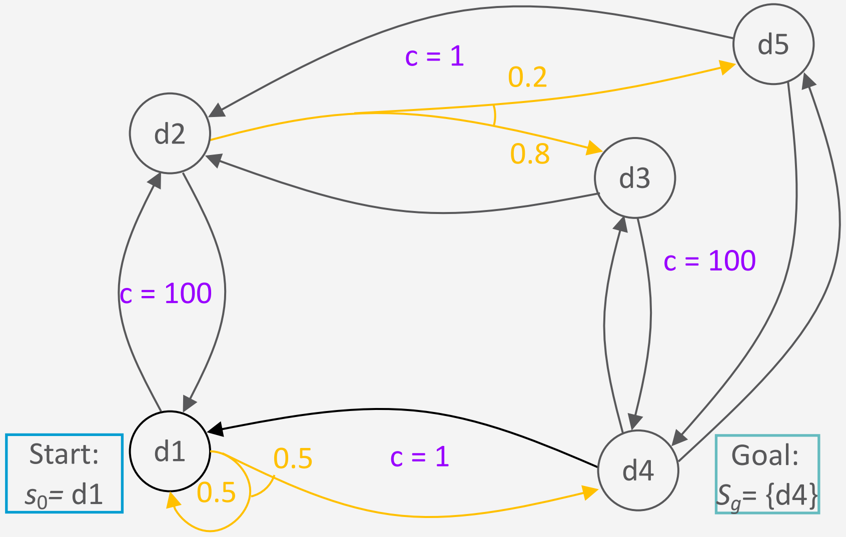
Universität
Münster

Automated Planning and Acting

Probabilistic Models



living.knowledge

Tanya Braun
Research Group Data Science, Computer Science Department

Content: Planning and Acting

Outline per the Book

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

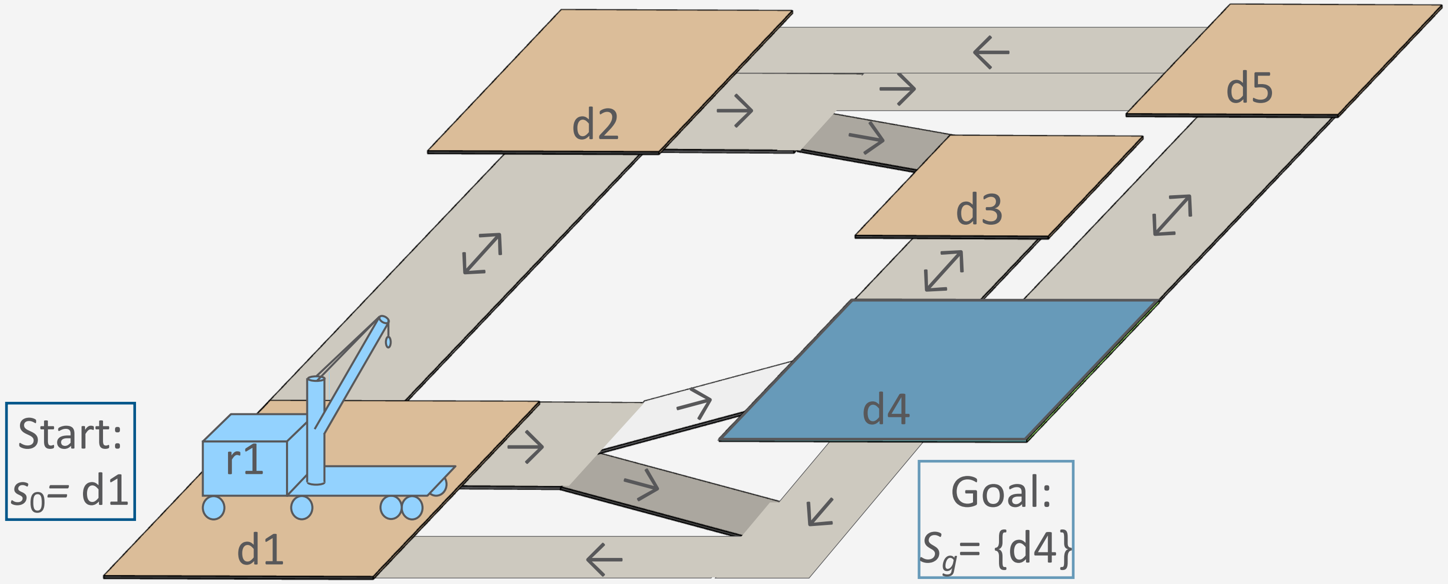
- Lookahead

Probabilistic Planning Domain

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

Example

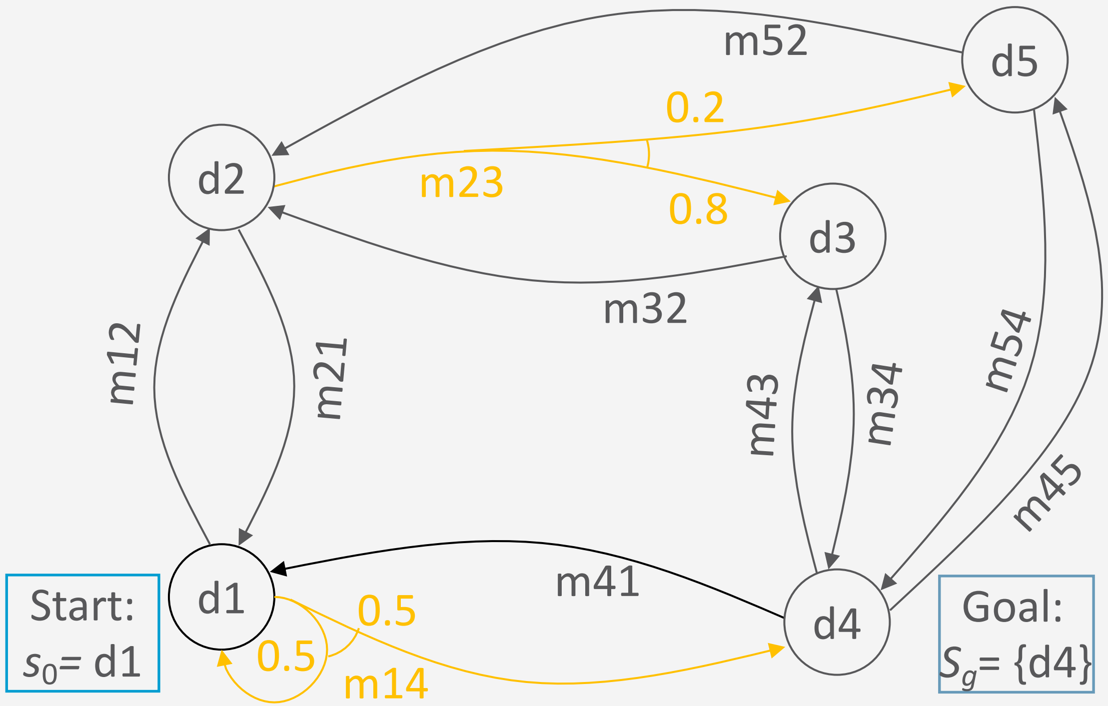
- Robot $r1$ starts at $d1$
- Objective: get to $d4$
- $r1$ has unreliable steering, especially on hills
  - May slip and go elsewhere
    - $m14$: $\mathrm{P}(d4 \mid d1, m14) = 0.5$
      $\mathrm{P}(d1 \mid d1, m14) = 0.5$
    - $m23$: $\mathrm{P}(d3 \mid d2, m23) = 0.8$
      $\mathrm{P}(d5 \mid d2, m23) = 0.2$
    - $m21$: $\mathrm{P}(d2 \mid d1, m21) = 1$
      - $m34, m41, m43, m45, m52, m54$: like $m21$
- Simplified state names:
  - Write $\{loc(r1) = d2\}$ as $d2$
- Simplified action names:
  - Write $move(r1, d2, d3)$ as $m23$



Policies, Problems, Solutions

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

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



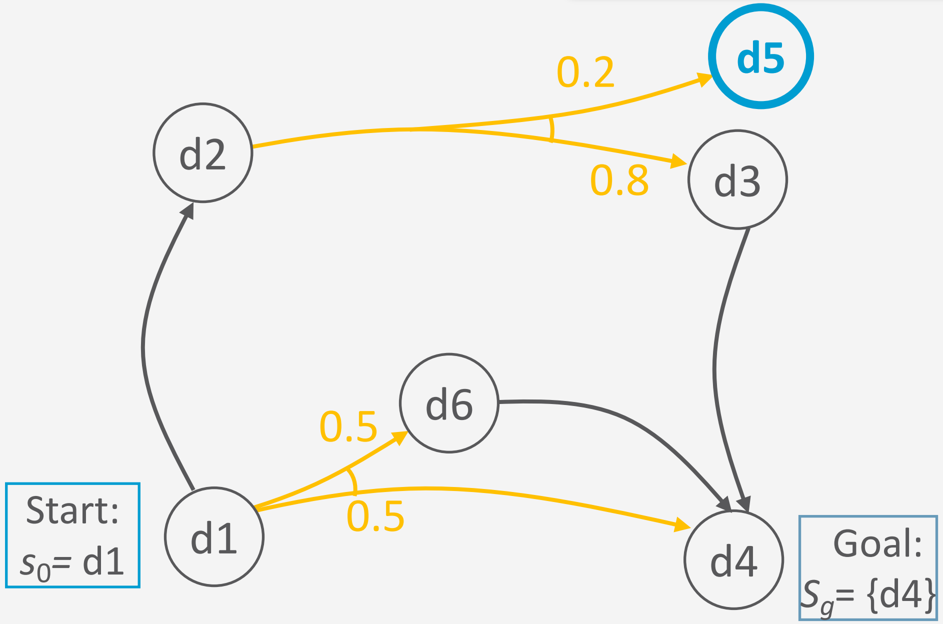
Notation and Terminology

- As before:
  - 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
  - $leaves(s, \pi) = \hat{\gamma}(s, \pi) \backslash \mathrm{dom}(\pi)$
- Solution policy π is closed if it does not stop at non-goal states unless there is no way to continue
  - Formally, for every state $s \in \hat{\gamma}(s, \pi)$, either
    - $s \in \mathrm{dom}(\pi)$ (i.e., π specifies an action at s),
    - $s \in S_g$ (i.e., s is a goal state), or
    - $Applicable(s) = \emptyset$ (i.e., there are no applicable actions at s)

