Advanced Topics Data Science and AI

Automated Planning and Acting

Probabilistic Models

Tanya Braun

UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR INFORMATIONSSYSTEME

Content

1. Planning and Acting with **Deterministic** Models
2. Planning and Acting with **Refinement** Methods
3. Planning and Acting with **Temporal** Models
4. Planning and Acting with **Nondeterministic** Models
5. Making Simple Decisions
6. Planning and Acting with **Probabilistic** Models
   a. Markov Decision Process
   b. Stochastic Shortest-Path Problems
   c. Heuristic Search Algorithms
   d. Online Probabilistic Approaches
7. Making Complex Decisions
8. Provably Beneficial AI

- Other: open world, perceiving, learning
  - If time permits

Outline

Markov decision process (MDP)

- Markov property
- Sequence of actions, history, policy
- Value iteration, policy iteration

6.2 Stochastic shortest path problems

- Safe/unsafe policies
- Optimality
- Policy iteration, value iteration

6.3 Heuristic search algorithms

- Best-first search
- Determinisation

6.4 Online probabilistic planning

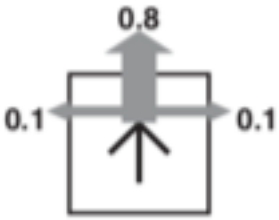
- Lookahead

Acknowledgements

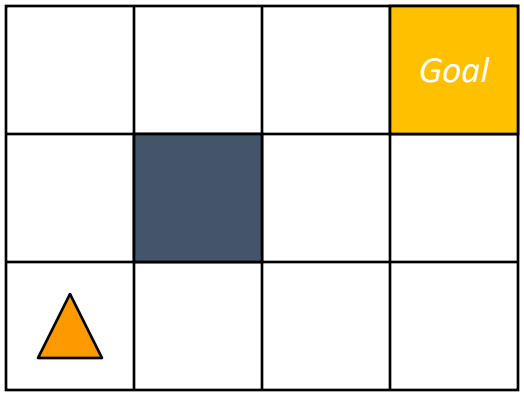
- Material from Lise Getoor, Jean-Claude Latombe, Daphne Koller, and Stuart Russell, Xiaoli Fern
- Compiled by Ralf Möller
- Based on Chapter 17 (up and including 17.3)

Simple Robot Navigation Problem

- In each state, the possible actions are U, D, R, and L
- The effect of U is as follows (transition model):
  - With probability 0.8, move up one square
    - If already in top row or blocked, no move
  - With probability 0.1, move right one square
    - If already in rightmost row or blocked, no move
  - With probability 0.1, move left one square
    - If already in leftmost row or blocked, no move



- Same transition model holds for D, R, and L and their respective directions



Markov Property

The transition properties depend only on the current state, not on previous history (how that state was reached).

- Also known as Markov-k with $k = 1$
  - $k \leq t$

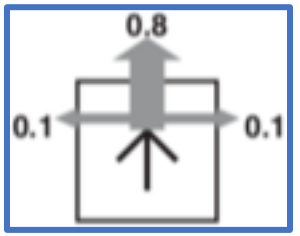
    $$P(x_{t+1} \mid x_t, \dots, x_0) = P(x_{t+1} \mid x_t, \dots, x_{t-k+1})$$

  - $k = 1$

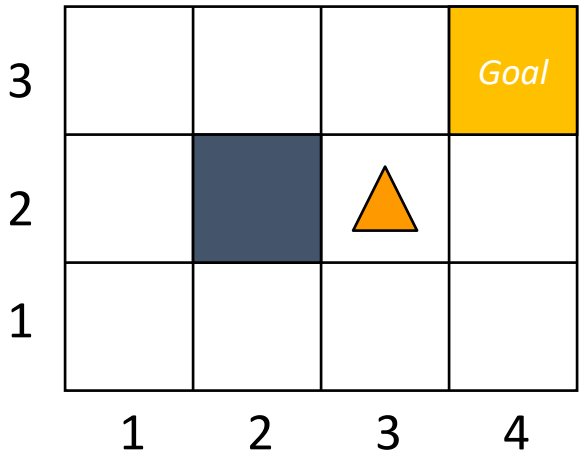
    $$P(x_{t+1} \mid x_t, \dots, x_0) = P(x_{t+1} \mid x_t)$$

Sequence of Actions

- In each state, the possible actions are U, D, R, and L; transition model:
- Current position: [3,2]
- Planned sequence of actions: (U, R)

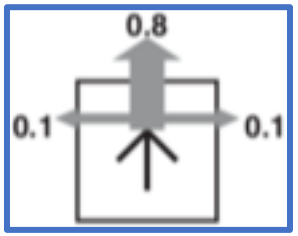


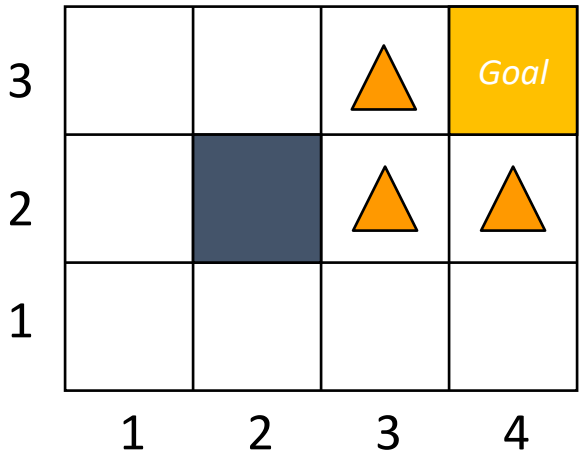
[3,2]

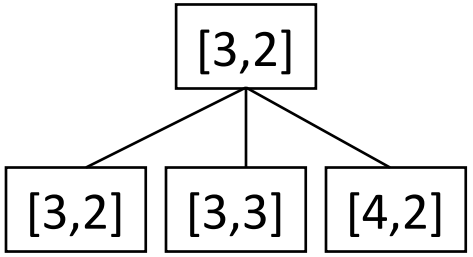


Sequence of Actions

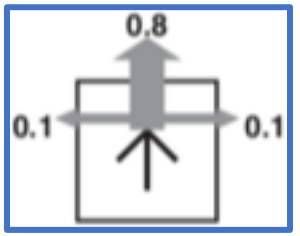
- In each state, the possible actions are U, D, R, and L; transition model:
- Current position: [3,2]
- Planned sequence of actions: (U, R)
  - U is executed



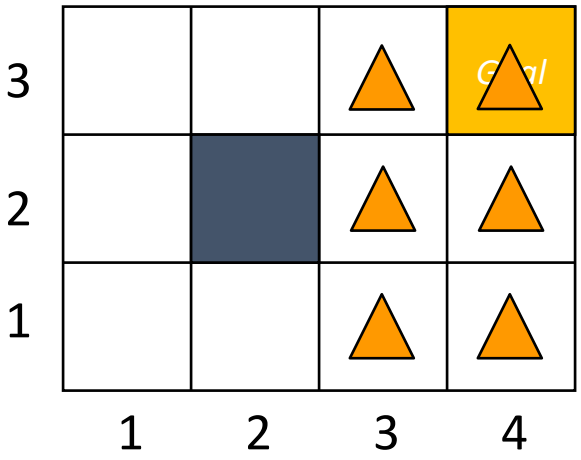


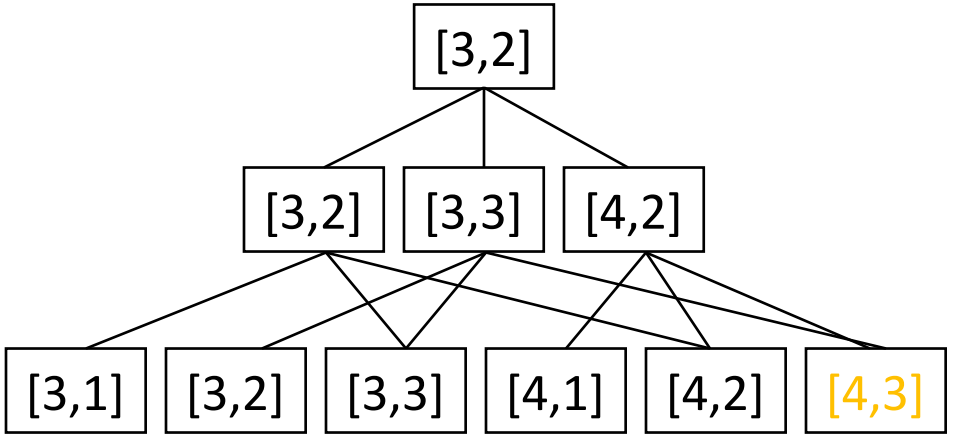


Sequence of Actions



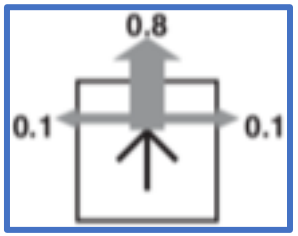
- In each state, the possible actions are U, D, R, and L; transition model:
- Current position: [3,2]
- Planned sequence of actions: (U, R)
  - U has been executed
  - R is executed



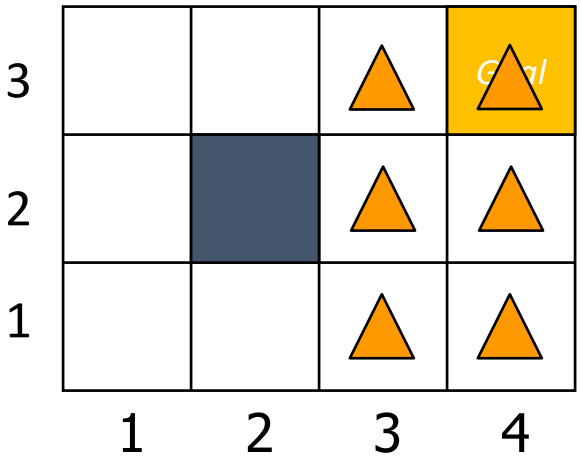


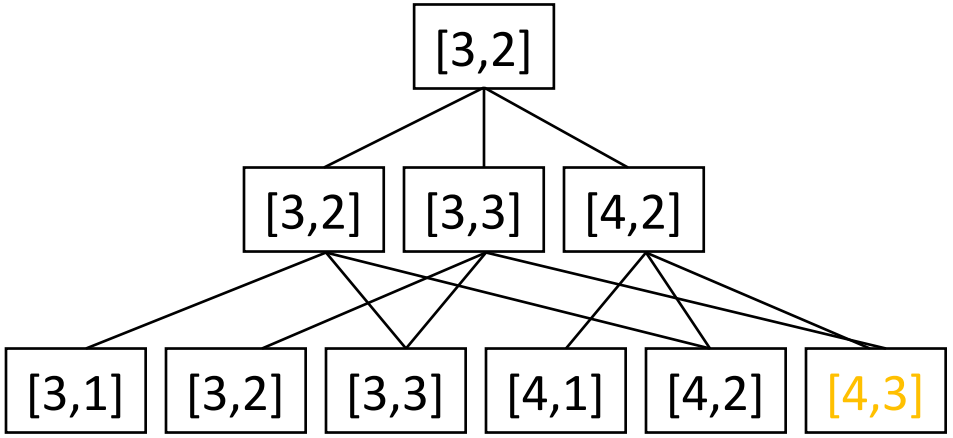
Histories

- In each state, the possible actions are U, D, R, and L; transition model:
- Current position: [3,2]
- Planned sequence of actions: (U, R)
  - U has been executed
  - R is executed



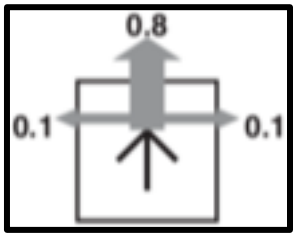
9 possible sequences of states, called histories, and 6 possible final states





Probability of Reaching the Goal

- In each state, the possible actions are U, D, R, and L; transition model:



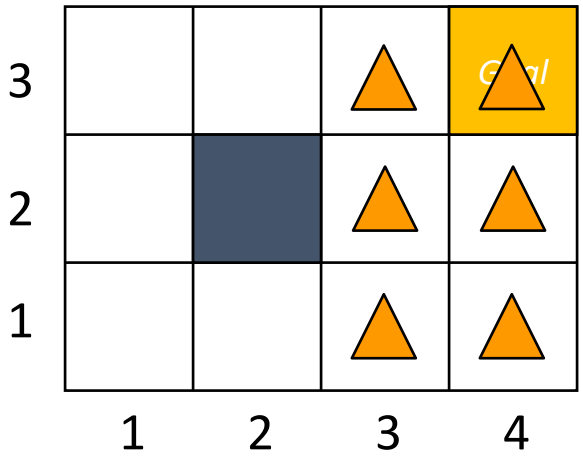
$$P([4,3] \mid (U,R).[3,2]) =$$
$$P([4,3] \mid R.[3,3]) \cdot P([3,3] \mid U.[3,2]) + P([4,3] \mid R.[4,2]) \cdot P([4,2] \mid U.[3,2])$$

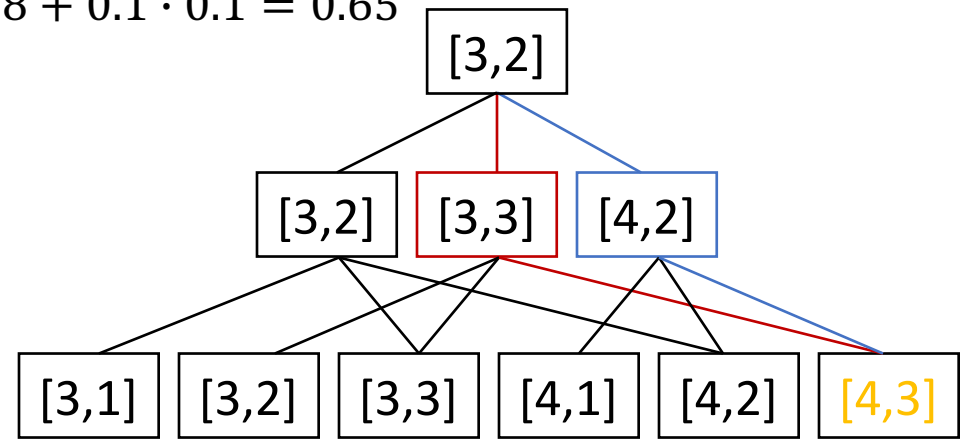
$P([4,3] \mid R.[3,3]) = 0.8$ $\quad$ $P([3,3] \mid U.[3,2]) = 0.8$

$P([4,3] \mid R.[4,2]) = 0.1$ $\quad$ $P([4,2] \mid U.[3,2]) = 0.1$

Note importance of Markov property in this derivation

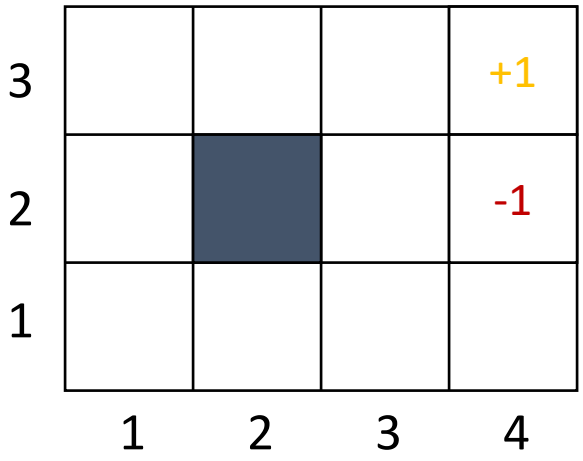
$$P([4,3] \mid (U,R).[3,2]) = 0.8 \cdot 0.8 + 0.1 \cdot 0.1 = 0.65$$





Utility Function

- [4,3] : power supply
- [4,2] : sand area the robot cannot escape
- Goal: robot needs to recharge its batteries
- [4,3] or [4,2] are terminal states
- In this example, we define the utility of a history by
  - the utility of the last state (+1 or –1) minus $0.04 \cdot n$
    - n is the number of moves
    - I.e., each move costs 0.04, which provides incentive to reach goal fast



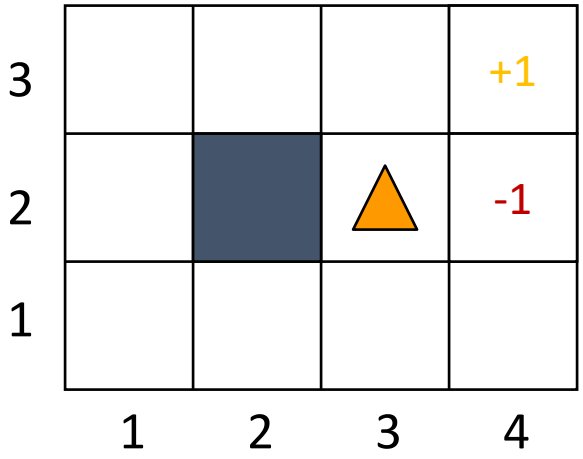
Utility of an Action Sequence

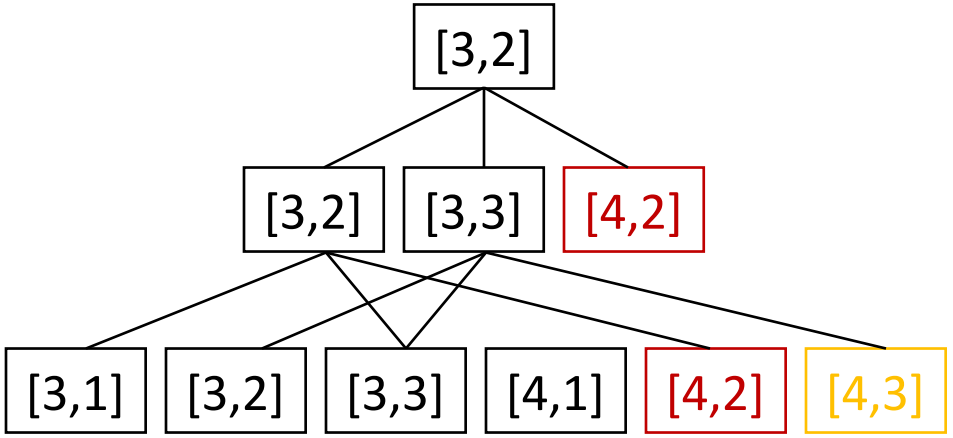
- Consider the action sequence (U,R) from [3,2]
- A run produces one among 7 possible histories, each with some probability
- Utility of the sequence is the expected utility of histories h:

$$U = \sum_h U_h P(h)$$

Is the optimal sequence what we want?

