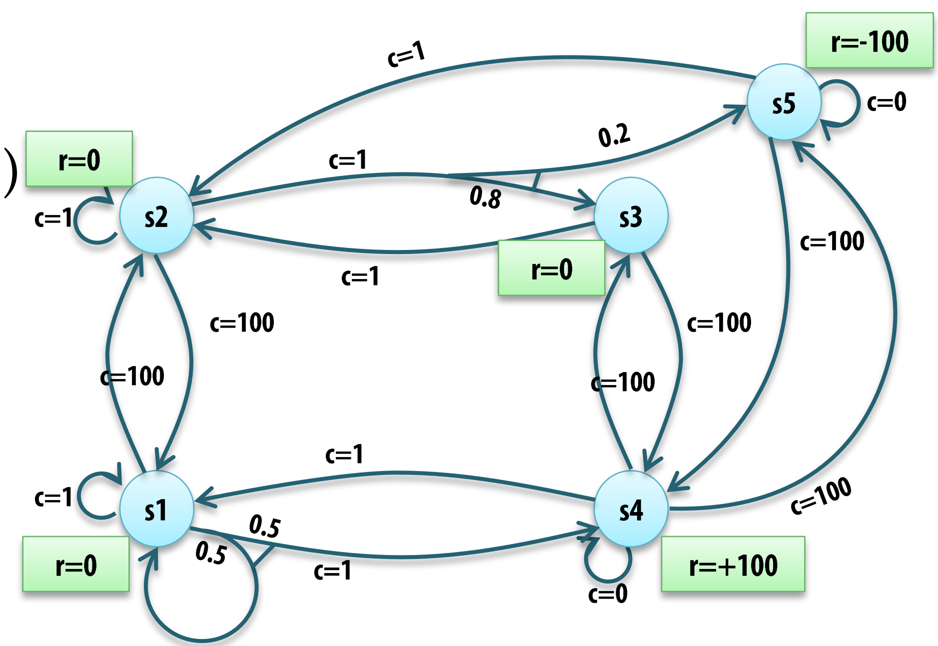
- What’s the problem, given that we "like" being in state s4?
  - We can’t distinguish between different ways of getting there!
  - s1→s2→s3→s4: -201 + ∞ = ∞
  - s1→s2→s1→s2→s3→s4: -401 + ∞ = ∞
  - Both appear equally good…



Discounted Utility

- Solution: Use a **discount factor**, γ, with $0 \le \gamma \le 1$
  - To avoid divergence (infinite utility values V(...))
  - To model "impatience": rewards and costs far in the future are less important to us

- **Discounted utility** of a history:
  - $V(h \mid \pi) = \sum_{i \geq 0} \boxed{\gamma^i} (R(s_i) - C(s_i, \pi(s_i)))$
  - Distant rewards/costs have **less influence**
  - **Convergence** (with finite results) is guaranteed if $0 \le \gamma < 1$



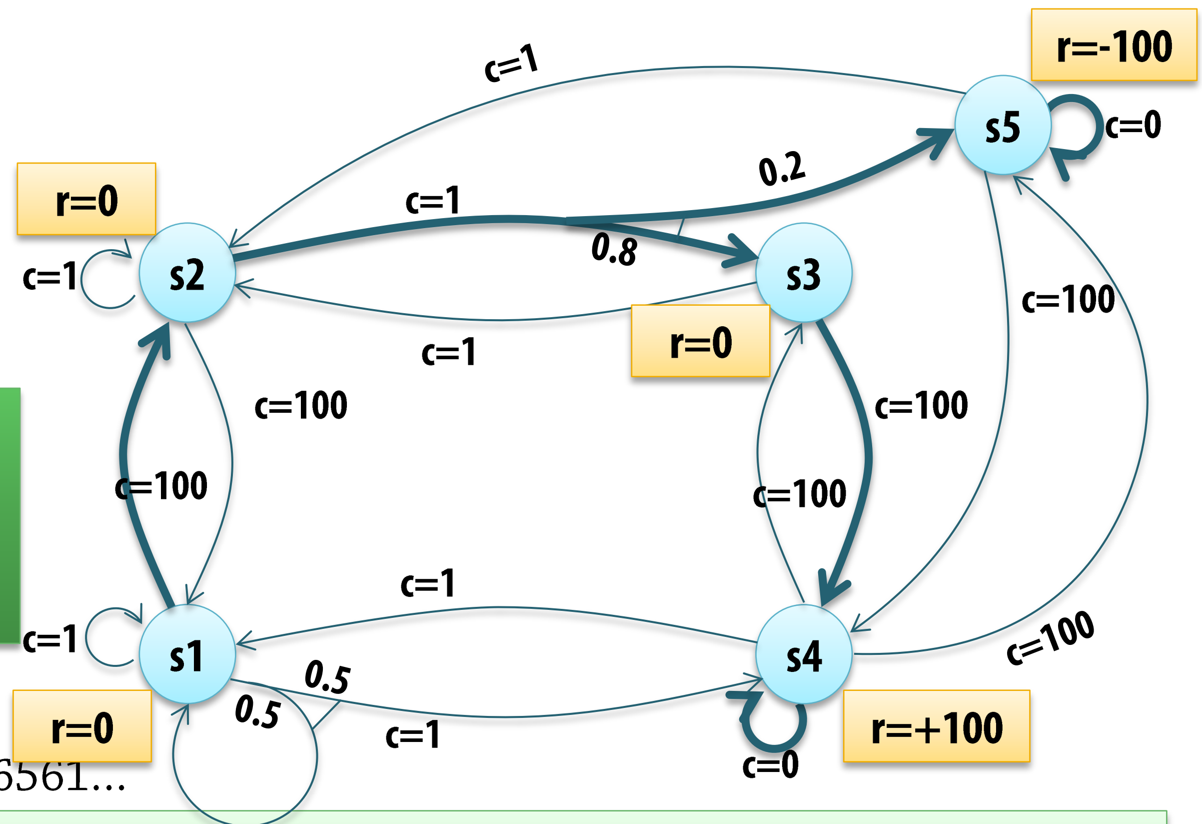
Expected Utility, Optimality, Solutions

- Still only tells us the utility of a **history**
  - But we can't force a history
  - Can only decide a policy – which can lead to **many** histories

- Assuming a known starting state:
  - **Expected utility** of a **policy**: $E(\pi) = \sum_h P(h \mid \pi) V(h \mid \pi)$
    - How **probable** is each history (outcome), and how **valuable** is it to us?
  - A policy π is **optimal** if no other policy has greater expected utility
    - For every π', $E(\pi) >= E(\pi')$
  - A solution is an **optimal policy**!
    - Gives us the greatest (expected) reward that we **can** get, given the specified probabilities, costs, and rewards

Example

π_1 = {(s1, move(l1,l2)),
(s2, move(l2,l3)),
(s3, move(l3,l4)),
(s4, wait),
(s5, wait)}



Given that we start in s1,
this simple policy can lead to only
__two__ different histories…
80% chance of history h1,
20% chance of history h2

$\gamma = 0.9$

Factors 1, 0.9, 0.81, 0.729, 0.6561…

$h_1 = \langle$s1, s2, s3, s4, s4, … $\rangle$

$V(h_1 \mid \pi_1) = .9^0(0 - 100) + .9^1(0 - 1) + .9^2(0 - 100) + .9^3\,100 + .9^4\,100 + \ldots = 547.9$

$h_2 = \langle$s1, s2, s5, s5 … $\rangle$

$V(h_2 \mid \pi_1) = .9^0(0 - 100) + .9^1(0 - 1) + .9^2(-100\text{-}0) + .9^3(-100\text{-}0) + \ldots = -910.1$

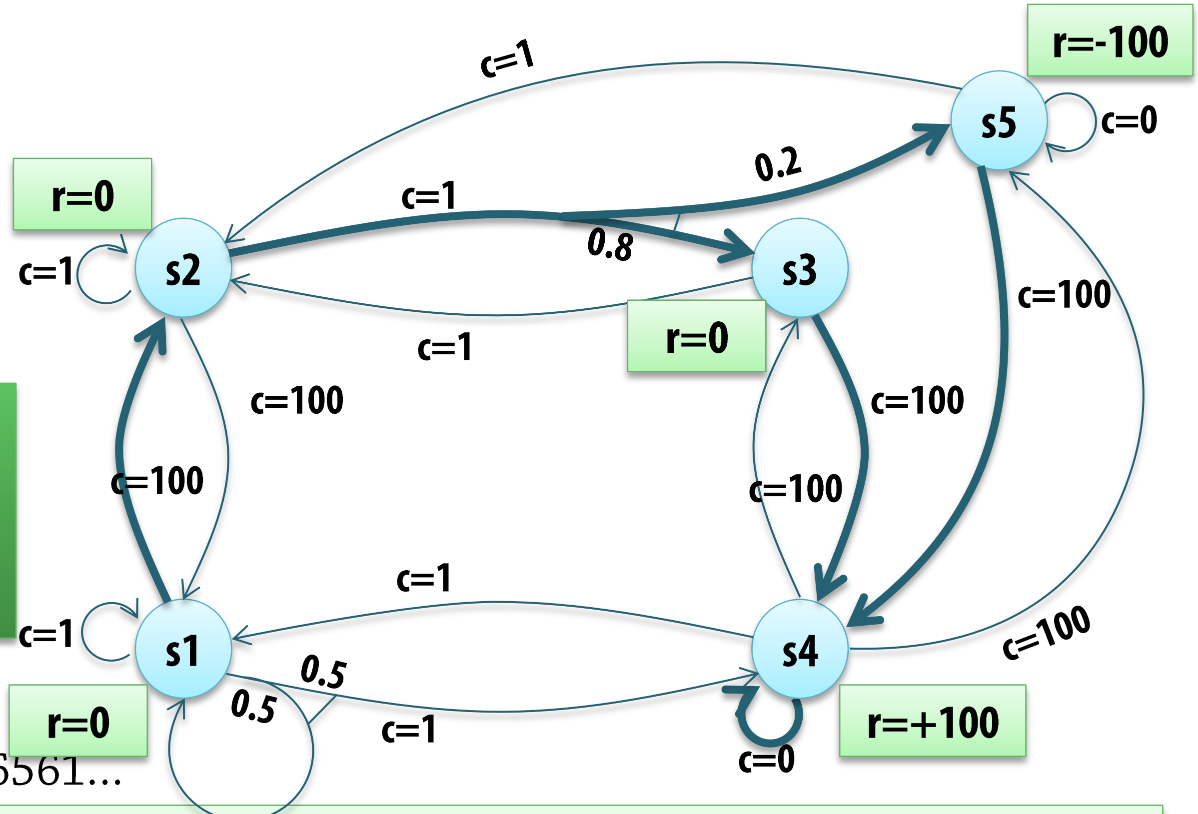
$E(\pi_1) = 0.8 * 547.9 + 0.2\,(-910.1) = 256.3$

We expect a reward of 256.3 on average

Example

π_2 = {(s1, move(l1,l2)),
(s2, move(l2,l3)),
(s3, move(l3,l4)),
(s4, wait),
(s5, **move(l5,l4)**}

Given that we start in s1,
also **two** different histories…
80% chance of history h1,
20% chance of history h2



$\gamma = 0.9$

Factors 1, 0.9, 0.81, 0.729, 0.6561…

$h_1 = \langle s1, s2, s3, s4, s4, \ldots \rangle$

$V(h_1 \mid \pi_1) = .9^0(0-100) + .9^1(0-1) + .9^2(0-100) + .9^3\,100 + .9^4\,100 + \ldots = 547.9$

$h_2 = \langle s1, s2, s5, s5 \ldots \rangle$

$V(h_2 \mid \pi_1) = .9^0(0-100) + .9^1(0-1) + .9^2(-100-100) + .9^3 100 + \ldots = 466.9$

$E(\pi_1) = 0.8 * 547.9 + 0.2\ (466.9) = 531{,}7$

Expect 531,7 on average (π_1 gave 256.3)

Summary

- **Markov Decision Processes**
  - Underlying world model: **Stochastic system**
  - Plan representation: **Policy** – which action to perform in **any** state
  - Goal representation: **Utility function** defining "solution quality"
  - Planning problem: **Optimization**: Maximize **expected** utility

Finding a Solution: Preliminaries

Special Case

- To simplify the presentation of important **principles**:
  - Let's consider a **special case**:
    - We start in a known state, s_0
    - All rewards are 0
  - Can easily be generalized

- We should **minimize** the expected **cost** of a policy:
  - $E(\pi) = \sum_h P(h \mid \pi)\, C(h \mid \pi)$
    - Where $C(h \mid \pi) = \sum_{i \geq 0} \gamma^i\, C(s_i, \pi(s_i))$ (discounted cost)
    - replaces $V(h \mid \pi) = \sum_{i \geq 0} \gamma^i\, (R(s_i) - C(s_i, \pi(s_i)))$ (discounted cost/reward)

- We will also need to know:
  - $E_\pi(s)$ = the expected cost of executing π **starting in some specific state s**

Calculating Costs

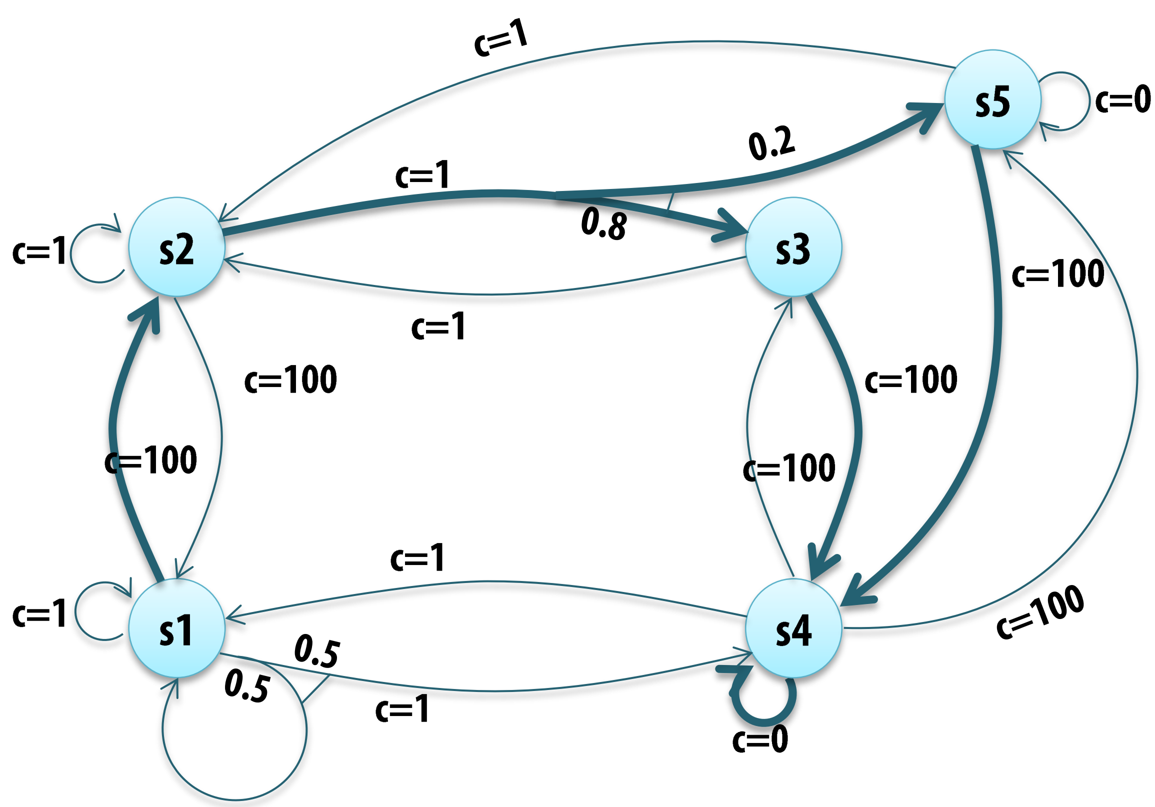
- How can we calculate $E_\pi(s)$?
  - If we visit the states ⟨s1, s2, s3, s4, s5,…⟩ where s1 = *s*:
    - $E_\pi(s) = \sum_{i \geq 0} \gamma^i C(s_i, \pi(s_i))$
  - But only the **first** state is known in advance!