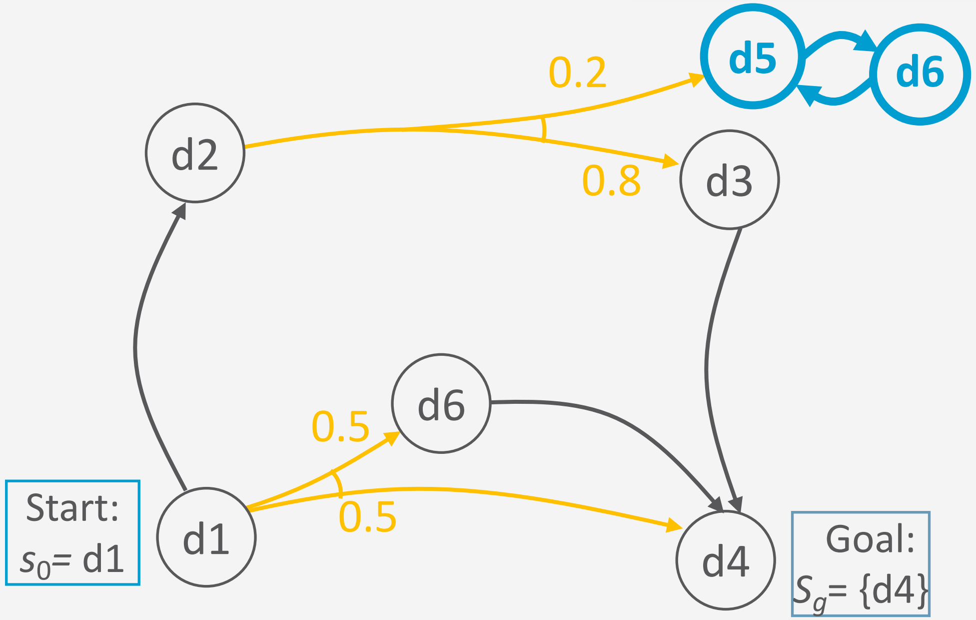
Dead Ends

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

Explicit dead end

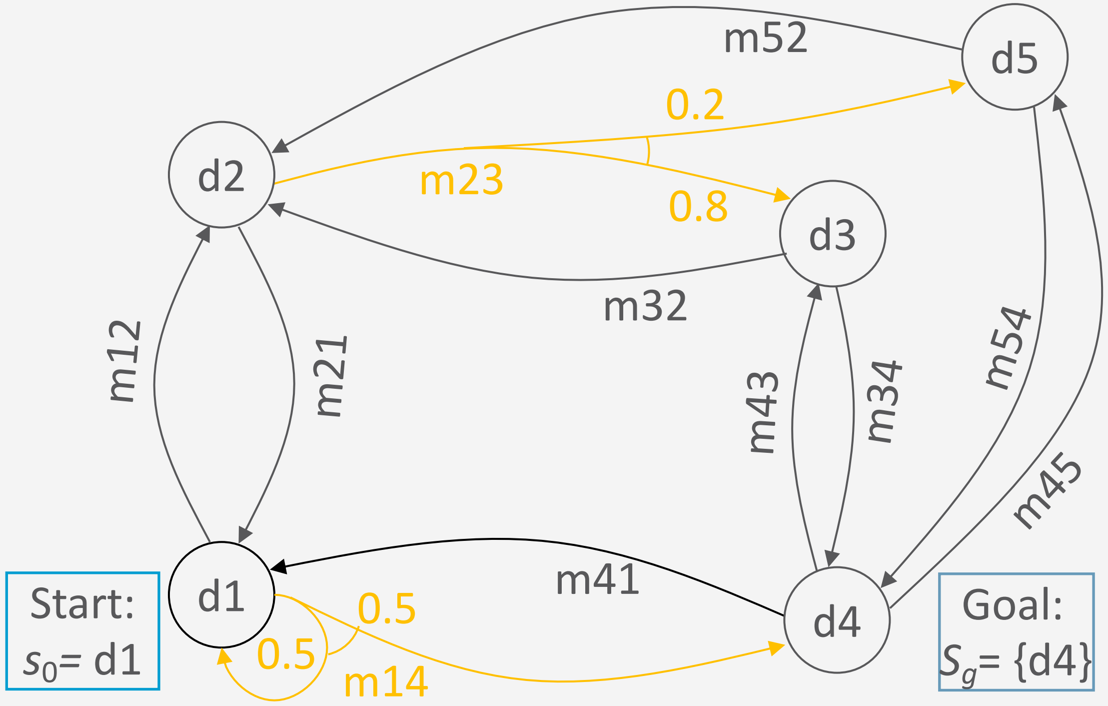


Implicit dead end



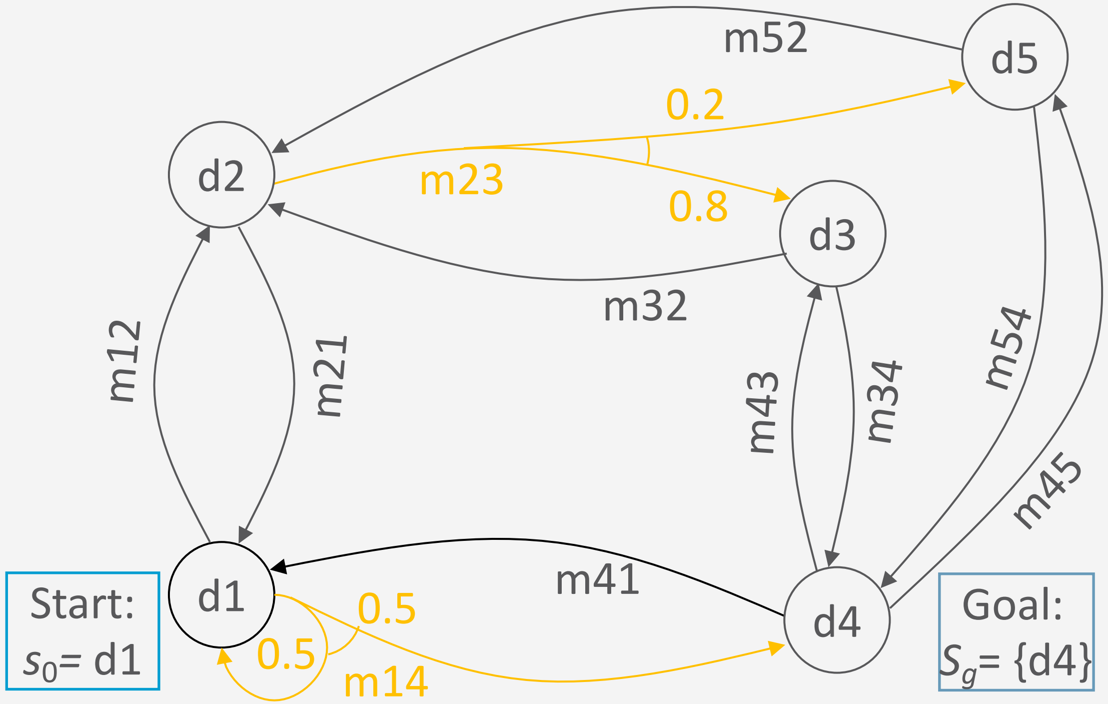
Histories

- 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}



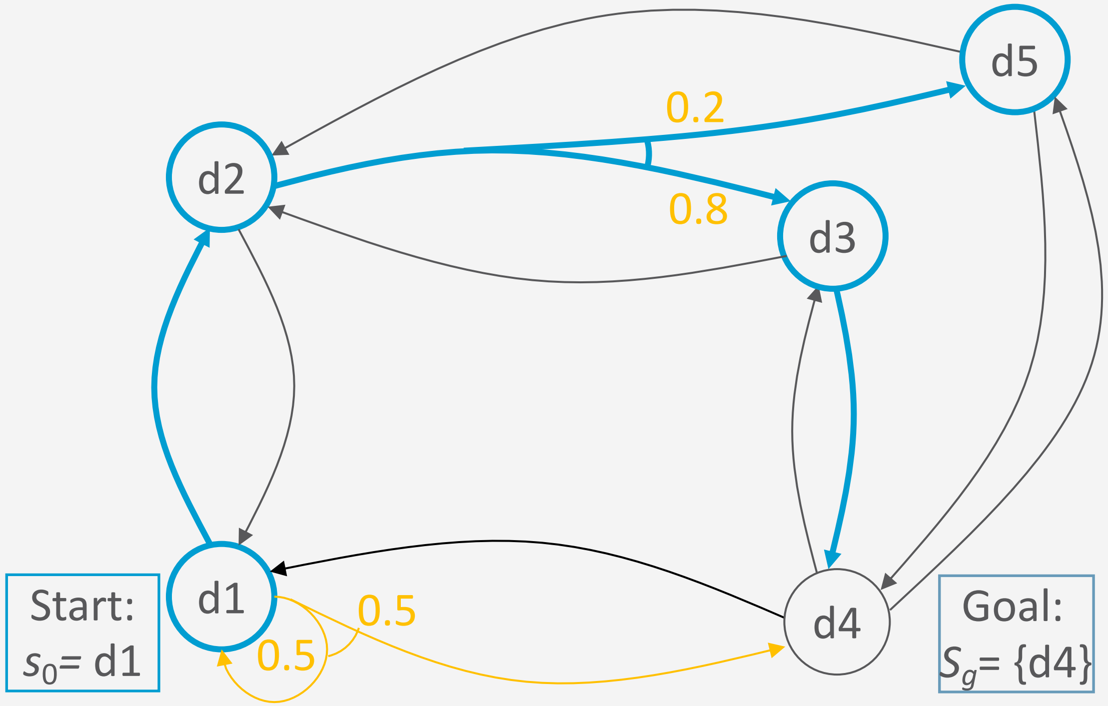
Histories

- If $\sigma \in H(s, \pi)$, then
$$P(\sigma \mid s, \pi) = \prod_i P\big(s_{i+1} \mid s_i, \pi(s_i)\big)$$
  - Thus
$$\sum_{\sigma \in H(s,\pi)} P(\sigma \mid s, \pi) = 1$$
- Probability of reaching a goal:
$$P\big(S_g \mid s, \pi\big) = \sum_{\substack{\sigma \in H(s,\pi), \\ \sigma \text{ ends at } s \in S_g}} P(\sigma \mid s, \pi)$$



Unsafe Solutions

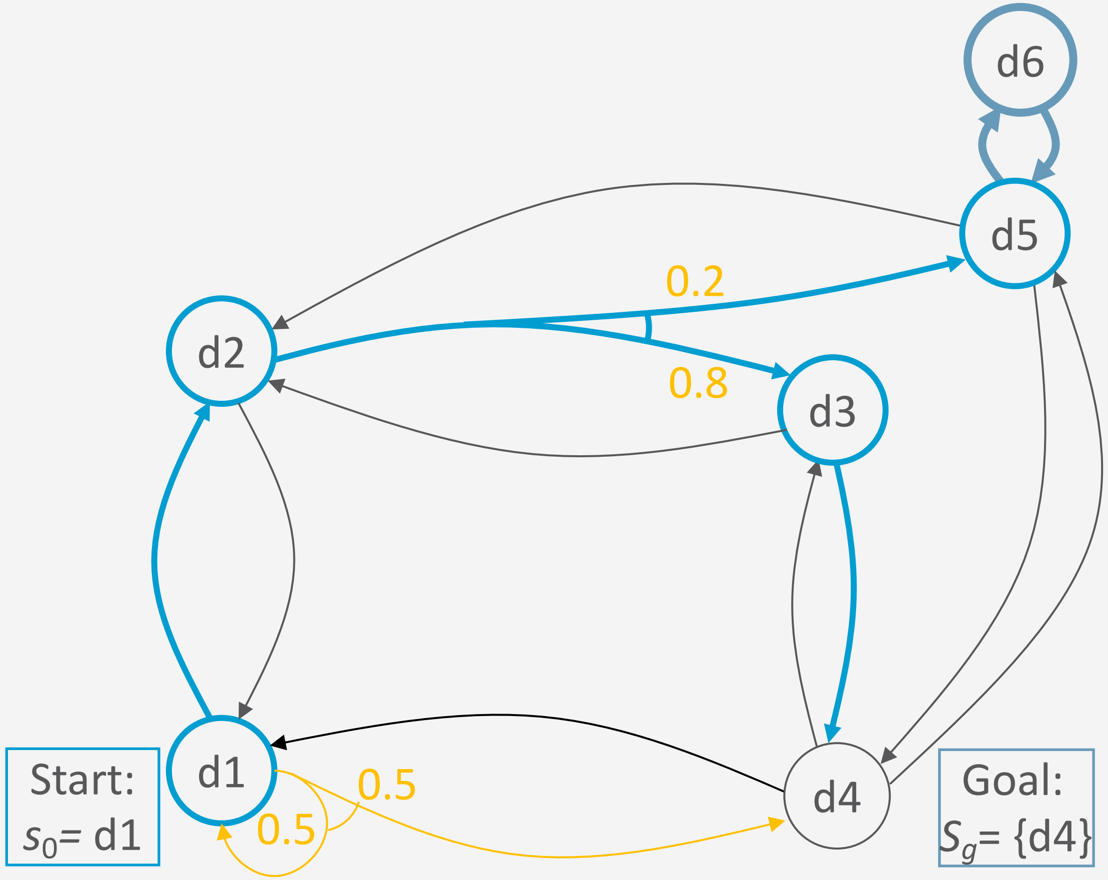
- Unsafe solution: $0 < P(S_g | s_0, \pi) < 1$
- Example:
  - $\pi_1 = \{(d1, m12), (d2, m23), (d3, m34)\}$
  - $H(s_0, \pi_1)$ contains two histories:
    - $\sigma_1 = \langle d1, d2, d3, d4 \rangle$
    - $P(\sigma_1 | s_0, \pi_1)$
      $= 1 \cdot 0.8 \cdot 1 = 0.8$
    - $\sigma_2 = \langle d1, d2, d5 \rangle$
    - $P(\sigma_2 | s_0, \pi_1)$
      $= 1 \cdot 0.2 = 0.2$
  - $P(S_g | s_0, \pi_1)$
    $= 0.8$



Unsafe Solutions

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

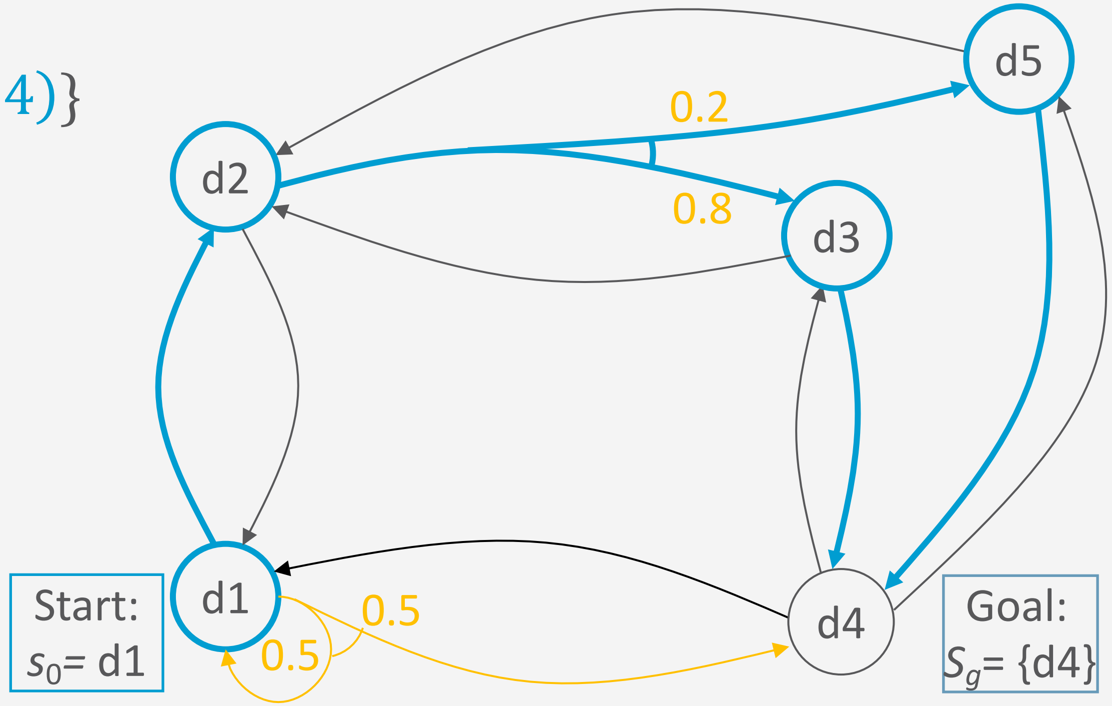
- Example:
  - $\pi_2 = \{(d1, m12), (d2, m23), (d3, m34), (d5, m56), (d6, m65)\}$
  - $H(s_0, \pi_2)$ contains two histories:
    - $\sigma_1 = \langle d1, d2, d3, d4 \rangle$
    - $P(\sigma_1|s_0, \pi_2) = 1 \cdot 0.8 \cdot 1 = 0.8$
    - $\sigma_3 = \langle d1, d2, d5, d6, \dots \rangle$
    - $P(\sigma_3|s_0,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