- Optimal sequence = the one with maximum utility

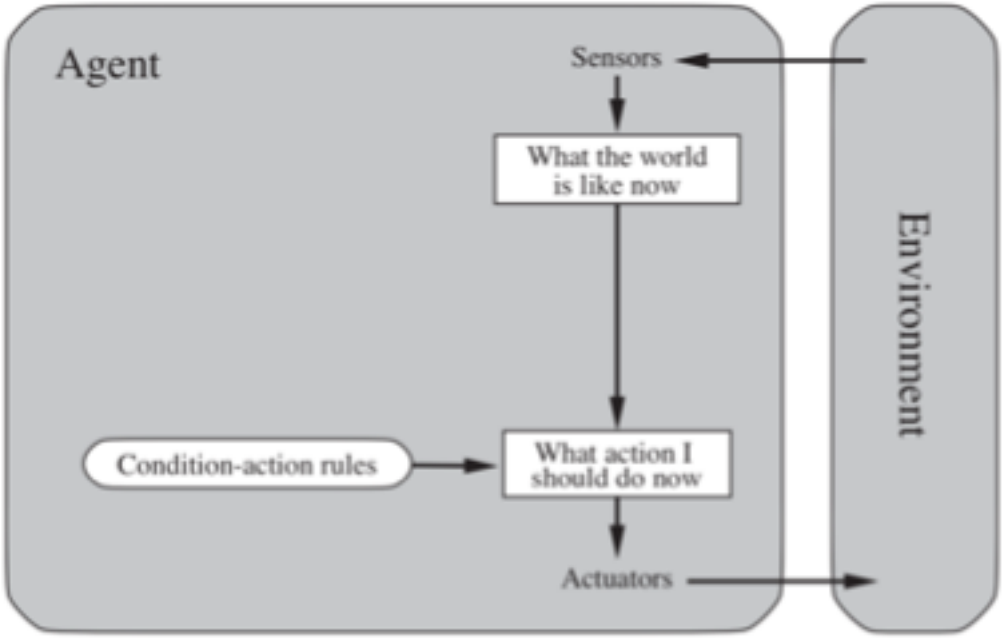




Reactive Agent Algorithm

Accessible or observable state

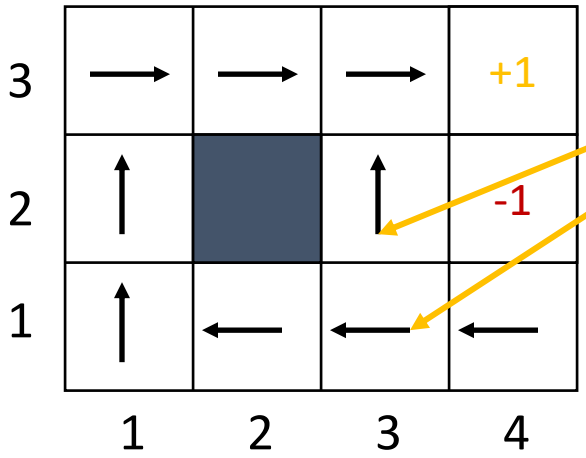
```
Act()
   repeat
      s ← sensed state
      if s is terminal then
         exit
      a ← choose action (given s)
      perform a
```



Policy (Reactive/Closed-loop Strategy)

- Policy π
  - Complete mapping from states to actions
- Optimal policy π^*
  - Always yields a history (ending at terminal state) with maximum expected utility
    - Due to Markov property

```
Act()
   repeat
      s ← sensed state
      if s is terminal then
         exit
      a ← π(s)
      perform a
```

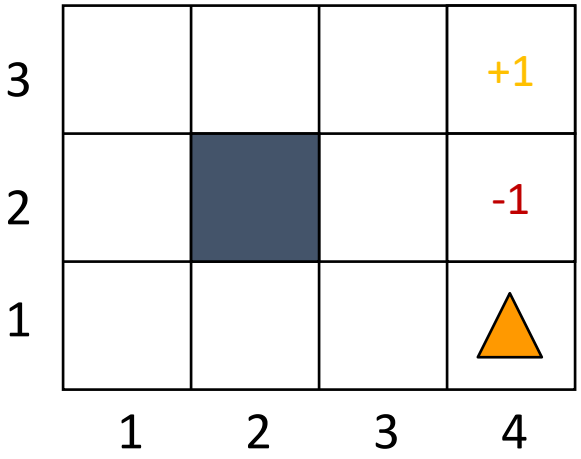


Note that [3,2] is a "dangerous" state that the optimal policy tries to avoid

How to compute π^*?
Solving a Markov Decision Process (MDP)

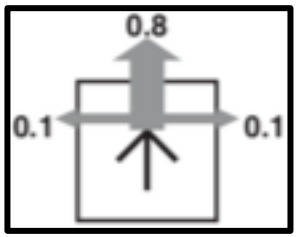
MDP

- Sequential decision problem for a fully observable, stochastic environment with a Markovian transition model and additive rewards (next slide)
- Components
  - a set of states S (with an initial state s_0)
  - a set $A(s)$ of actions in each state
  - a transition model $P(s'|s,a)$
  - a reward function $R(s)$



U, D, L, R

each move costs 0.04

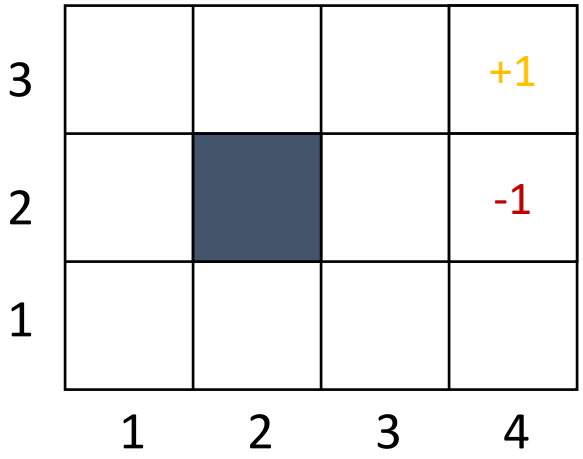


Additive Utility

- History $H = (s_0, s_1, \dots, s_n)$
- In each state s, agent receives reward $R(s)$
- Utility of H is additive iff

$$U(s_0, s_1, \dots, s_n) = R(s_0) + U(s_1, \dots, s_n) = \sum_{i=0}^{n} R(s_i)$$

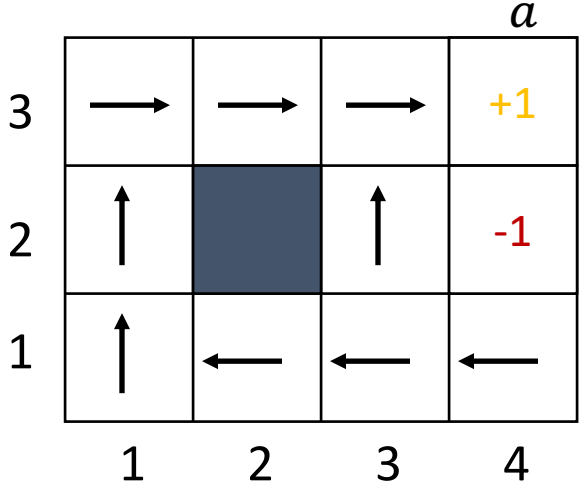
  - Discount factor $\gamma \in]0,1]$: $U(s_0, s_1, \dots, s_n) = \sum_{i=0}^{n} \gamma^i R(s_i)$
    - Close to 0: future rewards insignificant
    - Corresponds to an interest rate of $^{1-\gamma}/_{\gamma}$



- Robot navigation example:
  - $R(s_n) = +1$ if $s_n = [4,3]$
  - $R(s_n) = -1$ if $s_n = [4,2]$
  - $R(s_i) = -0.04$ if $i = 0, \dots, n-1$
  - $\gamma = 1$

Principle of MEU

- History $H = (s_0, s_1, \dots, s_n)$
  - Utility of H: $U(s_0, s_1, \dots, s_n) = \sum_{i=0}^{n} R(s_i)$
- Bellman equation:
  - $U(s_i) = R(s_i) + \gamma \max_a \sum_{s_j} P(s_j | a. s_i) U(s_j)$
- Optimal policy:
  - $\pi^*(s_i) = \operatorname{argmax}_a \sum_{s_j} P(s_j | a. s_i) U(s_j)$



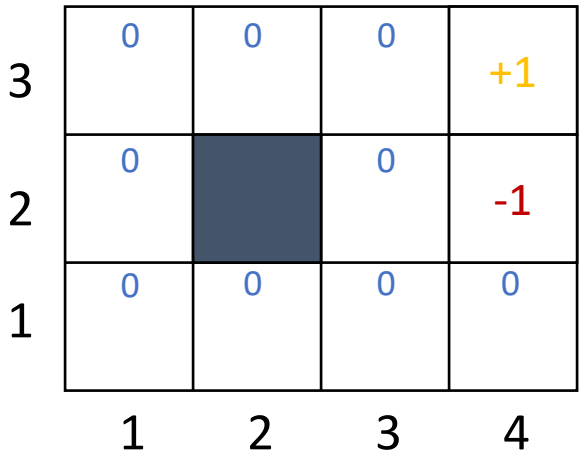
- Bellman equation for [1,1]
  - $U(1,1) = -0.04 + \gamma \max$

$$\begin{bmatrix} 0.8U(1,2) + 0.1U(2,1) + 0.1U(1,1), & (Up) \\ 0.9U(1,1) + 0.1U(1,2), & (Left) \\ 0.9U(1,1) + 0.1U(2,1), & (Down) \\ 0.8U(2,1) + 0.1U(1,2) + 0.1U(1,1) \end{bmatrix}. \quad (Right)$$

  - with $\gamma = 1$ as discount factor

Value Iteration

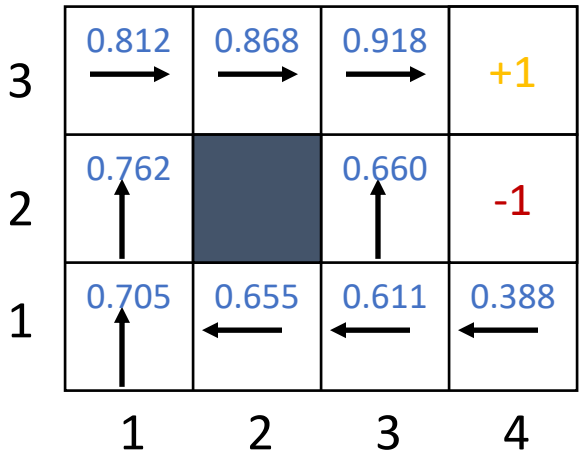
- Initialise the utility of each non-terminal state s_i to $U_0(s_i) = 0$
- For $t = 0, 1, 2, \ldots,$ do
  - $U_{t+1}(s_i) \leftarrow R(s_i) + \gamma \max_a \sum_{s_j} P(s_j | a. s_i) U_t(s_j)$
    - So called Bellman update

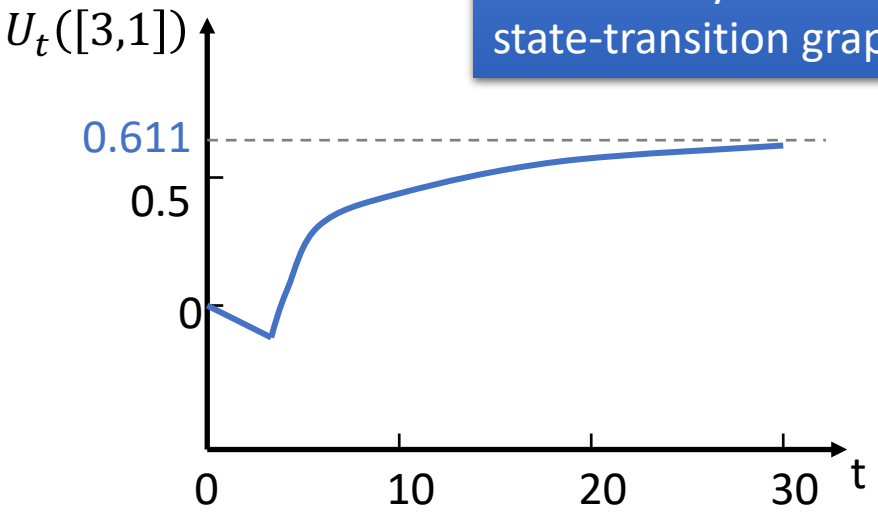


Value Iteration

- Initialise the utility of each non-terminal state s_i to $U_0(s_i) = 0$
- For $t = 0, 1, 2, \ldots$, do
  - $U_{t+1}(s_i) = R(s_i) + \gamma \max_a \sum_{s_j} P(s_j | a.s_i) U_t(s_j)$
    - So called Bellman update

Note the importance of terminal states and connectivity of the state-transition graph





Value Iteration: Algorithm

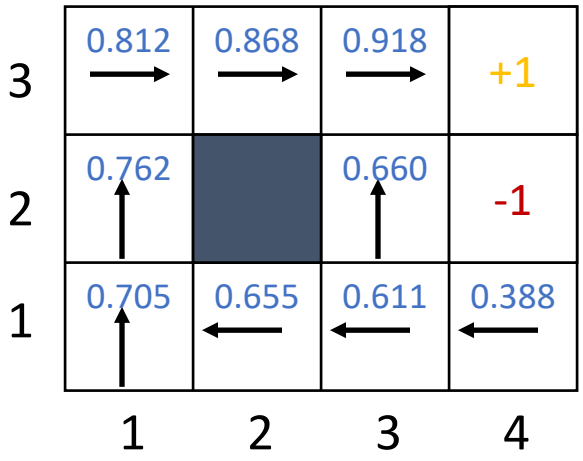
```
function value-iteration(mdp,ϵ)
    U′ ← 0
    repeat
        U ← U′
        δ ← 0
        for each state s ∈ S do
            U′[s] ← R(s) + γ max_{a∈A(s)} Σ_{s′} P(s′|a.s) U[s′]
            if |U′[s] - U[s]| > δ then
                δ ← |U′[s] - U[s]|
    until δ < ϵ(1−γ)/γ
```

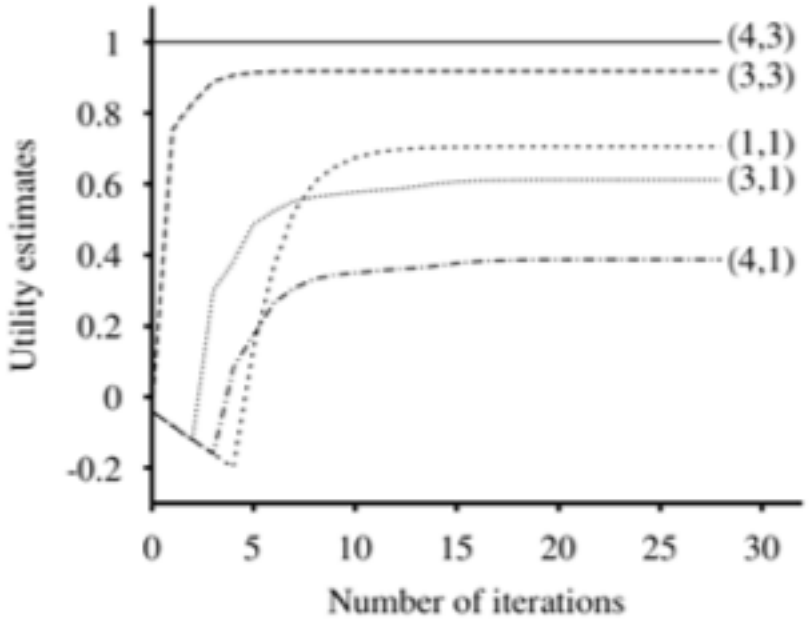
- Inputs
  - an MDP, which includes
    - States S
    - For all $s \in S$, actions $A(s)$, transition model $P(s' | a.s)$, rewards $R(s)$
    - Discount γ
  - Maximum error allowed ϵ
- Local variables
  - U, U' vectors of utilities for states in S, initially 0
  - δ maximum change in utility of any state in an iteration

Evolution of Utilities

- For $t = 0, 1, 2, \ldots,$ do
  - $U_{t+1}(s_i) = R(s_i) + \gamma \max_a \sum_{s_j} P(s_j | a. s_i) U_t(s_j)$

- Value iteration ≈ information propagation

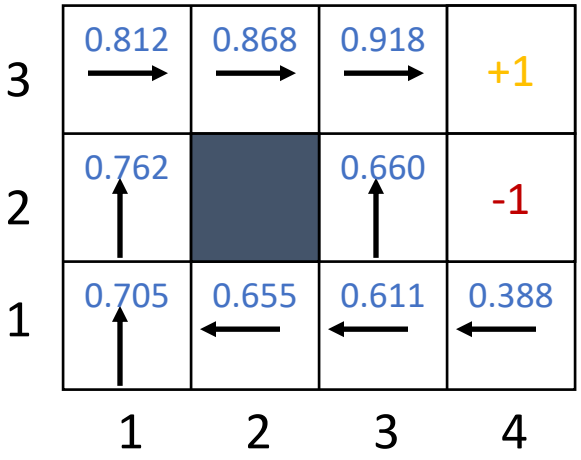




Argmax Action

- For $t = 0, 1, 2, \ldots,$ do
  - $U_{t+1}(s_i) = R(s_i) + \gamma \max_a \sum_{s_j} P(s_j | a. s_i) U_t(s_j)$