Bellman's Theorem: Background

- If π is a policy, then
  - $E_\pi(s) = C(s, \pi(s)) + \gamma \sum_{s' \in S} P(s, \pi(s), s') \; E_\pi(s')$
  - The expected cost of executing π starting in s
  - Is the cost of executing the action chosen by the policy, π(s), in s
  - Plus the discount factor γ times…
    - …the sum, for all possible states $s' \in S$ that you **might** end up in,
      - of the probability $P(s, \pi(s), s')$ of actually ending up in that state given the action π(s) chosen by the policy
      - times the expected cost $E_\pi(s')$ of executing π starting in that new state s'
- (If you expand in one step…)
  - $E_\pi(s) = C(s, \pi(s)) + \gamma \sum_{s' \in S} P(s, \pi(s), s')$ [
    $C(s', \pi(s')) + \gamma \sum_{s'' \in S} P(s', \pi(s''), s'') \quad E_\pi(s'')$
    ]

Example 1

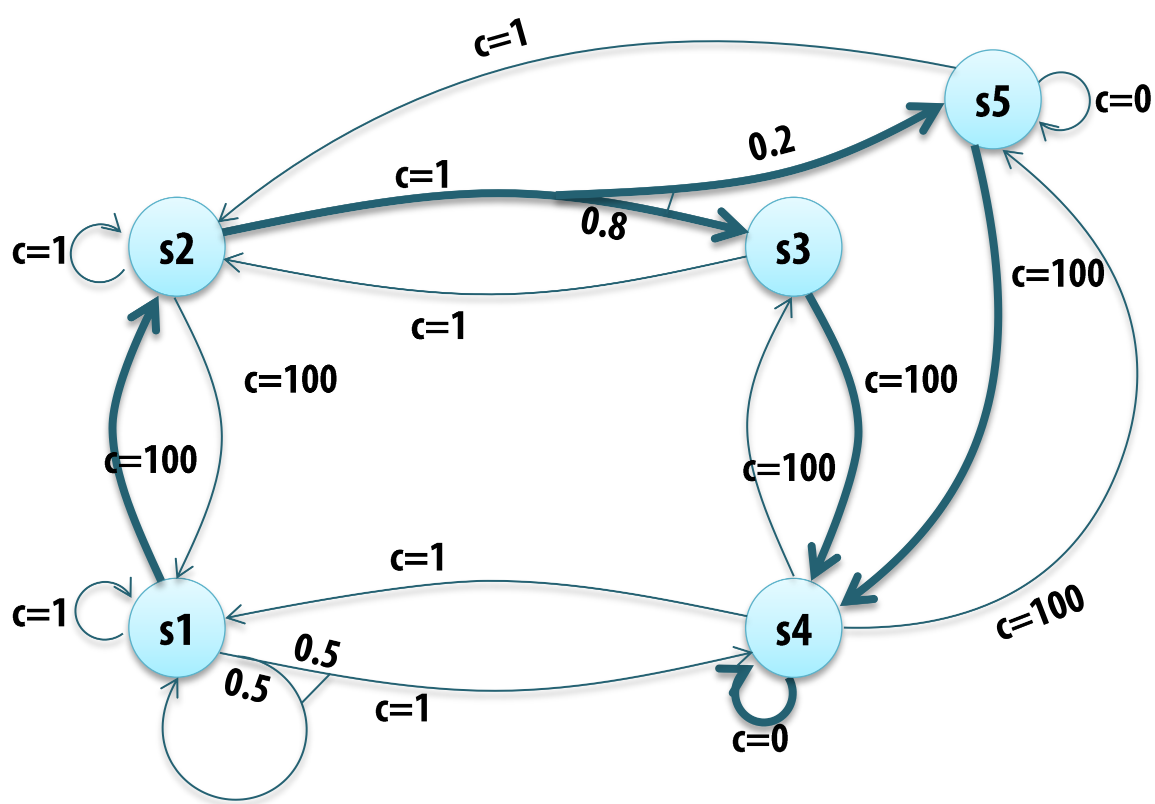
- $E_{\pi 2}$(s1) = The expected cost of executing π_2 starting in **s1**:
  - The cost of the first action: move(l1,l2)
  - Plus the discount factor γ times…
    - [Ending up in s2]
      100% * $E_{\pi 2}$(s2)



π_2 = {(s1, move(l1,l2)),
(s2, move(l2,l3)),
(s3, move(l3,l4)),
(s4, wait),
(s5, move(l5,l4)}

Example 2

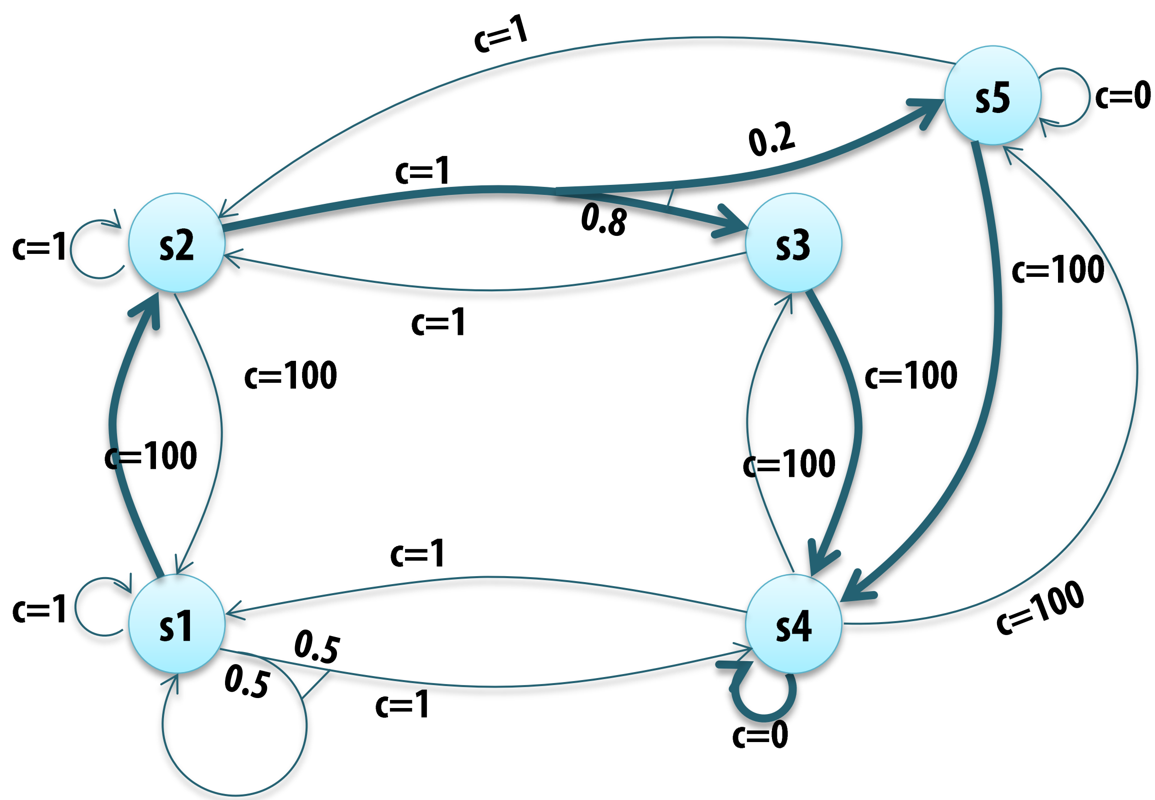
- $E_{\pi 2}(s2)$ = the expected cost of executing π_2 starting in **s2**:
  - The cost of the first action: move(l2,l3)
    - (Which has multiple outcomes!)
  - Plus the discount factor γ times…
    - [Ending up in s3]
      80% * $E_{\pi 2}(s3)$
    - Plus
      [Ending up in s5]
      20% * $E_{\pi 2}(s5)$



π_2 = {(s1, move(l1,l2)),
(s2, move(l2,l3)),
(s3, move(l3,l4)),
(s4, wait),
(s5, move(l5,l4)}

Recursive?

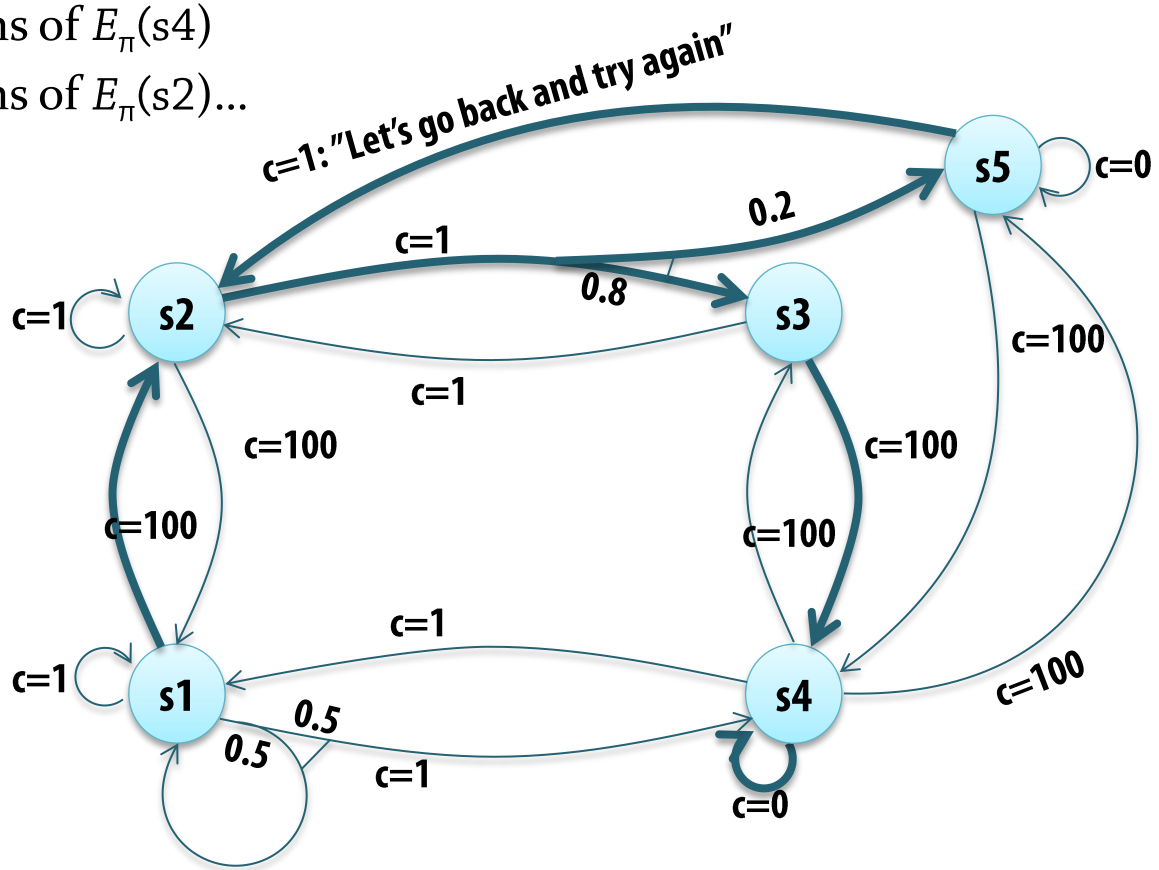
- Seems like we could easily calculate this **recursively**!
  - $E_{\pi 2}$(s1) defined in terms of $E_{\pi 2}$(s2)
  - $E_{\pi 2}$(s2) defined in terms of $E_{\pi 2}$(s3) and $E_{\pi 2}$(s5)
  - …
  - Just continue until you reach the end!



π_2 = {(s1, move(l1,l2)),
(s2, move(l2,l3)),
(s3, move(l3,l4)),
(s4, wait),
(s5, move(l5,l4)}

Not Recursive!

- But **<u>there isn't always an "end"</u>**!
  - Modified example below is a valid policy π:
    - E_π(s1) defined in terms of E_π(s2)
    - E_π(s2) defined in terms of E_π(s3) and E_π(s5)
    - E_π(s3) defined in terms of E_π(s4)
    - E_π(s5) defined in terms of E_π(s2)…



Bellman's Theorem: Equation System

- If π is a policy, then for all states s:
  - $E_\pi(s) = C(s, \pi(s)) + \gamma \sum_{s' \in S} P(s, \pi(s), s')\ E_\pi(s')$
  - The expected cost of executing π starting in s
  - Is the cost of executing the action chosen by the policy, π(s), in s
  - Plus the discount factor γ times…
    - …the sum, for all possible states $s' \in S$ that you **might** end up in,
      - of the probability $P(s, \pi(s), s')$ of actually ending up in that state given the action π(s) chosen by the policy
      - times the expected cost $E_\pi(s')$ of executing π starting in that new state s'

This is an **equation system**: |S| equations, |S| variables!

Requires different solution methods…

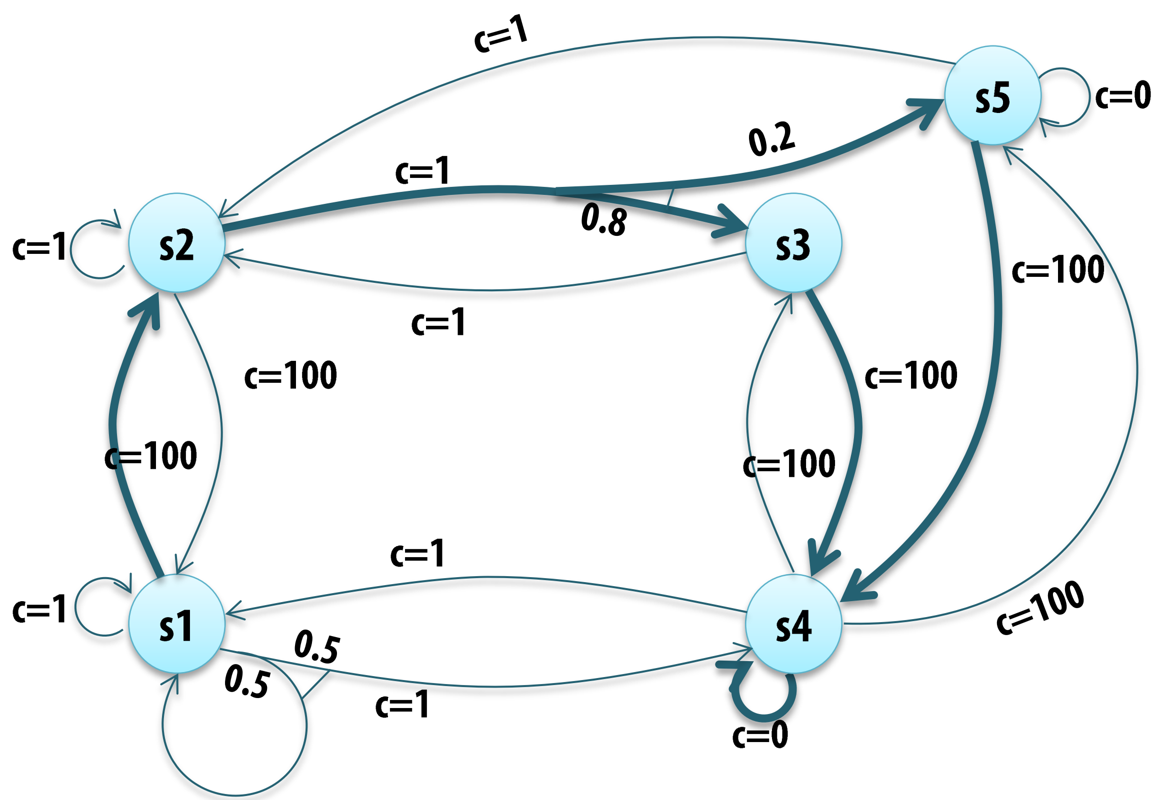
Principle of Optimality

- Bellman's **Principle of Optimality**:
  - An **optimal policy** has the property that whatever the initial state and initial decision are, the **remaining decisions must constitute an optimal policy** with regard to the state resulting from the first decision

Problem: Find a policy that minimizes cost given that we start in s1.