Safe Solutions

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

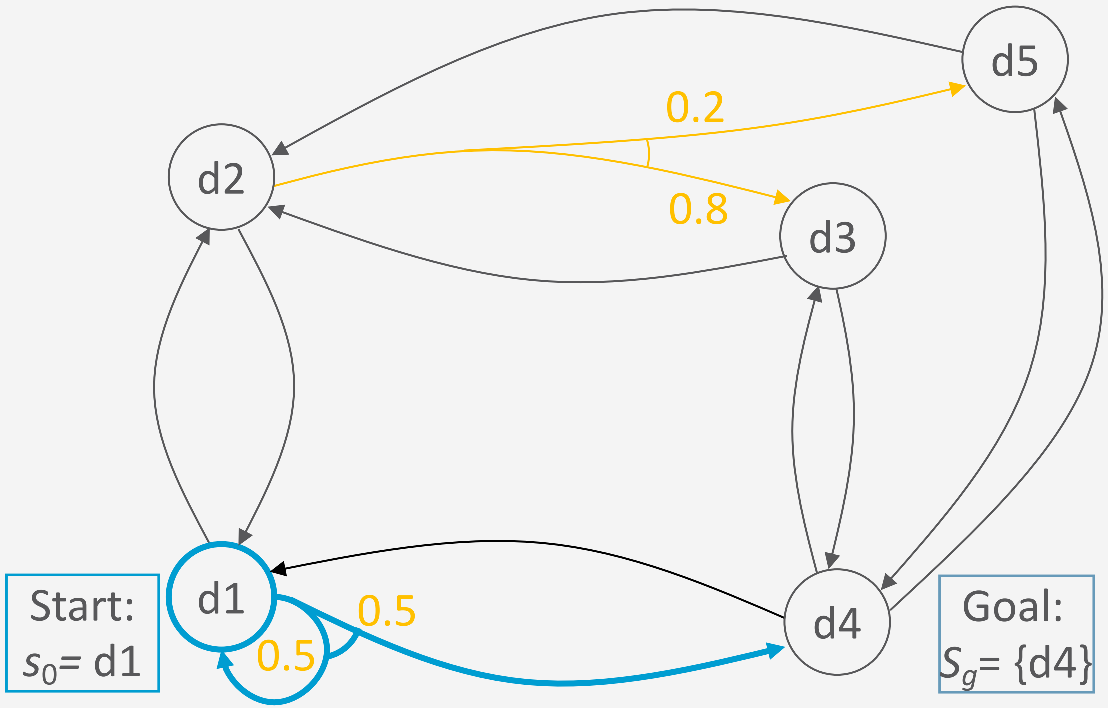
- An acyclic safe solution:
  - $\pi_3 = \{(d1, m12), (d2, m23), (d3, m34), (d5, m54)\}$
  - $H(s_0, \pi_3)$ contains two histories:
    - $\sigma_1 = \langle d1, d2, d3, d4 \rangle$
    - $P(\sigma_1|s_0, \pi_3) = 1 \cdot 0.8 \cdot 1 = 0.8$
    - $\sigma_4 = \langle d1, d2, d5, d4 \rangle$
    - $P(\sigma_4|s_0,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



Safe Solutions

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

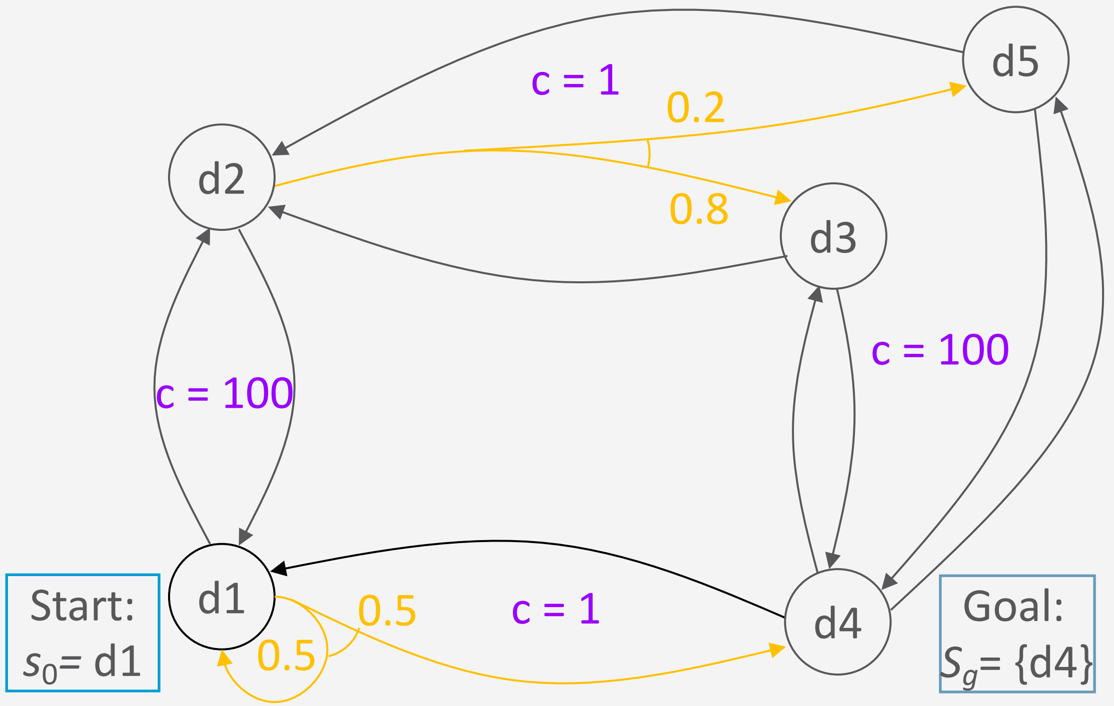
- A cyclic safe solution:
  - $\pi_4 = \{(d1, m14)\}$
  - $H(s_0, \pi_4)$ contains infinitely many histories:
    - $\sigma_5 = \langle d1, d4 \rangle$
    - $P(\sigma_5|s_0, \pi_4) = 0.5 = (^1/_2)^1$
    - $\sigma_6 = \langle d1, d1, d4 \rangle$
    - $P(\sigma_6|s_0, \pi_4) = 0.5 \cdot 0.5 = (^1/_2)^2$
    - …
  - $P\left(S_g|s_0,\pi_4\right) = \frac{1}{2} + \frac{1}{4} + \ \ldots = 1$



