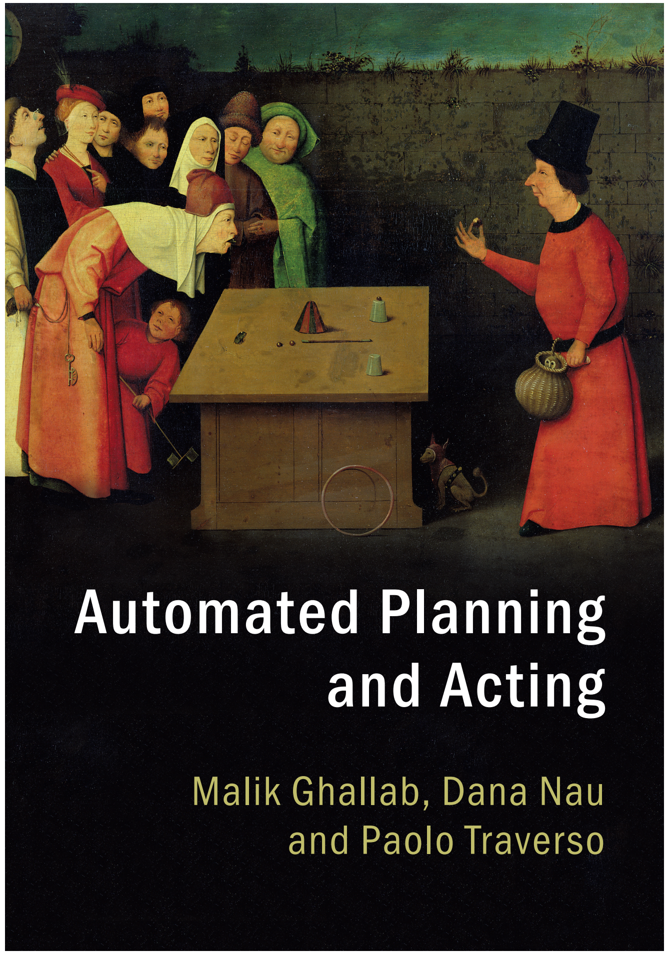
Last update: April 21, 2022

Chapter 6

Deliberation with Probabilistic Domain Models

Dana S. Nau
University of Maryland



http://www.laas.fr/planning

Motivation

- Situations where actions have multiple possible outcomes and each outcome has a probability

- Several possible action representations
  - Bayes nets, probabilistic actions, …
- Book doesn't commit to any representation
  - Mainly concentrates on the underlying semantics

roll-die(d)
    pre: holding(d) = true
    eff:
        1/6:   top(d) $\leftarrow$ 1
        1/6:   top(d) $\leftarrow$ 2
        1/6:   top(d) $\leftarrow$ 3
        1/6:   top(d) $\leftarrow$ 4
        1/6:   top(d) $\leftarrow$ 5
        1/6:   top(d) $\leftarrow$ 6

Probabilistic planning domain

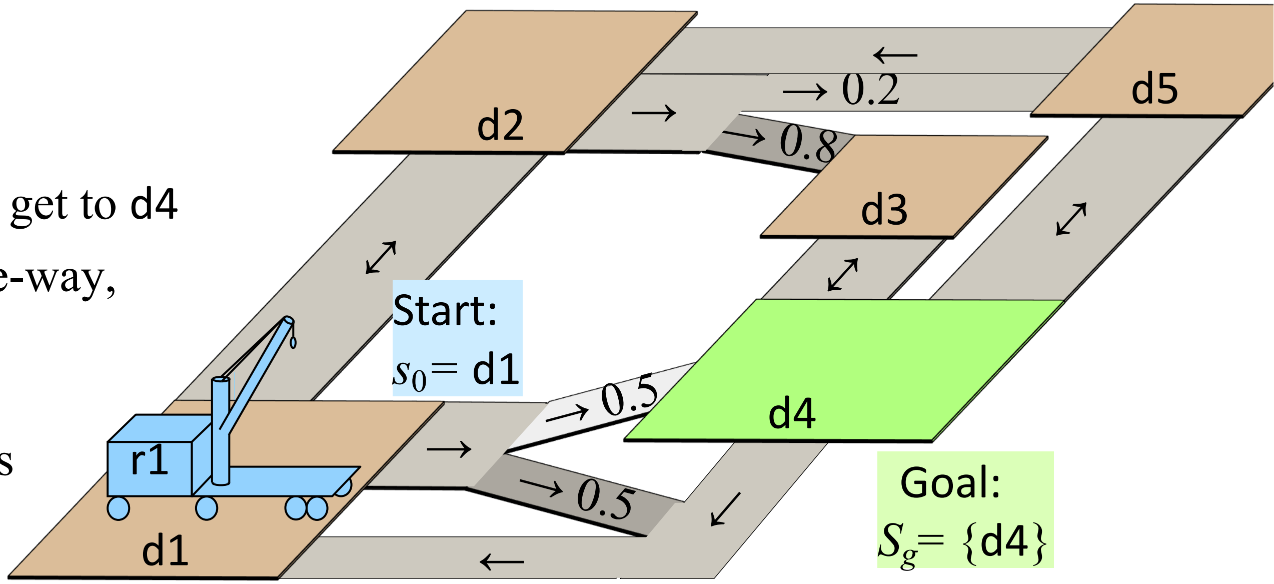
Definitions

$\Sigma = (S, A, \gamma, \Pr, \text{cost})$

- S = {states}
- A = {actions}
- $\gamma : S \times A \to 2^S$
- $\Pr(s' \mid s, a)$ = probability of going to state s' if we're in s and apply a
  - $\Pr(s' \mid s, a) \neq 0$ iff $s' \in \gamma(s,a)$
- cost: $S \times A \to \mathbb{R}_{\geq 0}$
  - cost(s,a) = cost of action a in state s
  - may omit, default is cost$(s,a) = 1$
- Applicable$(s) = \{a \mid \gamma(s,a) \neq \emptyset\}$

Example

- Start at d1, want to get to d4
- Some roads are one-way, some are two-way
- Unreliable steering when the road forks
  - may take the wrong fork



- Simplified state and action names:
  - write {loc(r1)=d2} as d2
  - write move(r1,d2,d3) as m23
- γ(d1,m12) = {d2}
  - Pr(d2 | d1,m12) = 1
- m21, m34, m41, m43, m45, m52, m54:
  - like m12
- γ(d1,m14) = {d1,d4}
  - Pr(d4 | d1,m14) = 0.5
  - Pr(d1 | d1,m14) = 0.5
- γ(d2,m23) = {d3,d5}
  - Pr(d3 | d2,m23) = 0.8
  - Pr(d5 | d2,m23) = 0.2
- No m11, no m25

Probabilistic planning domain

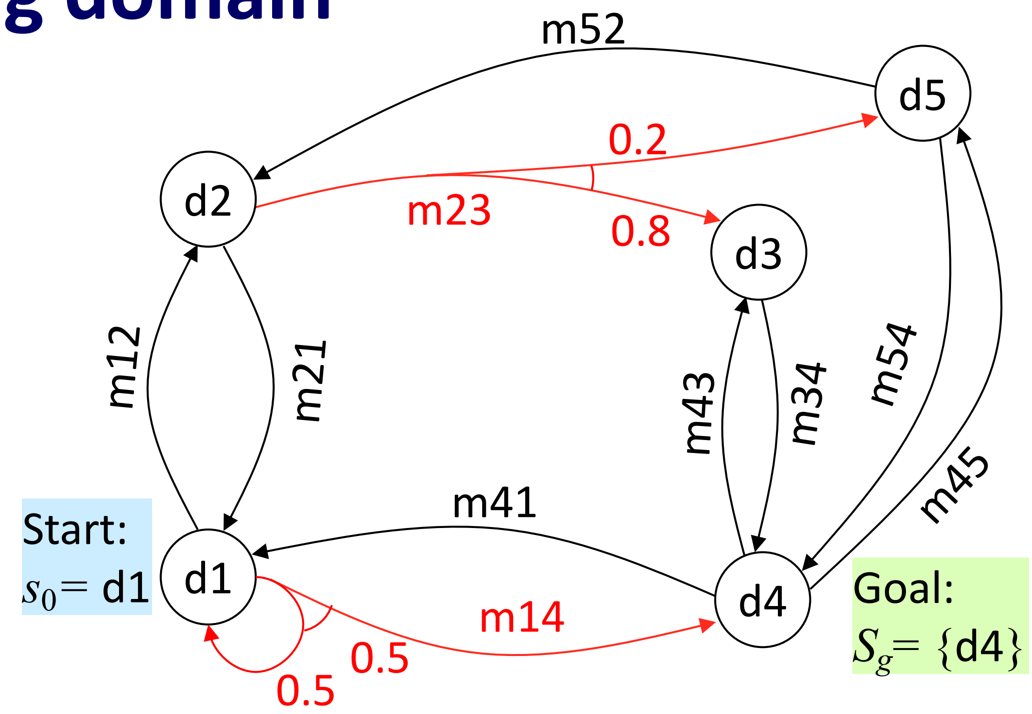
Definitions

$\Sigma = (S, A, \gamma, \Pr, \text{cost})$

- S = {states}
- A = {actions}
- $\gamma : S \times A \rightarrow 2^S$
- $\Pr(s' \mid s, a)$ = probability of going to state s' if we're in s and apply a
  - $\Pr(s' \mid s, a) \neq 0$ iff $s' \in \gamma(s,a)$
- cost: $S \times A \rightarrow \mathbb{R}_{\geq 0}$
  - cost(s,a) = cost of action a in state s
  - may omit, default is cost(s,a) = 1
- Applicable(s) = $\{a \mid \gamma(s,a) \neq \emptyset\}$