- Argmax action may change over iterations

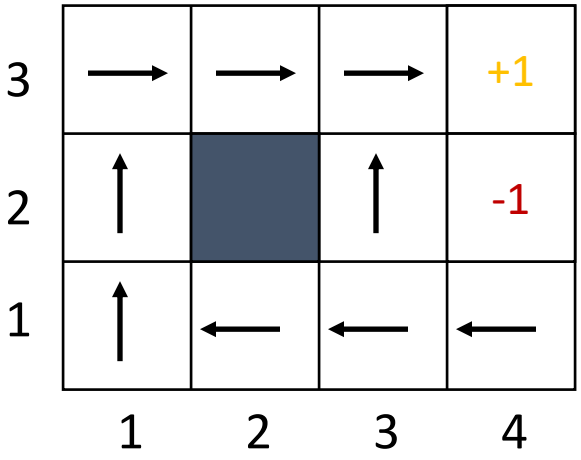


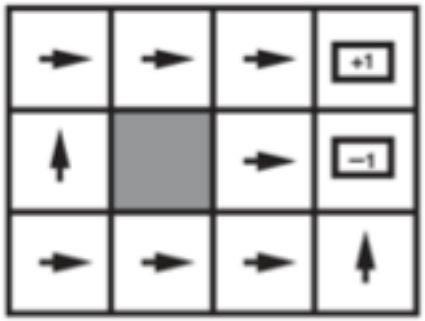
- Bellman equation for $[1,1]$
  - $U(1,1) = -0.04 + \gamma \max$

$$\begin{bmatrix} 0.8U(1,2) + 0.1U(2,1) + 0.1U(1,1), & (Up) \\ 0.9U(1,1) + 0.1U(1,2), & (Left) \\ 0.9U(1,1) + 0.1U(2,1), & (Down) \\ 0.8U(2,1) + 0.1U(1,2) + 0.1U(1,1) \end{bmatrix}. \quad (Right)$$

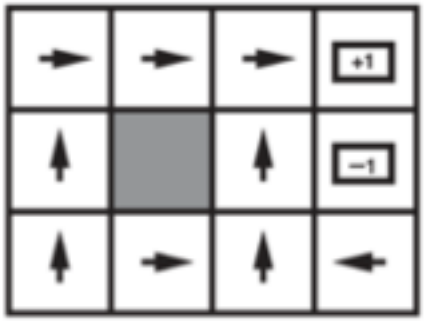
Effect of Rewards

- For $t = 0, 1, 2, \ldots$, do
  - $U_{t+1}(s_i) = R(s_i) + \gamma \max_a \sum_{s_j} P(s_j | a. s_i) U_t(s_j)$
- Optimal policies for different rewards
  - For $R(s) = 0.04$, see below (⇩)

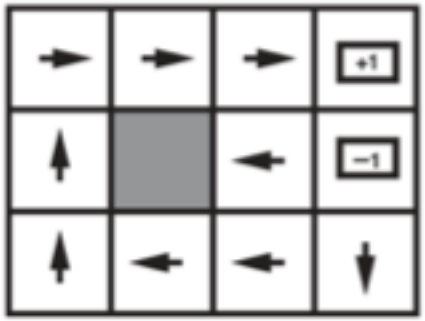




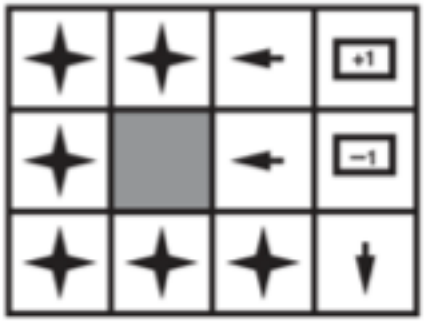
$R(s) < -1.6284$



$-0.4278 < R(s) < -0.0850$



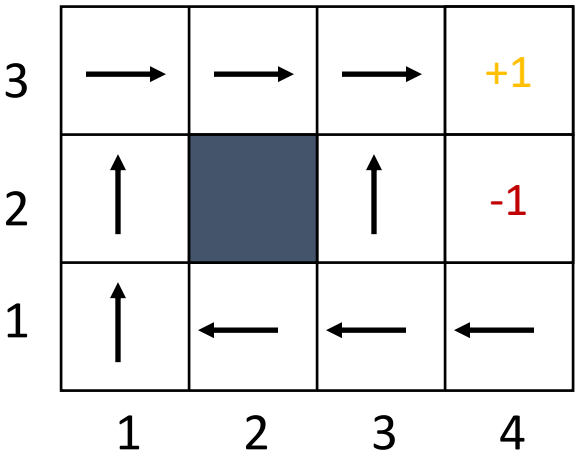
$-0.0221 < R(s) < 0$

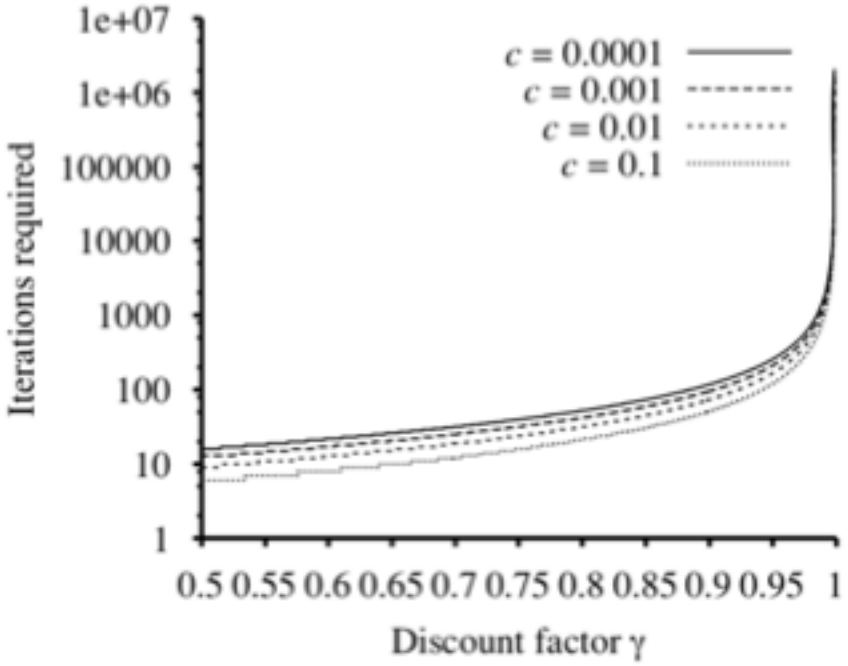


$R(s) > 0$

Effect of Allowed Error & Discount

- For $t = 0, 1, 2, \ldots$, do
  - $U_{t+1}(s_i) = R(s_i) + \gamma \max_a \sum_{s_j} P(s_j | a. s_i) U_t(s_j)$
- Left figure: Iterations required to ensure a maximum error of $\varepsilon = c \cdot R_{max}$
  - R_{max} maximum reward
    - +1 in the example





Policy Iteration

- Pick a policy π_0 at random
- Repeat:
  - Policy evaluation: Compute the utility of each state for π_t
    - $U_t(s_i) = R(s_i) + \gamma \sum_{s_j} P(s_j|\pi_t(s_i).s_i)U_t(s_j)$
      - No longer involves a max operation as action is determined by π_t
  - Policy improvement: Compute the policy π_{t+1} given U_t
    - $\pi_{t+1}(s_i) = \arg\max_a \sum_{s_j} P(s_j|\pi_t(s_i).s_i)U_t(s_j)$
  - If $\pi_{t+1} = \pi_t$, then return π_t

Solve the set of linear equations:

$$U(s_i) = R(s_i) + \gamma \sum_{s_j} P(s_j|\pi(s_i).s_i)U(s_j)$$

(often a sparse system)

Policy Iteration: Algorithm

```
function policy-iteration(mdp)
    repeat
        U ← policy-evaluation(π,U,mdp)
        unchanged ← true
        for each state s ∈ S do
            if max_{a∈A(s)} Σ_{s'} P(s'|a.s)U[s'] > Σ_{s'} P(s'|π[s].s)U[s'] then
                π[s] ← argmax_{a∈A(s)} Σ_{s'} P(s'|a.s)U[s']
                unchanged ← false
    until unchanged
    return π
```

- Inputs
  - an MDP, which includes
    - States S
    - For all $s \in S$, actions $A(s)$, transition model $P(s' | a.s)$, rewards $R(s)$
- Local variables
  - U vectors of utilities for states in S, initially 0
  - π a policy vector indexed by state, initially random

Policy Evaluation

- Compute the utility of each state for π
  - $U_t(s_i) = R(s_i) + \gamma \sum_{s_j} P\left(s_j | \pi_t(s_i).\, s_i\right) U_t\left(s_j\right)$
- Complexity of policy evaluation: $O(n^3)$
  - For n states, n linear equations with n unknowns
  - Prohibitive for large n
- Approximation of utilities
  - Perform k value iteration steps with fixed policy π_t, return utilities
    - Simplified Bellman update: $U_{t+1}(s_i) = R(s_i) + \gamma \sum_{s_j} P\left(s_j | \pi(s_i).\, s_i\right) U_t\left(s_j\right)$

1. Asynchronous policy iteration
   - Pick any subset of states and either
     - Update utilities (simplified value iteration) or
     - Update the policy (policy improvement)

Asynchronous Policy Iteration

- Further approximation of policy iteration
  - Pick any subset of states and do one of the following
    - Update utilities
      - Using simplified value iteration as described on previous slide
    - Update the policy
      - Policy improvement as before
- Is not guaranteed to converge to an optimal policy
  - Possible if each state is still visited infinitely often, knowledge about unimportant states, etc.
- Freedom to work on any states allows for design of domain-specific heuristics
  - Update states that are likely to be reached by a good policy

Intermediate Summary

- MDP
  - Markov property
    - Current state depends only on previous state
  - Sequence of actions, history, policy
    - Sequence of actions may yield multiple histories, i.e., sequences of states, with a utility
    - Policy: complete mapping of states to actions
    - Optimal policy: policy with maximum expected utility
  - Value iteration, policy iteration
    - Algorithms for calculating an optimal policy for an MDP

How is that used in planning?

Outline

Markov decision process (MDP)

- Markov property
- Sequence of actions, history, policy
- Value iteration, policy iteration

6.2 Stochastic shortest path problems

- Safe/unsafe policies
- Optimality
- Policy iteration, value iteration

6.3 Heuristic search algorithms

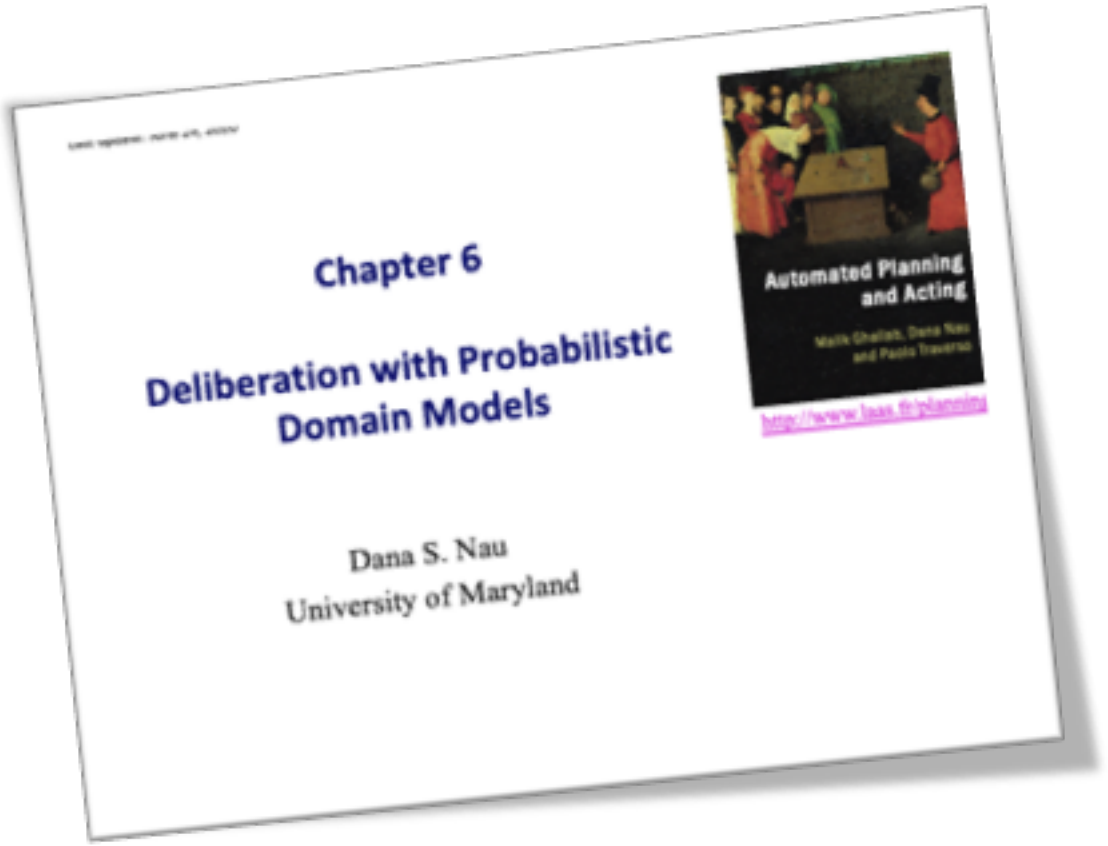
- Best-first search
- Determinisation

6.4 Online probabilistic planning

- Lookahead

Acknowledgements

- Back to slides adapted from material provided by Dana Nau and Chapter 6 of the book



Probabilistic Planning Domain

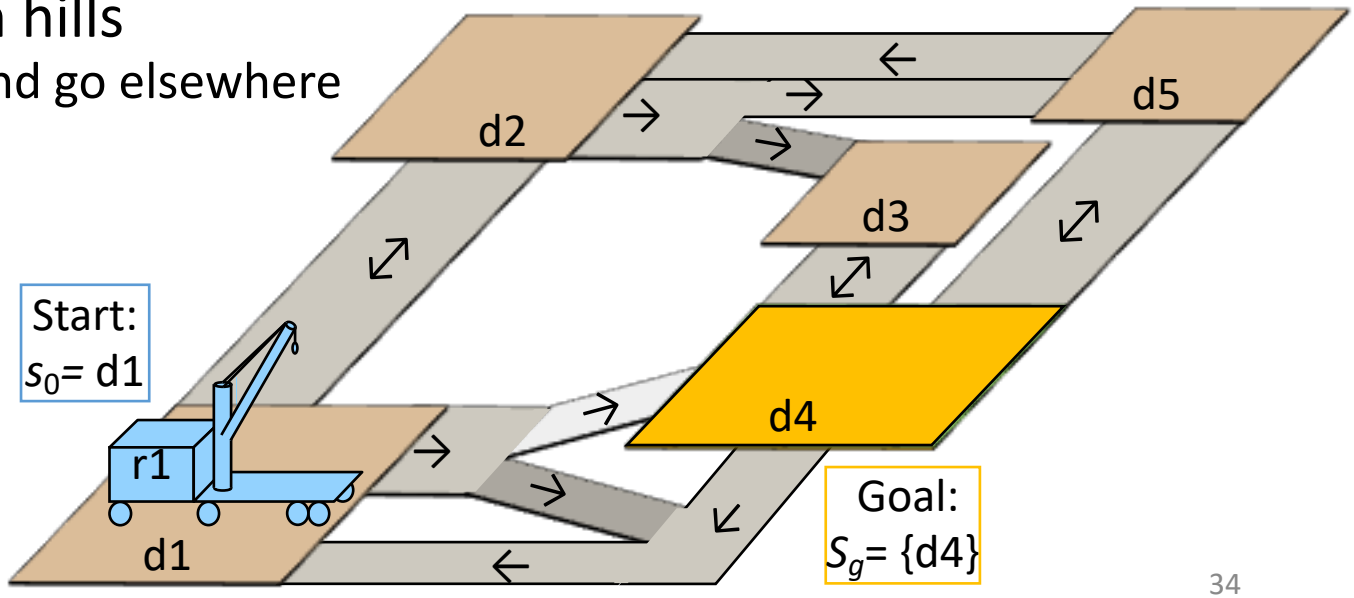
- $\Sigma = (S, A, \gamma, P, cost)$
  - S = set of states
  - A = set of actions
  - $\gamma : S \times A \rightarrow 2^S$
  - $P(s' \mid s, a)$ = probability of going to state s' if we perform a in s
    - Require $P(s' \mid s, a) \neq 0$ iff $s' \in \gamma(s, a)$
  - $cost$: $S \times A \rightarrow \mathbb{R}^{>0}$
    - $cost(s, a)$ = cost of action a in state s
    - may omit, default is $cost(s, a) = 1$

Instead of maximising expected utility as before: *Minimise expected cost*

Example

- Robot $r1$ starts at $d1$
- Objective: get to $d4$
- Simplified state names: write $\{loc(r1) = d2\}$ as $d2$
- Simplified action names: write $move(r1, d2, d3)$ as $m23$
- $r1$ has unreliable steering, especially on hills
  - May slip and go elsewhere