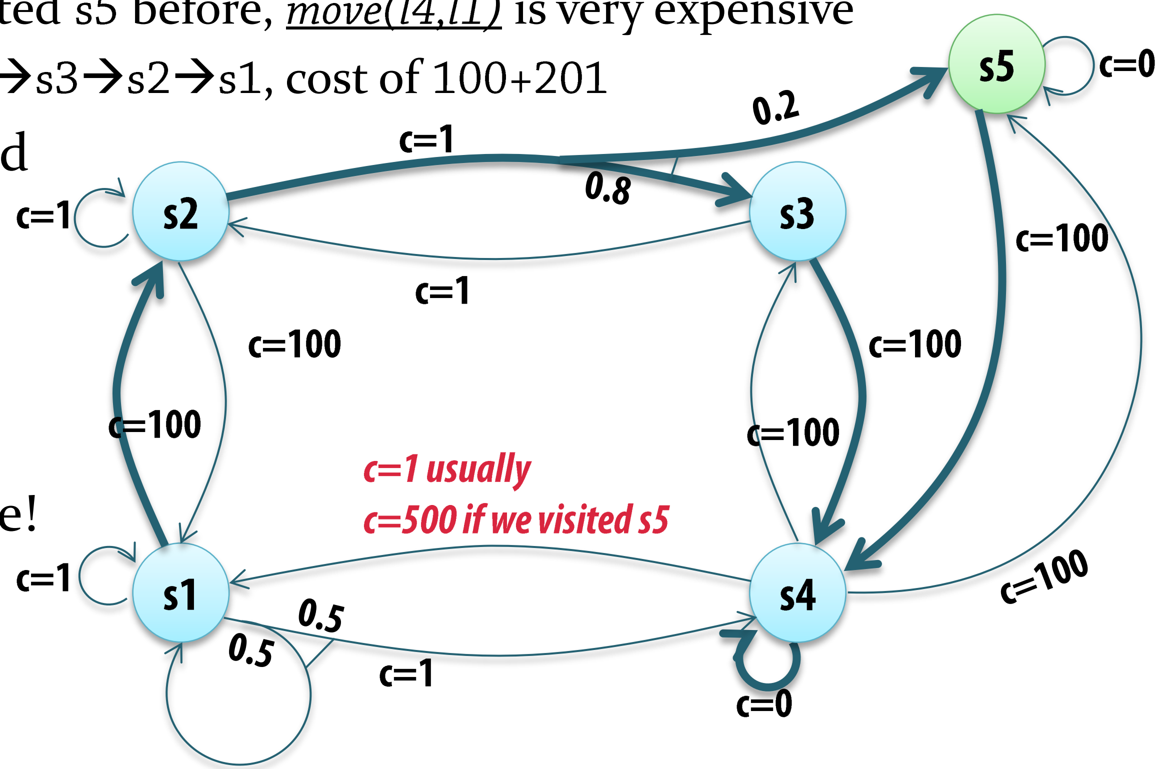
Suppose that an **optimal policy** π* begins with move(l1,l2), so that the next state is s2.

Then π* **must also minimize cost** given that we start in s2!



Principle of Optimality (2)

- Sounds trivial? Depends on the **Markov Property**!
  - Suppose **costs** depended on **which states you had visited before**
  - Suppose you want to go s5 → s1
    - First action should be *move(l5,l4)*
  - Now you need to go s4 → s1
    - Because you have visited s5 before, *move(l4,l1)* is very expensive
    - Best solution: s5→s4→s3→s2→s1, cost of 100+201
  - But if you **only** wanted to go s4→s1:
    - *move(l4,l1)*, with a cost of 1
  - This can't happen here!
    - **Markovian**!



Solution Methods (1)

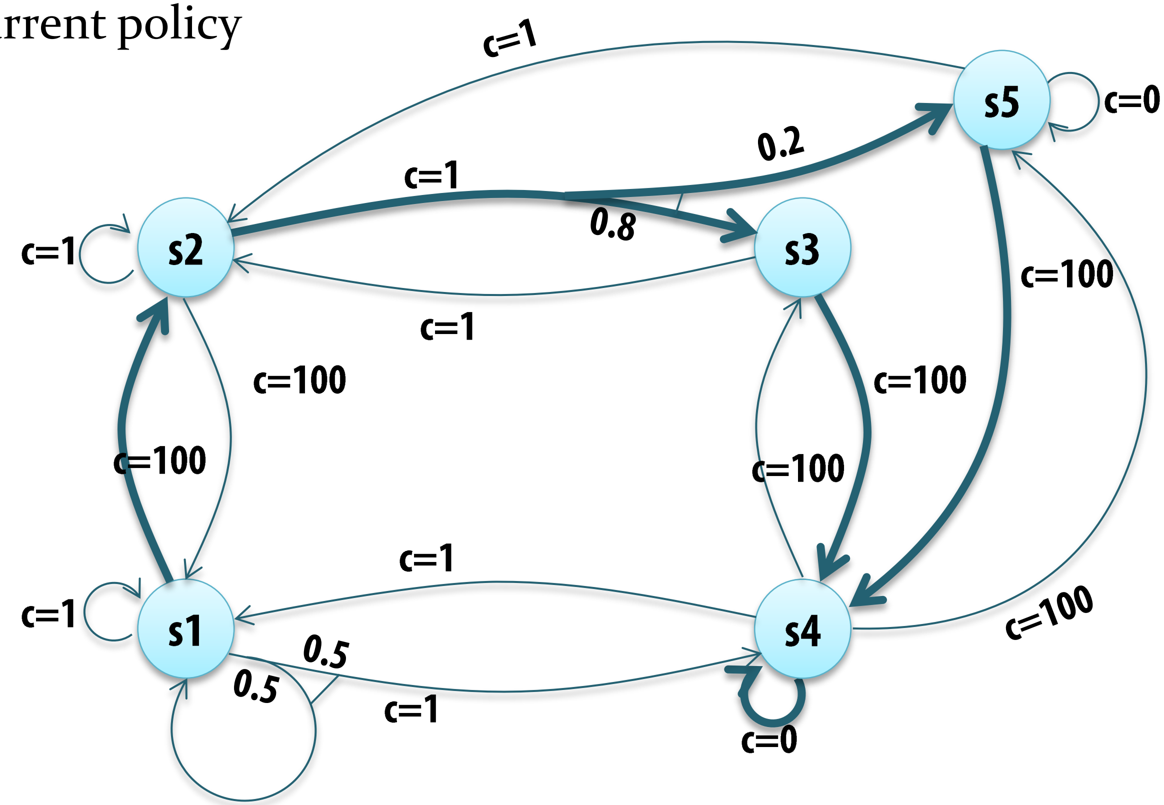
- Let's **hypothesize**:
  What if I made **this** local change, but kept everything else?
  - Let $Q_\pi(s,a)$ be the expected cost of π in a state s if we **start** by executing the given action a, but we use the **policy** π from then onward

Local change!

  - $E_\pi(s) = C(s, \pi(s)) + \gamma \sum_{s' \in S} P(s, \pi(s), s') \; E_\pi(s')$
  - $Q_\pi(s,a) = C(s, a) + \gamma \sum_{s' \in S} P(s, a, s') \; E_\pi(s')$

Example

- Example: $E_\pi(s1)$
  - The expected cost of following the current policy
  - Starting in s1, beginning with move(l1,l2)
- $Q_\pi(s1,$ move(l1,l4))
  - The expected cost of first trying to move from l1 to l4, then following the current policy



Solution Methods (2)

- Suppose you have an **<u>everywhere optimal</u>** policy π^*
  - That is, no other policy gives a better result for **<u>any</u>** starting state

- Then, because of the principle of optimality:
  - For all states s, $E_{\pi^*}(s) = \min_a Q_{\pi^*}(s,a)$
  - For all states s, $E_{\pi^*}(s) = \min_a (C(s,a) + \gamma \sum_{s' \in S} P(s, a, s') E_{\pi^*}(s'))$

Choice **<u>now</u>**

"The rest"

  - In every state,
    the **<u>local</u>** choice made by the policy
    is **<u>locally</u>** optimal

Solution Methods (3)

- Suggests a **specific type** of solution method:
  - Try to separate the decision in **this** state
    from the decisions in the **remainder** of the policy
  - Use **iterative refinement**
    - Start with some initial values (for example, a random policy)
    - Find local improvements

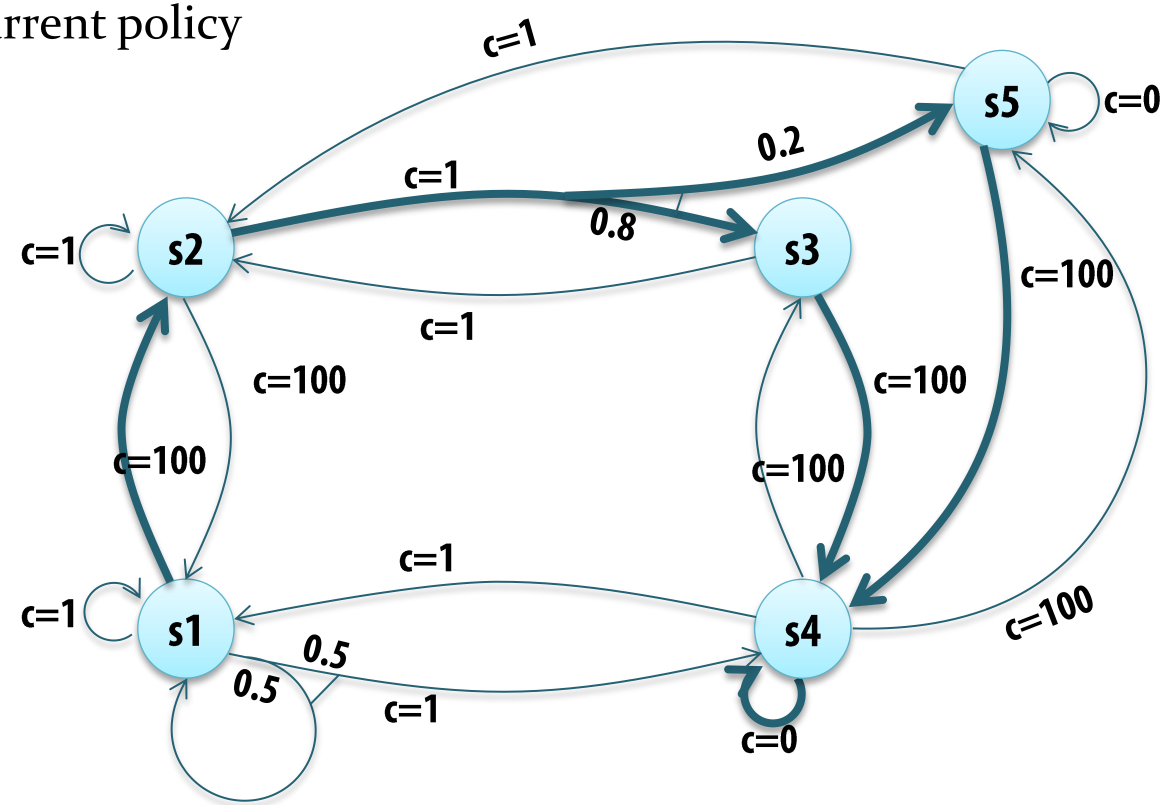
Example, Revisited

- Example: $E_\pi(s1)$
  - The expected cost of following the current policy
  - Starting in s1, beginning with move(l1,l2)
- $Q_\pi(s1, \text{move(l1,l4)})$
  - The expected cost of first trying to move from l1 to l4, then following the current policy

If doing move(l1,l4) first has a lower expected cost, we may want to modify the current policy:

(s1, move(l1,l4))

Details: Next time!



Action Representations

Action Representations

- **Action representations**:
  - The book only deals with the **underlying semantics**: Explicit enumeration of each P(s, a, s')
  - Several "convenient" representations possible, such as Bayes networks, probabilistic operators

Representation Example: PPDDL

- **Probabilistic PDDL**: new constructs for effects, initial state
  - (probabilistic p_1 e_1 … p_k e_k)
    - Effect e_1 takes place with probability p_1, etc.
    - **Sum** of probabilities <= 1 (can be strictly less ➔ implicit empty effect)
    - ```
      (define (domain bomb-and-toilet)
          (:requirements :conditional-effects :probabilistic-effects)
          (:predicates (bomb-in-package ?pkg) (toilet-clogged) (bomb-defused))
          (:action dunk-package
              :parameters (?pkg)
              :effect (and
                  (when (bomb-in-package ?pkg) (bomb-defused))
                  (probabilistic 0.05 (toilet-clogged)))))
      ```
    - ```
      (define (problem bomb-and-toilet)
          (:domain bomb-and-toilet)
          (:requirements :negative-preconditions)
          (:objects package1 package2)
          (:init (probabilistic   0.5 (bomb-in-package package1)
                                  0.5 (bomb-in-package package2)))
          (:goal (and (bomb-defused) (not (toilet-clogged)))))
      ```

First, a "standard" effect

5% chance of toilet-clogged, 95% chance of no effect

Probabilistic initial state

Ladder 1

- ;; Authors: Sylvie Thiébaux and Iain Little
- ;; Story: You are stuck on a roof because the ladder you climbed up on
- ;; fell down. There are plenty of people around; if you call out for
- ;; help someone will certainly lift the ladder up again. Or you can
- ;; try the climb down without it. You aren't a very good climber
- ;; though, so there is a 50-50 chance that you will fall and break
- ;; your neck if you go it alone. What do you do?

```
(define (domain climber)
  (:requirements :typing :strips :probabilistic-effects)
  (:predicates (on-roof) (on-ground)
               (ladder-raised) (ladder-on-ground) (alive))
  (:action climb-without-ladder :parameters ()
    :precondition (and (on-roof) (alive))
    :effect (and (not (on-roof))
                 (on-ground)
                 (probabilistic 0.4 (not (alive)))))
```

Ladder 2

- (:action climb-with-ladder :parameters ()
-    :precondition (and (on-roof) (alive) (ladder-raised))
-    :effect (and (not (on-roof)) (on-ground)))
- (:action call-for-help :parameters ()
-    :precondition (and (on-roof) (alive) (ladder-on-ground))
-    :effect (and (not (ladder-on-ground))
-                                   (ladder-raised))))

- (define (problem climber-problem)
- (:domain climber)
- (:init (on-roof) (alive) (ladder-on-ground))
- (:goal (and (on-ground) (alive))))