Example

- γ(d1,m12) = {d2}
  - Pr(d2 | d1,m12) = 1
- m21, m34, m41, m43, m45, m52, m54:
  - like m12
- γ(d1,m14) = {d1,d4}
  - Pr(d4 | d1,m14) = 0.5
  - Pr(d1 | d1,m14) = 0.5
- γ(d2,m23) = {d3,d5}
  - Pr(d3 | d2,m23) = 0.8
  - Pr(d5 | d2,m23) = 0.2
- there's no m25

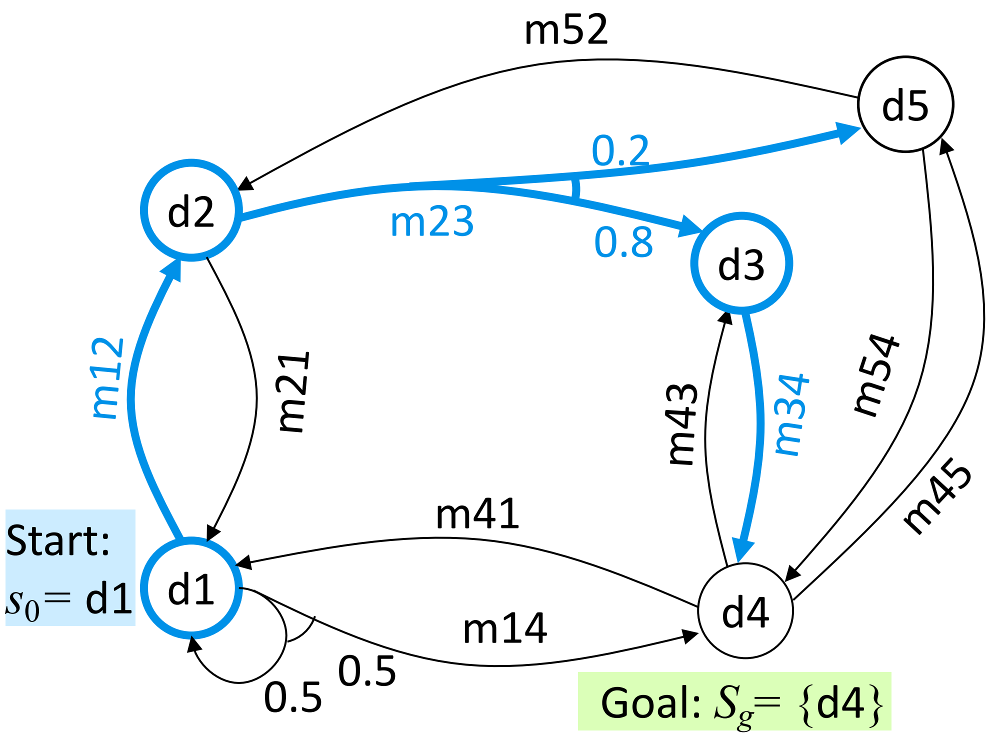


Poll: Can a plan (sequence of actions) be a solution for this problem?

1. yes
2. no

Policies, Problems

- Same as in Chapter 5:
- *Policy*
  - partial function $\pi : S \rightarrow A$ such that for every $s \in \text{Dom}(\pi) \subseteq S$, $\pi(s) \in \text{Applicable}(s)$
- *Transitive closure*
  - $\hat{\gamma}(s,\pi)$ = {s and all states reachable from s using π}
- Graph(s,π) = rooted graph induced by π at s
  - nodes: $\hat{\gamma}(s,\pi)$
  - edges: state transitions
- $\textit{leaves}(s,\pi) = \hat{\gamma}(s,\pi) \setminus \text{Dom}(\pi)$



π_1 = {(d1, m12), (d2, m23), (d3, m34)}

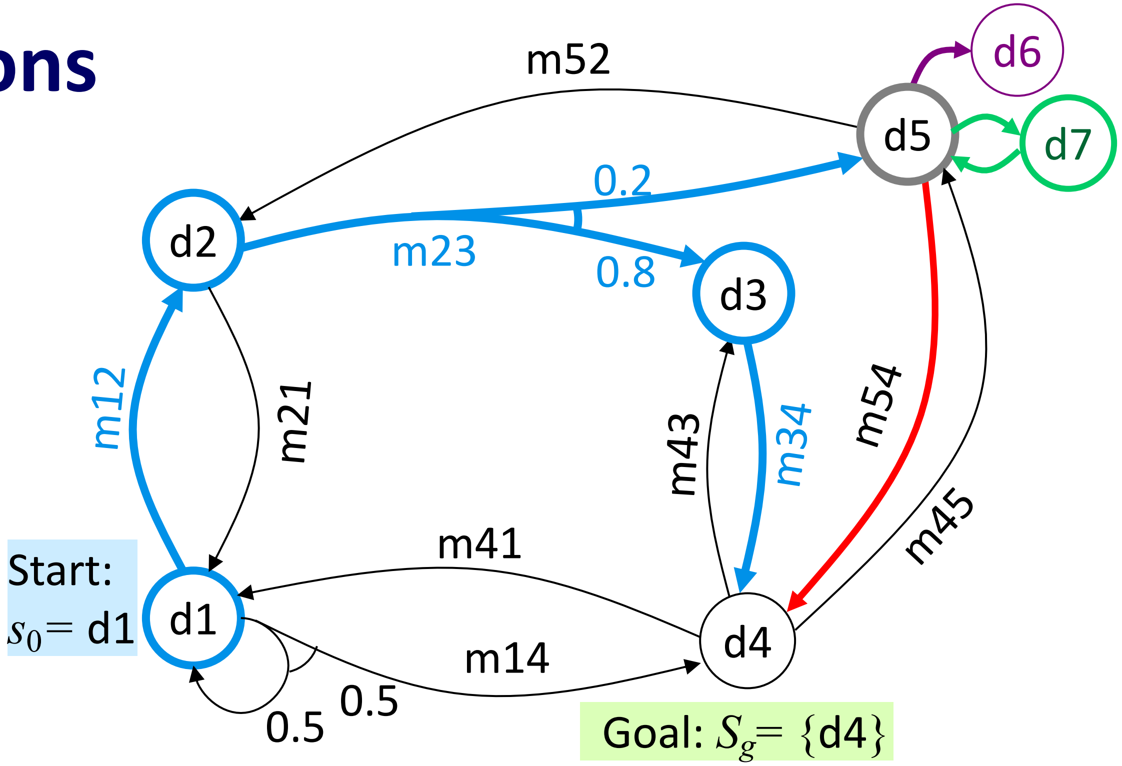
Dom(π_1) = {d1, d2, d3}

$\hat{\gamma}$(d1,π_1) = {d1, d2, d3, d4, d5}

leaves(d1,π_1) = $\hat{\gamma}$(d1,π_1) \ Dom(π_1)
= {d4, d5}

Solutions

- *Stochastic shortest path (SSP)* problem:
  - a triple (Σ, s_0, S_g)
- *Solution* for (Σ, s_0, S_g):
  - A policy π for Σ such that $\hat{\gamma}((s_0,\pi) \cap S_g \neq \emptyset$
- Unlike Chapter 5, don't require π to end at S_g
  http://www.cs.umd.edu/users/nau/apa/slides/errata.pdf
- A solution policy π is *closed* if it doesn't stop at non-goal states unless there's no way to continue
  - for every state s in $\hat{\gamma}(s_0,\pi)$, either
    $s \in \text{Dom}(\pi)$ (i.e., $\pi(s)$ is defined)
    or $s \in S_g$
    or $\text{Applicable}(s) = \emptyset$
- For the rest of this chapter we require all solutions to be closed



π_1 = {(d1,m12), (d2,m23), (d3,m34)}

π_2 = {(d1,m12), (d2,m23), (d3,m34), (d5,m54)}

π_3 = {(d1,m12), (d2,m23), (d3,m34), (d5,m56)}

π_4 = {(d1,m12), (d2,m23), (d3,m34), (d5,m57), (d7,m75)}

Histories

- **History**: sequence of states $\sigma = \langle s_0, s_1, s_2, \ldots \rangle$
  - May be finite or infinite