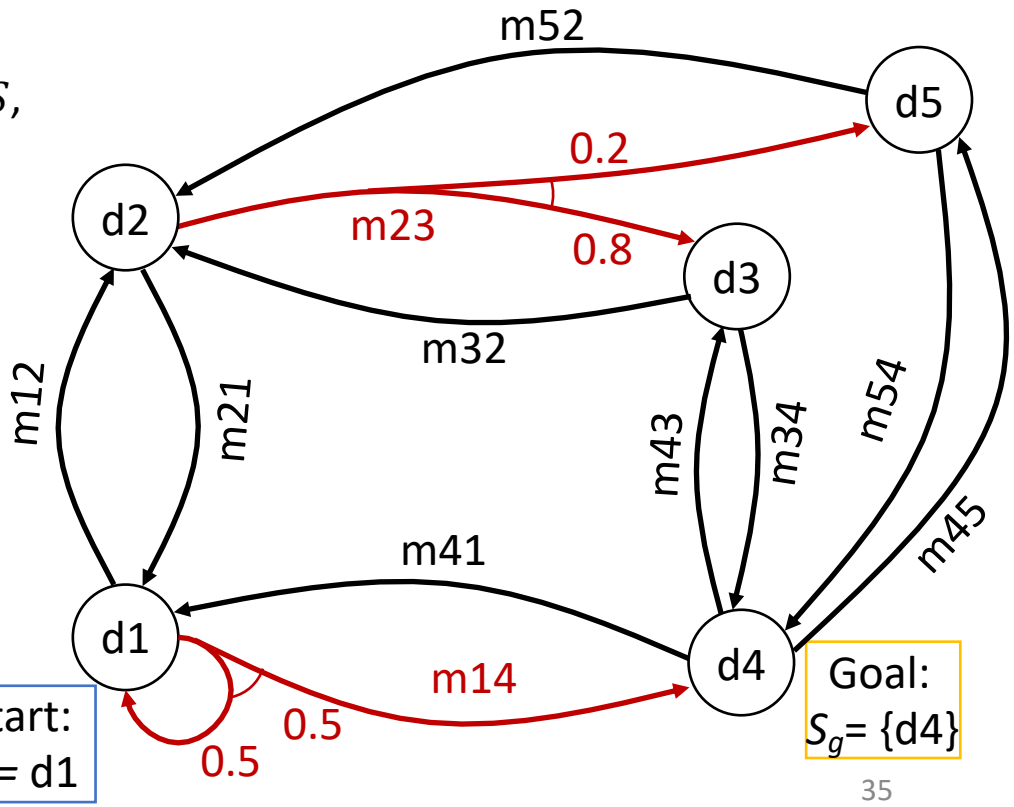
- $m12$: $\mathrm{P}(d2 \mid d1, m12) = 1$
- $m21, m34, m41, m43, m45, m52, m54$: like above
- $m14$: $\mathrm{P}(d4 \mid d1, m14) = 0.5$
  $\mathrm{P}(d1 \mid d1, m14) = 0.5$
- $m23$: $\mathrm{P}(d3 \mid d1, m23) = 0.8$
  $\mathrm{P}(d5 \mid d1, m23) = 0.2$



Policies, Problems, Solutions

- Stochastic shortest path (SSP) problem:
  - a triple (Σ, s_0, S_g)
- Policy:
  - partial function $\pi : S \rightarrow A$ s.t.
    - for every $s \in Dom(\pi) \subseteq S$, $\pi(s) \in Applicable(s)$
- Solution for (Σ, s_0, S_g) :
  - a policy π s.t.
    - $s_0 \in Dom(\pi)$ and
    - $\hat{\gamma}(s_0, \pi) \cap S_g \neq \emptyset$

- $m14$: $P(d4 \mid d1, m14) = 0.5$
  $P(d1 \mid d1, m14) = 0.5$
- $m23$: $P(d3 \mid d1, m23) = 0.8$
  $P(d5 \mid d1, m23) = 0.2$



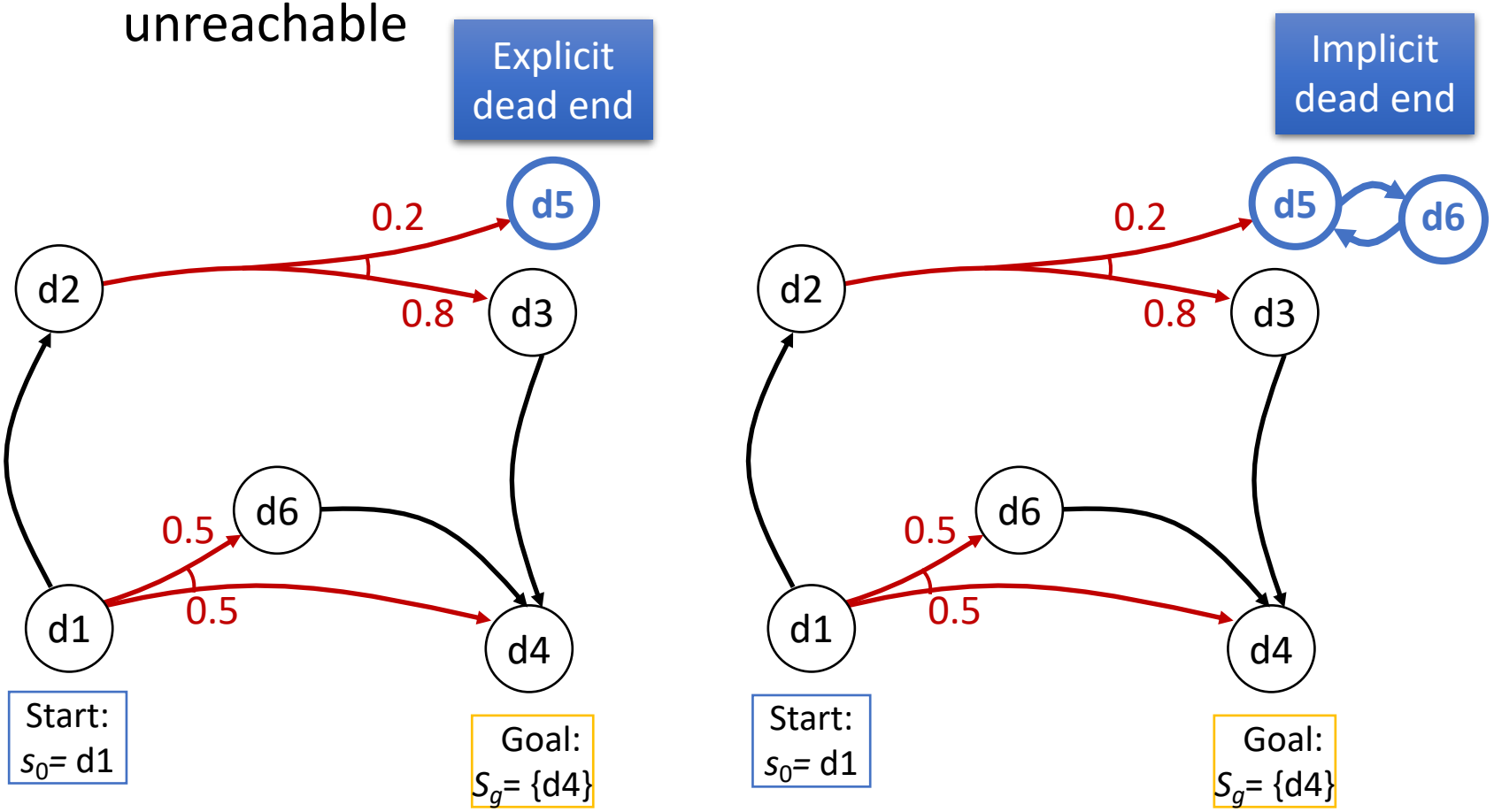
Notation and Terminology

- Transitive closure
  - $\hat{\gamma}(s, \pi)$ = {s and all states reachable from s using π}
- $Graph(s, \pi)$ = rooted graph induced by π at s
  - Nodes: $\hat{\gamma}(s, \pi)$
  - Edges: state transitions
- $leaves(s, \pi) = \hat{\gamma}(s, \pi) \backslash Dom(\pi)$

- A solution policy π is closed if it does not stop at non-goal states unless there's no way to continue
  - for every state $s \in \hat{\gamma}(s, \pi)$, either
    - $s \in Dom(\pi)$ (i.e., π specifies an action at s)
    - $s \in S_g$ (i.e., s is a goal state)
    - $Applicable(s) = \emptyset$ (i.e., there are no applicable actions at s)

Dead Ends

- Dead end
  - A state or set of states from which the goal is unreachable



Histories

- History: sequence of states $\sigma = \langle s_0, s_1, s_2, \dots \rangle$
  - May be finite or infinite
    - $\sigma = \langle d1, d2, d3, d4 \rangle$
    - $\sigma = \langle d1, d2, d1, d2, \dots \rangle$
- $H(s, \pi)$ = {all possible histories if we start at s and follow π, stopping if $\pi(s)$ is undefined or if we reach a goal state}
- If $\sigma \in H(s, \pi)$, then

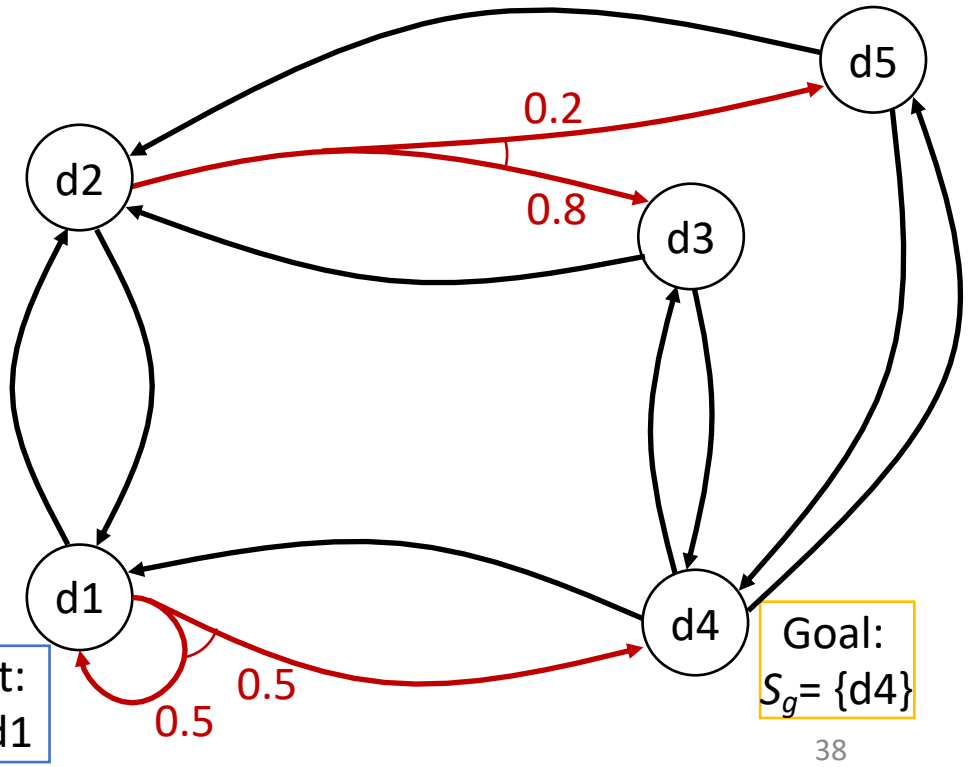
$$P(\sigma \mid s, \pi) = \prod_i P\left(s_{i+1} \mid s_i, \pi(s_i)\right)$$

  - Thus

$$\sum_{\sigma \in H(s,\pi)} P(\sigma \mid s, \pi) = 1$$

- Probability of reaching a goal:

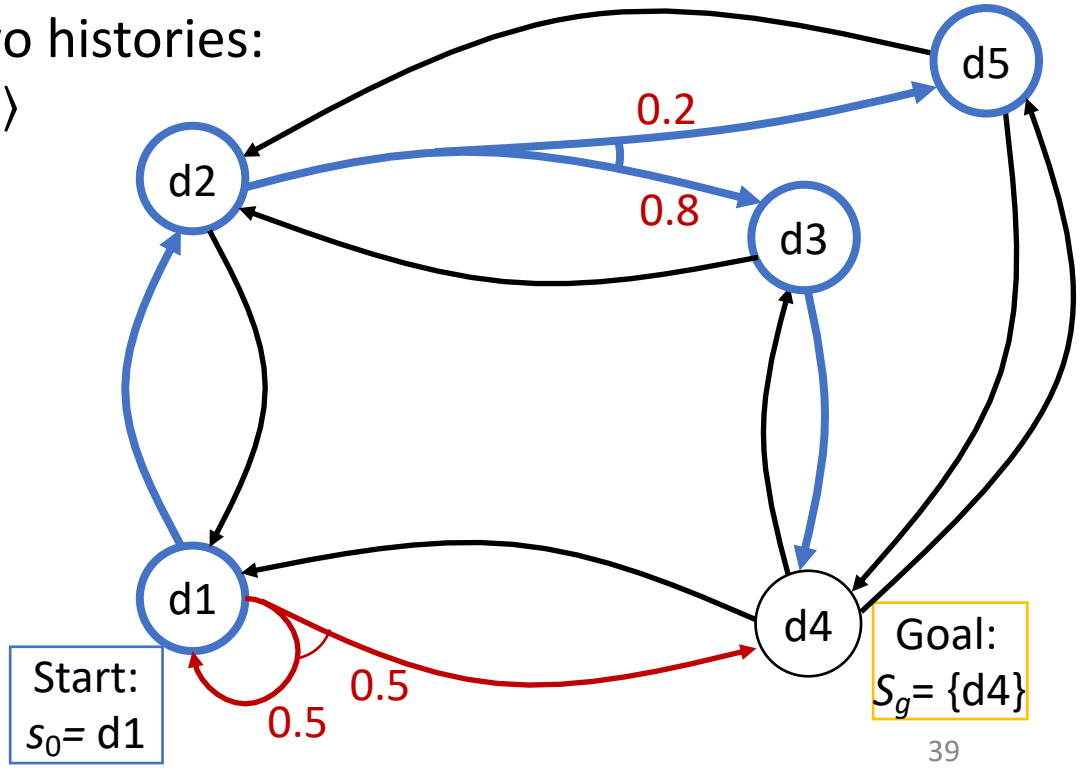
$$P(S_g \mid s, \pi) = \sum_{\sigma \in H(s,\pi),\ \sigma \text{ ends at a state in } S_g} P(\sigma \mid s, \pi)$$



Unsafe Solutions

- Unsafe solution: $0 < P(S_g|s_0, \pi) < 1$

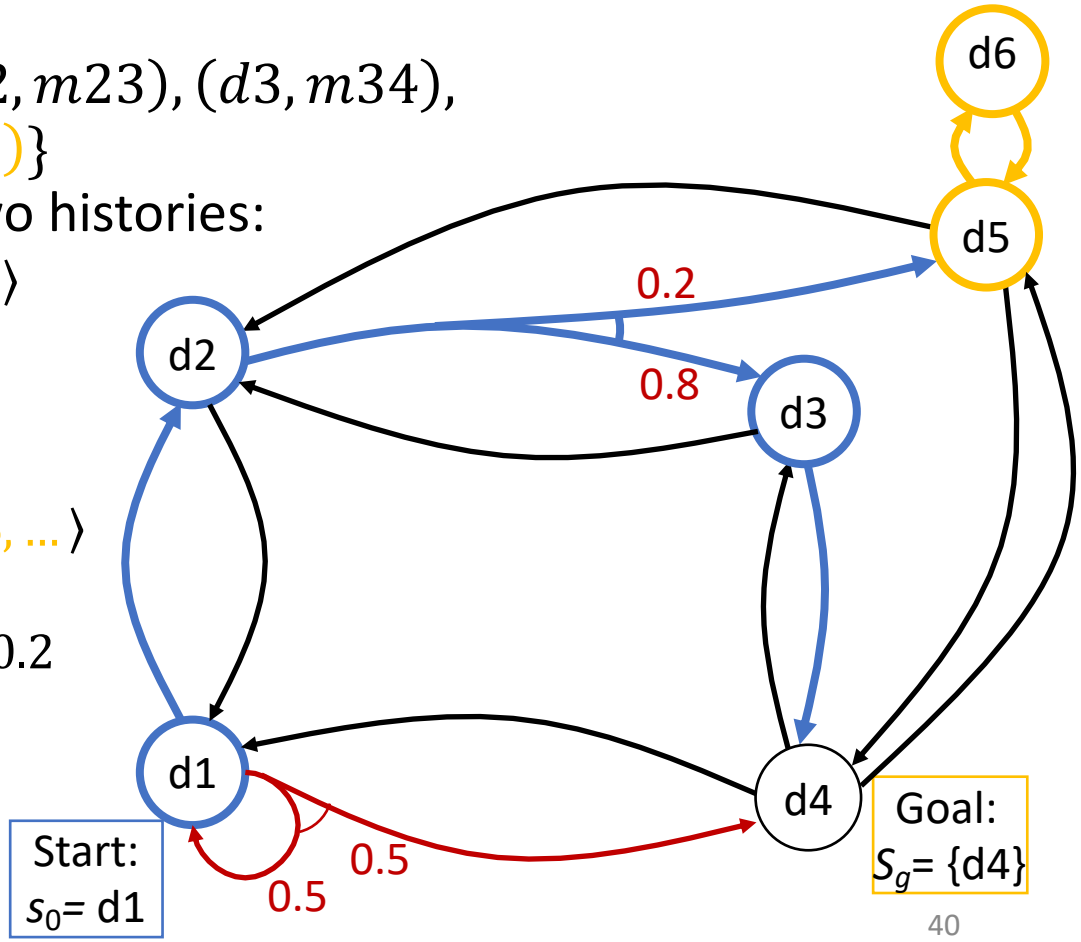
- Example:
  - $\pi_1 = \{(d1, m12), (d2, m23), (d3, m34)\}$
  - $H(s_0, \pi_1)$ contains two histories:
    - $\sigma_1 = \langle d1, d2, d3, d4 \rangle$
    - $P(\sigma_1|s_0, \pi_1) = 1 \cdot 0.8 \cdot 1 = 0.8$
    - $\sigma_2 = \langle d1, d2, d5 \rangle$
    - $P(\sigma_2|s_0, \pi_1) = 1 \cdot 0.2 = 0.2$
  - $P(S_g|s_0, \pi_1) = 0.8$