Representation Example: RDDL

- domain prop_dbn {

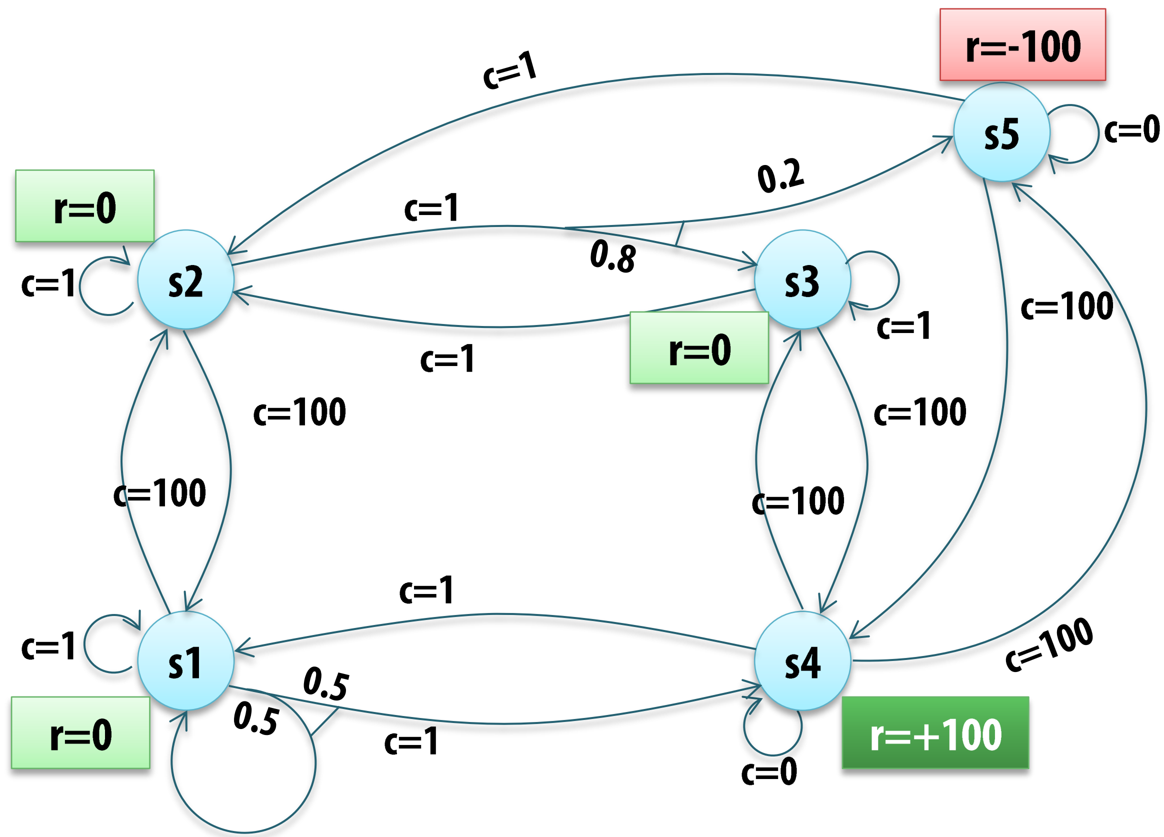
```
        requirements = { reward - deterministic };
        // Define the state and action variables ( not parameterized here )
        pvariables {
                p : { state - fluent , bool , default = false };
                q : { state - fluent , bool , default = false };
                r : { state - fluent , bool , default = false };
                a : { action - fluent , bool , default = false };
        };
        // Define the conditional probability function for each next
        // state variable in terms of previous state and action
        cpfs {
                p’ = if (p ^ r) then Bernoulli (.9) else Bernoulli (.3);
                q’ = if (q ^ r) then Bernoulli (.9)
                else if (a) then Bernoulli (.3) else Bernoulli (.8);
                r’ = if (~q) then KronDelta (r) else KronDelta (r <=> q);
        };
        // Define the reward function ; note that boolean functions are
        // treated as 0/1 integers in arithmetic expressions
        reward = p + q - r;
}
```

Automated Planning

Planning Based on Markov Decision Processes, part 2

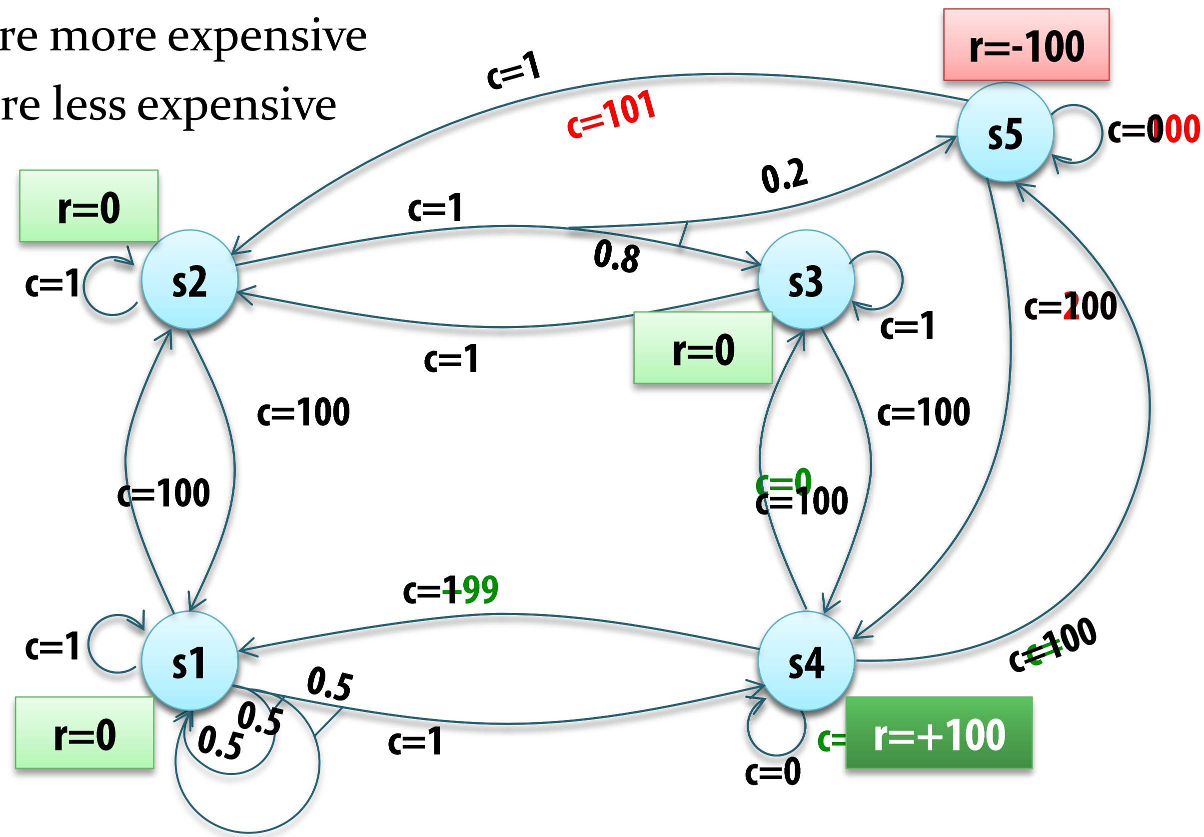
Example Problem

- Example Problem with Rewards and Costs



Example Problem, Simplified

- In the model we used, rewards and costs are always "taken together"
  - Can't get a reward without a cost or vice versa
  - R(s) – C(s, a): You are **<u>in a state</u>**, and then you execute an action **<u>in that state</u>**
- To simplify, we **<u>include the reward in the cost</u>**!
  - Decrease each C(s,a) by R(s)
  - Transitions from s5 are more expensive
  - Transitions from s4 are less expensive
  - Sometimes negative costs – not a problem!



- **<u>Objective</u>** is to **<u>minimize cost</u>**
  - Automatically takes rewards into account

Finding a Solution (Optimal Policy): Algorithm 1, Policy Iteration

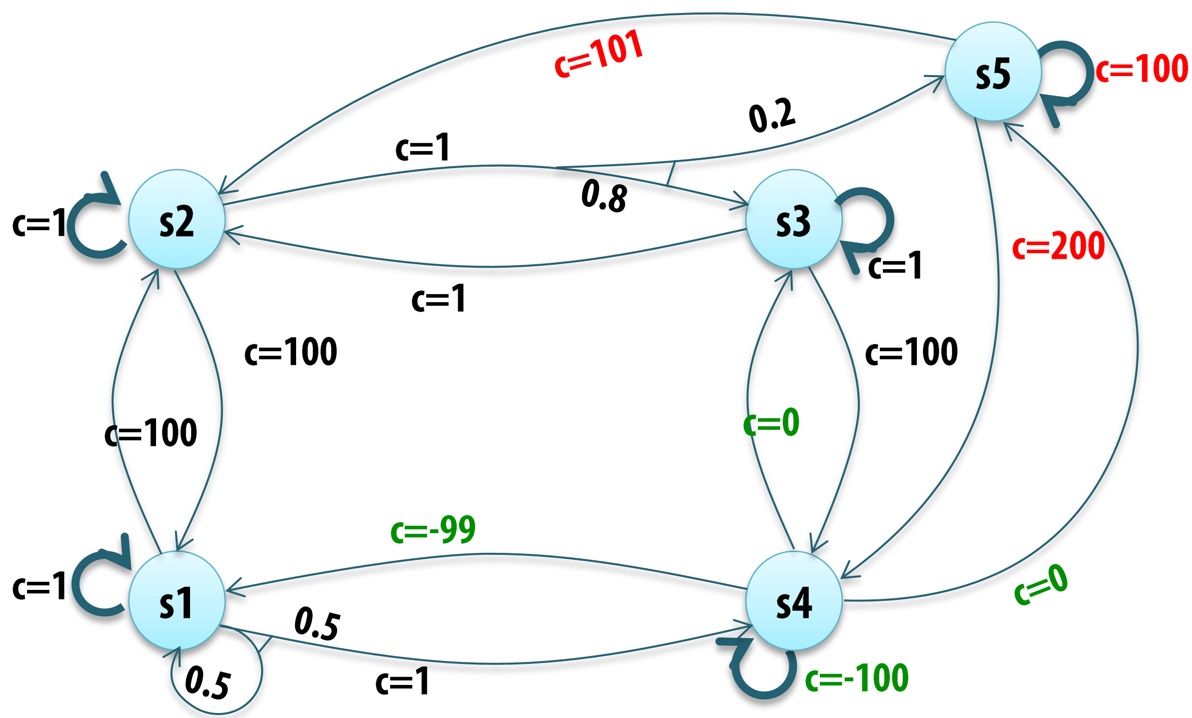
Policy Iteration

- First algorithm: **Policy iteration**
  - General idea:
    - Start out with an **initial policy**, maybe randomly chosen
    - Calculate the **expected cost** of executing that policy from each state
    - **Update** the policy by making a **local** decision **for each state** : "Which action should my **improved** policy choose in this state, given the expected costs of the **current** policy?"
    - Iterate until convergence (the policy no longer changes)

Policy Iteration 2: Initial Policy π_1

- Policy iteration requires an **initial policy**
  - Let's start by choosing "wait" in every state
  - Let's set a discount factor: $\gamma = 0.9$
    - Easy to use in calculations on these slides, but in reality we might use a larger factor (we're not **that** short-sighted!)

$$\pi_1 = \{(s1, wait), (s2, wait), (s3, wait), (s4, wait), (s5, wait)\}$$



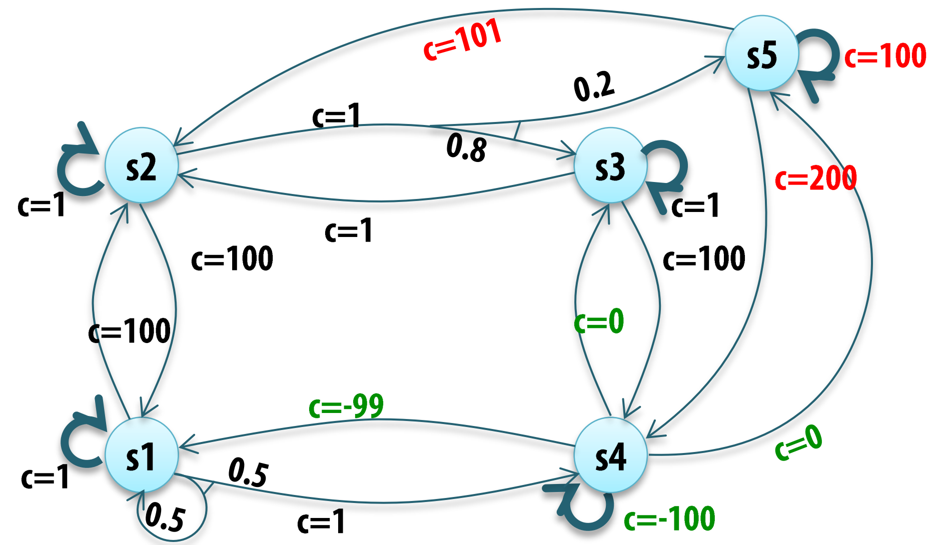
Policy Iteration 3: Expected Costs for π_1

- Calculate expected costs for the **current** policy π_1
  - Simple: Chosen transitions are deterministic + return to the same state!
    - $E_\pi(s) = C(s, \pi(s)) + \gamma \sum_{s' \in S} P(s, \pi(s), s')\ E_\pi(s')$
    - $E_{\pi 1}(s1) = C(s1, wait) + \gamma\, E_{\pi 1}(s1) = 1 + 0.9\, E_{\pi 1}(s1)$
    - $E_{\pi 1}(s2) = C(s2, wait) + \gamma\, E_{\pi 1}(s2) = 1 + 0.9\, E_{\pi 1}(s2)$
    - $E_{\pi 1}(s3) = C(s3, wait) + \gamma\, E_{\pi 1}(s3) = 1 + 0.9\, E_{\pi 1}(s3)$
    - $E_{\pi 1}(s4) = C(s4, wait) + \gamma\, E_{\pi 1}(s4) = -100 + 0.9\, E_{\pi 1}(s4)$
    - $E_{\pi 1}(s5) = C(s5, wait) + \gamma\, E_{\pi 1}(s5) = 100 + 0.9\, E_{\pi 1}(s5)$
  - Simple equations to solve:
    - $0.1E_{\pi 1}(s1) = 1 \rightarrow E_{\pi 1}(s1) = 10$
    - $0.1E_{\pi 1}(s2) = 1 \rightarrow E_{\pi 1}(s2) = 10$
    - $0.1E_{\pi 1}(s3) = 1 \rightarrow E_{\pi 1}(s3) = 10$
    - $0.1E_{\pi 1}(s4) = -100 \rightarrow E_{\pi 1}(s4) = -1000$
    - $0.1E_{\pi 1}(s5) = 100 \rightarrow E_{\pi 1}(s5) = 1000$

Given this policy π_1:
High costs if we start in s5,
high rewards if we start in s4

Policy Iteration 4: Update 1a

What is the best **local** modification according to the **expected cost** of the **current** policy?	$E_{\pi1}(s1) = 10$ $E_{\pi1}(s2) = 10$ $E_{\pi1}(s3) = 10$ $E_{\pi1}(s4) = -1000$ $E_{\pi1}(s5) = 1000$



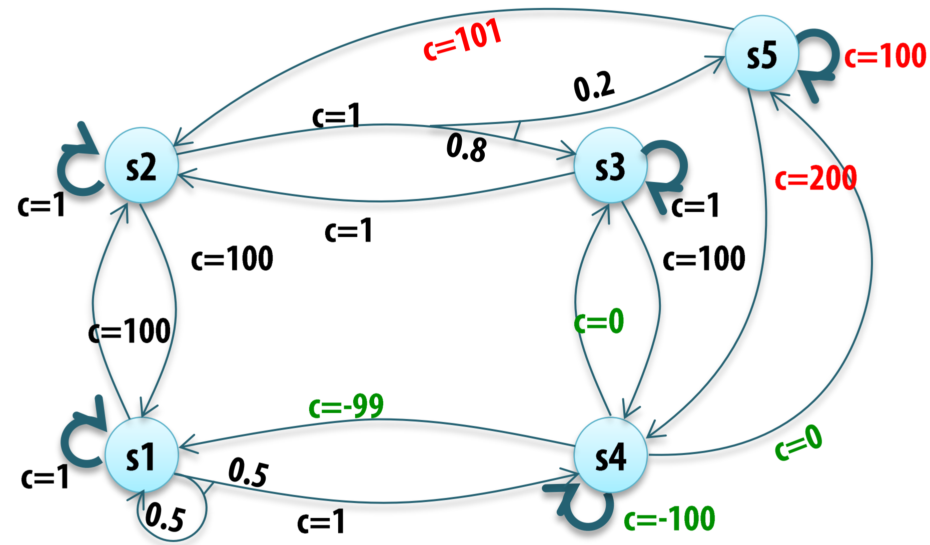
- For every state *s*:
  - Let $\pi_2(s) = \text{argmin}_{a \in A}\, Q_{\pi 1}(s,a)$
  - That is, find the action *a* that minimizes $C(s, a) + \gamma \sum_{s' \in S} P(s, a, s')\ E_{\pi 1}(s')$
    - s1: wait — 1 + 0.9 * 10 = 10
    - move(l1,l2) — 100 + 0.9 * 10 = 109
    - move(l1,l4) — 1 + 0.9 * (0.5*10 + 0.5*-1000) = -444,5

Seems best – chosen!