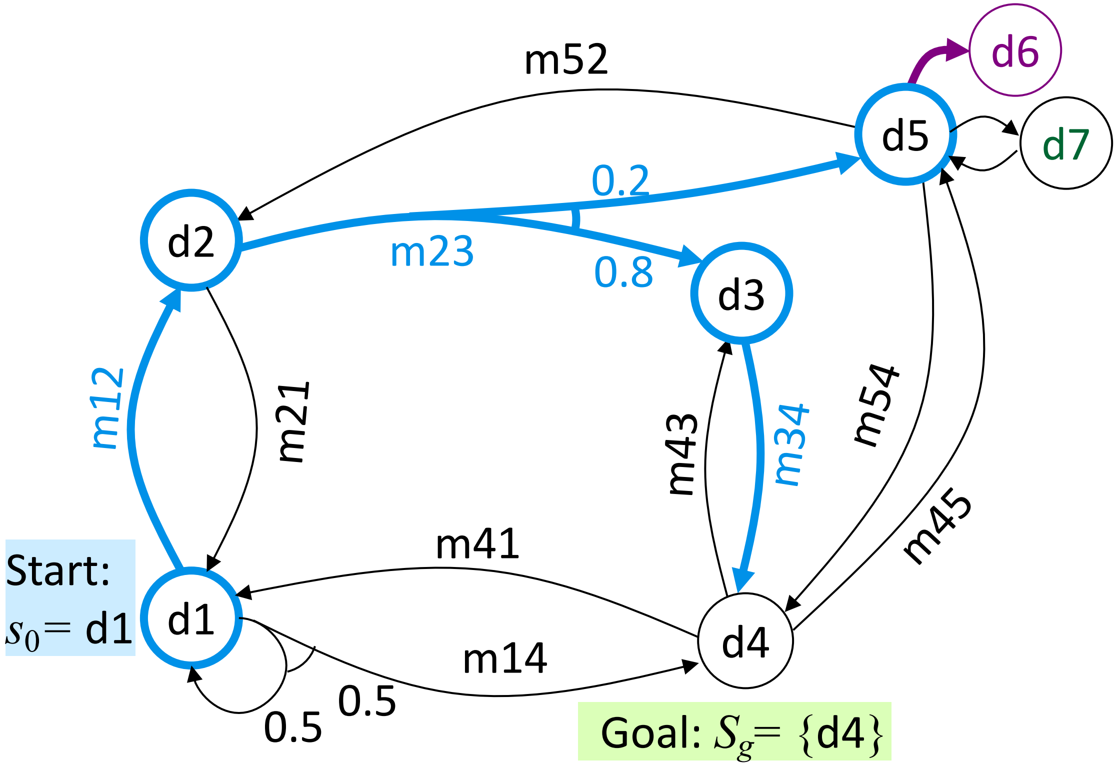
    ⟨d1, d2, d3, d4⟩

    ⟨d1, d2, d1, d2, …⟩

- Let $H(s,\pi)$ = set of all possible histories if we start at s and follow π
  - Stop if we reach a state s' such that $s' \notin \mathrm{Dom}(\pi)$ or $s' \in S_g$

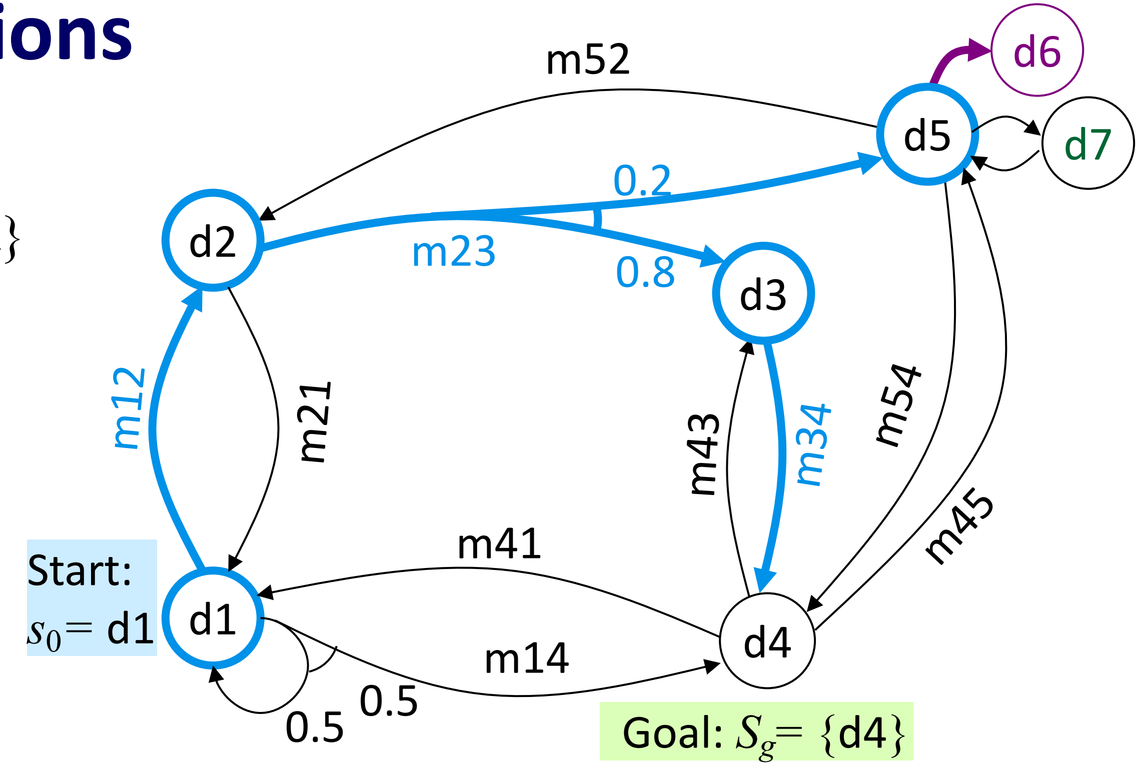
- If $\sigma \in H(s, \pi)$ then
  - $\Pr(\sigma \mid s, \pi) = \prod_{s_i, s_{i+1} \in \sigma} \Pr(s_{i+1} \mid s_i, \pi(s_i))$
  - Product of the probabilities of the state transitions

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



- $\pi_3 = \{$(d1,m12), (d2,m23), (d3,m34), (d5,m56)$\}$
- $H(s_0, \pi_3) = \{\sigma_1, \sigma_2\}$, where:
  - σ_1 = ⟨d1,d2,d3,d4⟩
  - σ_2 = ⟨d1,d2,d5,d6⟩
- $\Pr(\sigma_1 \mid s_0, \pi_3) = 1 \times 0.8 \times 1 = 0.8$
- $\Pr(\sigma_2 \mid s_0, \pi_3) = 1 \times 0.2 \times 1 = 0.2$

Unsafe Solutions



- Probability of reaching a goal state:

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

- Equivalently:

$$\Pr(S_g \mid s, \pi) = \begin{cases} 1, & \text{if } s \in S_g \\ \sum_{s' \in \gamma(s,\pi(s))} \{\Pr(S_g \mid s', \pi)\}, & \text{otherwise} \end{cases}$$

- A solution is *unsafe* if $0 < \Pr(S_g \mid s_0, \pi) < 1$

- π_3 = {(d1,m12), (d2,m23), (d3,m34), (d5,m56)}

- Two possible histories:
  - σ_1 = ⟨d1,d2,d3,d4⟩ ends at a goal state; $\Pr(\sigma_1 \mid s_0, \pi_3) = 1 \times 0.8 \times 1 = 0.8$
  - σ_2 = ⟨d1,d2,d5,d6⟩ doesn't; $\Pr(\sigma_2 \mid s_0, \pi_3) = 1 \times 0.2 \times 1 = 0.2$

- $\Pr(S_g \mid s_0, \pi_3) = \Pr(\sigma_1 \mid s_0, \pi_3) = 0.8$

Unsafe Solutions

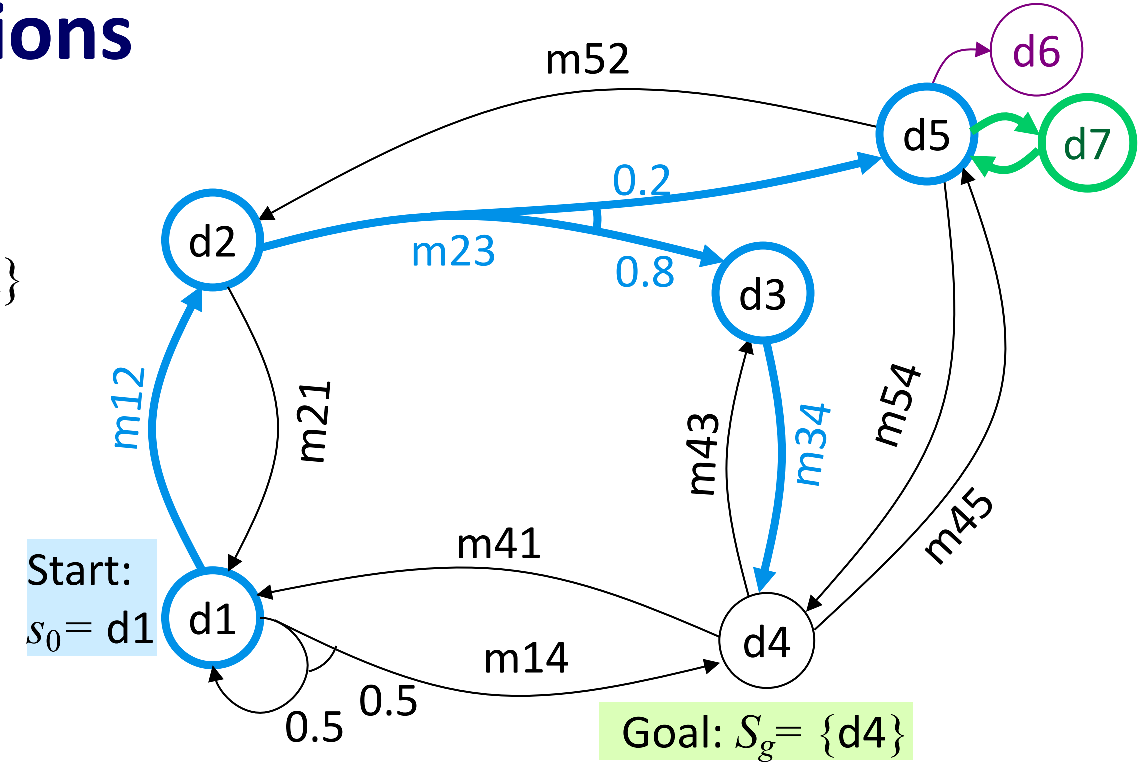
- Probability of reaching a goal state:

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

- Equivalently:

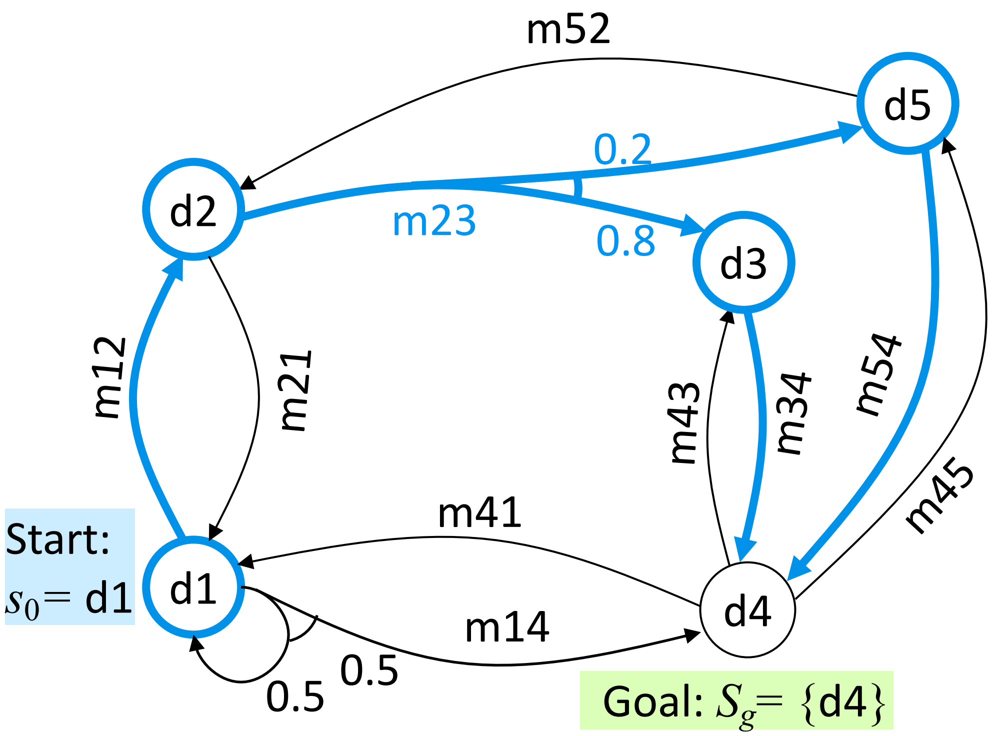
$$\Pr(S_g \mid s, \pi) = \begin{cases} 1, & \text{if } s \in S_g \\ \sum_{s' \in \gamma(s,\pi(s))} \{\Pr(S_g \mid s', \pi)\}, & \text{otherwise} \end{cases}$$

- A solution is *unsafe* if $0 < \Pr(S_g \mid s_0, \pi) < 1$



- $\pi_4 = \{$(d1,m12), (d2,m23), (d3,m34), (d5,m57), (d7,m75)$\}$
- Two possible histories
  - $\sigma_1 = \langle$d1,d2,d3,d4$\rangle$ ends at a goal state; $\Pr(\sigma_1 \mid s_0, \pi_4) = 1 \times .8 \times 1 = 0.8$
  - $\sigma_3 = \langle$d1,d2,d5,d7,d5,d7,…$\rangle$ doesn't; $\Pr(\sigma_3 \mid s_0, \pi_4) = 1 \times .2 \times 1 \times 1 \times 1 \times \ldots = 0.2$
  - $\Pr(S_g \mid s_0, \pi_4) = \Pr(\sigma_1 \mid s_0, \pi_4) = 0.8$

Safe Solutions



- A solution is *safe* if $\Pr(S_g \mid s_0, \pi) = 1$

- An *acyclic* safe solution:
  - π_2 = {(d1, m12), (d2, m23), (d3, m34), (d5, m54)}

- $H(s_0, \pi_2)$ contains two histories:
  - σ_1 = ⟨d1, d2, d3, d4⟩ $\Pr(\sigma_1 \mid s_0, \pi_2) = 1 \times .8 \times 1 = .8$
  - σ_4 = ⟨d1, d2, d5, d4⟩ $\Pr(\sigma_4 \mid s_0, \pi_2) = 1 \times .2 \times 1 = .2$

- $\Pr(S_g \mid s_0, \pi_2) = .8 + .2 = 1$

Safe Solutions

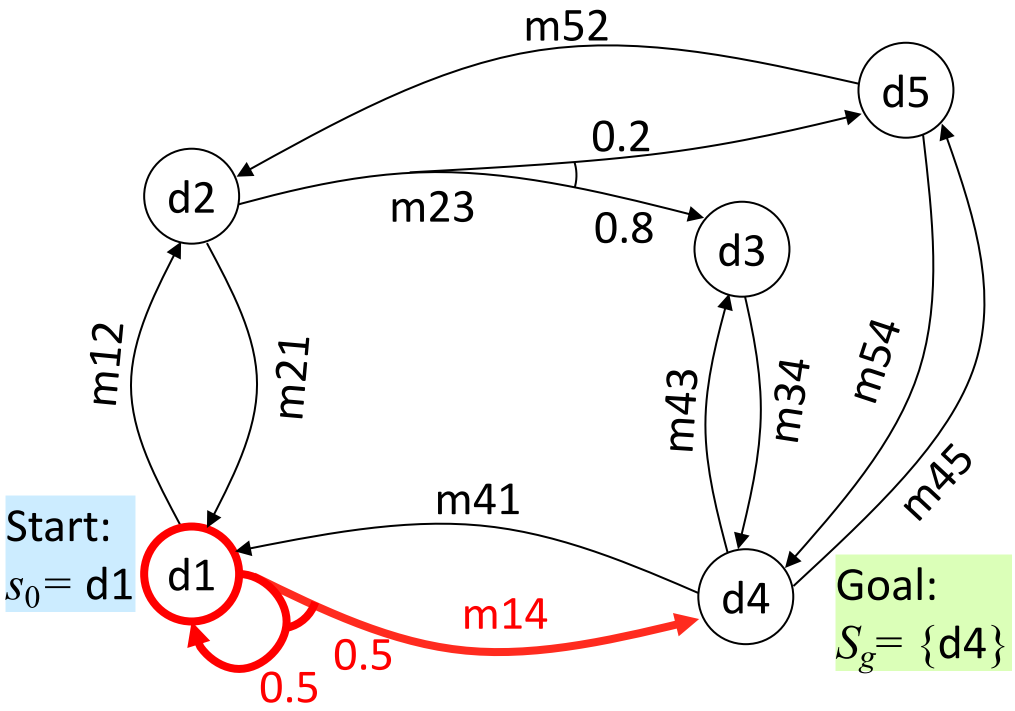
- A solution is *safe* if $\Pr(S_g \mid s_0, \pi) = 1$

- A *cyclic* safe solution:
  - $\pi_5 = \{(\text{d1, m14})\}$

- $H(s_0, \pi_5)$ contains infinitely many histories:
  - $\sigma_5 = \langle \text{d1, d4} \rangle$ $\Pr(\sigma_5 \mid s_0, \pi_5) = ½$
  - $\sigma_6 = \langle \text{d1, d1, d4} \rangle$ $\Pr(\sigma_6 \mid s_0, \pi_5) = (½)^2 = ¼$
  - $\sigma_7 = \langle \text{d1, d1, d1, d4} \rangle$ $\Pr(\sigma_7 \mid s_0, \pi_5) = (½)^3 = 1/8$

    …
  - $\sigma_\infty = \langle \text{d1, d1, d1, d1, d1, } \ldots \rangle$

- $\Pr(S_g \mid s_0, \pi_5) = ½ + ¼ + 1/8 + \ldots = 1$



Poll: what is $\Pr(\sigma_\infty \mid s_0, \pi_5)$?