Unsafe Solutions

- Unsafe solution: $0 < P(S_g|s_0, \pi) < 1$

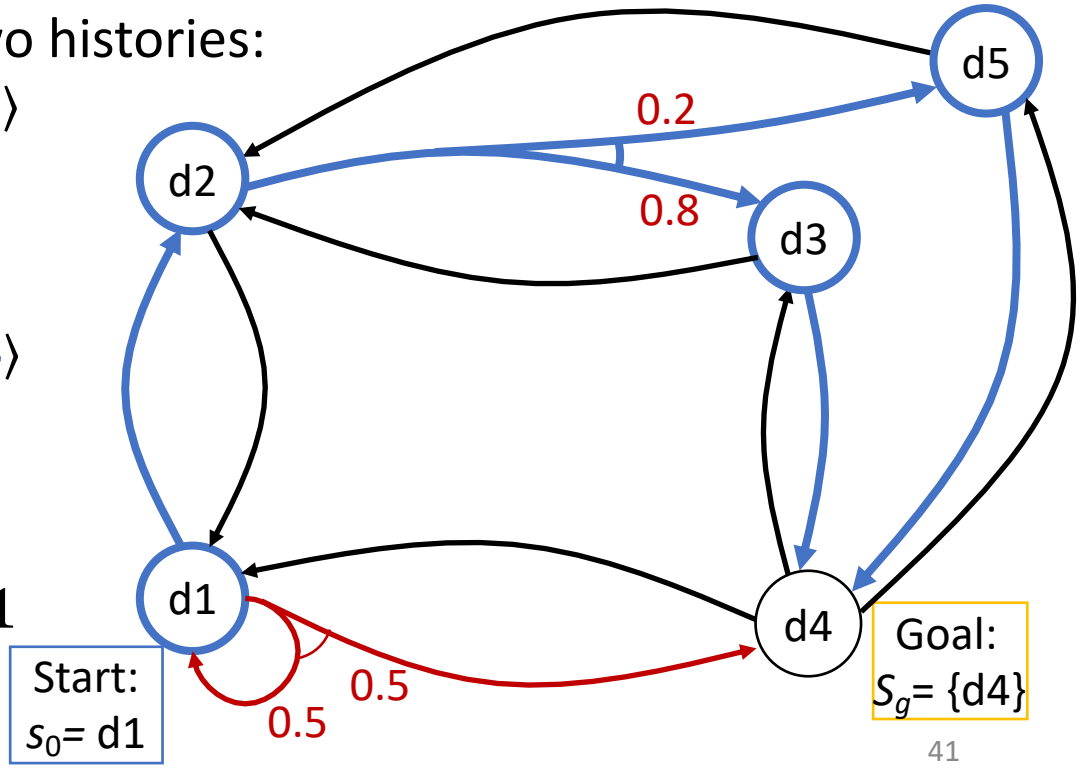
- Example:
  - $\pi_2 = \{(d1, m12), (d2, m23), (d3, m34), (d5, m56), (d6, m65)\}$
  - $H(s_0, \pi_2)$ contains two histories:
    - $\sigma_1 = \langle d1, d2, d3, d4 \rangle$
    - $P(\sigma_1|s_0, \pi_2) = 1 \cdot 0.8 \cdot 1 = 0.8$
    - $\sigma_3 = \langle d1, d2, d5, d6, \dots \rangle$
    - $P(\sigma_3|s_0, \pi_2) = 1 \cdot 0.2 \cdot 1 \cdot \dots = 0.2$
  - $P(S_g|s_0, \pi_2) = 0.8$



Safe Solutions

- Safe solution: $P(S_g|s_0,\pi) = 1$

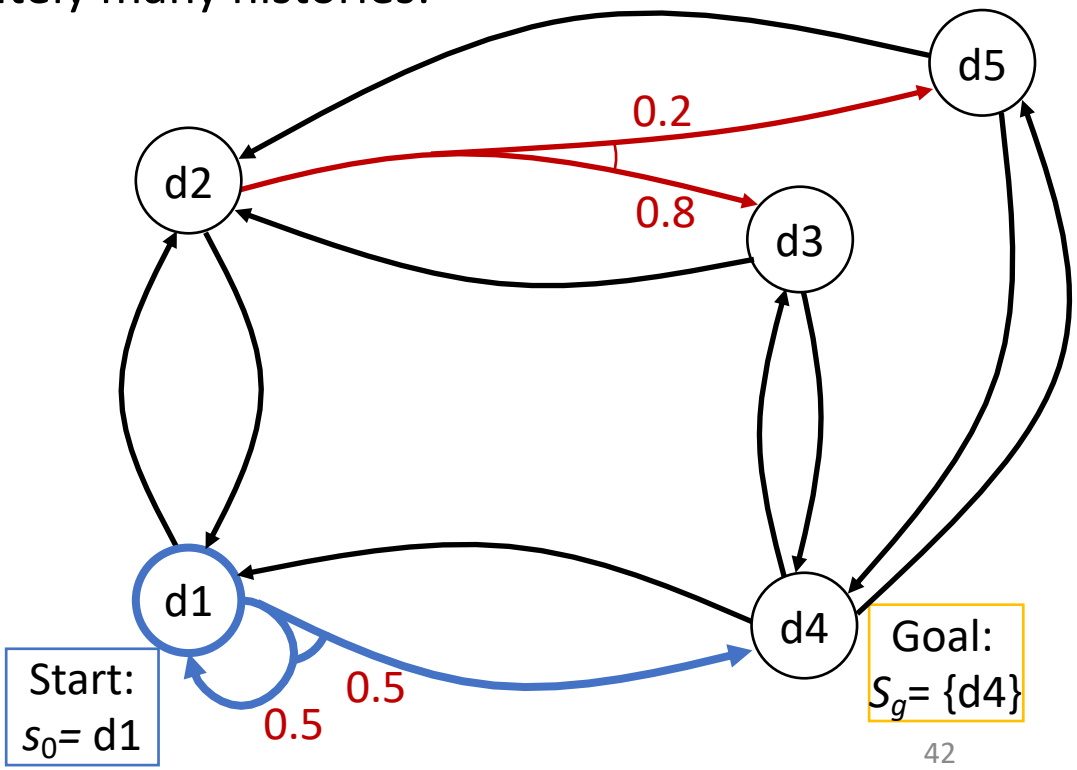
- An acyclic safe solution:
  - $\pi_3 = \{(d1, m12), (d2, m23), (d3, m34), (d5, m54)\}$
  - $H(s_0, \pi_3)$ contains two histories:
    - $\sigma_1 = \langle d1, d2, d3, d4\rangle$
    - $P(\sigma_1|s_0,\pi_3) = 1 \cdot 0.8 \cdot 1 = 0.8$
    - $\sigma_4 = \langle d1, d2, d5, d4\rangle$
    - $P(\sigma_4|s_0,\pi_3) = 1 \cdot 0.2 \cdot 1 = 0.2$
  - $P(S_g|s_0,\pi_3) = 0.8 + 0.2 = 1$



Safe Solutions

- Safe solution: $P(S_g|s_0,\pi) = 1$

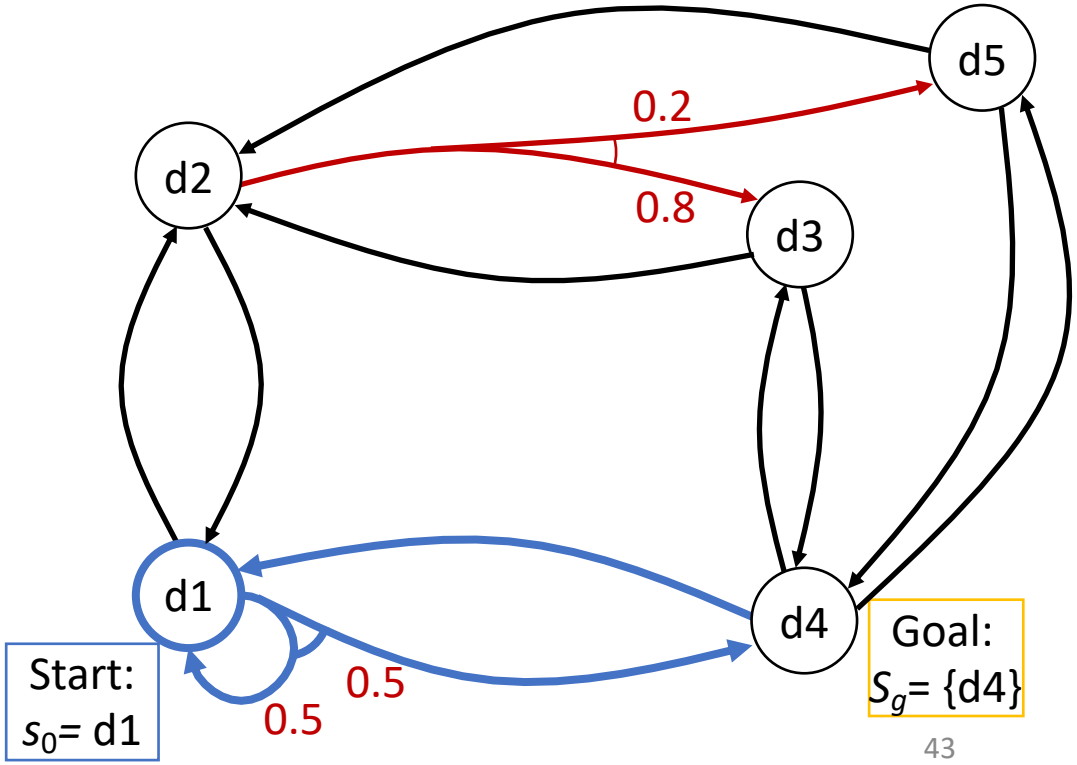
- A cyclic safe solution:
  - $\pi_4 = \{(d1, m14)\}$
  - $H(s_0, \pi_4)$ contains infinitely many histories:
    - $\sigma_5 = \langle d1, d4 \rangle$
    - $P(\sigma_5|s_0,\pi_4)$ $= 0.5 = \left(\frac{1}{2}\right)^1$
    - $\sigma_6 = \langle d1, d1, d4 \rangle$
    - $P(\sigma_6|s_0,\pi_4)$ $= 0.5 \cdot 0.5 = \left(\frac{1}{2}\right)^2$
    - ...
  - $P(S_g|s_0,\pi_4)$ $= \frac{1}{2} + \frac{1}{4} + \ \ldots = 1$



Safe Solutions

- Safe solution: $P(S_g|s_0,\pi) = 1$

- Another cyclic safe solution:
  - $\pi_5 = \{(d1, m14), (d4, m41)\}$
  - $H(s_0,\pi_5) = H(s_0,\pi_4)$:
    - $\sigma_5 = \langle d1, d4\rangle$
    - $P(\sigma_5|s_0,\pi_5)$
      $= 0.5 = \left(\frac{1}{2}\right)^1$
    - $\sigma_6 = \langle d1, d1, d4\rangle$
    - $P(\sigma_6|s_0,\pi_6)$
      $= 0.5 \cdot 0.5 = \left(\frac{1}{2}\right)^2$
    - ...
  - $P(S_g|s_0,\pi_5)$
    $= \frac{1}{2} + \frac{1}{4} + \ \ldots = 1$

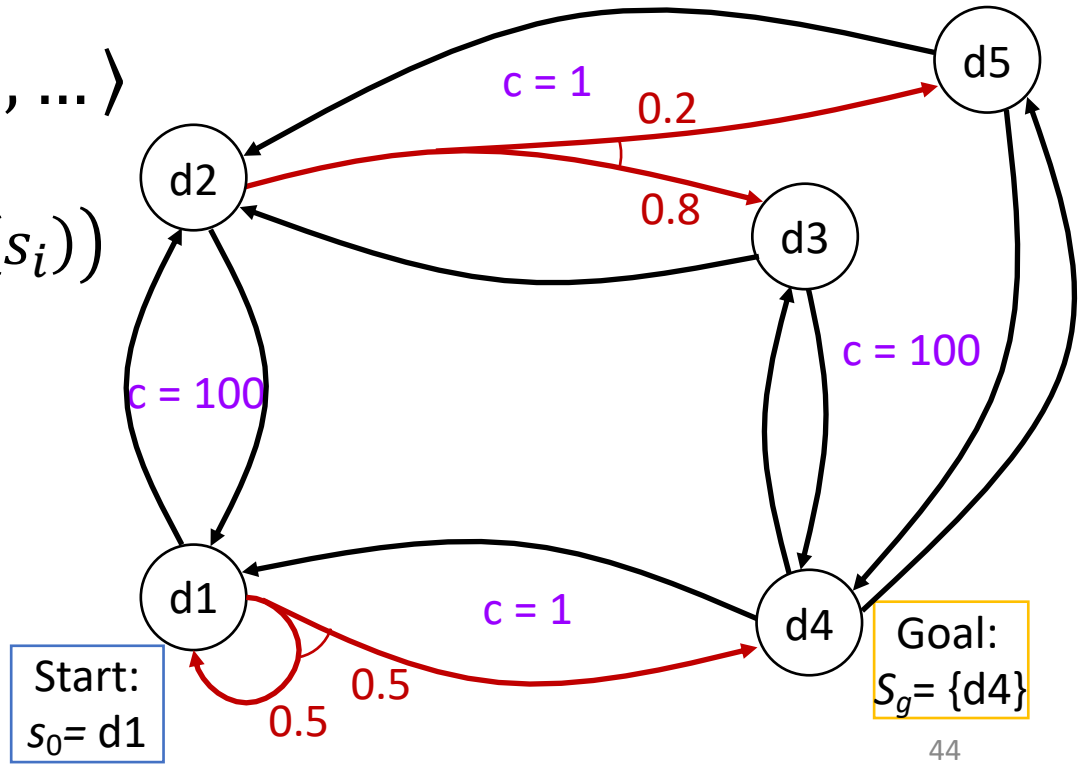


Expected Cost

- $cost(s, a)$ = cost of using a in s
  - Example
    - Each ”horizontal” action costs 1
    - Each ”vertical” action costs 100
- Costs of a history

$$\sigma = \langle s_0, s_1, s_2, \dots \rangle$$

  - $cost(\sigma \mid s_0, \pi) = \sum_{s_i \in \sigma} cost\big(s_i, \pi(s_i)\big)$



Expected Cost

- Let π be a safe solution
- At each state $s \in Dom(\pi)$, expected cost of following π to goal:
  - Weighted sum of history costs:

$$V^{\pi}(s) = cost\big(s, \pi(s)\big) + \sum_{\sigma \in H(s,\pi)} P(\sigma \mid s, \pi)\, cost(\sigma \mid s, \pi)$$

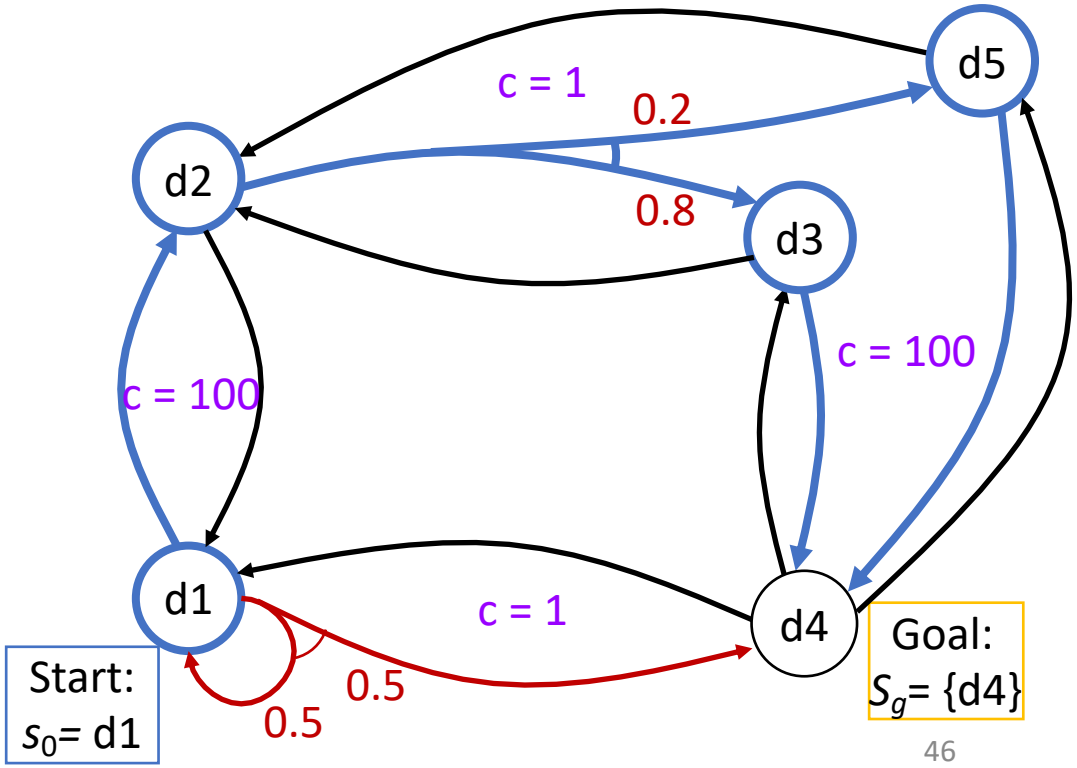
  - Recursive formulation

$$V^{\pi}(s) = \begin{cases} 0 & \text{if } s \in S_g \\ cost\big(s, \pi(s)\big) + \sum_{s' \in \gamma(s,\pi(s))} P\big(s' \mid s, \pi(s)\big) V^{\pi}(s') & \text{otherwise} \end{cases}$$