  - These are not the **true** expected costs for starting in state s1!
    - They are only correct if we locally change the **first** action to execute and then *go on to use the previous policy* (in this case, always waiting)!
    - But they can be proven to yield good guidance, as long as you apply the improvements repeatedly (as policy iteration does)

Policy Iteration 5: Update 1b

What is the best <u>local</u> modification according to the <u>expected cost</u> of the <u>current</u> policy?	$E_{\pi 1}(s1) = 10$ $E_{\pi 1}(s2) = 10$ $E_{\pi 1}(s3) = 10$ $E_{\pi 1}(s4) = -1000$ $E_{\pi 1}(s5) = 1000$



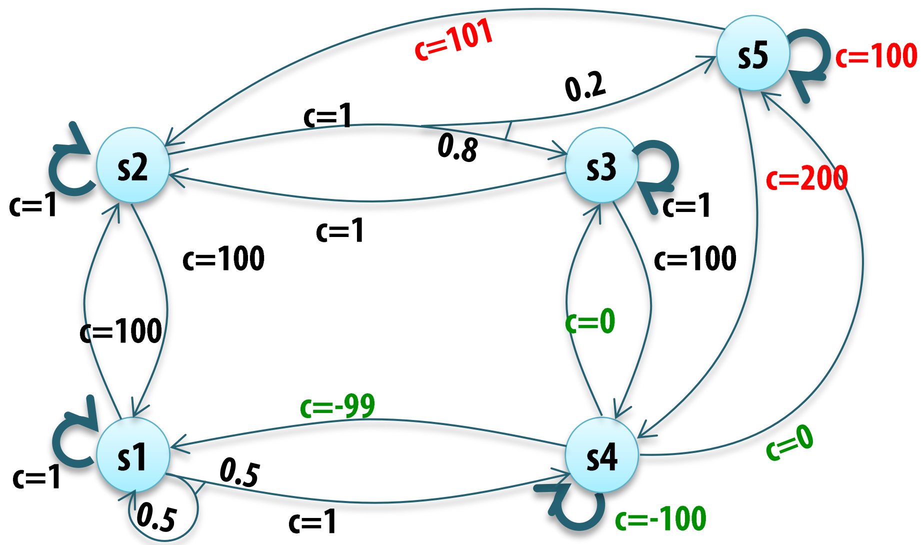
- For every state s:
  - Let $\pi_2(s) = \text{argmin}_{a \in A}\, Q_{\pi 1}(s,a)$
  - That is, find the action a that minimizes $C(s, a) + \gamma \sum_{s' \in S} P(s, a, s')\ E_{\pi 1}(s')$
    - s2: wait $\quad 1 + 0.9 * 10 \quad = 10$

      move(l2,l1) $\quad 100 + 0.9 * 10 \quad = 109$

      move(l2,l3) $\quad 1 + 0.9 * (0.8*10 + 0.2*1000) \quad = 188,2$

Policy Iteration 6: Update 1c

What is the best **local** modification according to the **expected cost** of the **current** policy?	$E_{\pi 1}(s1) = 10$ $E_{\pi 1}(s2) = 10$ $E_{\pi 1}(s3) = 10$ $E_{\pi 1}(s4) = -1000$ $E_{\pi 1}(s5) = 1000$



- For every state s:
  - Let $\pi_2(s) = \text{argmin}_{a \in A}\, Q_{\pi 1}(s,a)$
  - That is, find the action a that minimizes $C(s, a) + \gamma \sum_{s' \in S} P(s, a, s')\ E_{\pi 1}(s')$
    - s3: wait — $1 + 0.9 * 10 = 10$
      move(l3,l2) — $1 + 0.9 * 10 = 10$
      move(l3,l4) — $100 + 0.9 * -1000 = -800$
    - s4: **wait** — $-100 + 0.9 * -1000 = -1000$
      move(l4,l1) — $-99 + 0.9 * 10 = -90$
      …
    - s5: wait — $100 + 0.9 * 1000 = 1000$
      move(l5,l2) — $101 + 0.9 * 10 = 110$
      move(l5,l4) — $200 + 0.9 * -1000 = -700$

Policy Iteration 7: Second Policy

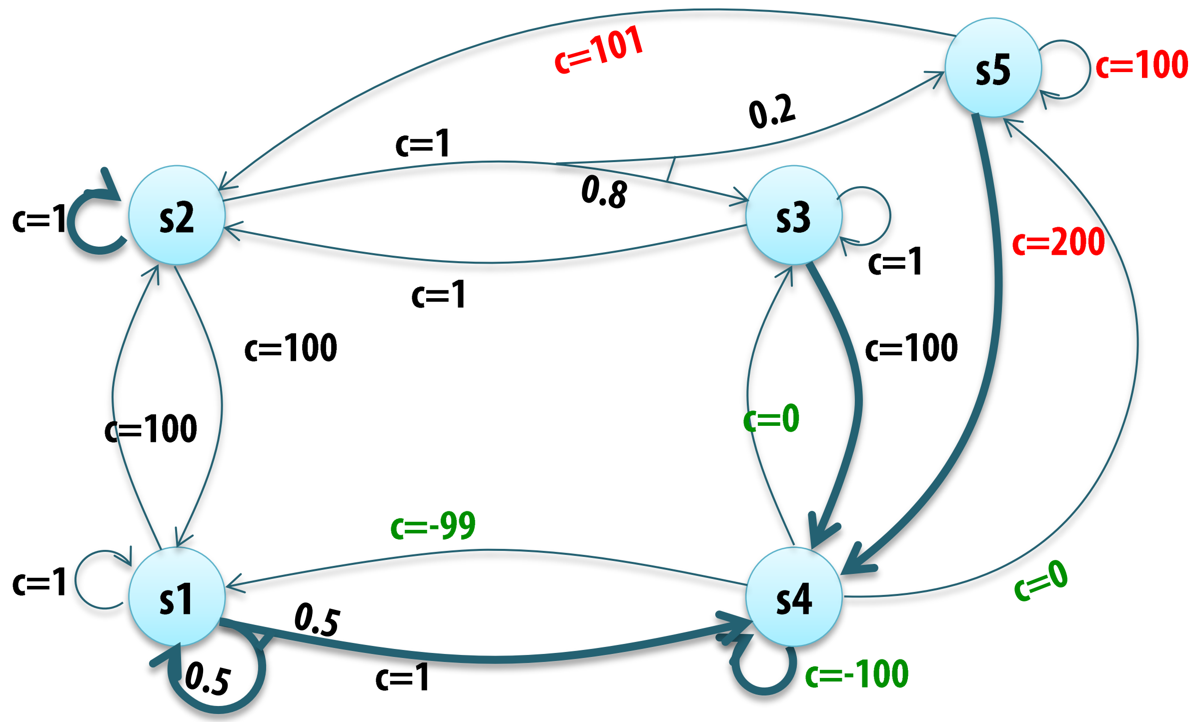
- This results in a **new policy**

π_1 = {(s1, wait), (s2, wait), (s3, wait), (s4, wait), (s5, wait)}	$E_{\pi 1}(s1) = 10$ $E_{\pi 1}(s2) = 10$ $E_{\pi 1}(s3) = 10$ $E_{\pi 1}(s4) = -1000$ $E_{\pi 1}(s5) = 1000$	π_2 = { (s1, move(l1,l4)), (s2, wait), (s3, move(l3,l4)), (s4, wait), (s5, move(l5,l4))}	<= **-444,5** <= 10 <= -800 <= -1000 <= -700	Costs based on one modified action + following π_1 (no increase!)

Now we have made use of earlier indications that s4 seems to be a good place

➔ Try to go there from s1 / s3 / s5!

No change in s2 yet…



Policy Iteration 8: Expected Costs for π_2

- Calculate **true** expected costs for the **new** policy π_2
  - $E_{\pi2}(s1) = C(s1, move(l1,l4)) + \gamma \ldots = 1 + 0.9\,(0.5E_{\pi2}(s1) + 0.5E_{\pi2}(s4))$
  - $E_{\pi2}(s2) = C(s2, wait) + \gamma E_{\pi2}(s2) = 1 + 0.9\,E_{\pi2}(s2)$
  - $E_{\pi2}(s3) = C(s3, move(l3,l4)) + \gamma E_{\pi2}(s4) = 100 + 0.9\,E_{\pi2}(s4)$
  - $E_{\pi2}(s4) = C(s4, wait) + \gamma E_{\pi2}(s4) = -100 + 0.9\,E_{\pi2}(s4)$
  - $E_{\pi2}(s5) = C(s5, move(l5,l4)) + \gamma E_{\pi2}(s4) = 200 + 0.9\,E_{\pi2}(s4)$
- Equations to solve:
  - $0.1E_{\pi2}(s2) = 1$ ➔ $E_{\pi2}(s2) = 10$
  - $0.1E_{\pi2}(s4) = -100$ ➔ $E_{\pi2}(s4) = -1000$
  - $E_{\pi2}(s3) = 100 + 0.9E_{\pi2}(s4) = 100 + 0.9*-1000 = -800$ ➔ $E_{\pi2}(s3) = -800$
  - $E_{\pi2}(s5) = 200 + 0.9E_{\pi2}(s4) = 200 + 0.9*-1000 = -700$ ➔ $E_{\pi2}(s5) = -700$
  - $E_{\pi2}(s1) = 1 + 0.45 * E_{\pi2}(s1) + 0.45 * E_{\pi2}(s4)$ ➔ $E_{\pi2}(s1) = -816,36$
    $0.55\, E_{\pi2}(s1) = 1 + 0.45 * E_{\pi2}(s4)$ ➔
    $0.55\, E_{\pi2}(s1) = 1 + (-450)$ ➔
    $0.55\, E_{\pi2}(s1) = -449$ ➔
    $E_{\pi2}(s1) = -816,3636\ldots$

π_2 = {(s1, move(l1,l4)),
(s2, wait),
(s3, move(l3,l4)),
(s4, wait),
(s5, move(l5,l4))}

Policy Iteration 9: Second Policy

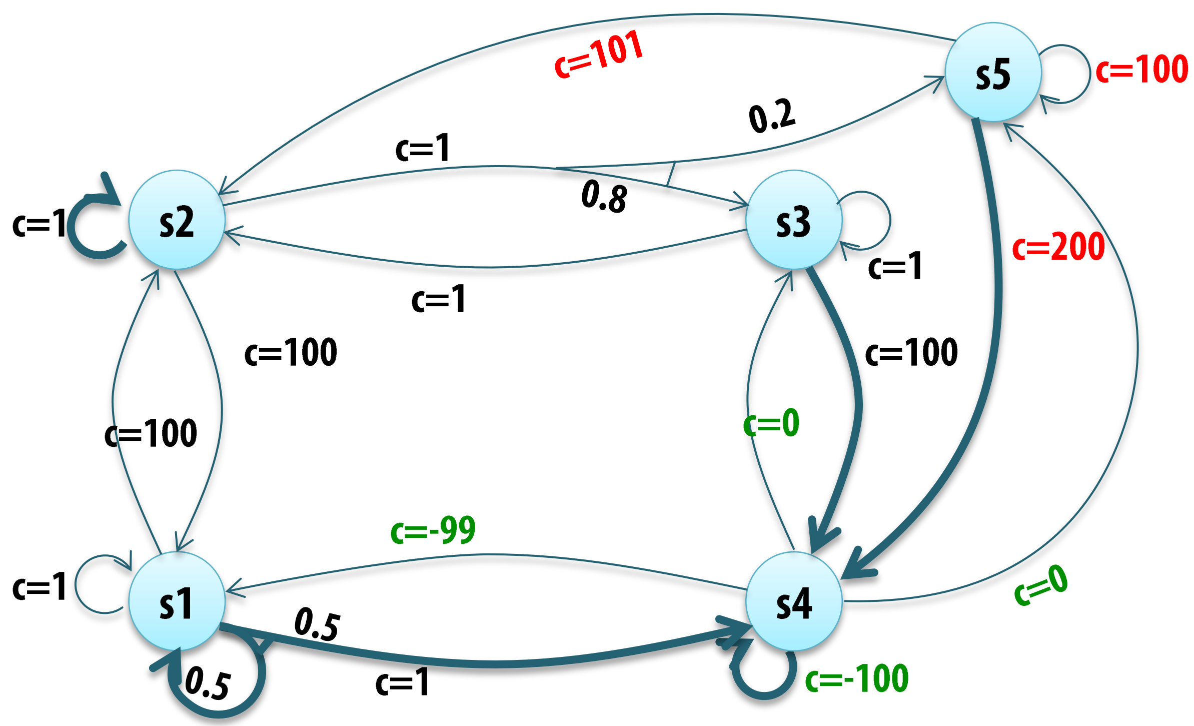
- Now we have the **true** expected costs of the second policy…

π_1 = {(s1, wait), (s2, wait), (s3, wait), (s4, wait), (s5, wait)}	$E_{\pi 1}(s1) = 10$ $E_{\pi 1}(s2) = 10$ $E_{\pi 1}(s3) = 10$ $E_{\pi 1}(s4) = -1000$ $E_{\pi 1}(s5) = 1000$	π_2 = { (s1, move(l1,l4), (s2, wait), (s3, move(l3,l4)), (s4, wait), (s5, move(l5,l4))}	<= ~~-444,5~~ <= 10 <= -800 <= -1000 <= -700	$E_{\pi 2}(s1)$ = **-816,36** $E_{\pi 2}(s2) = 10$ $E_{\pi 2}(s3) = -800$ $E_{\pi 2}(s4) = -1000$ $E_{\pi 2}(s5) = -700$

S5 wasn't so bad after all,
since you can reach s4
in a single step!