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

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



Expected Cost

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

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

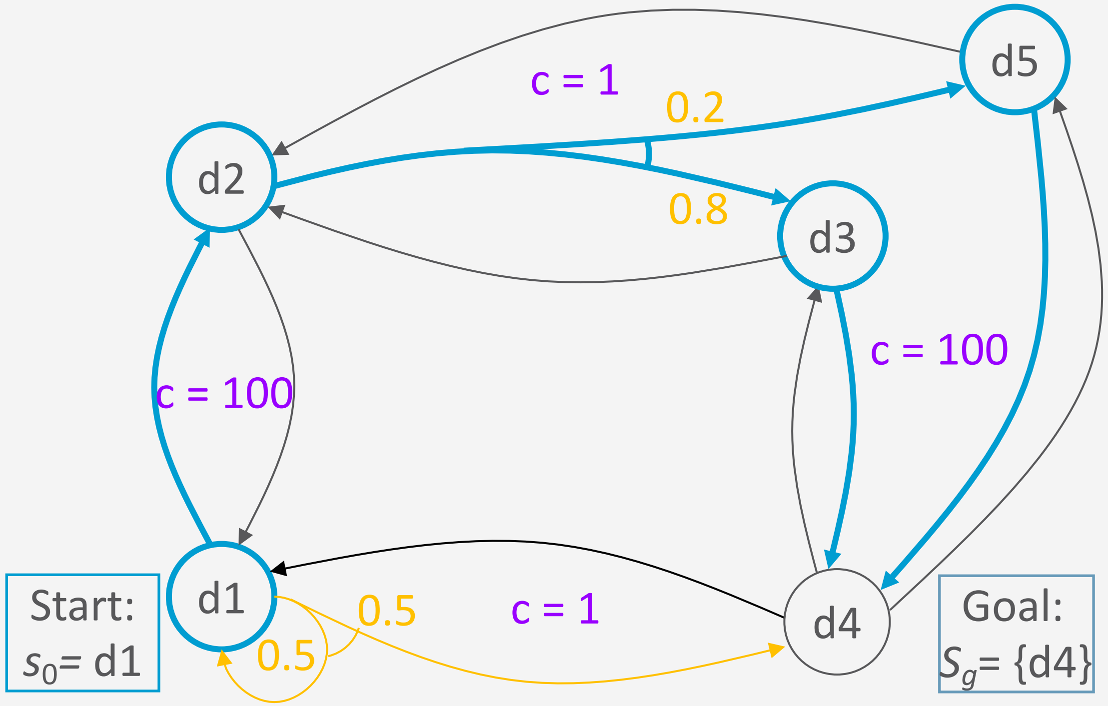
  - Recursive formulation

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

Example

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

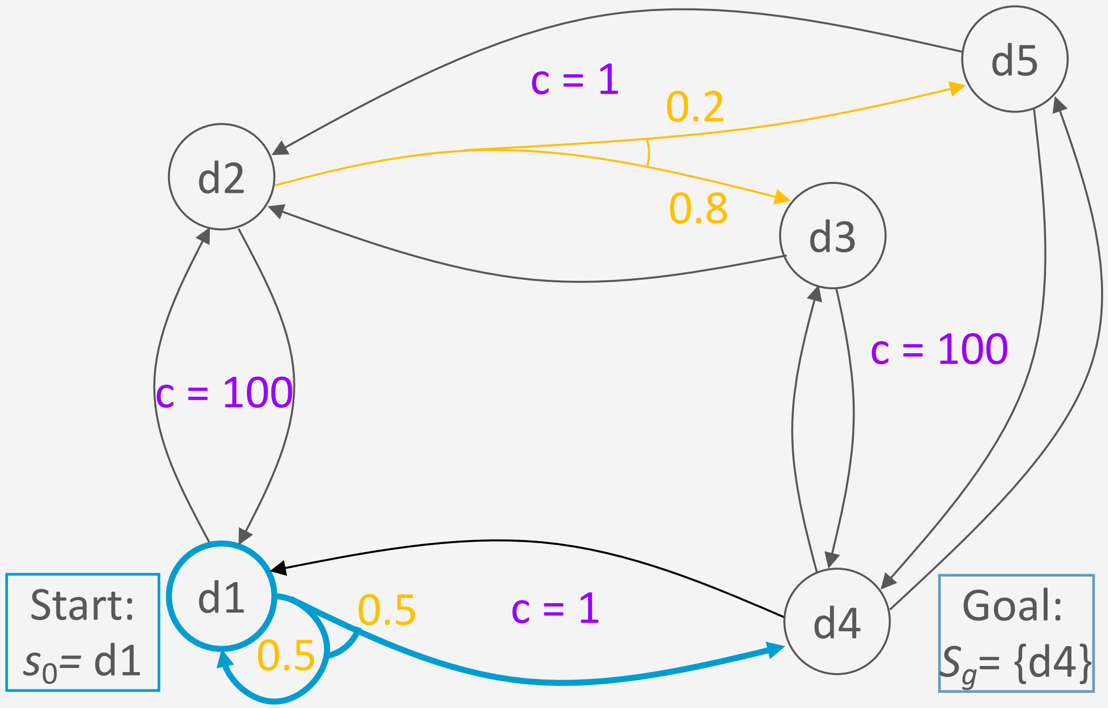
Example policy
$\pi_3 = \{(d1, m12), (d2, m23), (d3, m34), (d5, m54)\}$



Safe Solutions

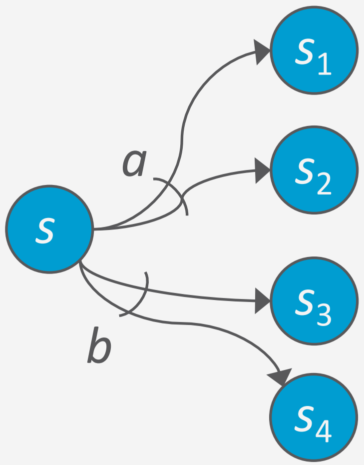
- Weighted sum of history cost:
  - $\sigma_5 = \langle d1, d4 \rangle$
    - $\mathrm{P}(\sigma_5 | s_0, \pi_5) = (^1/_2)^1$
    - $cost(\sigma_5 | s_0, \pi_5) = 1$
  - $\sigma_6 = \langle d1, d1, d4 \rangle$
    - $\mathrm{P}(\sigma_6 | s_0, \pi_6) = (^1/_2)^2$
    - $cost(\sigma_6 | s_0, \pi_5) = 2$
  - …
  - $V^{\pi_4}(d1) = \frac{1}{2}(1) + \frac{1}{4}(2) + \cdots = 2$
  - Recursive equation
    - $V^{\pi_4}(d1) = 1 + 0.5(0) + 0.5\big(V^{\pi_4}(d1)\big)$
      $\Leftrightarrow 0.5V^{\pi_4}(d1) = 1 \Leftrightarrow V^{\pi_4}(d1) = 2$

Example policy
$\pi_4 = \{(d1, m14)\}$



Planning as Optimisation

- Let π and π' be safe solutions
  - π dominates π' if $\forall s \in Dom(\pi) \cap Dom(\pi') : V^{\pi}(s) \leq V^{\pi'}(s)$
- 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 the goal using an optimal safe solution
- Optimality principle (Bellman's theorem):



$$V^*(s) = \begin{cases} 0 & \text{if } s \in S_g \\ \min_{a \in Applicable(S)} \left\{ cost(s,a) + \sum_{s' \in \gamma(s,a)} P(s'|s,a) V^*(s') \right\} & \text{otherwise} \end{cases}$$

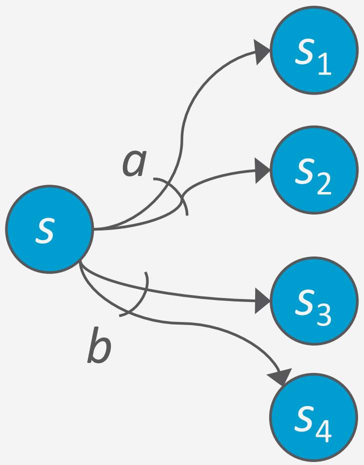
Cost to Go

- 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., $\mathrm{dom}(\pi) = S \setminus S_g$
- 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

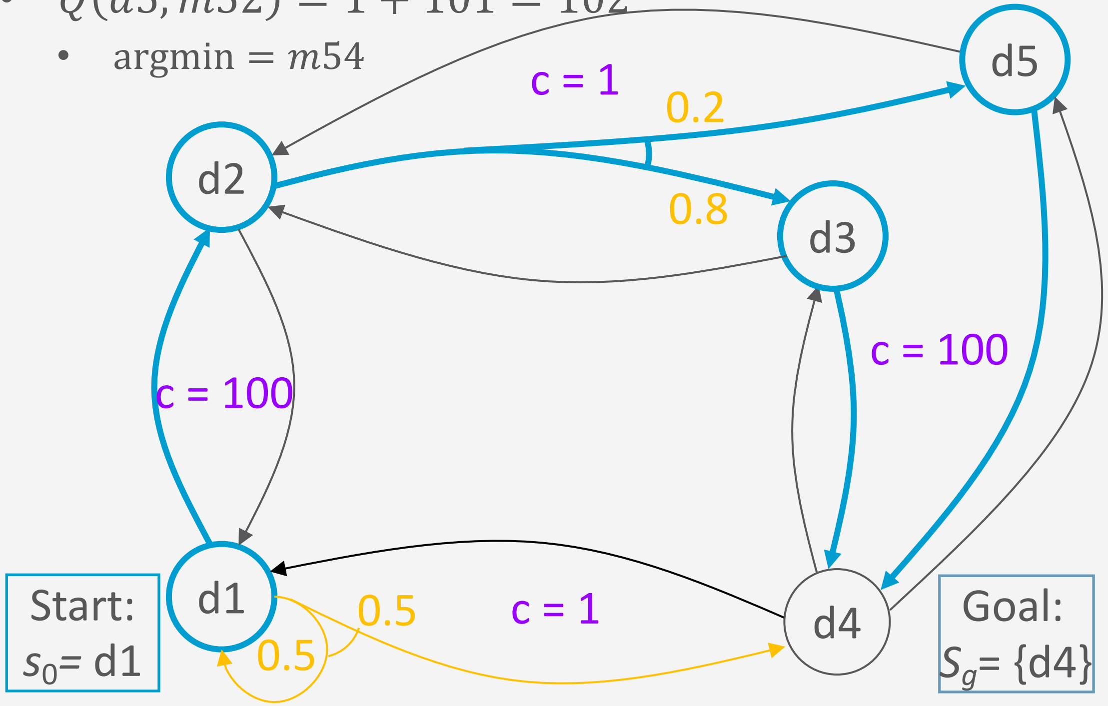
- Inputs
  - SSP problem (Σ, s_0, S_g)
  - Initial policy π_0
- Finds an optimal policy
- Converges in a finite number of steps

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

```
policy-iteration(Σ,s0,Sg,π0)
    π ← π0
    loop
        compute{V^π(s)|s ∈ S}
        for every state s ∈ S \ Sg 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′
```