Example

- $\pi_3 = \{(d1, m12), (d2, m23), (d3, m34), (d5, m54)\}$
- Weighted sum of history cost:
  - $\sigma_1 = \langle d1, d2, d3, d4 \rangle$
  - $\mathrm{P}(\sigma_1 | s_0, \pi_1) = 0.8$
  - $cost(\sigma_1 | s_0, \pi_3) = 100 + 1 + 100 = 201$
  - $\sigma_4 = \langle d1, d2, d5, d4 \rangle$
  - $\mathrm{P}(\sigma_4 | s_0, \pi_1) = 0.2$
  - $cost(\sigma_4 | s_0, \pi_3) = 100 + 1 + 100 = 201$
- $V^{\pi_1}(d1) = 0.8(201) + 0.2(201) = 201$

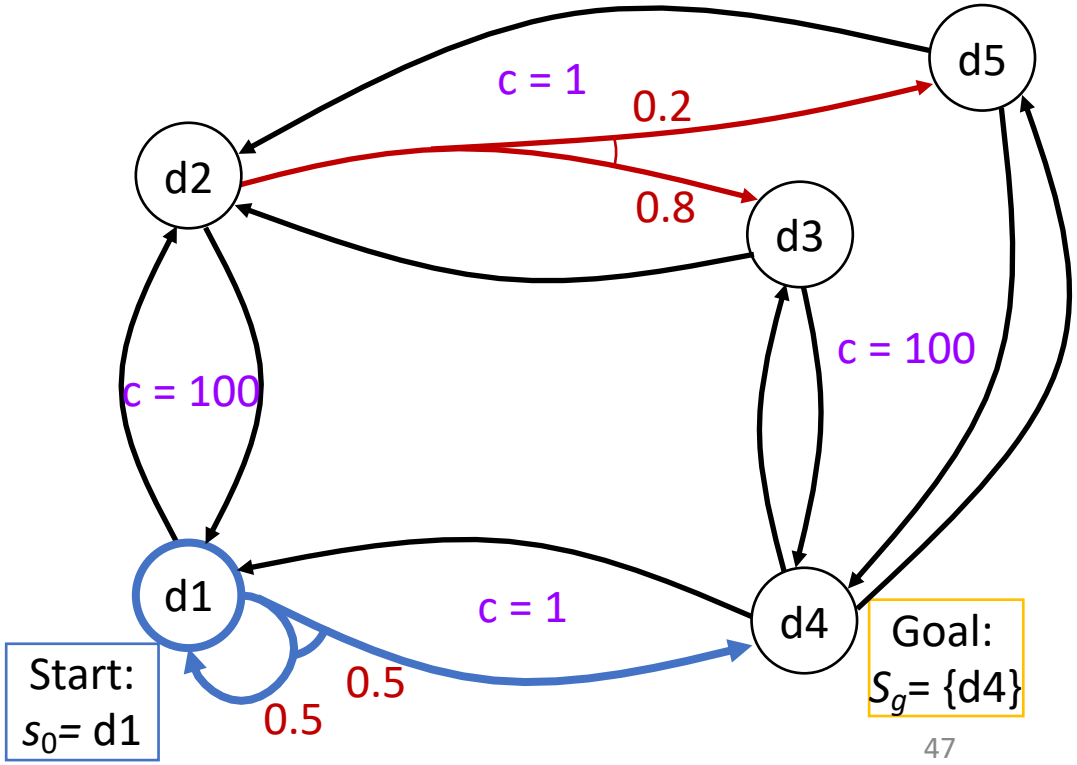
- Recursive equation
  - $V^{\pi_1}(d1) = 100 + V^{\pi_1}(d2) = 100 + 1 + 0.8V^{\pi_1}(d3) + 0.2V^{\pi_1}(d5) = 100 + 1 + 0.8(100) + 0.2(100) = 201$



Safe Solutions

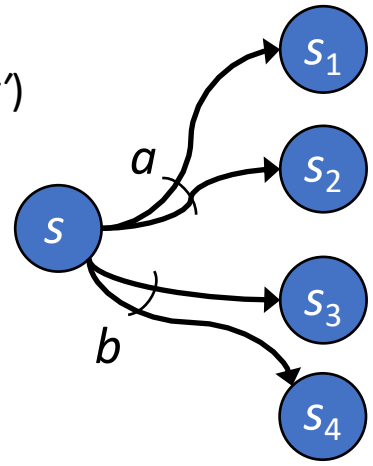
- $\pi_4 = \{(d1, m14)\}$
- Weighted sum of history cost:
  - $\sigma_5 = \langle d1, d4 \rangle$
  - $\mathrm{P}(\sigma_5 | s_0, \pi_5) = \left(\frac{1}{2}\right)^1$
  - $cost(\sigma_5 | s_0, \pi_5) = 1$
  - $\sigma_6 = \langle d1, d1, d4 \rangle$
  - $\mathrm{P}(\sigma_6 | s_0, \pi_6) = \left(\frac{1}{2}\right)^2$
  - $cost(\sigma_6 | s_0, \pi_5) = 2$
  - …
- $V^{\pi_4}(d1)$
  $= \frac{1}{2}(1) + \frac{1}{4}(2) + \ \ldots$
  $= 2$

- Recursive equation
  $V^{\pi_4}(d1)$
  $= 1 + 0.5(0) + 0.5\left(V^{\pi_4}(d1)\right)$
  $\Leftrightarrow 0.5V^{\pi_4}(d1) = 1$
  $\Leftrightarrow \ V^{\pi_4}(d1) = 2$



Planning as Optimization

- Let π and π' be safe solutions
  - π dominates π' if $V^{\pi}(s) \leq V^{\pi'}(s)$ for every $s \in \text{Dom}(\pi) \cap \text{Dom}(\pi')$
- π is optimal if π dominates *every* safe solution
  - If π and π' are both optimal, then $V^{\pi}(s) = V^{\pi'}(s)$ at every state where they are both defined
- $V^*(s)$ = expected cost of getting to goal using an optimal safe solution
- Recall that
  - $V^{\pi}(s) = \begin{cases} 0 & \text{if } s \in S_g \\ cost\big(s, \pi(s)\big) + \sum_{s' \in \gamma(s,\pi(s))} P\big(s'|s, \pi(s)\big) V^{\pi}(s') & \text{otherwise} \end{cases}$
    - Expected cost of following π to goal starting in s



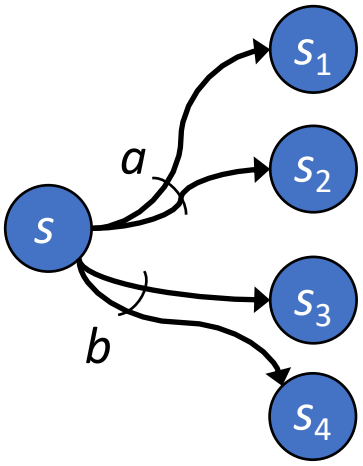
- Optimality principle (Bellman's theorem):
  - $V^*(s) = \begin{cases} 0 & \text{if } s \in S_g \\ \min\limits_{a \in Applicable(S)} \left\{ cost\big(s, \pi(s)\big) + \sum_{s' \in \gamma(s,\pi(s))} P\big(s'|s, \pi(s)\big) V^*(s') \right\} & \text{otherwise} \end{cases}$

Cost to Go

- Let (Σ, s_0, S_g) be a safe SSP
  - I.e., S_g is reachable from every state
  - Same as safely explorable in non-deterministic models
- Let π be a safe solution that is defined at all non-goal states
  - I.e., $Dom(\pi) = S \setminus S_g$
- Let $a \in Applicable(s)$
- Cost-to-go

$$Q^{\pi}(s,a) = cost(s,a) + \sum_{s' \in \gamma(s,a)} P(s'|s,a)\, V^{\pi}(s')$$

  - Expected cost if we start at s, use a, and use π afterward



- For every $s \in S \setminus S_g$, let

$$\pi'(s) \in \operatorname*{argmin}_{a \in Applicable(s)} Q^{\pi}(s,a)$$

Policy Iteration

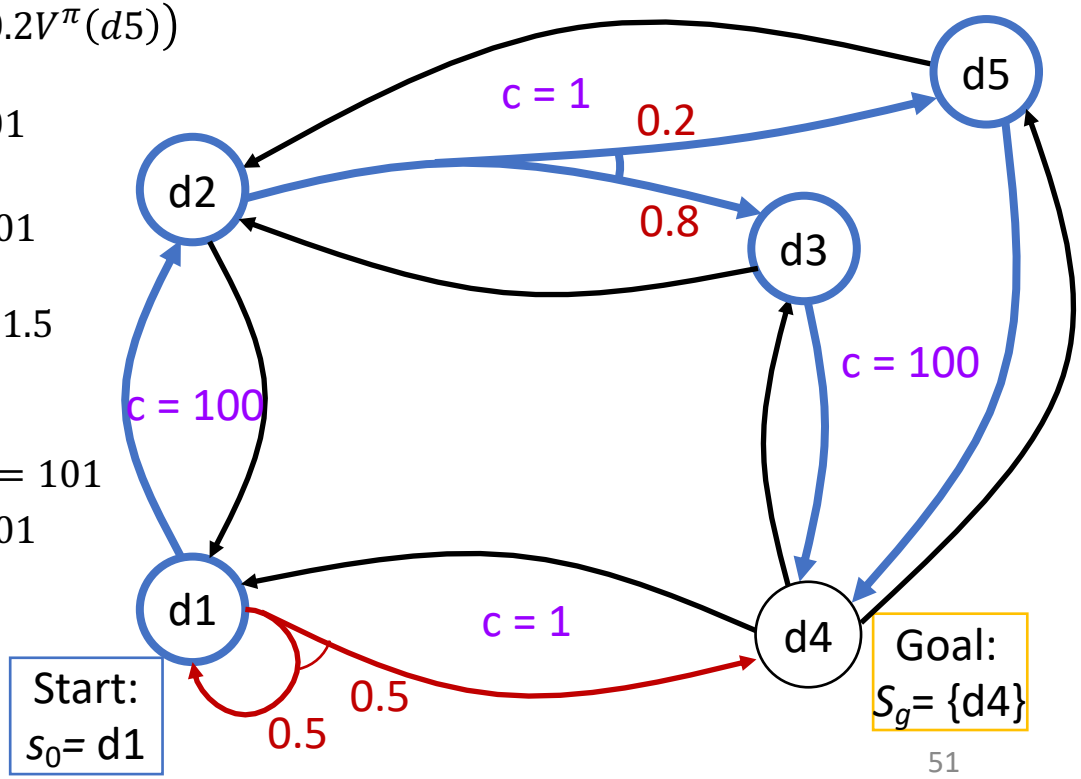
- Converges in a finite number of steps

n equations,
n unknowns,
where $n = |S|$

```
policy-iteration(Σ, s_0, S_g, π_0)
    π ← π_0
    loop
        compute{V^π(s) | s ∈ S}
        for every state s ∈ S \ S_g do
            A ← argmin_{a∈Applicable(s)} Q^π(s, a)
            if π(s) ∈ A then
                π'(s) ← π(s)
            else
                π'(s) ← any action in A
            if π' = π then
                return π
        π ← π'
```

Example

- Start with
  - $\pi = \pi_0 = \{(d1, m12), (d2, m23), (d3, m34), (d5, m54)\}$
- Expected cost
  - $V^{\pi}(d4) = 0$
  - $V^{\pi}(d3) = 100 + V^{\pi}(d4) = 100$
  - $V^{\pi}(d5) = 100 + V^{\pi}(d5) = 100$
  - $V^{\pi}(d2) = 1 + \left(0.8V^{\pi}(d3) + 0.2V^{\pi}(d5)\right) = 101$
  - $V^{\pi}(d1) = 100 + V^{\pi}(d2) = 201$
- Cost-to-go
  - $Q(d1, m12) = 100 + 101 = 201$
  - $Q(d1, m14) = 1 + 0.5(201) + 0.5(0) = 101.5$
    - argmin $= m14$
  - $Q(d2, m23) = 1 + \left(0.8(100) + 0.2(100)\right) = 101$
  - $Q(d2, m21) = 100 + 201 = 301$
    - argmin $= m23$
- Cost-to-go continued
  - $Q(d3, m34) = 100 + 0 = 100$
  - $Q(d3, m32) = 100 + 101 = 201$
    - argmin $= m34$
  - $Q(d5, m54) = 100 + 0 = 100$
  - $Q(d5, m54) = 100 + 101 = 201$
    - argmin $= m54$

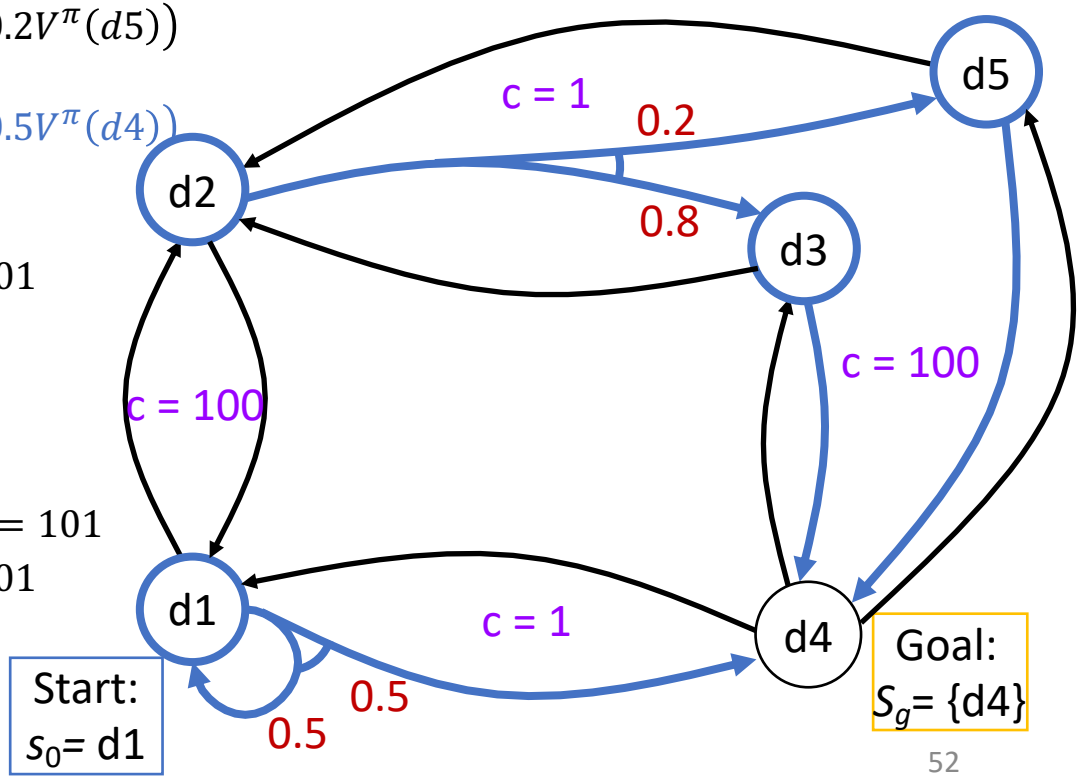


Example

- Continue with
  - $\pi = \{(d1, m14),\ (d2, m23), (d3, m34), (d5, m54)\}$
- Expected cost
  - $V^{\pi}(d4) = 0$
  - $V^{\pi}(d3) = 100 + V^{\pi}(d4) = 100$
  - $V^{\pi}(d5) = 100 + V^{\pi}(d5) = 100$
  - $V^{\pi}(d2) = 1 + \left(0.8V^{\pi}(d3) + 0.2V^{\pi}(d5)\right) = 101$
  - $V^{\pi}(d1) = 1 + \left(0.5V^{\pi}(d1) + 0.5V^{\pi}(d4)\right) = 2$
- Cost-to-go
  - $Q(d1, m12) = 100 + 101 = 201$
  - $Q(d1, m14) = 1 + 0.5(2) + 0.5(0) = 2$
    - argmin $= m14$
  - $Q(d2, m23) = 1 + \left(0.8(100) + 0.2(100)\right) = 101$
  - $Q(d2, m21) = 100 + 201 = 301$
    - argmin $= m23$

π unchanged

- Cost-to-go continued
  - $Q(d3, m34) = 100 + 0 = 100$
  - $Q(d3, m32) = 100 + 101 = 201$
    - argmin $= m34$
  - $Q(d5, m54) = 100 + 0 = 100$
  - $Q(d5, m54) = 100 + 101 = 201$
    - argmin $= m54$



Value Iteration

- $\eta > 0$:
  - for testing approx. convergence
- V_0 is a heuristic fct. for initial values
  - $V_0(s) = 0 \; \forall s \in S_g$
  - E.g., adapt a heuristic from Ch. 2
- V_i = values computed at i'th iteration
- π_i = plan computed from V_i
- Synchronous version computes V_i and π_i from old V_{i-1} and π_{i-1}
- Asynchronous version updates V and π in place
  - New values available immediately
  - More efficient than synchronous version

```
sync-value-iteration(Σ, s_0, S_g, V_0)
    for i = 1,2,… do
        for every state s ∈ S \ S_g do
            for every a ∈ Applicable(s) do
                Q(s,a) ← cost(s,a)+Σ_{s'∈S} PR(s'|s,a) V_{i-1}(s')
            V_i(s) ← min_{a∈Applicable(s)} Q(s,a)
            π_i(s) ← argmin_{a∈Applicable(s)} Q(s,a)
            if max_{s∈S} |V_i(s) - V_{i-1}(s)| ≤ η then
                return π_i
```