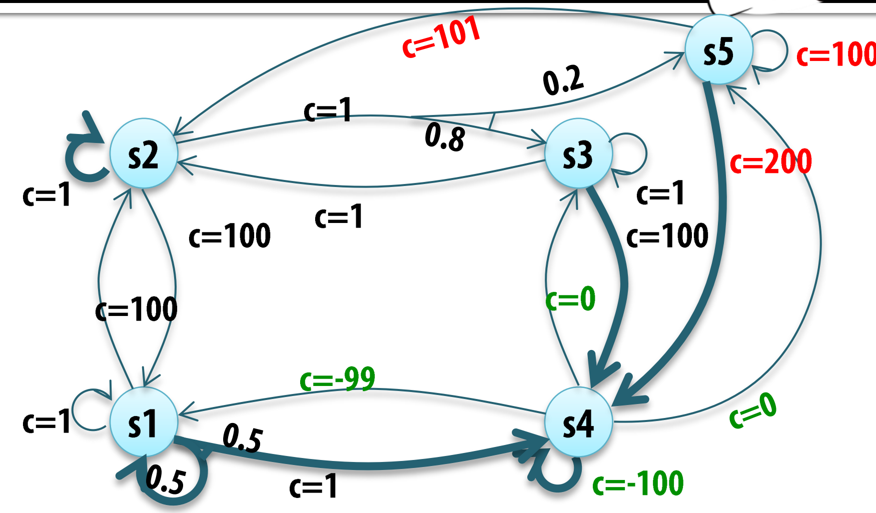
S1 / s3 are even better.

S2 seems much worse
in comparison,
since the benefits of s4
haven't "propagated" that far.



Policy Iteration 10: Update 2a

What is the best **local** modification according to the **expected cost** of the **current** policy?	$E_{\pi 2}(s1) = -816{,}36$ $E_{\pi 2}(s2) = 10$ $E_{\pi 2}(s3) = -800$ $E_{\pi 2}(s4) = -1000$ $E_{\pi 2}(s5) = -700$

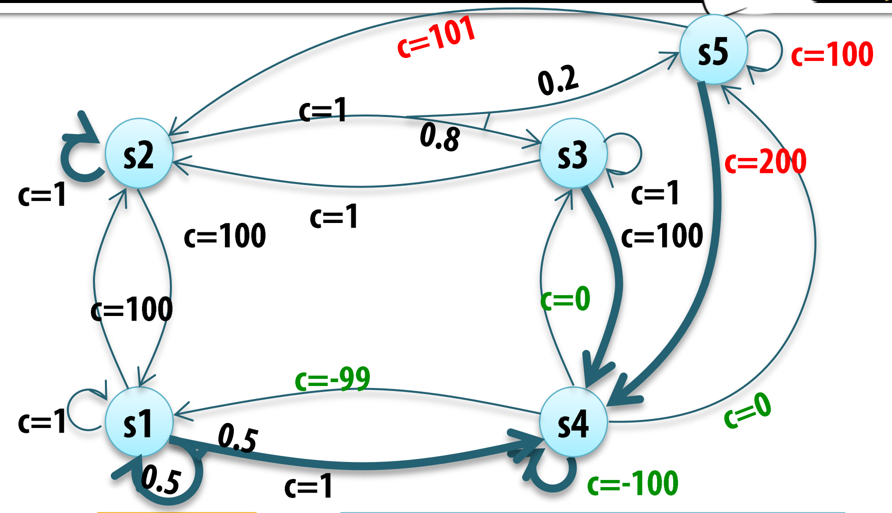


- For every state *s*:
  - Let $\pi_3(s) = \mathrm{argmin}_{a \in A}\, Q_{\pi 2}(s,a)$
  - That is, find the action *a* that minimizes $C(s, a) + \gamma \sum_{s' \in S} P(s, a, s')\; E_{\pi 2}(s')$
    - s1: wait — $1 + 0.9 * -816{,}36 = -733{,}72$
      move(l1,l2) — $100 + 0.9 * 10 = 109$
      move(l1,l4) — $1 + 0.9 * (.5*-1000+.5*-816.36) = -816{,}36$
      Seems best – chosen!
    - s2: wait — $1 + 0.9 * 10 = 10$
      move(l2,l1) — $100 + 0.9 * -816{,}36 = -634{,}72$
      move(l2,l3) — $1 + 0.9 * (0.8*-800 + 0.2*-700) = -701$

Now we will change the action taken at s2,
since we have better expected costs for s1, s3, s5…

Policy Iteration 11: Update 2b

What is the best **local** modification according to the **expected cost** of the **current** policy?	$E_{\pi 2}(s1) = -816{,}36$ $E_{\pi 2}(s2) = 10$ $E_{\pi 2}(s3) = -800$ $E_{\pi 2}(s4) = -1000$ $E_{\pi 2}(s5) = -700$



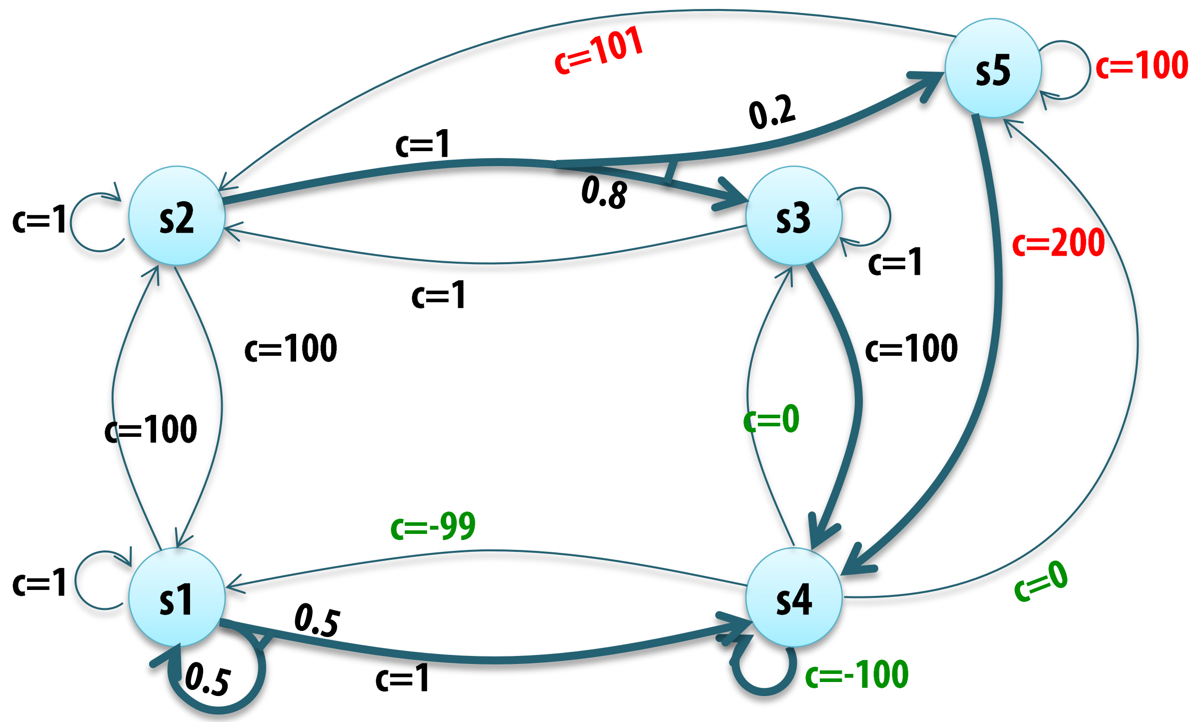
- For every state *s*:
  - Let $\pi_3(s) = \text{argmin}_{a \in A}\, Q_{\pi 2}(s,a)$
  - That is, find the action *a* that minimizes $C(s, a) + \gamma \sum_{s' \in S} P(s, a, s')\ E_{\pi 1}(s')$
    - s3: wait / move(l3,l2) / move(l3,l4)
    - s4: wait / move(l4,l1) / …
    - s5: wait / move(l5,l2) / move(l5,l4)

s3:	wait	1 + 0.9 *	-800	= -719
	move(l3,l2)	1 + 0.9 *	10	= 10
	move(l3,l4)	100 + 0.9 *	-1000	= -800
s4:	**wait**	-100 + 0.9 *	-1000	= -1000
	move(l4,l1)	-99 + 0.9 *	-816,36	= -833,72
	…			
s5:	wait	100 + 0.9 *	-700	= -530
	move(l5,l2)	101 + 0.9 *	10	= 110
	move(l5,l4)	200 + 0.9 *	-1000	= -700

Policy Iteration 12: Third Policy

- This results in a **new policy** π_3
  - **True expected costs** are updated by solving an equation system
  - The algorithm will iterate once more
  - No changes will be made to the policy
  - ➔ Termination with optimal policy!

π_3 = {(s1, move(l1,l4),
(s2, move(l2,l3)),
(s3, move(l3,l4)),
(s4, wait),
(s5, move(l5,l4))}



Policy Iteration 13: Algorithm

- **Policy iteration** is a way to find an optimal policy π^*
  - Start with an **arbitrary** initial policy π_1. Then, for i = 1, 2, …
    - Compute expected costs $E\pi_i(s)$ for every s by **solving a system of equations**

Find costs according to current policy

- System: For all s, $E\pi_i(s) = C(s, \pi_i(s)) + \gamma \sum_{s' \in S} P(s, \pi_i(s), s') E\pi_i(s')$
- Result: The expected cost of the "current" policy in **any** given state s
- Not a simple recursive calculation – the state graph is generally cyclic!

    - Compute an improved policy π_{i+1} "locally" for every s

Find best policy according to current costs

- $\pi_{i+1}(s) := \text{argmin}_{a \in A}\, C(s, a) + \gamma \sum_{s' \in S} P(s, a, s') E\pi_i(s')$
- Tells us the best action in **any** given state s given **current** expected costs
- But this is a new policy – with **new** expected costs!
- Loop back and calculate **those** costs

    - If $\pi_{i+1} = \pi_i$ then exit
      - We have found an optimal solution – cannot be improved anywhere
      - Otherwise, loop and calculate the expected cost for π_{i+1}, etc.

Convergence

- Converges in a finite number of iterations!
  - We change which action to execute
    if this **improves expected cost** for this state
    - This can sometimes decrease,
      and **never increase**, the cost of other states!
    - So costs are **monotonically improving**
      and we only have to consider a finite number of policies

- In general:
  - May take **many** iterations
  - Each iteration involves can be slow
  - Partly because of the need to **solve a large equation system**!

Finding a Solution: Value Iteration

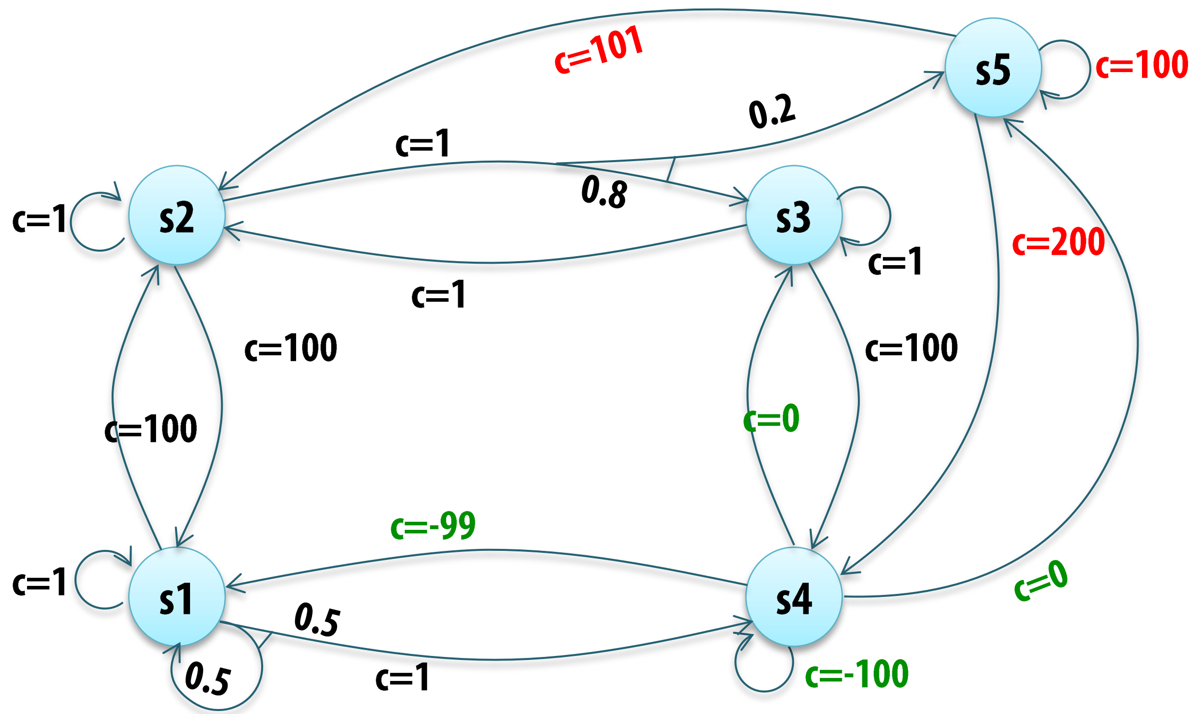
Value Iteration

- Second algorithm: **Value iteration**
  - An **intuitive** explanation:
    - Start by considering the minimum cost of proceeding **zero** steps
      - $E_0(s) = 0$ for every state
    - Then consider the reward we can get in **one** step
      - For each state s, create $E_1(s)$ using values of E_0 as a basis
    - …
    - Then consider the reward we can get in **n** steps
      - For each state s, create $E_n(s)$ using values of E_{n-1} as a basis
  - No need to solve an expensive equation system
    - Only local calculations using the **previous estimate**
    - The policy is implicit in the calculations
  - Will always converge towards an **optimal value function**
    - Will converge **faster** if $E_0(s)$ is close to the true value function
    - **Will** actually converge regardless of the initial value of $E_0(s)$
      - Intuition: As n goes to infinity, the importance of $E_0(s)$ goes to zero

Value Iteration 2: Initial Guess E0

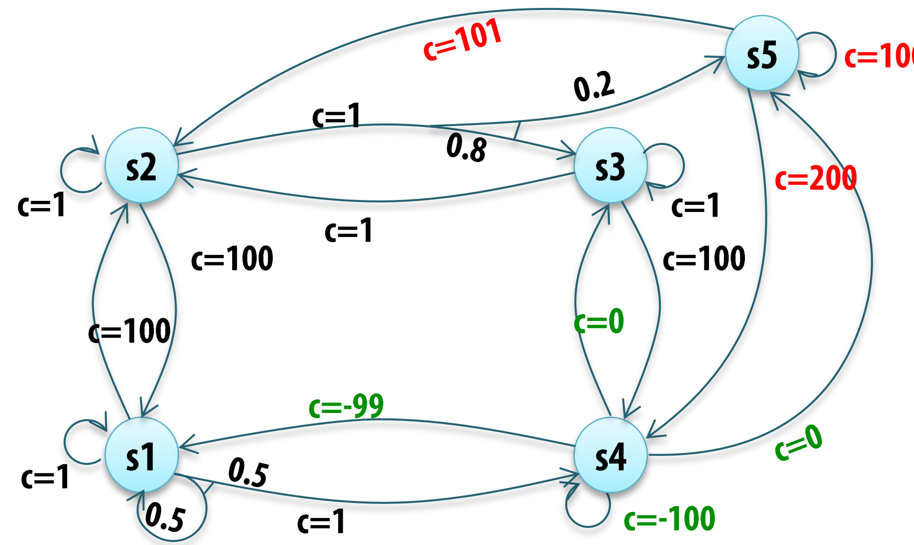
- Value iteration requires an **initial approximation**
  - Let's start with $E_0(s) = 0$ for each s
  - *Does not correspond to any actual policy!*
    - Does correspond to the optimal expected cost of executing zero steps…