A. 1

B. 0

C. a number between 0 and 1

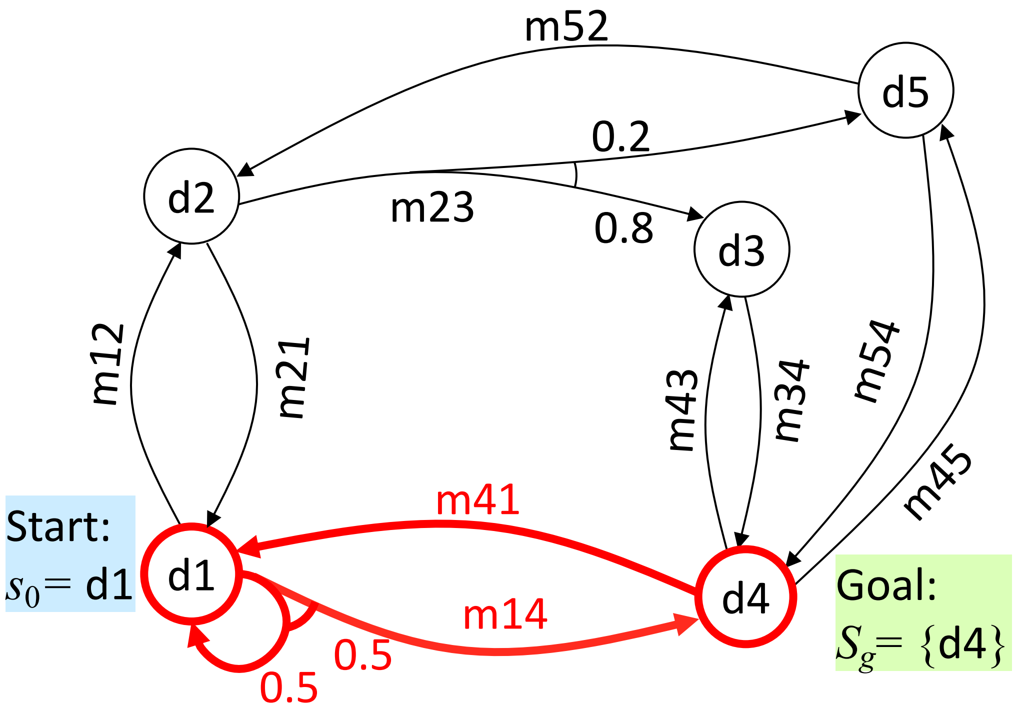
D. undefined

Safe Solutions

- A solution is *safe* if $\Pr(S_g \mid s_0, \pi) = 1$

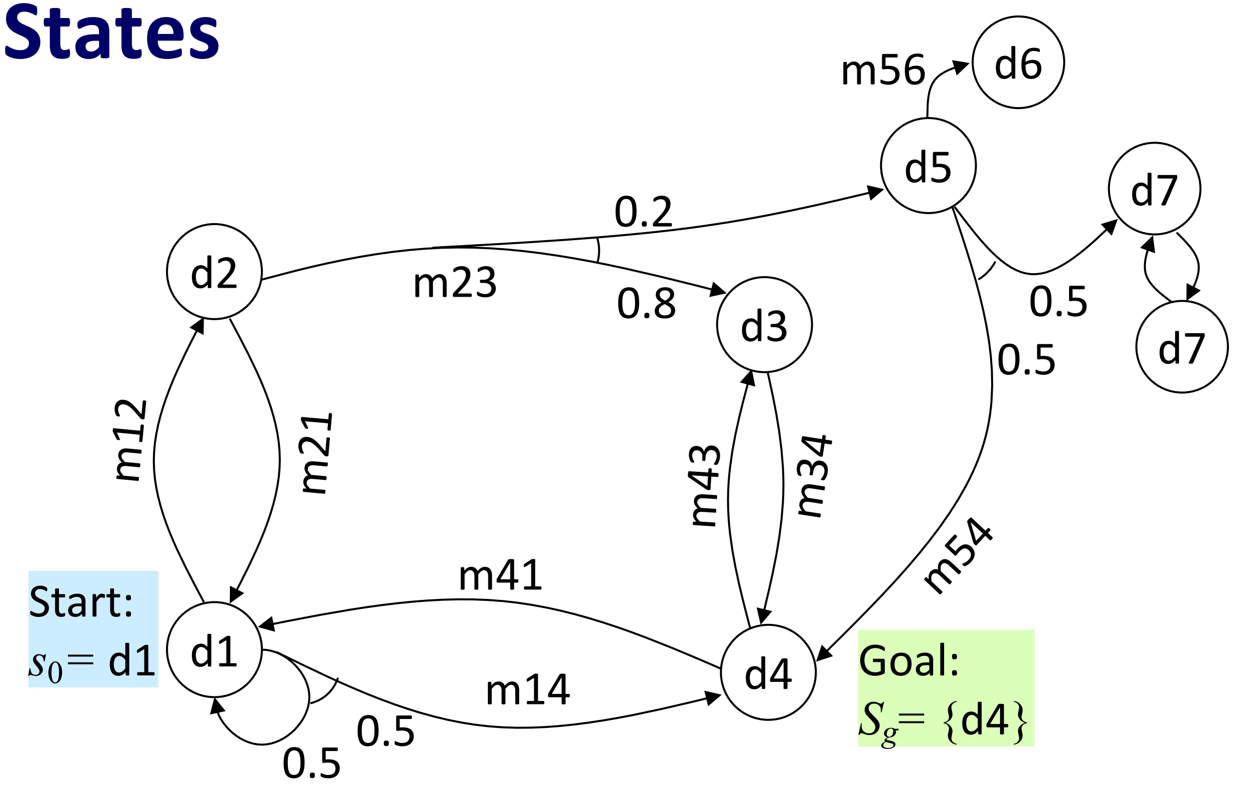
- Another cyclic safe solution:
  π_6 = {(d1, m54), (d4, m41)}

- We stop when we reach a goal, so (d4, m41) doesn't matter
  - Same histories and probabilities as for π_5



Unsafe States

- s is *safe* if $\exists\pi$ such that $\Pr(S_g \mid s, \pi) = 1$
  - e.g., d3 is safe
  - same as saying $(\Sigma$, d3, $S_g)$ has a safe solution
- s is *unsafe* if $\forall$ closed policy π defined at s, $0 < \Pr(S_g \mid s, \pi) < 1$
  - e.g., d5 is unsafe
  - same as saying $(\Sigma$, d5, $S_g)$ has an unsafe solution but no safe solution
- s is a *dead end* if $\forall\pi\ \Pr(S_g \mid s, \pi) = 0$

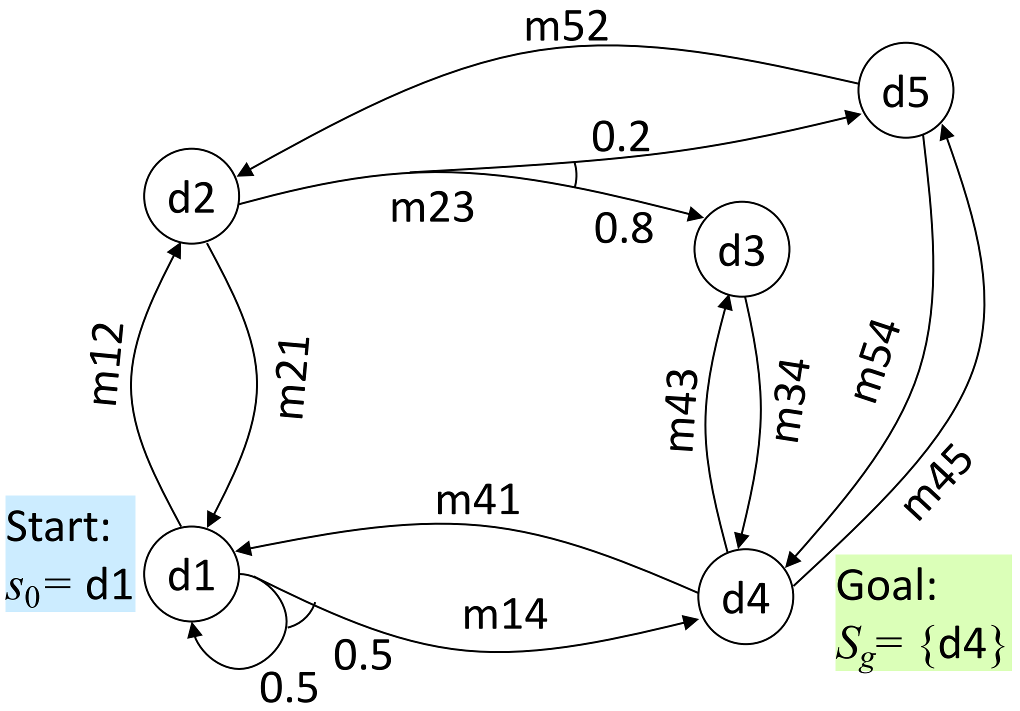


- d6 is an *explicit* dead end
  - No applicable actions
- d7 and d8 are *implicit* dead ends
  - Applicable actions, but no way to reach S_g

Safe SSPs

- An SSP is *safe* if every state in the SSP is safe
  - Same as a *safely explorable* SSP in Chapter 5

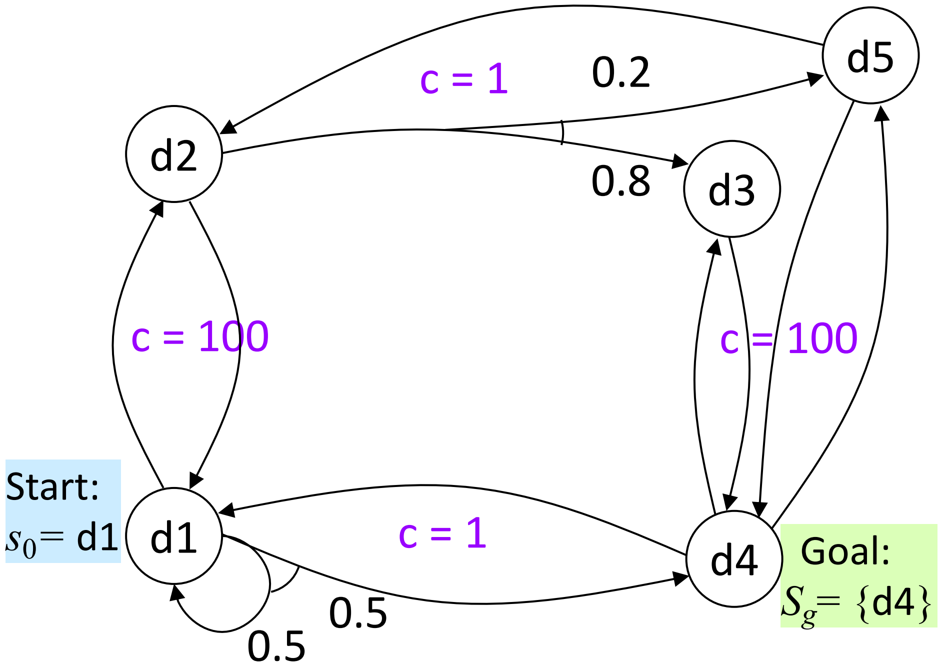
- In what follows, only consider safe SSPs



Poll: Is the above SSP safe?
A. yes
B. no

Expected Cost

- $\text{cost}(s,a)$ = cost of using a in s
- Example:
  - each "horizontal" action costs 1
  - each "vertical" action costs 100



- Let $\sigma = \langle s_0, s_1, s_2, \ldots \rangle \in H(s_0,\pi)$
  - i.e., starting at s_0, π can produce history σ
- Then $\text{cost}(\sigma \,|\, s_0, \pi) = \sum_i \text{cost}(s_i,\pi(s_i))$