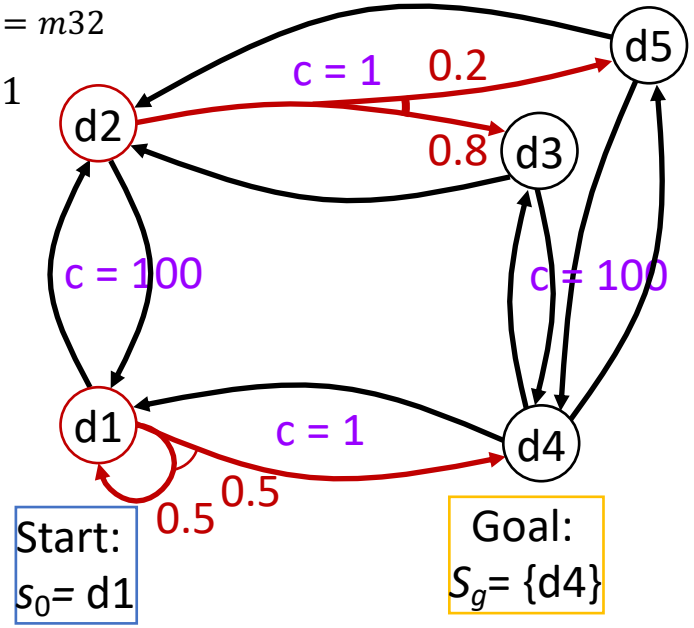
```
async-value-iteration(Σ, s_0, S_g, V_0)
    global π ← ∅
    global V(s) ← V_0(s) ∀ s
    loop
        r ← max_{s∈S\S_g} Bellman-Update(s)
        if r ≤ η then
            return π

Bellman-Update(s)
    ν_old ← V(s)
    for every a ∈ Applicable(s) do
        Q(s,a) ← cost(s,a)+Σ_{s'∈S} PR(s'|s,a) V(s')
    V(s) ← min_{a∈Applicable(s)} Q(s,a)
    π(s) ← argmin_{a∈Applicable(s)} Q(s,a)
    return |V(s) - ν_old|
```

Synchronous

$\eta = 0.2$
$V_0(s) = 0 \; \forall s$

Asynchronous

- $Q(d1, m12) = 100 + 0 = 100$
- $Q(d1, m14) = 1 + \big(0.5(0) + 0.5(0)\big) = 1$
  - $V_1(d1) = 1;\ \pi_1(d1) = m14$
- $Q(d2, m21) = 100 + 0 = 100$
- $Q(d2, m23) = \ 1 + \big(0.2(0) + 0.8(0)\big) = 1$
  - $V_1(d2) = 1;\ \pi_1(d2) = m23$
- $Q(d3, m32) = 1 + 0 = 1$
- $Q(d3, m34) = 100 + 0 = 100$
  - $V_1(d3) = 1;\ \pi_1(d3) = m32$
- $Q(d5, m52) = 1 + 0 = 1$
- $Q(d5, m54) = 100 + 0 = 100$
  - $V_1(d5) = 1;\ \pi_1(d5) = m52$
- $r = \max(1 - 0, 1 - 0{,}1 - 0{,}1 - 0) = 1$



- $Q(d1, m12) = 100 + 0 = 100$
- $Q(d1, m14) = 1 + \big(0.5(0) + 0.5(0)\big) = 1$
  - $V(d1) = 1;\ \pi(d1) = m14$
- $Q(d2, m21) = 100 + 1 = 101$
- $Q(d2, m23) = 1 + \big(0.2(0) + 0.8(0)\big) = 1$
  - $V(d2) = 1;\ \pi(d2) = m23$
- $Q(d3, m32) = 1 + 1 = 2$
- $Q(d3, m34) = 100 + 0 = 100$
  - $V(d3) = 2;\ \pi(d3) = m32$
- $Q(d5, m52) = 1 + 1 = 2$
- $Q(d5, m54) = 100 + 0 = 100$
  - $V(d5) = 2;\ \pi(d5) = m52$
- $r = \max(1 - 0, 1 - 0, 2 - 0, 2 - 0) = 2$

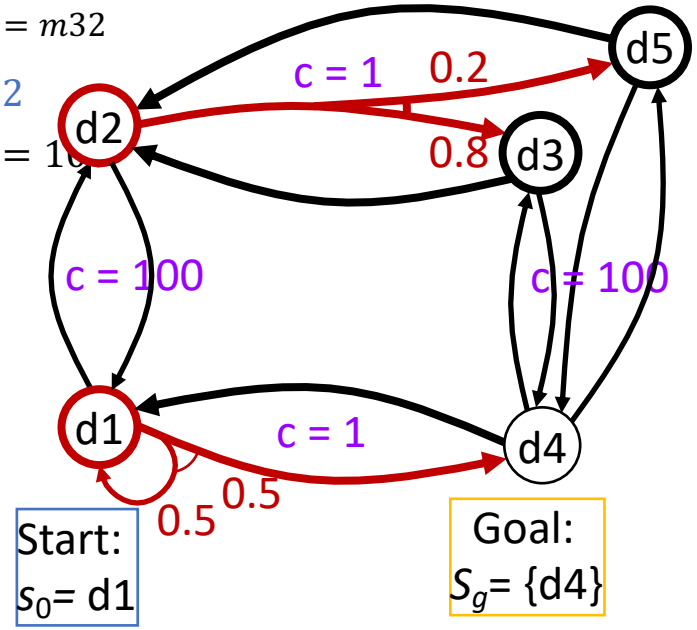
Synchronous

$\eta = 0.2$

Asynchronous

- $Q(d1, m12) = 100 + 1 = 101$
- $Q(d1, m14) = 1 + \big(0.5(1) + 0.5(0)\big) = 1.5$
  - $V_1(d1) = 1.5;\ \pi_1(d1) = m14$
- $Q(d2, m21) = 100 + 1 = 101$
- $Q(d2, m23) = 1 + \big(0.2(1) + 0.8(1)\big) = 2$
  - $V_1(d2) = 2;\ \pi_1(d2) = m23$
- $Q(d3, m32) = 1 + 1 = 2$
- $Q(d3, m34) = 100 + 0 = 100$
  - $V_1(d3) = 2;\ \pi_1(d3) = m32$
- $Q(d5, m52) = 1 + 1 = 2$
- $Q(d5, m54) = 100 + 0 = 1$
  - $V_1(d5) = 1;$ $\pi_1(d5) = m52$
- $r = \max(1.5 - 1,$ $2 - 1, 2 - 1, 2 - 1) = 1$

$V(d1) = 1$
$V(d2) = 1$
$V(d3) = 1$
$V(d5) = 1$



- $Q(d1, m12) = 100 + 1 = 101$
- $Q(d1, m14) = 1 + \big(0.5(1) + 0.5(0)\big) = 1.5$
  - $V(d1) = 1.5;\ \pi(d1) = m14$
- $Q(d2, m21) = 100 + 1.5 = 101.5$
- $Q(d2, m23) = 1 + \big(0.2(2) + 0.8(2)\big) = 3$
  - $V(d2) = 3;\ \pi(d2) = m23$
- $Q(d3, m32) = 1 + 3 = 4$
- $Q(d3, m34) = 100 + 0 = 100$
  - $V(d3) = 4;\ \pi(d3) = m32$
- $Q(d5, m52) = 1 + 3 = 4$
- $Q(d5, m54) = 100 + 0 = 100$
  - $V(d5) = 4;\ \pi(d5) = m52$
- $r = \max(1.5 - 1, 3 - 1,$ $4 - 2, 4 - 2) = 2$

$V(d1) = 1$
$V(d2) = 1$
$V(d3) = 2$
$V(d5) = 2$

Synchronous

$\eta = 0.2$

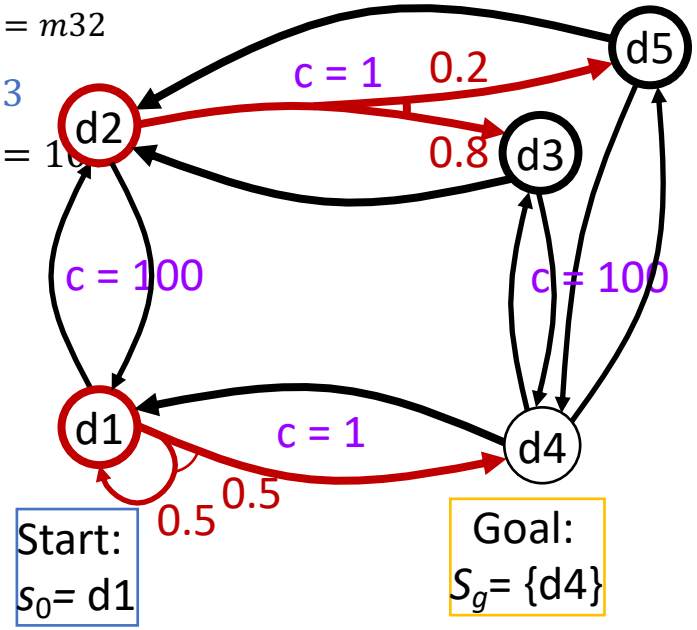
Asynchronous

- $Q(d1, m12) = 100 + 2 = 102$
- $Q(d1, m14) = 1 + \big(0.5(1.5) + 0.5(0)\big) = 1.75$
  - $V_1(d1) = 1.75;\ \pi_1(d1) = m14$
- $Q(d2, m21) = 100 + 1.5 = 101.5$
- $Q(d2, m23) = 1 + \big(0.2(2) + 0.8(2)\big) = 3$
  - $V_1(d2) = 3;\ \pi_1(d2) = m23$
- $Q(d3, m32) = 1 + 2 = 3$
- $Q(d3, m34) = 100 + 0 = 100$
  - $V_1(d3) = 3;\ \pi_1(d3) = m32$
- $Q(d5, m52) = 1 + 2 = 3$
- $Q(d5, m54) = 100 + 0 = 1$
  - $V_1(d5) = 3;$ $\pi_1(d5) = m52$
- $r = \max(1.75 - 1.5,$ $3 - 2, 3 - 2, 3 - 2) = 1$

$V(d1) = 1.5$
$V(d2) = 2$
$V(d3) = 2$
$V(d5) = 2$

- $Q(d1, m12) = 100 + 3 = 103$
- $Q(d1, m14) = 1 + \big(0.5(1.5) + 0.5(0)\big) = 1.75$
  - $V(d1) = 1.75;\ \pi(d1) = m14$
- $Q(d2, m21) = 100 + 1.75 = 101.75$
- $Q(d2, m23) = 1 + \big(0.2(4) + 0.8(4)\big) = 5$
  - $V(d2) = 5;\ \pi(d2) = m23$
- $Q(d3, m32) = 1 + 5 = 6$
- $Q(d3, m34) = 100 + 0 = 100$
  - $V(d3) = 6;\ \pi(d3) = m32$
- $Q(d5, m52) = 1 + 5 = 6$
- $Q(d5, m54) = 100 + 0 = 100$
  - $V(d5) = 6;\ \pi(d5) = m52$
- $r = \max(1.75 - 1.5, 5 - 3,$ $6 - 4, 6 - 4) = 2$

$V(d1) = 1.5$
$V(d2) = 3$
$V(d3) = 4$
$V(d5) = 4$



Synchronous

$\eta = 0.2$

Asynchronous

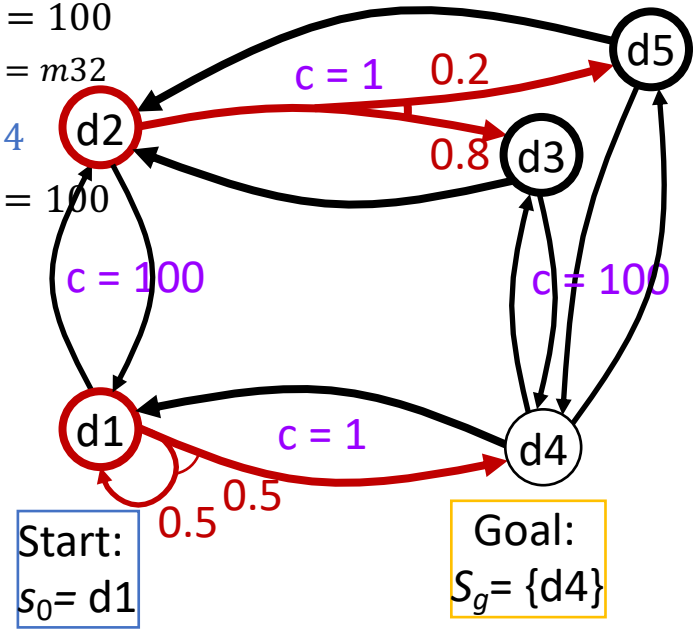
- $Q(d1, m12) = 100 + 3 = 103$
- $Q(d1, m14) = 1 + \big(0.5(1.75) + 0.5(0)\big) = 1.875$
  - $V_1(d1) = 1.875$; $\pi_1(d1) = m14$
- $Q(d2, m21) = 100 + 1.75 = 101.75$
- $Q(d2, m23) = 1 + \big(0.2(3) + 0.8(3)\big) = 4$
  - $V_1(d2) = 4$; $\pi_1(d2) = m23$
- $Q(d3, m32) = 1 + 3 = 4$
- $Q(d3, m34) = 100 + 0 = 100$
  - $V_1(d3) = 4$; $\pi_1(d3) = m32$
- $Q(d5, m52) = 1 + 3 = 4$
- $Q(d5, m54) = 100 + 0 = 100$
  - $V_1(d5) = 4$; $\pi_1(d5) = m52$
- $r = \max(1.875 - 1.75, 4 - 3, 4 - 3, 4 - 3) = 1$

$$V(d1) = 1.75$$
$$V(d2) = 3$$
$$V(d3) = 3$$
$$V(d5) = 3$$

How long before $r \leq \eta$?
How long, if the "vertical" actions cost 10 instead of 100?

- $0.5(0)\big) = 1.875$
- $m14$
- $0 + 1.875 = 101.875$
- $Q(d2, m23) = 1 + \big(0.2(6) + 0.8(6)\big) = 7$
  - $V(d2) = 7$; $\pi(d2) = m23$
- $Q(d3, m32) = 1 + 7 = 8$
- $Q(d3, m34) = 100 + 0 = 100$
  - $V(d3) = 8$; $\pi(d3) = m32$
- $Q(d5, m52) = 1 + 7 = 8$
- $Q(d5, m54) = 100 + 0 = 100$
  - $V(d5) = 8$; $\pi(d5) = m52$
- $r = \max(1.875 - 1.75, 7 - 5, 8 - 6, 8 - 6) = 2$

$$V(d1) = 1.75$$
$$V(d2) = 5$$
$$V(d3) = 6$$
$$V(d5) = 6$$



Discussion

- Policy iteration computes new π in each iteration; computes V^{π} from π
  - More work per iteration than value iteration
    - Needs to solve a set of simultaneous equations
  - Usually converges in a smaller number of iterations
- Value iteration
  - Computes new V in each iteration; chooses π based on V
  - New V is a revised set of heuristic estimates
    - Not V^{π} for π or any other policy
  - Less work per iteration: does not need to solve a set of equations
  - Usually takes more iterations to converge
- At each iteration, both algorithms need to examine the entire state space
  - Number of iterations polynomial in $|S|$, but $|S|$ may be quite large
- Next: use search techniques to avoid searching the entire space

Summary

- SSPs
- Solutions, closed solutions, histories
- Unsafe solutions, acyclic safe solutions, cyclic safe solutions
- Expected cost, planning as optimization
- Policy iteration
- Value iteration (synchronous, asynchronous)
  - Bellman-update

Outline

Markov decision process (MDP)

- Markov property
- Sequence of actions, history, policy
- Value iteration, policy iteration

6.2 Stochastic shortest path problems

- Safe/unsafe policies
- Optimality
- Policy iteration, value iteration

6.3 Heuristic search algorithms

- Best-first search
- Determinisation

6.4 Online probabilistic planning

- Lookahead

AO*

- Best-first search for acyclic domains

Requires acyclic Σ

not in book

```
AO*(Σ, s_0, S_g, V_0)
    global π ← ∅; V(s_0) ← V_0(s_0); Envelope ← {s_0}
    while leaves(s_0,π) \ S_g ≠ ∅ do
        select s ∈ leaves(s_0,π) \ S_g
        for all a ∈ Applicable(s) do
            for all s' ∈ γ(s,a) \ Envelope do
                V(s') ← V_0(s')
                Add s' to Envelope
        AO-Update(s)
    return π
```

no π-descendants in Z but s itself
• ensures bottom-up updates

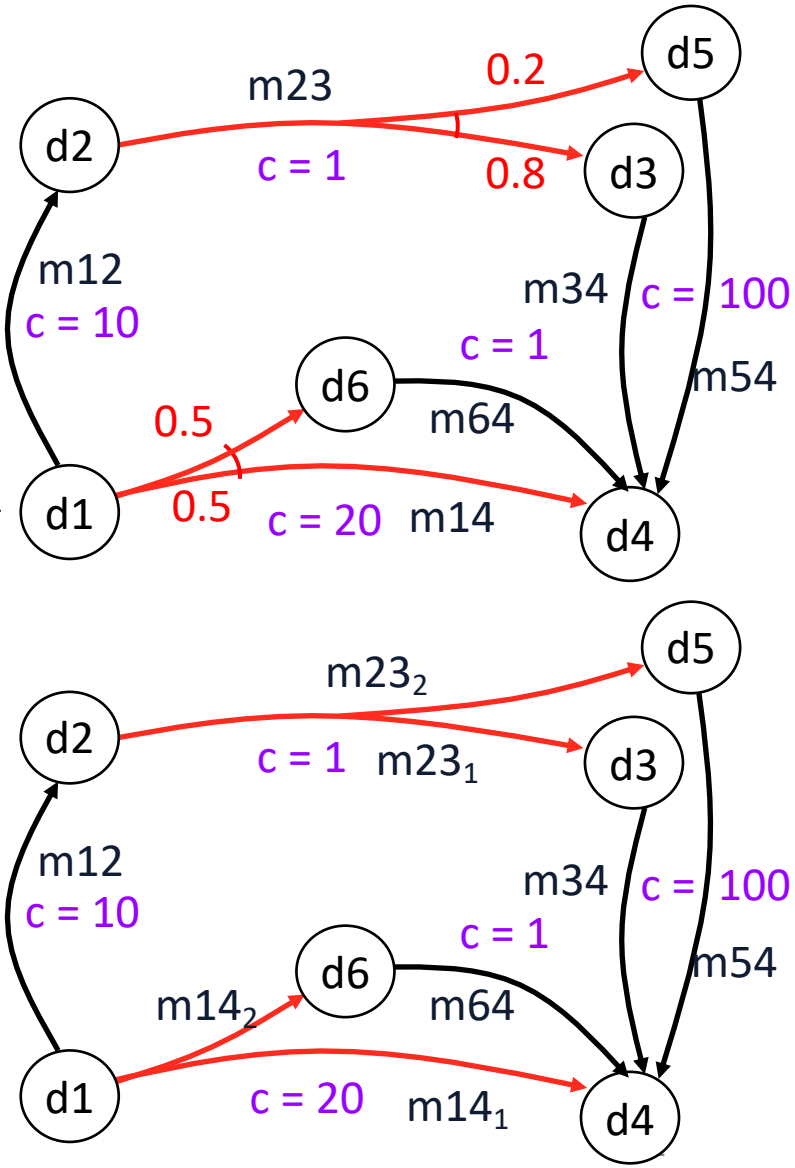
```
AO-Update(s)
    Z ← {s}  // nodes that need updating
    while Z ≠ ∅ do
        select s ∈ Z s.t. γ̂(s,π(s))∩Z = {s}
        remove s from Z
        Bellman-Update(s)
        Z ← Z ∪ {s' ∈ Envelope | s ∈ γ(s',π)}
```

the states "just above" s

```
Bellman-Update(s)
    for every a ∈ Applicable(s) do
        Q(s,a) ← cost(s,a)+Σ_{s'∈S} PR(s'|s,a)V(s')
    V(s) ← min_{a∈Applicable(s)} Q(s,a)
    π(s) ← argmin_{a∈Applicable(s)} Q(s,a)
```