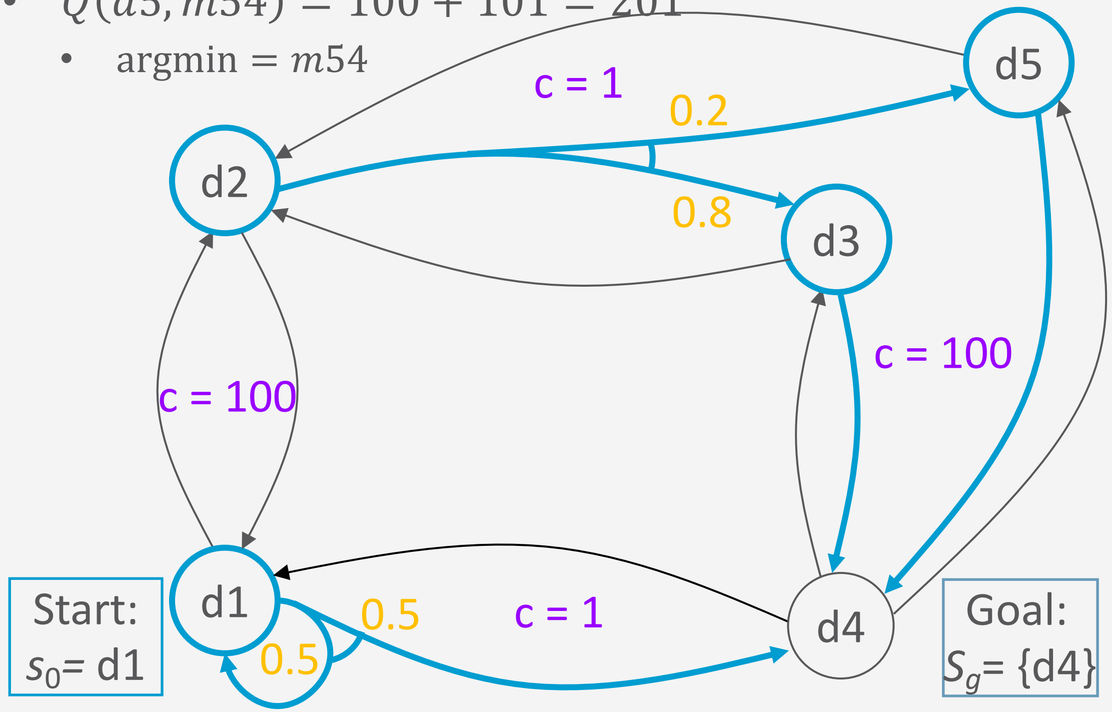
Example

- Start with
  - $\pi = \pi_0 = \{(d1, m12), (d2, m23), (d3, m34), (d5, m54)\}$
- Expected cost
  - $V^\pi(d4) = 0$
  - $V^\pi(d3) = 100 + 1 \cdot V^\pi(d4) = 100$
  - $V^\pi(d5) = 100 + 1 \cdot V^\pi(d4) = 100$
  - $V^\pi(d2) = 1 + \left(0.8 \cdot V^\pi(d3) + 0.2 \cdot V^\pi(d5)\right) = 101$
  - $V^\pi(d1) = 100 + 1 \cdot V^\pi(d2) = 201$
- Cost-to-go
  - $Q(d1, m12) = 100 + 1(101) = 201$
  - $Q(d1, m14) = 1 + 0.5(201) + 0.5(0) = 101.5$
    - $\text{argmin} = m14$
  - $Q(d2, m23) = 1 + \left(0.8(100) + 0.2(100)\right) = 101$
  - $Q(d2, m21) = 100 + 201 = 301$
    - $\text{argmin} = m23$
- Cost-to-go continued
  - $Q(d3, m34) = 100 + 0 = 100$
  - $Q(d3, m32) = 1 + 101 = 102$
    - $\text{argmin} = m34$
  - $Q(d5, m54) = 100 + 0 = 100$
  - $Q(d5, m52) = 1 + 101 = 102$
    - $\text{argmin} = m54$



Example

- Continue with
  - $\pi = \{(d1, m14), (d2, m23), (d3, m34), (d5, m54)\}$
- Expected cost
  - $V^\pi(d4) = 0$
  - $V^\pi(d3) = 100 + V^\pi(d4) = 100$
  - $V^\pi(d5) = 100 + V^\pi(d4) = 100$
  - $V^\pi(d2) = 1 + \left(0.8V^\pi(d3) + 0.2V^\pi(d5)\right) = 101$
  - $V^\pi(d1) = 1 + \left(0.5V^\pi(d1) + 0.5V^\pi(d4)\right) = 2$
- Cost-to-go
  - $Q(d1, m12) = 100 + 101 = 201$
  - $Q(d1, m14) = 1 + 0.5(2) + 0.5(0) = 2$
    - $\text{argmin} = m14$
  - $Q(d2, m23) = 1 + \left(0.8(100) + 0.2(100)\right) = 101$
  - $Q(d2, m21) = 100 + 201 = 301$
    - $\text{argmin} = m23$
- Cost-to-go continued
  - $Q(d3, m34) = 100 + 0 = 100$
  - $Q(d3, m32) = 100 + 101 = 201$
    - $\text{argmin} = m34$
  - $Q(d5, m54) = 100 + 0 = 100$
  - $Q(d5, m54) = 100 + 101 = 201$
    - $\text{argmin} = m54$



π unchanged

Universität Münster

Value Iteration

- Inputs
  - SSP problem (Σ, s_0, S_g)
  - Convergence criterion $\eta > 0$
  - V_0 is a heuristic fct. for initial values
    - E.g., $V_0(s) = 0 \; \forall s \in S_g$ or adapt a heuristics from Ch. 2
- Returns optimal plan π
  - V_i = values computed at i'th iteration
  - π_i = plan computed from V_i
  - Synchronous: Computes V_i and π_i from V_{i-1}, π_{i-1}
  - Asynchronous: Update V and π in place
    - New values available immediately
    - More efficient than synchronous version

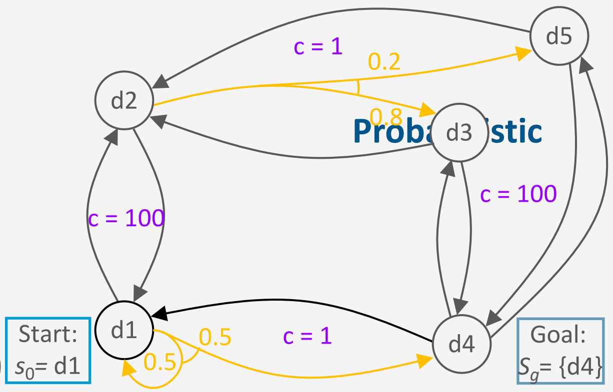
```
sync-value-iteration(Σ,s₀,S_g,V₀,η)
    for i = 1,2,… do
        for every state s ∈ S \ S_g do
            for every a ∈ Applicable(s) do
                Q(s,a) ← cost(s,a)+Σ_{s'∈S} P(s'|s,a) V_{i-1}(s')
            V_i(s) ← min_{a∈Applicable(s)} Q(s,a)
            π_i(s) ← argmin_{a∈Applicable(s)} Q(s,a)
            if max_{s∈S}|V_i(s)- V_{i-1}(s)| ≤ η then
                return π_i
```

```
async-value-iteration(Σ,s₀,S_g,V₀,η)
    global π ← ∅
    global V(s) ← V₀(s) ∀ s
    loop
        r ← max_{s∈S\Sg} Bellman-Update(s)
        if r ≤ η then
            return π

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

Universität Münster

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



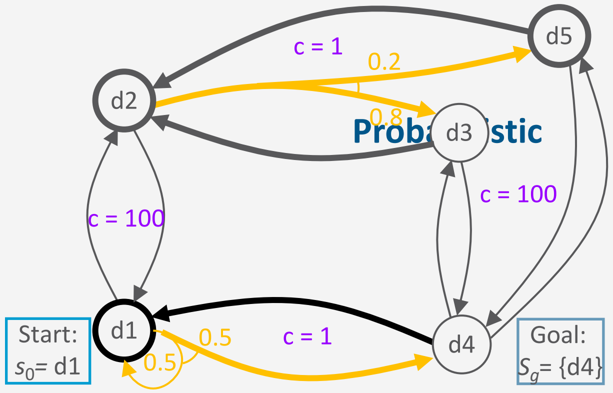
Synchronous

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

Asynchronous

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

$\eta = 0.2$



Synchronous

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

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

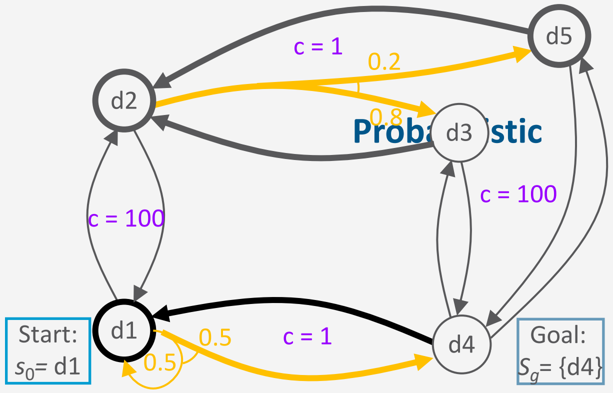
Asynchronous

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

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

Universität Münster

$\eta = 0.2$



Synchronous

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

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

Asynchronous

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

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

Universität Münster

$\eta = 0.2$

Synchronous

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

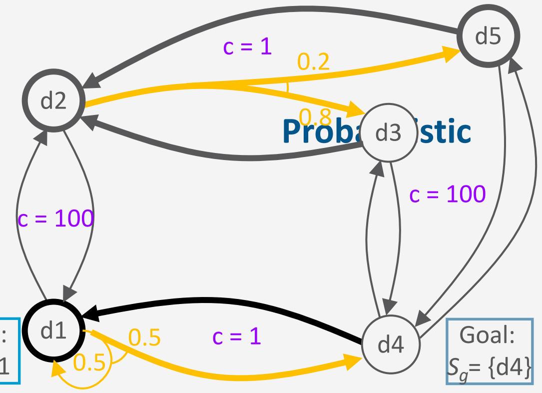
$V(d1) = 1.75$
$V(d2) = 3$
$V(d3) = 3$
$V(d5) = 3$

A

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

- $\ldots 105$
- $\ldots .5(1.75) + 0.5(0)\big) = 1.875$
  - $V(d1) = 1.875$; $\pi(d1) = m14$
- $Q(d2, m21) = 100 + 1.875 = 101.875$
- $Q(d2, m23) = 1 + \big(0.2(6) + 0.8(6)\big) = 7$
  - $V(d2) = 7$; $\pi(d2) = m23$
- $Q(d3, m32) = 1 + 7 = 8$
- $Q(d3, m34) = 100 + 0 = 100$
  - $V(d3) = 8$; $\pi(d3) = m32$
- $Q(d5, m52) = 1 + 7 = 8$
- $Q(d5, m54) = 100 + 0 = 100$
  - $V(d5) = 8$; $\pi(d5) = m52$
- $r = \max(1.875 - 1.75, 7 - 5, 8 - 6, 8 - 6) = 2$

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



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

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
- Inputs: SSP problem (Σ, s_0, S_g), initial values V_0
- Envelope: set of states that have been generated

Requires acyclic Σ

not in book