- Let π be a safe solution
- At each state $s \in \text{Dom}(\pi)$, expected cost of following π to goal:
  - Weighted sum of history costs:
    - $V^\pi(s) = \sum_{\sigma \in H(s,\pi)} \Pr(\sigma \,|\, s, \pi)\, \text{cost}(\sigma \,|\, s, \pi)$ My version
  - Recursive equation

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

From the book

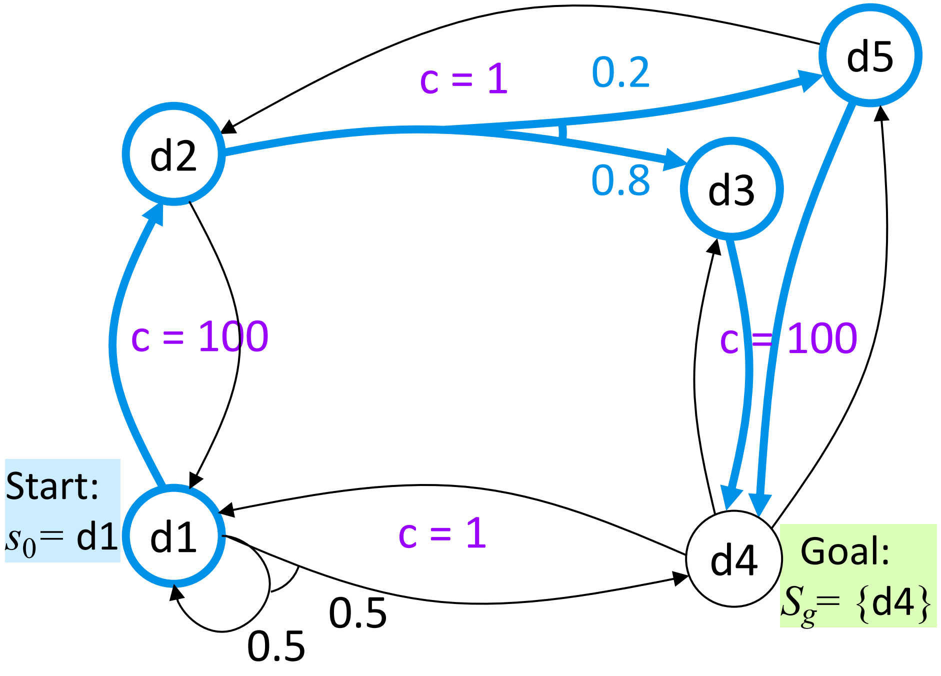
Poll: Are the two equations equivalent?
A. yes
B. no

Example

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

- Weighted sum of history costs:
  - $\sigma_1 = \langle$d1, d2, d3, d4$\rangle$
    - $\Pr(\sigma_1 \mid s_0, \pi_3) = 0.8$
    - $\text{cost}(\sigma_1 \mid s_0, \pi_3) = 100 + 1 + 100 = 201$
  - $\sigma_2 = \langle$d1, d2, d5, d4$\rangle$
    - $\Pr(\sigma_2 \mid s_0, \pi_3) = 0.2$
    - $\text{cost}(\sigma_2 \mid s_0, \pi_3) = 100 + 1 + 100 = 201$

- $V^{\pi_3}(\text{d1}) = .8(201) + .2(201) = 201$



- Recursive equation ⇒ 4 equations, 4 unknowns

$$V^{\pi_3}(\text{d1}) = 100 + V^{\pi_3}(\text{d2})$$
$$V^{\pi_3}(\text{d2}) = 1 + .8(V^{\pi_3}(\text{d3})) + .2(V^{\pi_3}(\text{d5}))$$
$$V^{\pi_3}(\text{d3}) = 100 + V^{\pi_3}(\text{d4})$$
$$V^{\pi_3}(\text{d5}) = 100 + V^{\pi_3}(\text{d4})$$
$$V^{\pi_3}(\text{d4}) = 0$$

- So $V^{\pi_3}(\text{d1}) = 100 + 1 + .8(100) + .2(100) = 201$

Example

- $\pi_7 = \{$(d1, m14), (d2, m23), (d3, m34), (d5, m54)$\}$
- Weighted sum of history costs:
  - $\sigma_5 = \langle$d1, d4 $\rangle$

    $\Pr(\sigma_5 \mid \pi_7) = ½$, $\quad$ $\text{cost}(\sigma_5 \mid \pi_7) = 1$
  - $\sigma_6 = \langle$d1, d1, d4$\rangle$

    $\Pr(\sigma_6 \mid \pi_7) = (½)^2$, $\quad$ $\text{cost}(\sigma_6 \mid \pi_7) = 2$
  - $\sigma_7 = \langle$d1, d1, d1, d4$\rangle$

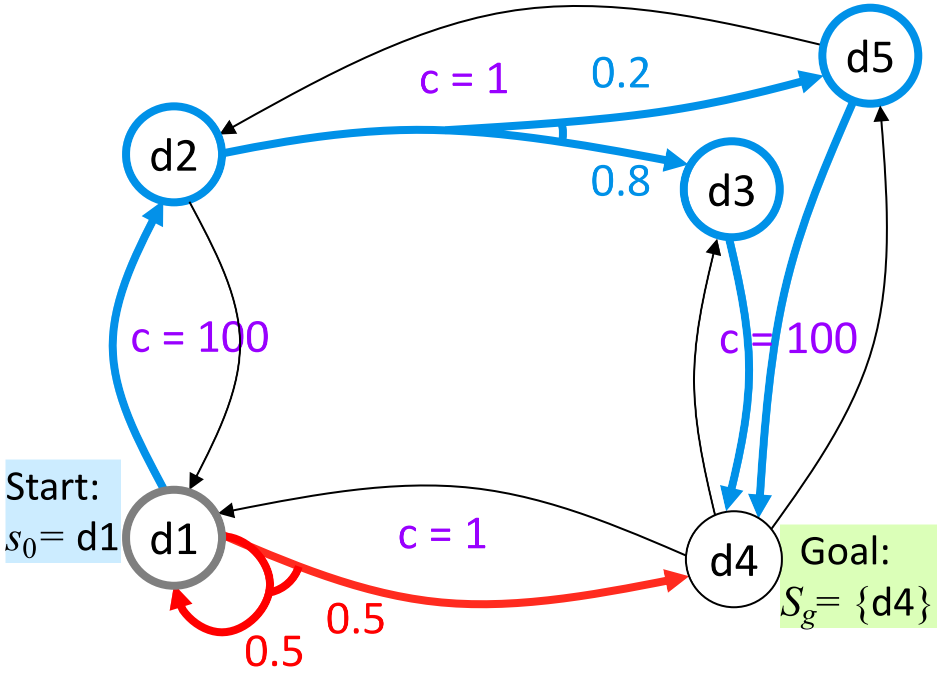
    $\Pr(\sigma_7 \mid \pi_7) = (½)^3$, $\quad$ $\text{cost}(\sigma_7 \mid \pi_7) = 3$

    …
- $V^{\pi_7}(\text{d1}) = (½)1 + (½)^2\,2 + (½)^3\,3 + \ldots = 2$
- Recursive equation:

  $V^{\pi_7}(\text{d1}) = 1 + ½(0) + ½(V^{\pi_7}(\text{d1}))$

  $½V^{\pi_7}(\text{d1}) = 1$

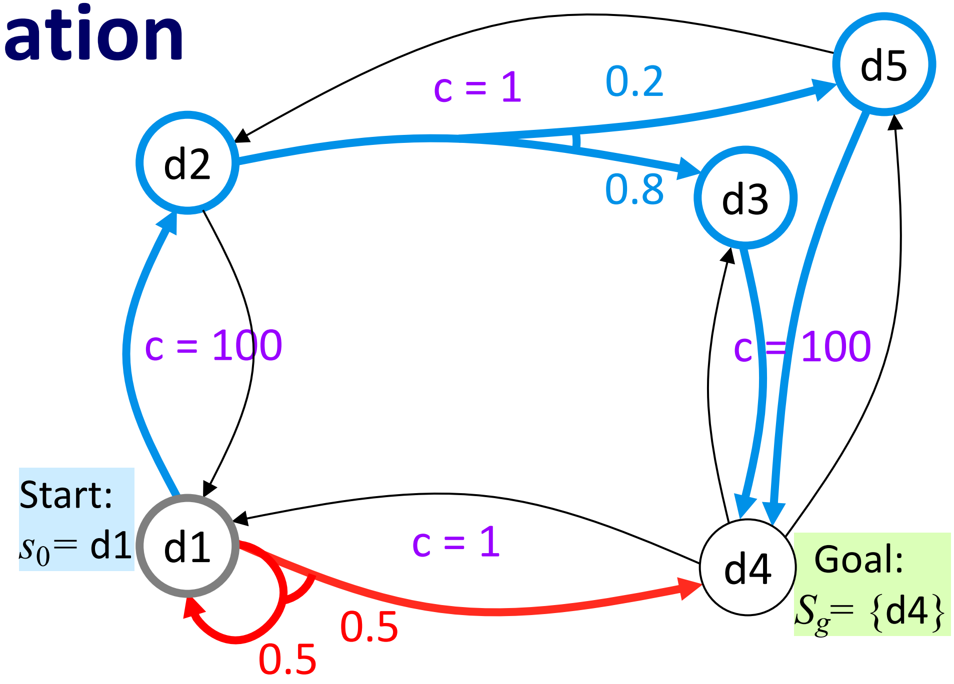
  $V^{\pi_7}(\text{d1}) = 2$



- Given safe solution π,
  - Compute V^π by solving n linear equations, n unknowns
  - n = number of states reachable from s_0 using π $= |\hat{\gamma}(s_0,\pi)|$