E0(s1) = 0
E0(s2) = 0
E0(s3) = 0
E0(s4) = 0
E0(s5) = 0



Value Iteration 3: Update 1a

What is the best <u>local</u> modification according to the <u>current approximation</u>?	E0(s1) = 0 E0(s2) = 0 E0(s3) = 0 E0(s4) = 0 E0(s5) = 0



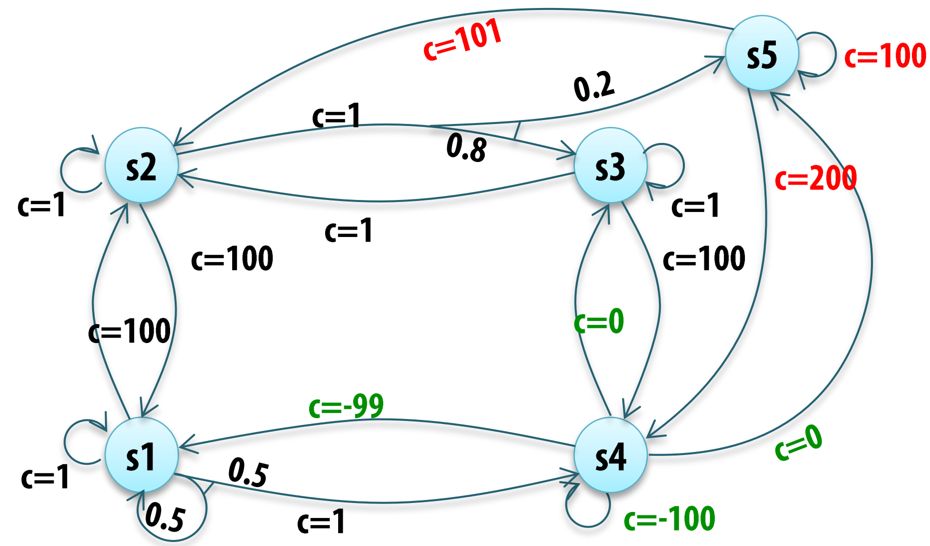
- For every state *s*:
  - **<u>PI</u>**: find the action *a* that minimizes $C(s, a) + \gamma \sum_{s' \in S} P(s, a, s')\ E_{\pi 1}(s')$
  - **<u>FI</u>**: find the action *a* that minimizes $C(s, a) + \gamma \sum_{s' \in S} P(s, a, s')\ E_0(s')$
    - s1:

s1:	wait	1 + 0.9 * 0	= 1
	move(l1,l2)	100 + 0.9 * 0	= 100
	move(l1,l4)	1 + 0.9 * (0.5*0 + 0.5*0)	= 1
s2:	wait	1 + 0.9 * 0	= 1
	move(l2,l1)	100 + 0.9 * 0	= 100
	move(l2,l3)	1 + 0.9 * (0.8*0 + 0.2*0)	= 1

Value Iteration 4: Update 1b

What is the best **local** modification according to the **current approximation**?

E0(s1) = 0
E0(s2) = 0
E0(s3) = 0
E0(s4) = 0
E0(s5) = 0



- For every state *s*:
  - FI: find the action *a* that minimizes $C(s, a) + \gamma \sum_{s' \in S} P(s, a, s') E_o(s')$
    - s3:
      - wait: 1 + 0.9 * 0 = 1
      - move(l3,l2): 1 + 0.9 * 0 = 1
      - move(l3,l4): 100 + 0.9 * 0 = 100
    - s4:
      - wait: -100 + 0.9 * 0 = -100
      - move(l4,l1): -99 + 0.9 * 0 = -99
      - …
    - s5:
      - wait: 100 + 0.9 * 0 = 100
      - move(l5,l2): 101 + 0.9 * 0 = 101
      - move(l5,l4): 200 + 0.9 * 0 = 200

Value Iteration 5: Second Policy

- This results in a **new approximation** of the lowest expected cost

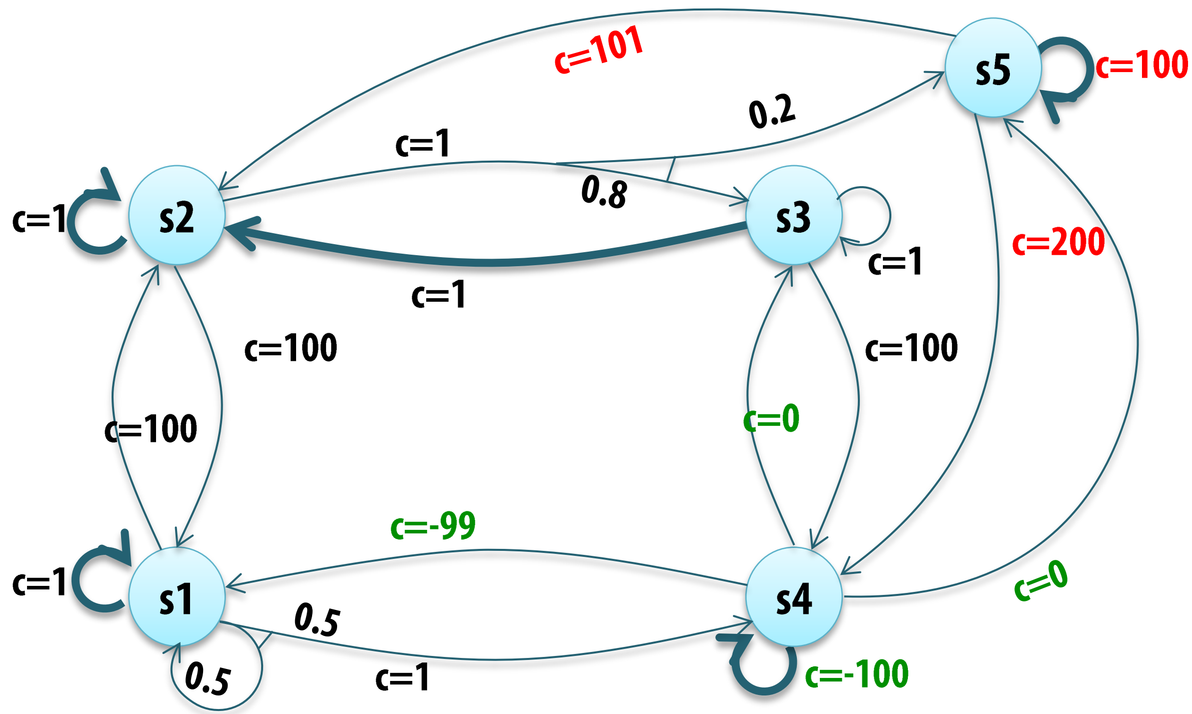
E0	π_1	E1
E0(s1) = 0	π_1 = { (s1, wait),	E1(s1) = 1
E0(s2) = 0	(s2, wait),	E1(s2) = 1
E0(s3) = 0	(s3, move(l3,l2)),	E1(s3) = 1
E0(s4) = 0	(s4, wait),	E1(s4) = -100
E0(s5) = 0	(s5, wait)}	E1(s5) = 100

For infinite execution,
$E\pi_1(s1) = 10$,
but this is not calculated…

E1 corresponds to **one step** of many polices, including the one shown here

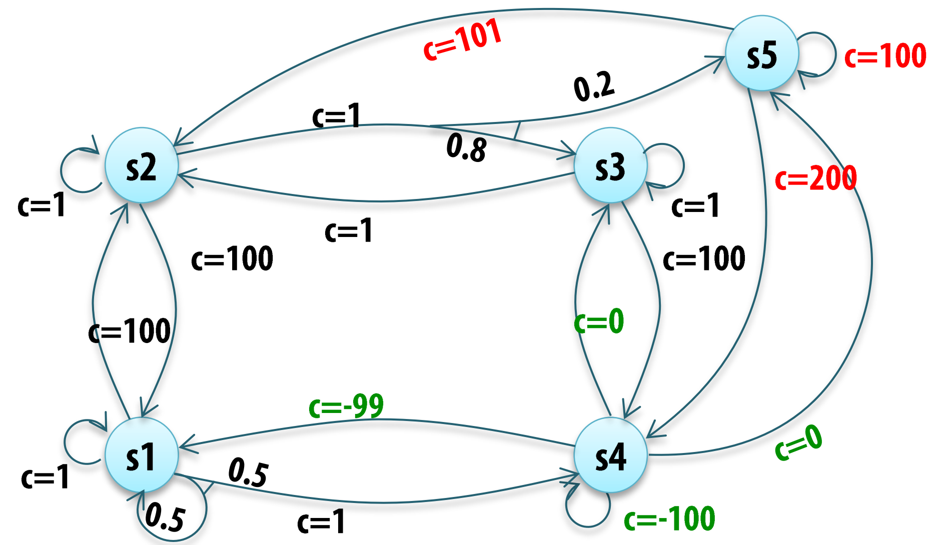
Policy iteration would now calculate the **true** expected cost for a chosen policy

Value iteration instead continues using E1, which is only a calculation guideline, not the true cost of **any** policy



Value Iteration 6: Update 2a

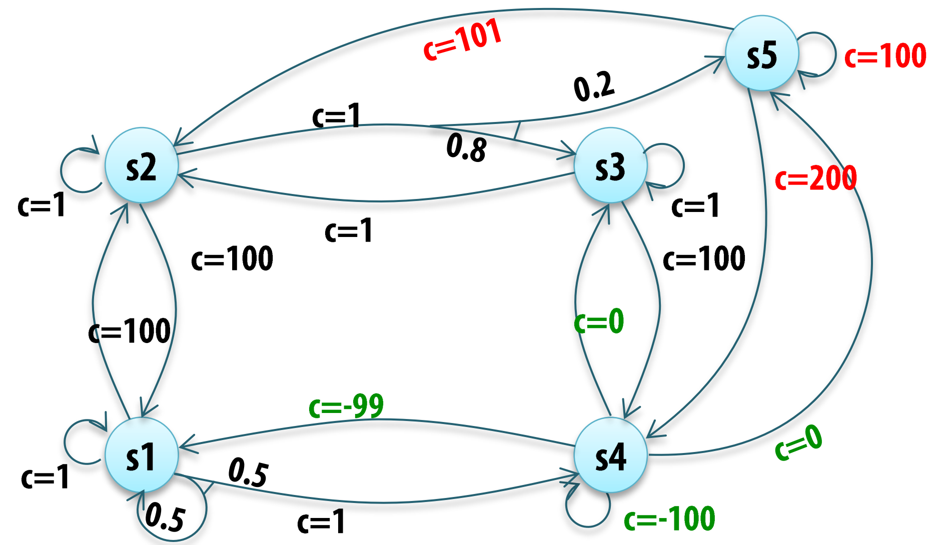
What is the best <u>local</u> modification according to the <u>current</u> <u>approximation</u>?	E1(s1) = 1 E1(s2) = 1 E1(s3) = 1 E1(s4) = -100 E1(s5) = 100



- For every state *s*:
  - PI: find the action *a* that minimizes $C(s, a) + \gamma \sum_{s' \in S} P(s, a, s')\ E_{\pi k}(s')$
  - FI: find the action *a* that minimizes $C(s, a) + \gamma \sum_{s' \in S} P(s, a, s')\ E_{k-1}(s')$
    - s1: wait — 1 + 0.9 * 1 = 1.9
      - move(l1,l2) — 100 + 0.9 * 1 = 100.9
      - move(l1,l4) — 1 + 0.9 * (0.5*1 + 0.5*-100) = -43,55
    - s2: wait — 1 + 0.9 * 1 = 1.9
      - move(l2,l1) — 100 + 0.9 * 1 = 100.9
      - move(l2,l3) — 1 + 0.9 * (0.8*1 + 0.2*1) = 1.9

Value Iteration 7: Update 2b

What is the best <u>local</u> modification according to the <u>current approximation</u>?	E1(s1) = 1 E1(s2) = 1 E1(s3) = 1 E1(s4) = -100 E1(s5) = 100



- For every state *s*:
  - FI: find the action *a* that minimizes $C(s, a) + \gamma \sum_{s' \in S} P(s, a, s')\, E_{k-1}(s')$
    - s3:

s3:	wait	1	+ 0.9 * 1	= 1.9
	move(l3,l2)	1	+ 0.9 * 1	= 1.9
	move(l3,l4)	100	+ 0.9 * -100	= 10
s4:	wait	-100	+ 0.9 * -100	= -190
	move(l4,l1)	-99	+ 0.9 * 1	= -98.1
	…			
s5:	wait	100	+ 0.9 * 1	= 100.9
	move(l5,l2)	101	+ 0.9 * 1	= 101.9
	move(l5,l4)	200	+ 0.9 * -100	= 110

Value Iteration 8: Second Policy

- This results in another **new approximation**

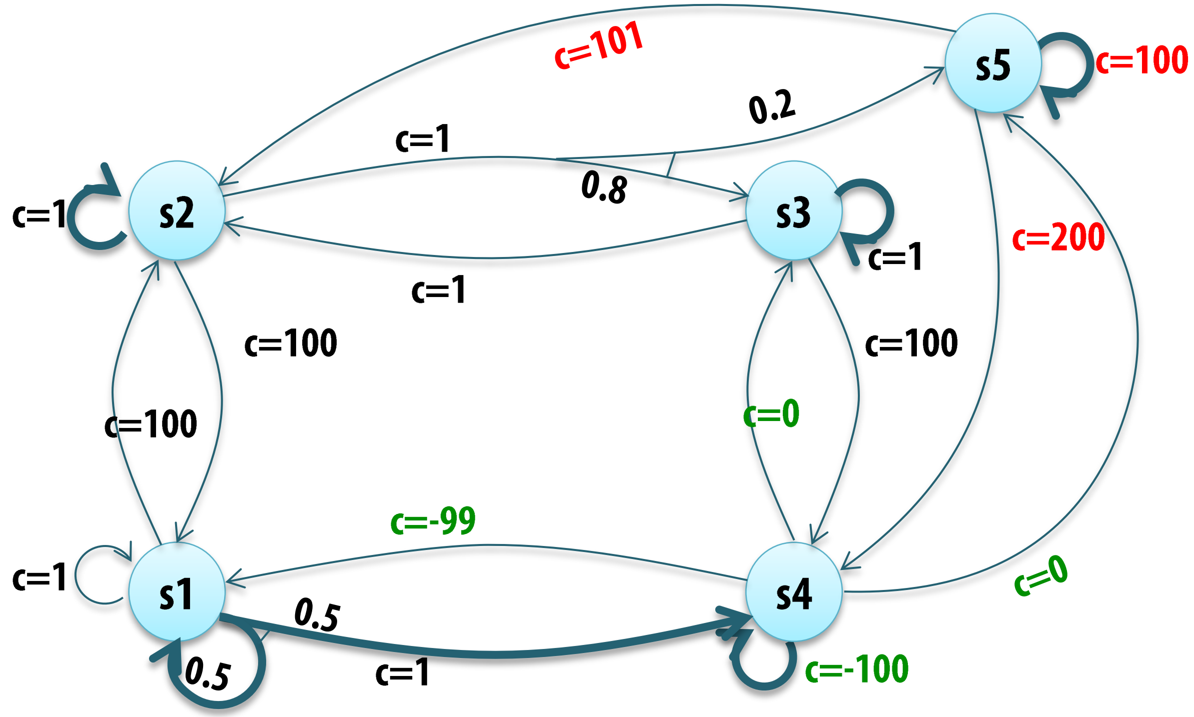
E0(s1) = 0 E0(s2) = 0 E0(s3) = 0 E0(s4) = 0 E0(s5) = 0	π_1 = { (s1, wait), (s2, wait), (s3, move(l3,l2)), (s4, wait), (s5, wait)}	E1(s1) = 1 E1(s2) = 1 E1(s3) = 1 E1(s4) = -100 E1(s5) = 100	π_2 = { (s1, move(l1,l4)), (s2, wait), (s3, wait), (s4, wait), (s5, wait)}	E2(s1) = -43.55 E2(s2) = 1.9 E2(s3) = 1.9 E2(s4) = -190 E2(s5) = 100.9

Again, E2 doesn't represent the true expected cost of π_2

Nor is it the true expected cost of executing two steps of E2

It is the true expected cost of one step of E2, then one of E1!