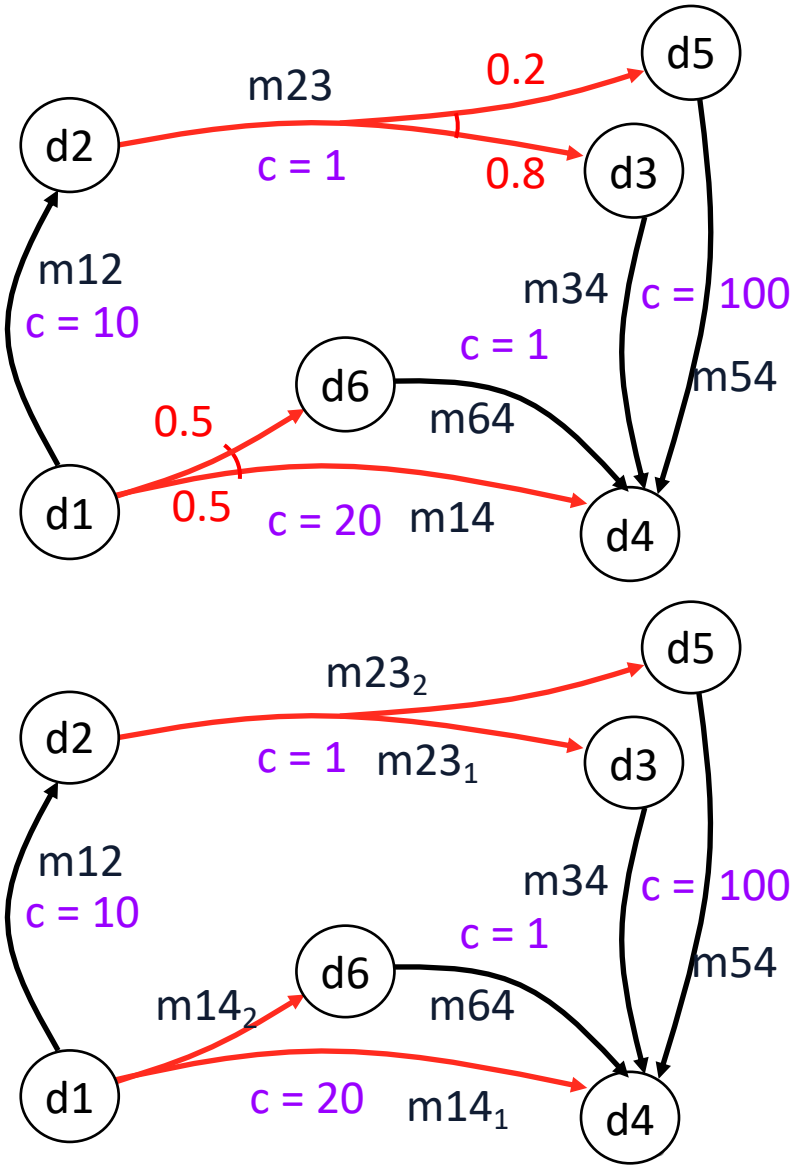
Heuristics through Determinization

- What to use for V_0?
  - One possibility: classical planner
  - Need to convert nondeterministic actions into something the classical planner can use
- Determinize the actions
  - Suppose $\gamma(s,a) = \{s_1, \ldots, s_n\}$
  - $Det(s,a) = \{n \text{ actions } a_1, a_2, \ldots, a_n\}$
    - $\gamma_d(s,a_i) = s_i$
    - $cost_d(s,a_i) = cost(s,a)$
- Classical domain $\Sigma_d = (S, A_d, \gamma_d, cost_d)$
  - S = same as in Σ
  - $A_d = \bigcup_{a \in A, s \in S} Det(s,a)$
  - γ_d and $cost_d$ as above



Heuristics through Determinization

- Suppose we want $V_0(s)$
- Call classical planner on (Σ_d, s, S_g)
  - Get plan $p = \langle a_1, a_2, \dots, a_n \rangle$
  - Go through states $\langle s, s_1, \dots, s_n \rangle$
    - $s_1 = \gamma(s, a_1), s_2 = \gamma(s_1, a_2), \dots$
  - Return $V_0(s) = cost(p) = \sum_i cost(a_i)$
- If the classical planner always returns optimal plans, then V_0 is admissible
- Outline of proof:
  - Let π be a safe solution in Σ
  - Every acyclic execution of π corresponds to a solution plan p' in Σ_d
    - Must have cost ≥ $V_0(s)$
    - Otherwise the classical planner would have chosen p' instead of p



LAO*

- Best-first search for both cyclic and acyclic domains

Σ may be cyclic or acyclic

not in book

all π-ancestors of s in *Envelope*

Asynchronous value iteration, restricted to Z

Different compared to AO*

```
LAO*(Σ, s0, Sg, V0)
    global π ← ∅; V(s0) ← V0(s0); Envelope ← {s0}
    loop
        if leaves(s0,π) ⊆ Sg ≠ ∅ then
            return π
        select s ∈ leaves(s0,π) \ Sg
        for all a ∈ Applicable(s) do
            for all s' ∈ γ(s,a) \ Envelope do
                V(s') ← V0(s'); add s' to Envelope
        LAO-Update(s)
    return π
```

```
LAO-Update(s)
    Z ← {s}∪{s' ∈ Envelope | s ∈ γ(s',π)}
    loop
        r ← max_{s∈Z} Bellman-Update(s)
        if leaves(s0,π) changed or r≤η then
            break
```

```
Bellman-Update(s)
    v_old ← V(s)
    for every a ∈ Applicable(s) do
        Q(s,a) ← cost(s,a)+Σ_{s'∈S} PR(s'|s,a) V(s')
    V(s) ← min_{a∈Applicable(s)} Q(s,a)
    π(s) ← argmin_{a∈Applicable(s)} Q(s,a)
    return |V(s)-v_old|
```

LAO* Example

1st iteration of main loop:

- Expand d1: add d2 and d4 to Envelope
- Call LAO-Update(d1)
  - π is empty, so $Z = \{d1\}$

  Iteration 1:
  - $Q(d1, m12) = 100 + 0 = 100$
  - $Q(d1, m14) = 1 + \big(0.5(0) + 0.5(0)\big) = 1$
    - $V(d1) = 1;\ \pi(d1) = m14; r = V(d1)\ - 0 = 1$

  Iteration 2:
  - $Q(d1, m12) = 100 + 0 = 100$
  - $Q(d1, m14) = 1 + \big(0.5(1) + 0.5(0)\big) = 1.5$
    - $V(d1) = 1.5;\ \pi(d1) = m14;$ $r\ =\ 1.5 - 1 = 0.5$

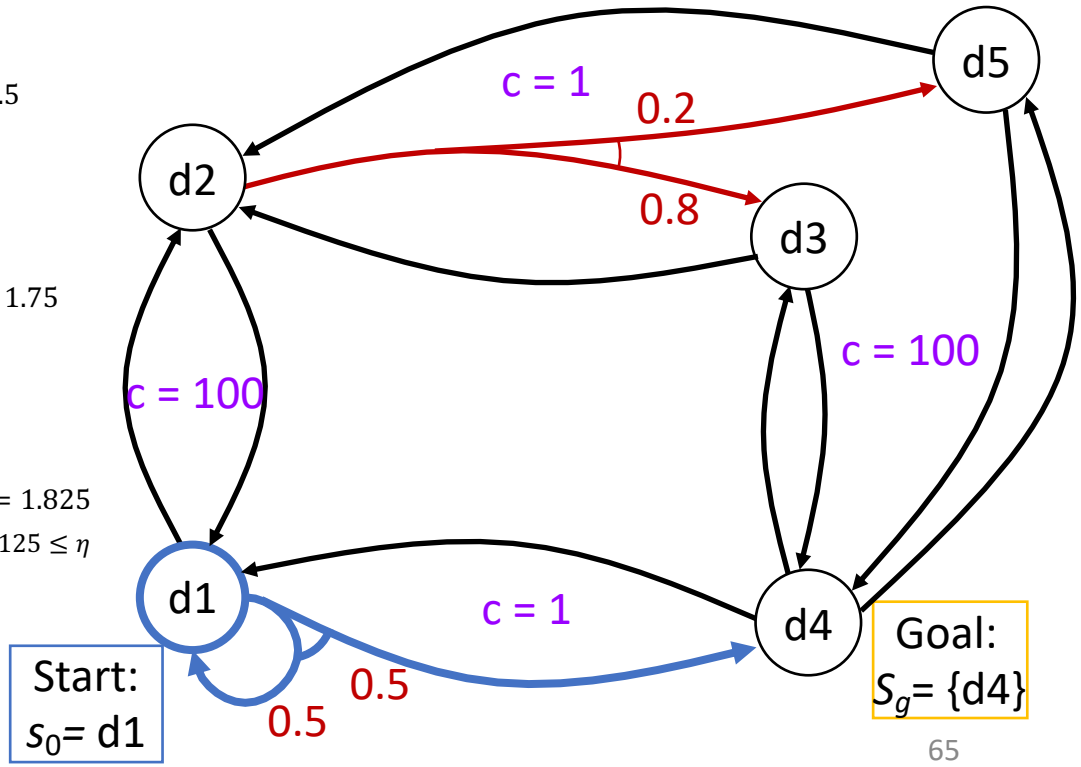
  Iteration 3:
  - $Q(d1, m12) = 100 + 0 = 100$
  - $Q(d1, m14) = 1 + \big(0.5(1.5) + 0.5(0)\big) = 1.75$
    - $V(d1) = 1.75; \pi(d1) = m14;$ $r = 1.75 - 1.5 = 0.25$

  Iteration 4:
  - $Q(d1, m12) = 100 + 0 = 100$
  - $Q(d1, m14) = 1 + \big(0.5(1.75) + 0.5(0)\big) = 1.825$
    - $V(d1) = 1.825; \pi(d1) = m14; r = 0.125 \leq \eta$
  - LAO-Update returns

2nd iteration of main loop:

- $leaves(\pi) = \{d4\} \subseteq S_g$
- return π

$\eta = 0.2$
$V_0(s) = 0\ \forall s$



Summary

- AO*
  - Acyclic
- LAO*
  - (A)cyclic

Outline

Markov decision process (MDP)

- Markov property
- Sequence of actions, history, policy
- Value iteration, policy iteration

6.2 Stochastic shortest path problems

- Safe/unsafe policies
- Optimality
- Policy iteration, value iteration

6.3 Heuristic search algorithms

- Best-first search
- Determinisation

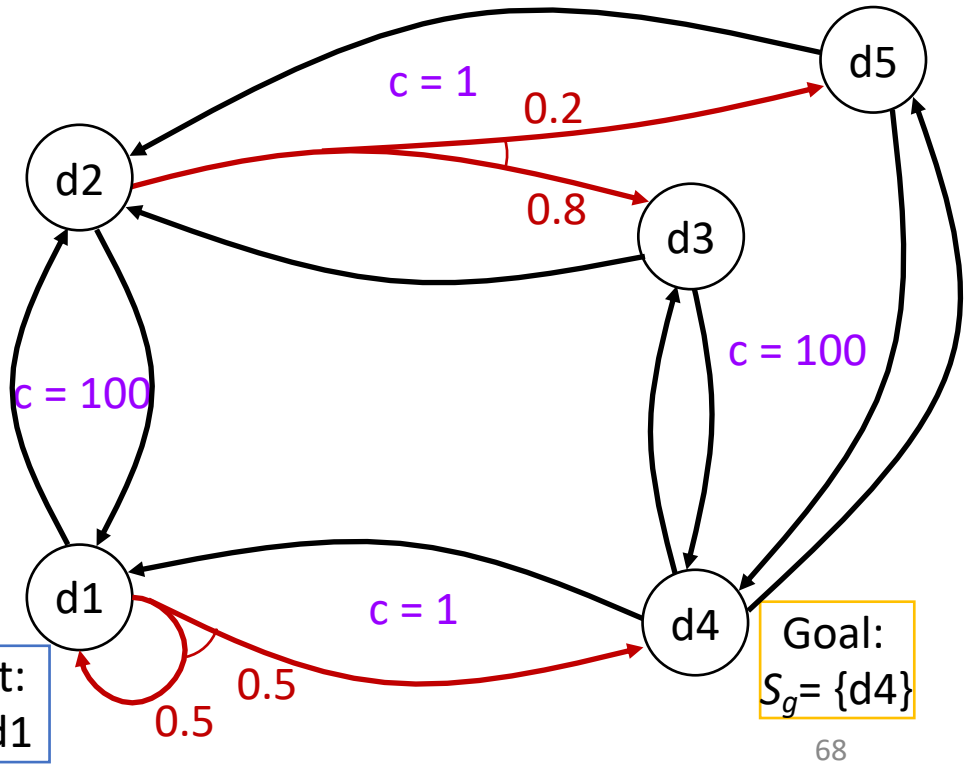
6.4 Online probabilistic planning

- Lookahead

Planning and Acting

- Same as in Ch. 2, except $s = \xi$
  - Could use $s \leftarrow$ abstraction of ξ as in Ch. 2
- Could also use Run-Lazy-Lookahead or Run-Concurrent-Lookahead
- What to use for Lookahead?
  - AO*, LAO*, …
    - Modify to search part of the space
  - Classical planner running on determinized domain
  - Stochastic sampling algorithms

```
Run-Lookahead(Σ, s_0, S_g)
    s ← s_0
    while s ∉ S_g and Applicable(s) ≠ ∅ do
        a ←Lookahead(s, θ)
        perform action a
        s ← observe resulting state
```



Planning and Acting

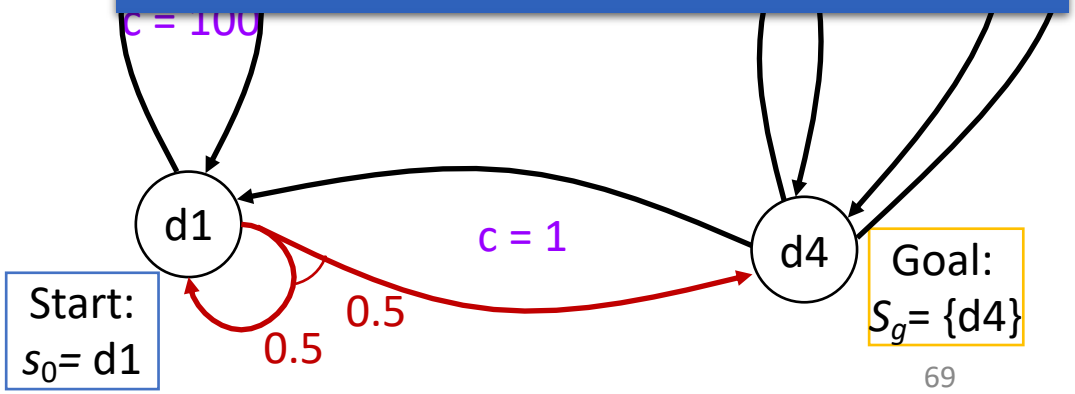
- If Lookahead = classical planner on determinized domain
  ⇒ FS-Replan (Ch. 5)
- Problem: Forward-search may choose a plan that depends on low-probability outcome
- RFF algorithm (see book) attempts to alleviate this

Run-Lookahead(Σ, s_0, S_g)

```
s ← s0
while s ∉ Sg and Applicable(s) ≠ ∅ do
    a ←Lookahead(s, θ)
    perform action a
    s ← observe resulting state
```

FS-Replan(Σ, s, S_g)

```
πd ← ∅
while s ∉ Sg and Applicable(s) ≠ ∅ do
    if πd undefined for s then
        πd ← Forward-Search(Σd, s, Sg)
        if πd = failure then
            return failure
    perform action πd(s)
    s ← observe resulting state
```



Summary

- Planning and Acting
  - Run-Lookahead

Outline per the Book

Markov decision problem (MDP)

- Markov property
- Sequence of actions, history, policy
- Value iteration, policy iteration

6.2 Stochastic shortest path problems

- Safe/unsafe policies
- Optimality
- Policy iteration, value iteration

6.3 Heuristic search algorithms

- Best-first search
- Determinisation

6.4 Online probabilistic planning

- Lookahead

⇒ Next: Complex Decision Making