Planning as Optimization

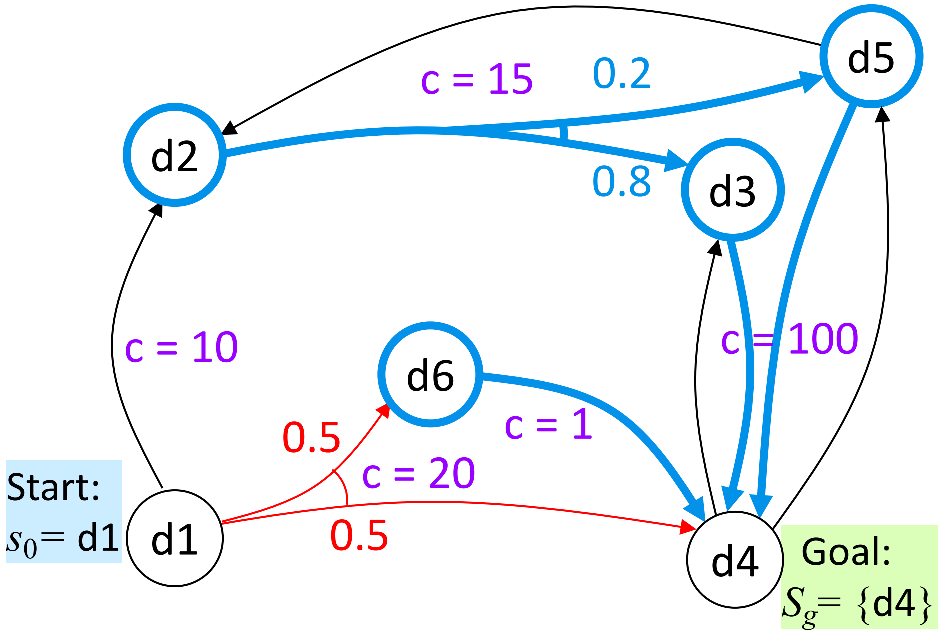
- Let π and π' be safe solutions
  - π *dominates* π′ if $V^\pi(s) \le V^{\pi'}(s)$ at every state s where they're both defined
    - i.e., every state $s \in \mathrm{Dom}(\pi) \cap \mathrm{Dom}(\pi')$
- On the previous two slides
  - π_3 = {(d1,m12), (d2,m23), (d3,m34), (d5,m54)}
  - π_7 = {(d1, m14), (d2, m23), (d3, m34), (d5, m54)}
  - d1 is the only state where they differ
    - V^{π_3}(d1) = 201; V^{π_7}(d1) = 2
  - π_7 dominates π_3
- On slide 11, π_5 = {(d1,m14)}
  - d1 is the only state where they're both defined
    - V^{π_3}(d1) = 201; V^{π_5}(d1) = 2
  - π_5 dominates π_3



- 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're both defined
  - e.g., π_5 and π_7

Planning as Optimization

- Let $V^*(s)$ = expected cost of an optimal safe solution
- Example:
  - At state d1, applicable actions m12 and m14
  - Suppose we know V^*(d2), V^*(d4), V^*(d6)
- Then
  - cost(d1,m12) = 10 + V^*(d2)
  - cost(d1,m14) = 20 + 0.5 V^*(d6) + 0.5 V^*(d4)
  - V^*(d1) = min(cost(d1,m12), cost(d1,m14))



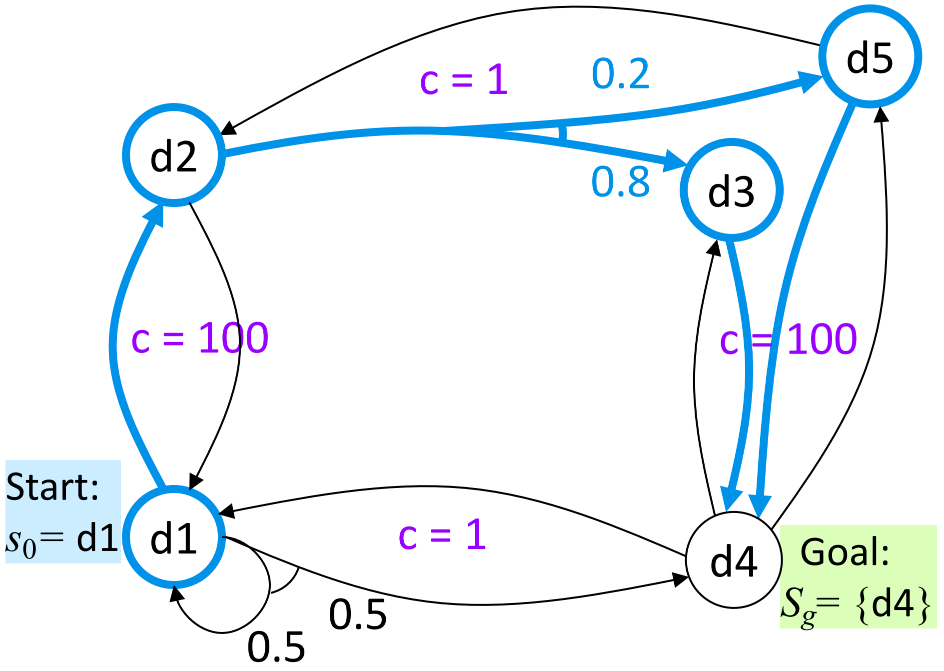
- In general,

$$V^*(s) = \begin{cases} 0, \text{ if } s \text{ is a goal} \\ \min_{a \in \text{Applicable}(s)} \{\text{cost}(s,a) + \sum_{s' \in \gamma(s,a)} \Pr(s' \mid s,a)\, V^*(s'), \text{ otherwise} \end{cases}$$

- *Optimality principle* (Bellman's theorem)

Cost to Go

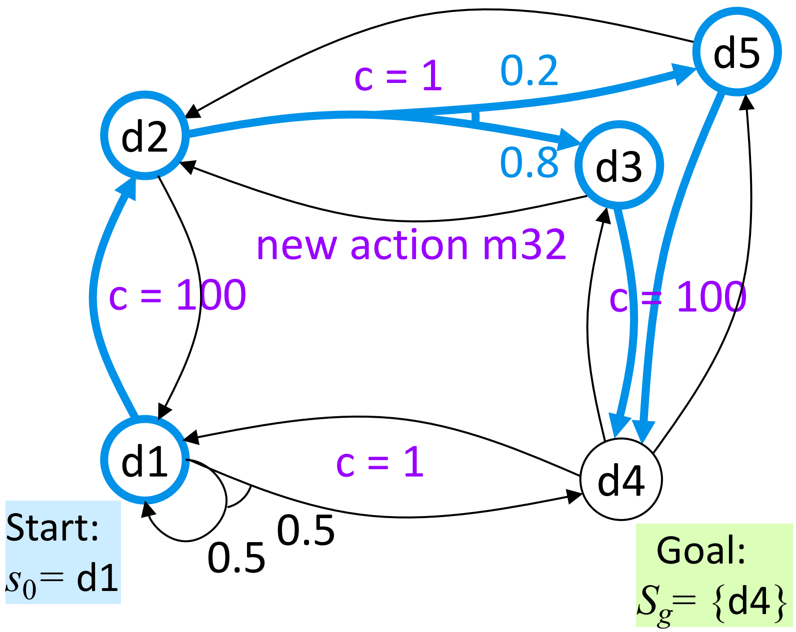
- Let (Σ, s_0, S_g) be a *safe* SSP
  - i.e., S_g is reachable from every state
  - same as *safely explorable* in Chapter 5
- Let π be a safe solution that's defined at all non-goal states
  - i.e., $\mathrm{Dom}(\pi) = S \setminus S_g$
- Compute V^π as $|S|$ equations, $|S|$ unknowns
- Let $s \in S$, $a \in \mathrm{Applicable}(s)$
  - *Cost-to-go*:
    - expected cost at s if we first use a, then use π afterward
  - $Q^\pi(s,a) = \mathrm{cost}(s,a) + \sum_{s' \in \gamma(s,a')} \Pr(s' \mid s,a')\, V^\pi(s')$



- For every $s \in S \setminus S_g$
  let $\pi'(s) \in \mathrm{argmin}_{a' \in \mathrm{Applicable}(s)}\, Q^\pi(s,a')$