(But it **will converge** towards true costs…)



Differences

- **Significant differences** from policy iteration
  - Less accurate basis for action selection
    - Based on **approximate costs**, not true expected costs
  - Policy does not necessarily change in each iteration
    - May first have to iterate *n* times, incrementally improving cost approximations
    - **Then** another action suddenly seems better in some state
  - ➔ Requires a larger number of iterations
    - But each iteration is cheaper
  - ➔ Can't terminate just because the policy does not change
    - Need another termination condition…

Illustration

- Illustration below, showing **<u>rewards</u>**
  - Notice that we already calculated rows 1 and 2
    - s1: wait — 1 + 0.9 * 1 = 1.9
      - move(l1,l2) — 100 + 0.9 * 1 = 100.9
      - move(l1,l4) — 1 + 0.9 * (0.5*1 + 0.5*-100) = -43,55

	s1			s2			s3			s4	s5		
Action	wait	move-s2	move-s4	wait	move-s1	move-s3	wait	move-s2	move-s4	wait	wait	move-s2	move-s4
	0	0	0	0	0	0	0	0	0	0	0	0	0
1	-1	-100	-1	-1	-100	-1	-1	-1	-100	100	-100	-101	-200
2	-1,9	-100,9	43,55	-1,9	-100,9	-1,9	-1,9	-1,9	-10	190	-190	-101,9	-110
3	38,195	-101,71	104,098	-2,71	-60,805	-2,71	-2,71	-2,71	71	271	-191,71	-102,71	-29
4	92,6878	-102,439	167,794	-3,439	-6,31225	62,9	62,9	-3,439	143,9	343,9	-126,1	-103,439	43,9
5	150,014	-43,39	229,262	55,61	51,0145	128,51	128,51	55,61	209,51	409,51	-60,49	-44,39	109,51
5	205,336	15,659	286,448	114,659	106,336	187,559	187,559	114,659	268,559	468,559	-1,441	14,659	168,559
6	256,803	68,8031	338,753	167,803	157,803	240,703	240,703	167,803	321,703	521,703	51,7031	67,8031	221,703
7	303,878	116,633	386,205	215,633	204,878	288,533	288,533	215,633	369,533	569,533	99,5328	115,633	269,533
8	346,585	159,68	429,082	258,68	247,585	331,58	331,58	258,68	412,58	612,58	142,58	158,68	312,58
9	385,174	198,422	467,748	297,422	286,174	370,322	370,322	297,422	451,322	651,322	181,322	197,422	351,322
10	419,973	233,289	502,581	332,289	320,973	405,189	405,189	332,289	486,189	686,189	216,189	232,289	386,189
11	451,323	264,67	533,947	363,67	352,323	436,57	436,57	363,67	517,57	717,57	247,57	263,67	417,57
12	479,552	292,913	562,183	391,913	380,552	464,813	464,813	391,913	545,813	745,813	275,813	291,913	445,813
13	504,964	318,332	587,598	417,332	405,964	490,232	490,232	417,332	571,232	771,232	301,232	317,332	471,232
14	527,838	341,209	610,474	440,209	428,838	513,109	513,109	440,209	594,109	794,109	324,109	340,209	494,109

Illustration

- Remember, these are “pseudo-rewards”!

	s1			s2			s3			s4	s5		
Action	wait	move-s2	**move-s4**	wait	move-s1	**move-s3**	wait	move-s2	**move-s4**	**wait**	wait	move-s2	**move-s4**
	0	0	0	0	0	0	0	0	0	0	0	0	0
1	-1	-100	-1	-1	-100	-1	-1	-1	-100	100	-100	-101	-200
2	-1,9	-100,9	43,55	-1,9	-100,9	-1,9	-1,9	-1,9	-10	190	-190	-101,9	-110
3	38,195	-101,71	104,098	-2,71	-60,805	-2,71	-2,71	-2,71	71	271	-191,71	-102,71	-29
4	92,6878	-102,439	167,794	-3,439	-6,31225	62,9	62,9	-3,439	143,9	343,9	-126,1	-103,439	43,9
5	150,014	-43,39	229,262	55,61	51,0145	128,51	128,51	55,61	209,51	409,51	-60,49	-44,39	109,51
5	205,336	15,659	286,448	114,659	106,336	187,559	187,559	114,659	268,559	468,559	-1,441	14,659	168,559
6	256,803	68,8031	338,753	167,803	157,803	240,703	240,703	167,803	321,703	521,703	51,7031	67,8031	221,703
7	303,878	116,633	386,205	215,633	204,878	288,533	288,533	215,633	369,533	569,533	99,5328	115,633	269,533
8	346,585	159,68	429,082	258,68	247,585	331,58	331,58	258,68	412,58	612,58	142,58	158,68	312,58
9	385,174	198,422	467,748	297,422	286,174	370,322	370,322	297,422	451,322	651,322	181,322	197,422	351,322
10	419,973	233,289	502,581	332,289	320,973	405,189	405,189	332,289	486,189	686,189	216,189	232,289	386,189
11	451,323	264,67	533,947	363,67	352,323	436,57	436,57	363,67	517,57	717,57	247,57	263,67	417,57
12	479,552	292,913	562,183	391,913	380,552	464,813	464,813	391,913	545,813	745,813	275,813	291,913	445,813
13	504,964	318,332	587,598	417,332	405,964	490,232	490,232	417,332	571,232	771,232	301,232	317,332	471,232
14	527,838	341,209	610,474	440,209	428,838	513,109	513,109	440,209	594,109	794,109	324,109	340,209	494,109

324,109 = cost of waiting **once** in s5,
then continuing according to the **previous** 14 policies for 14 steps,
then **doing nothing** (which is impossible according to the model)

How Many Iterations?

- Illustration, only showing **best** reward at each step
  - We actually have the optimal policy after iteration 4
    - But we can't know this unless we calculate true expected costs as in policy iteration
  - Here we only see that the pseudo-expected costs continue changing…
    - Maybe at some point in the future, they will change enough to yield another policy?

Iteration	E(s1)	E(s2)	E(s3)	E(s4)	E(s5)
0	0	0	0	0	0
1	-1	-1	-1	100	-100
2	43,55	-1,9	-1,9	190	-110
3	104,098	-2,71	71	271	-29
4	167,794	62,9	143,9	343,9	43,9
5	229,262	128,51	209,51	409,51	109,51
6	286,448	187,559	268,559	468,559	168,559
7	338,753	240,703	321,703	521,703	221,703
8	386,205	288,533	369,533	569,533	269,533
9	429,082	331,58	412,58	612,58	312,58
10	467,748	370,322	451,322	651,322	351,322
11	502,581	405,189	486,189	686,189	386,189
12	533,947	436,57	517,57	717,57	417,57
13	562,183	464,813	545,813	745,813	445,813
14	587,598	490,232	571,232	771,232	471,232
15	610,474	513,109	594,109	794,109	494,109
16	631,062	533,698	614,698	814,698	514,698
17	649,592	552,228	633,228	833,228	533,228
18	666,269	568,905	649,905	849,905	549,905
19	681,279	583,915	664,915	864,915	564,915
20	694,787	597,423	678,423	878,423	578,423

Different Discount Factors

- Suppose discount factor is 0.99 instead
  - Much slower convergence
    - Change at step 20: 2% ➔ 5%
    - Change at step 50: 0.07% ➔ 1.63%

  - Care more about the future ➔ need to consider many more steps!

Iteration	s1	s2	s3	s4	s5
0	0	0	0	0	0
1	-1	-1	-1	100	-100
2	48,005	-1,99	-1	199	-101
3	121,267	-1,99	97,01	297,01	-2,99
4	206,047	95,0399	194,04	394,04	94,0399
5	296,043	191,1	290,1	490,1	190,1
6	388,141	286,199	385,199	585,199	285,199
7	480,803	380,347	479,347	679,347	379,347
8	573,274	473,553	572,553	772,553	472,553
9	665,184	565,828	664,828	864,828	564,828
10	756,356	657,179	756,179	956,179	656,179
11	846,705	747,617	846,617	1046,62	746,617
12	936,195	837,151	936,151	1136,15	836,151
13	1024,81	925,79	1024,79	1224,79	924,79
14	1112,55	1013,54	1112,54	1312,54	1012,54
15	1199,42	1100,42	1199,42	1399,42	1099,42
16	1285,42	1186,42	1285,42	1485,42	1185,42
17	1370,57	1271,57	1370,57	1570,57	1270,57
18	1454,86	1355,86	1454,86	1654,86	1354,86
19	1538,31	1439,31	1538,31	1738,31	1438,31
20	1620,93	1521,93	1620,93	1820,93	1520,93

How Many Iterations?

- We can find bounds!
  - Let M be the maximum change in pseudo-cost between two iterations
  - Then we can find a bound on how far from the optimal cost the current policy may be
  - Cost of current policy – cost of optimal policy <=
    M * (2*discount) / (1-discount)

Absolute cost difference M between two iterations	Discount factor 0,5	0,9	0,95	0,99	0,999
0,001	0,002	0,018	0,038	0,198	1,998
0,01	0,02	0,18	0,38	1,98	19,98
0,1	0,2	1,8	3,8	19,8	199,8
1	2	18	38	198	1998
5	10	90	190	990	9990
10	20	180	380	1980	19980
100	200	1800	3800	19800	199800

How Many Iterations? Discount 0.90

							Possible diff from optimal policy
Iteration	s1	s2	s3	s4	s5	Greatest change	
0	0	0	0	0	0		
1	-1	-1	-1	100	-100	100	1800
2	43,55	-1,9	-1,9	190	-110	90	1620
3	104,0975	-2,71	71	271	-29	81	1458
4	167,7939	62,9	143,9	343,9	43,9	72,9	1312,2
5	229,2622	128,51	209,51	409,51	109,51	65,61	1180,98
6	286,4475	187,559	268,559	468,559	168,559	59,049	1062,882
7	338,7529	240,7031	321,7031	521,7031	221,7031	53,1441	956,5938
8	386,2052	288,5328	369,5328	569,5328	269,5328	47,82969	860,9344
9	429,0821	331,5795	412,5795	612,5795	312,5795	43,04672	774,841
10	467,7477	370,3216	451,3216	651,3216	351,3216	38,74205	697,3569
20	694,787	597,4233	678,4233	878,4233	578,4233	13,50852	243,1533
30	773,9725	676,6088	757,6088	957,6088	657,6088	4,710129	84,78232
40	801,5828	704,2191	785,2191	985,2191	685,2191	1,64232	29,56177
50	811,2099	713,8462	794,8462	994,8462	694,8462	0,572642	10,30755
60	814,5666	717,203	798,203	998,203	698,203	0,199668	3,594021
70	815,7371	718,3734	799,3734	999,3734	699,3734	0,06962	1,253157
80	816,1452	718,7815	799,7815	999,7815	699,7815	0,024275	0,436949
90	816,2875	718,9238	799,9238	999,9238	699,9238	0,008464	0,152355
100	816,3371	718,9734	799,9734	999,9734	699,9734	0,002951	0,053123

Bounds are incrementally tightened!

Quit after 10 iterations ➔ policy **appears** to cost -467. **Guarantee**: <= -467 + 697.

Quit after 50 iterations ➔ policy **appears** to cost -811. **Guarantee**: <= -811 + 10.

How Many Iterations? Discount 0.99

Iteration	s1	s2	s3	s4	s5	Greatest change	Possible diff from optimal policy
0	0	0	0	0	0		
1	-1	-1	-1	100	-100	100	19800
10	756,356	657,179	756,179	956,179	656,179	91,3517	18087,6
20	1620,93	1521,93	1620,93	1820,93	1520,93	82,6169	16358,1
30	2403	2304	2403	2603	2303	74,7172	14794
50	3749,94	3650,94	3749,94	3949,94	3649,94	61,1117	12100,1
100	6139,68	6040,68	6139,68	6339,68	6039,68	36,973	7320,65
150	7585,48	7486,48	7585,48	7785,48	7485,48	22,3689	4429,04
200	8460,2	8361,2	8460,2	8660,2	8360,2	13,5333	2679,59
250	8989,41	8890,41	8989,41	9189,41	8889,41	8,18773	1621,17
300	9309,59	9210,59	9309,59	9509,59	9209,59	4,95363	980,818
400	9620,49	9521,49	9620,49	9820,49	9520,49	1,81319	359,011
500	9734,3	9635,3	9734,3	9934,3	9634,3	0,66369	131,41
600	9775,95	9676,95	9775,95	9975,95	9675,95	0,24293	48,1002
700	9791,2	9692,2	9791,2	9991,2	9691,2	0,08892	17,6062
800	9796,78	9697,78	9796,78	9996,78	9696,78	0,03255	6,44445
900	9798,82	9699,82	9798,82	9998,82	9698,82	0,01191	2,35888
1000	9799,57	9700,57	9799,57	9999,57	9699,57	0,00436	0,86342

Bounds are incrementally tightened!

Quit after 250 iterations ➔ policy **appears** to cost 8989. **Guarantee**: <= 8989+1621.

Quit after 600 iterations ➔ policy **appears** to cost 9775. **Guarantee**: <= 9775+48.

Value Iteration

- **Value iteration** to find π^*:
  - Start with an **arbitrary cost** $E_0(s)$ for each s and an arbitrary $\varepsilon > 0$
  - For $k = 1, 2, \ldots$
    - for each s in S do
      - for each a in A do $Q(s,a) := C(s,a) + \gamma \sum_{s' \in S} P_a(s' \mid s)\, \boxed{E_{k-1}(s')}$
      - $E_k(s) = \min_{a \in A} Q(s,a)$
      - $\pi(s) = \operatorname{argmin}_{a \in A} Q(s,a)$
    - If $\max_{s \in S} |E_k(s) - E_{k-1}(s)| < \varepsilon$ for every s then exit *// Almost no change!*

Almost as in the definition of Q(s,a), but we use the **previous** expected cost

- On an acyclic graph, the values converge in finitely many iterations
- On a cyclic graph, **value** convergence can take infinitely many iterations
- That's why $\varepsilon > 0$ is needed

Discussion

- Both algorithms converge in a polynomial number of iterations
  - But the variable in the polynomial is *the number of states*
    - The number of states is usually huge
  - Need to examine the entire state space in each iteration

- Thus, these algorithms take huge amounts of time and space
  - Probabilistic set-theoretic planning is EXPTIME-complete
    - Much harder than ordinary set-theoretic planning, which was only PSPACE-complete
    - Methods exist for reducing the search space, and for approximating optimal solutions
    - Beyond the scope of this course