```
AO*(Σ, s₀, S_g, V₀)
    global π ← ∅; V(s₀) ← V₀(s₀); Envelope ← {s₀}
    while leaves(s₀,π) \ S_g ≠ ∅ do
        select s ∈ leaves(s₀,π) \ S_g
        for all a ∈ Applicable(s) do
            for all s' ∈ γ(s,a) \ Envelope do
                V(s') ← V₀(s')
                Add s' to Envelope
        AO-Update(s)
    return π
```

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

the states "just above" s

```
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)}
```

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

LAO*

Different compared to AO*

- Best-first search for both cyclic and acyclic domains
- Inputs: SSP problem (Σ, s_0, S_g), initial values V_0

Σ may be cyclic or acyclic

not in book

```
LAO*(Σ, s₀, S_g, V₀)
    global π ← ∅; V(s₀) ← V₀(s₀); Envelope ← {s₀}
    while leaves(s₀,π) \ S_g ≠ ∅ do
        select s ∈ leaves(s₀,π) \ S_g
        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
```

Asynchronous value iteration, restricted to Z

all π-ancestors of s in $Envelope$

```
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

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

1st iteration of main loop:

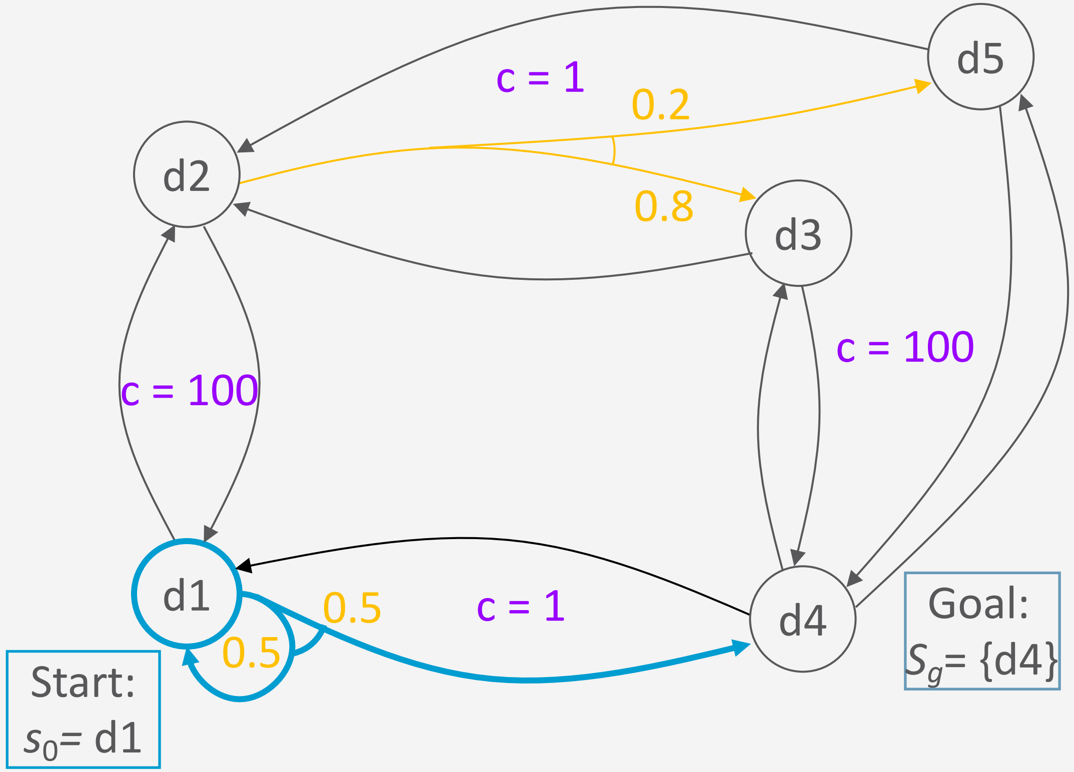
- Expand d1: add d2 and d4 to Envelope
- Call LAO-Update(d1)
  - π is empty, so $Z = \{d1\}$

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

  → $V(d1) = 1$

  → $\pi(d1) = m14$

  → $r = 1 - 0 = 1$



LAO* Example

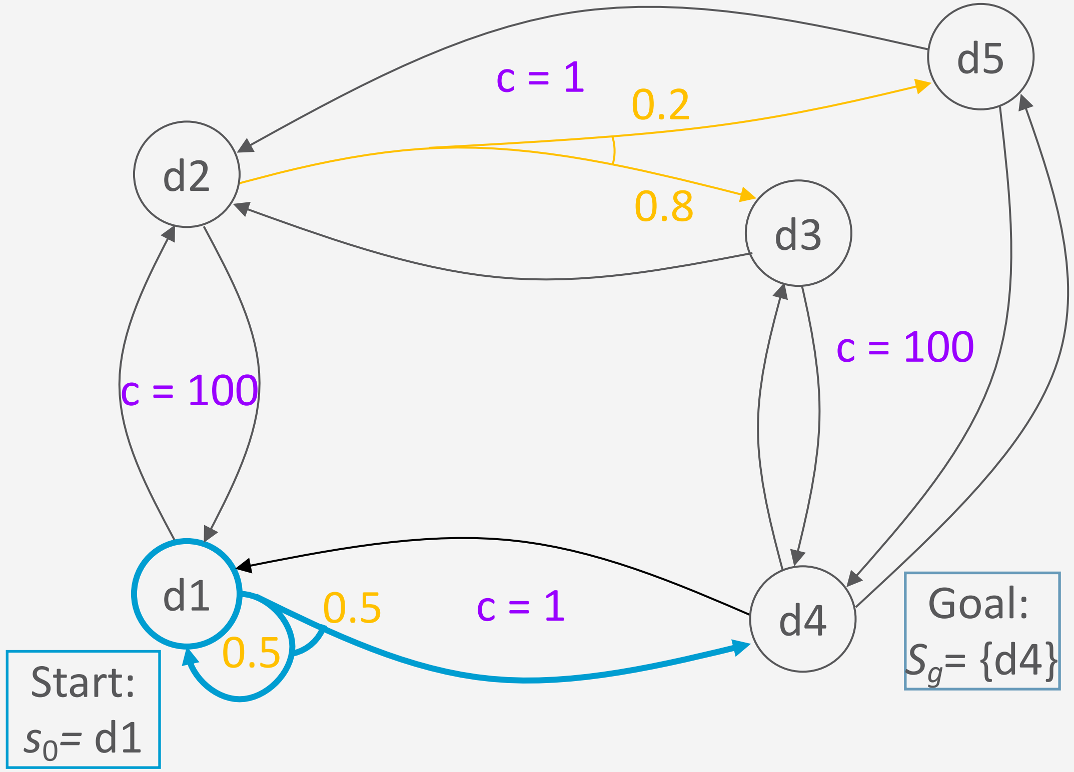
$\eta = 0.2$
$V_0(s) = 0\ \forall s$

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:

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



LAO* Example

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

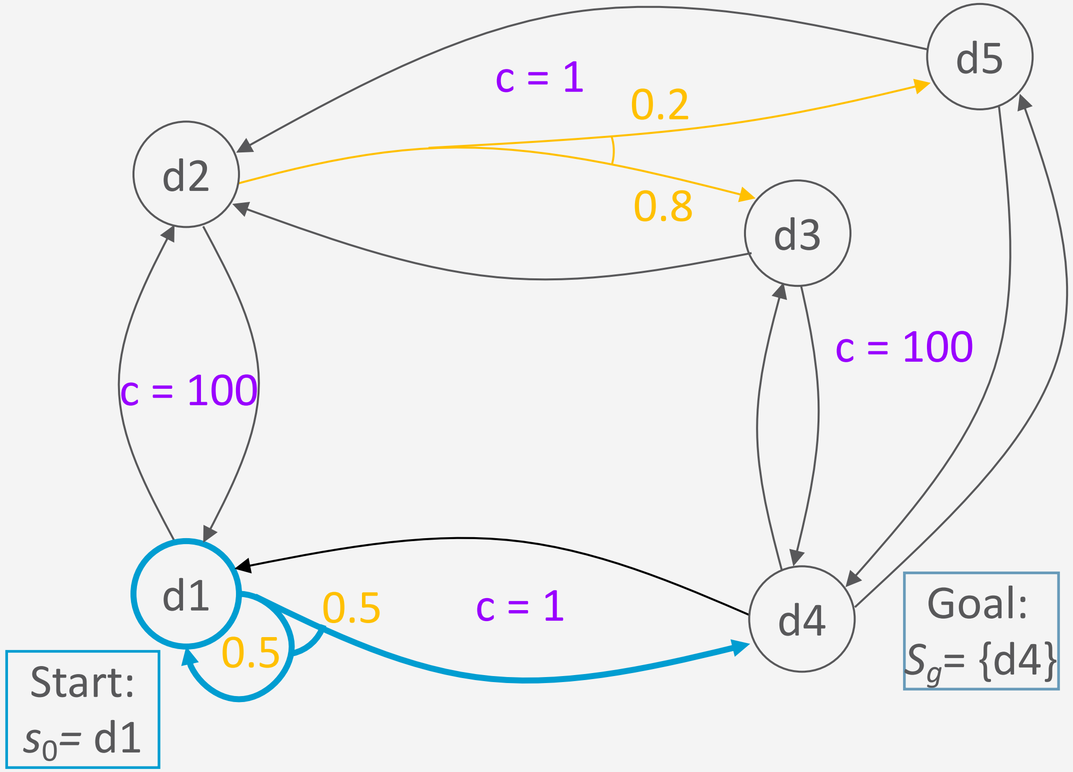
Iteration 4:

- $Q(d1, m12) = 100 + 0 = 100$
- $Q(d1, m14) = 1 + \big(0.5(1.75) + 0.5(0)\big) = 1.825$
  - $V(d1) = 1.825$
  - $\pi(d1) = m14$
  - $r = 0.125 \leq \eta$

  LAO-Update returns

2nd iteration of main loop:

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

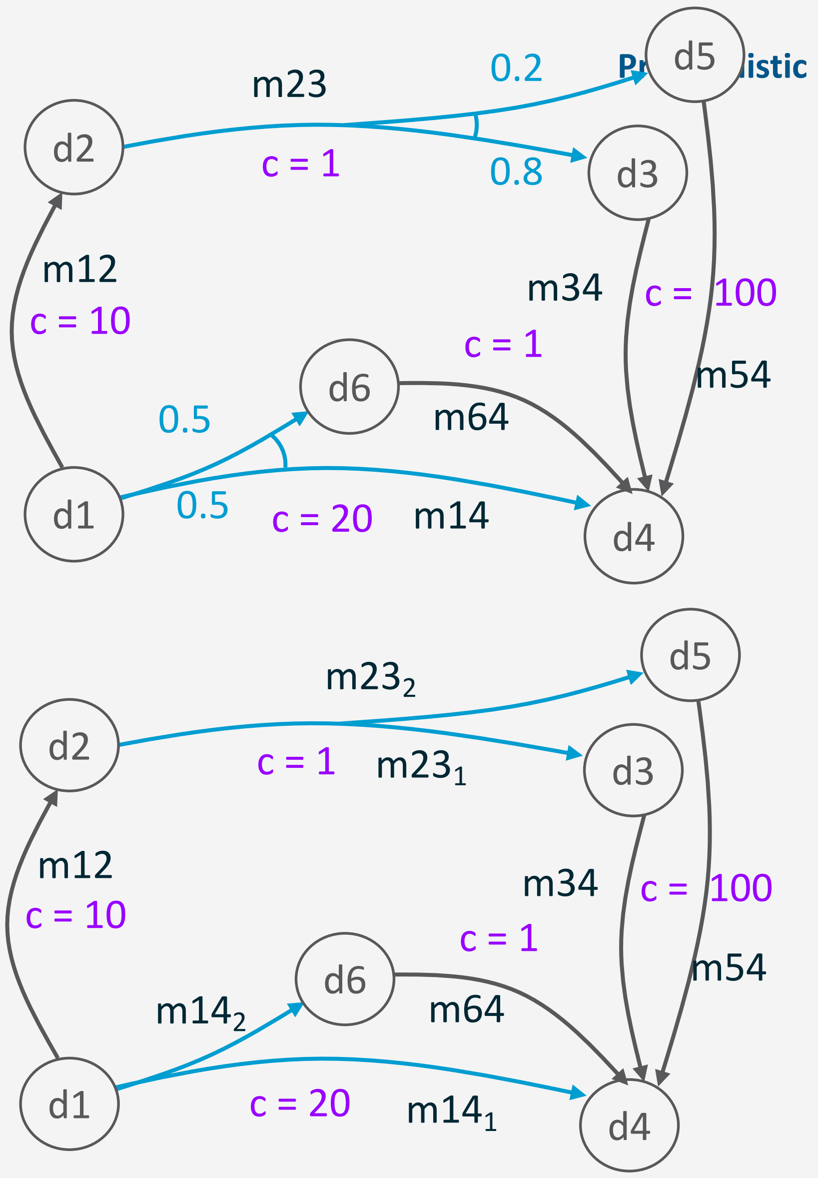


Heuristics through Determinisation

- What to use for V_0? One possibility: classical planner
  - Need to convert nondeterministic actions into something a classical planner can use
- Determinise the actions
  - Suppose $\gamma(s, a) = \{s_1, \dots, s_n\}$
  - $Det(s, a) = \{n \text{ actions } a_1, a_2, \dots, a_n\}$
    - $\gamma_d(s, a_i) = s_i$
    - $cost_d(s, a_i) = cost(s, a)$

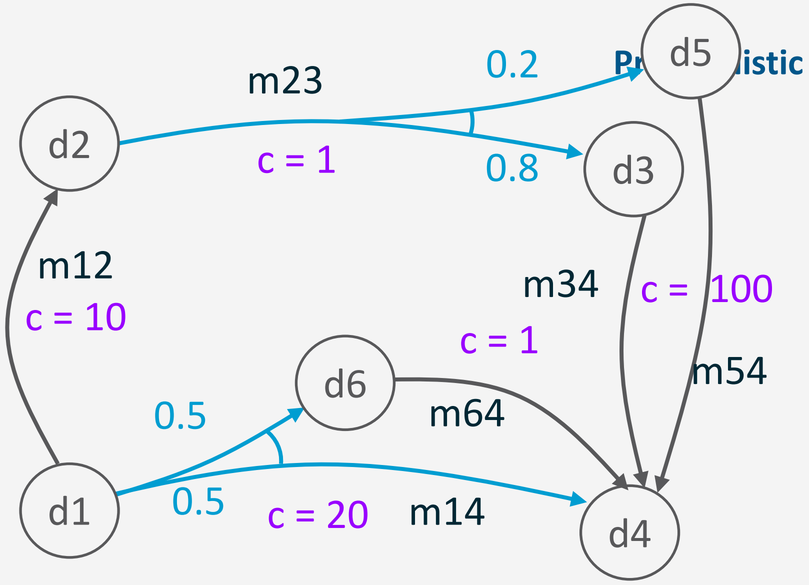
→ Classical domain $\Sigma_d = (S, A_d, \gamma_d, cost_d)$

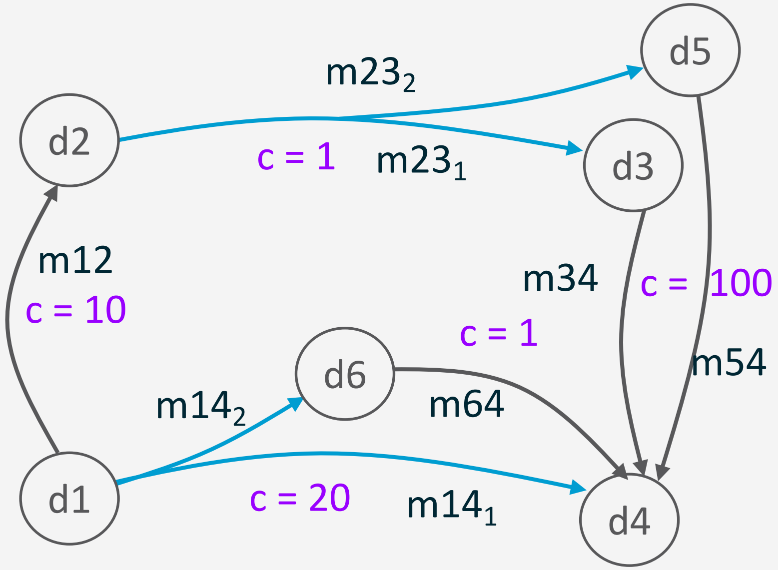
  - S = same as in Σ
  - $A_d = \bigcup_{a \in A, s \in S} Det(s, a)$
  - γ_d and $cost_d$ as above



Heuristics through Determinisation

- Call classical planner on (Σ_d, s, S_g)
  - Get plan $p = \langle a_1, a_2, \dots, a_n \rangle$
  - Return $V_0(s) = cost(p) = \sum_{i=1}^{n} cost(a_i)$
- If the classical planner always returns optimal plans p, then V_0 is admissible
  - Outline of proof:
    - Let π be a safe solution in Σ and p be an optimal plan in Σ_d with $cost(p) = V_0(s)$
    - Every acyclic execution of π corresponds to a plan p' in Σ_d
      - p' must have cost $\geq V_0(s)$
      - Otherwise, the classical planner would have chosen p' instead of p





Summary

- AO*
  - Acyclic
- LAO*
  - (A)cyclic
- Heuristics through determinisation

Outline

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 instead of ξ
  - Could use $s \leftarrow$ abstraction of ξ as in Ch. 2
  - Inputs:
    - SSP problem (Σ, s_0, S_g)
    - Vector of parameters θ
- 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 determinised domain
  - Stochastic sampling algorithms

```
Run-Lookahead(Σ,s₀,S_g,θ)
    s ← s₀
    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

- Pointer to the next chapter:
  Acting as *Reinforcement learning*
  - Learning to act / finding the optimal policy in an unknown environment (no model available)

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

```
FS-Replan(Σ, s, S_g)
    π_d ← ∅
    while s ∉ S_g and Applicable(s) ≠ ∅ do
        if π_d undefined for s then
            π_d ← Forward-Search(Σ_d, s, S_g)
            if π_d = failure then
                return failure
        perform action π_d(s)
        s ← observe resulting state
```

Outline per the Book

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: Decision Making