Poll: Does π' dominate π?
A. always
B. sometimes
C. never

Example



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

V^π(d4) = 0

V^π(d3) = 100 + V^π(d4) = 100

V^π(d5) = 100 + V^π(d4) = 100

V^π(d2) = 1 + (0.8 V^π(d3) + 0.2 V^π(d5))
= 101

V^π(d1) = 100 + V^π(d2) = 201

Q^π(d1,m12) = 100 + 101 = 201

Q^π(d1,m14) = 1 + ½(201) + ½(0) = 101.5

⇒ argmin$_a$ Q^π(d1,a) = m14

Q^π(d2,m23) = 1 + (0.8(100) + 0.2(100))
= 101

Q^π(d2,m21) = 100 + 201 = 301

⇒ argmin$_a$ Q^π(d2,a) = m23

Q^π(d3,m34) = 100 + 0 = 100

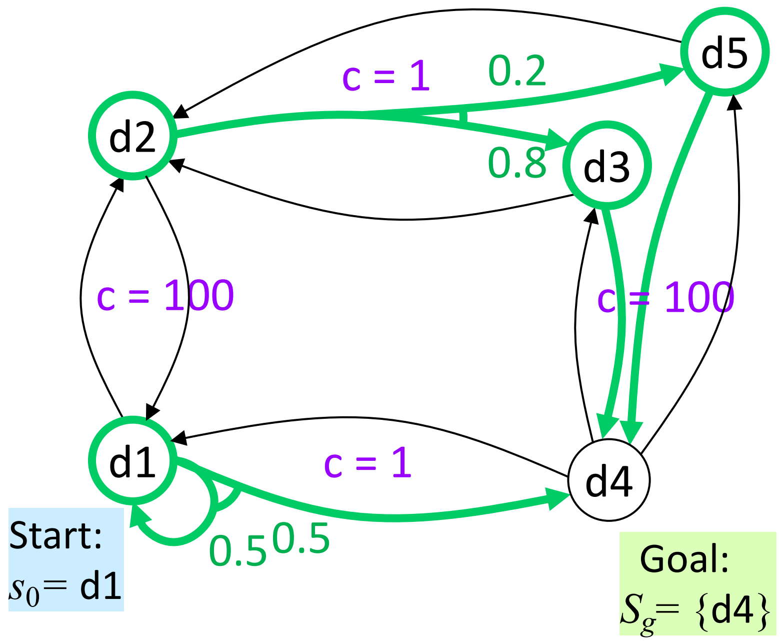
Q^π(d3,m32) = 1 + 101 = 102

⇒ argmin$_a$ Q^π(d3,a) = m34

Q^π(d5,m54) = 100 + 0 = 100

Q^π(d5,m52) = 1+101 = 102

⇒ argmin$_a$ Q^π(d5,a) = m54



$\pi' = \{$(d1,m14), (d2,m23), (d3,m34), (d5,m54)$\}$

$V^{\pi'}$(d4) = 0

$V^{\pi'}$(d3) = 100 + $V^{\pi'}$(d4) = 100

$V^{\pi'}$(d5) = 100 + $V^{\pi'}$(d4) = 100

$V^{\pi'}$(d2) = 1 + (0.8 $V^{\pi'}$(d3) + 0.2 $V^{\pi'}$(d5))
= 101

$V^{\pi'}$(d1) = 1 + ½$V^{\pi'}$(d1) + ½$V^{\pi'}$(d4)
⇒ $V^{\pi'}$(d1) = 2

Policy Iteration

- PI$(\Sigma, s_0, S_g, \pi_0)$

  $\pi \leftarrow \pi_0$

  loop

  compute $\{V^\pi(s) \mid s \in S\}$ ← $|S|$ equations, $|S|$ unknowns

  for every $s \in S \setminus S_g$ do

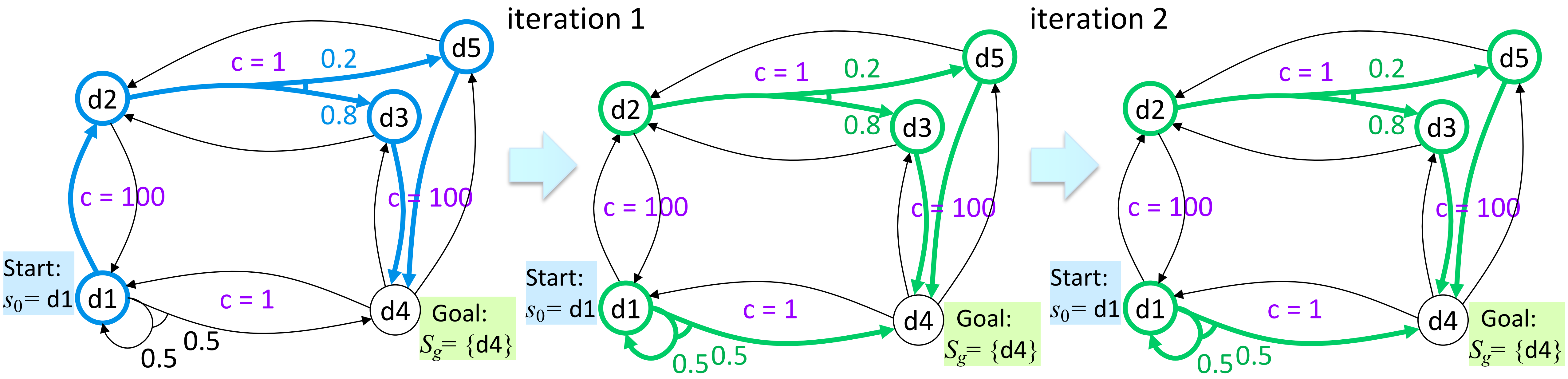
  $\pi'(s) \leftarrow \operatorname{argmin}_{a \in \text{Applicable}(s)} Q^\pi(s,a)$ ← E(cost of using a then π)

  if $\pi' = \pi$ then

  return π

  $\pi \leftarrow \pi'$

- Converges in a finite number of iterations
- Example:



Value Iteration

- V_0: a heuristic function
  - e.g., adapt an h heuristic from Chapter 2
  - $V_0(s) \approx$ optimal expected cost of getting from s to S_g
  - Require $V_0(s) = 0$ for every $s \in S_g$
- $\eta > 0$: for testing approximate convergence
- V (global variable): an array
  - $V(s) \approx$ optimal expected cost of getting from s to S_g
  - Initially $V(s) = V_0(s)\ \forall s$
  - Improve the estimates as we go along
- π (global variable): policy computed from V
- Difference from the book:
  - In the book, VI computes r as a separate step, not in Bellman-Update

VI(Σ, s_0, S_g, V_0, η)
  global $V \leftarrow V_0$
  global $\pi \leftarrow \emptyset$
  loop
    $r \leftarrow \max\{$Bellman-Update$(s) \mid s \in S \setminus S_g\}$
    if $r \le \eta$ then return π

Bellman-Update(s)
  global V, π
  $v_{\text{old}} \leftarrow V(s)$
  for every $a \in$ Applicable(s) do
    $Q(s,a) \leftarrow \text{cost}(s,a) + \sum_{s' \in S} \Pr(s'|s,a)\, V(s')$
  $V(s) \leftarrow \min_{a \in \text{Applicable}(s)} Q(s,a)$
  $\pi(s) \leftarrow \text{argmin}_{a \in \text{Applicable}(s)} Q(s,a)$
  return $|V(s) - v_{\text{old}}|$

Iteration 1

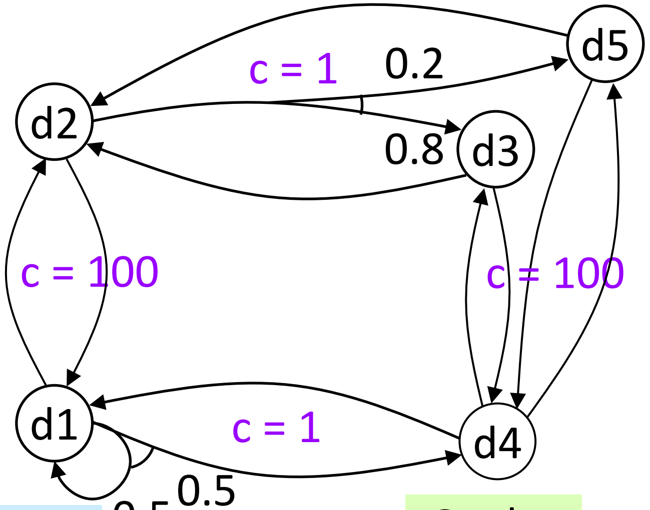
VI(Σ, s_0, S_g, V_0, η)
global $V \leftarrow V_0$
global $\pi \leftarrow \emptyset$
loop
$r \leftarrow \max\{$Bellman-Update$(s) \mid s \in S \setminus S_g\}$
if $r \leq \eta$ then return π

Bellman-Update(s)
global V, π
$v_{old} \leftarrow V(s)$
for every $a \in$ Applicable(s) do
$Q(s,a) \leftarrow \text{cost}(s,a) + \sum_{s' \in S} \Pr(s'|s,a)\ V(s')$
$V(s) \leftarrow \min_{a \in \text{Applicable}(s)} Q(s,a)$
$\pi(s) \leftarrow \text{argmin}_{a \in \text{Applicable}(s)} Q(s,a)$
return $|V(s) - v_{old}|$

$\eta = 0.2$

initial values

$V(\text{d1}) = 0$
$V(\text{d2}) = 0$
$V(\text{d3}) = 0$
$V(\text{d5}) = 0$



Start:
s_0= d1

Goal:
S_g= {d4}

$Q(\text{d1,m12}) = 100 + 0 = 100$
$Q(\text{d1,m14}) = 1 + ½(0) + ½(0) = 1$
$V(\text{d1}) = 1$; $\pi(\text{d1}) = \text{m14}$; $1 - 0 = 1$

$Q(\text{d2,m21}) = 100 + 1 = 101$
$Q(\text{d2,m23}) = 1 + .8(0) + .2(0) = 1$
$V(\text{d2}) = 1$; $\pi(\text{d2}) = \text{m23}$; $1 - 0 = 1$

$Q(\text{d3,m32}) = 1 + 1 = 2$
$Q(\text{d3,m34}) = 100 + 0 = 100$
$V(\text{d3}) = 2$; $\pi(\text{d3}) = \text{m32}$; $2 - 0 = 2$

$Q(\text{d5,m52}) = 1 + 1 = 2$
$Q(\text{d5,m54}) = 100 + 0 = 100$
$V(\text{d5}) = 2$; $\pi(\text{d5}) = \text{m52}$; $2 - 0 = 2$

$r = \max(1, 1, 2, 2) = 2$

Iteration 2

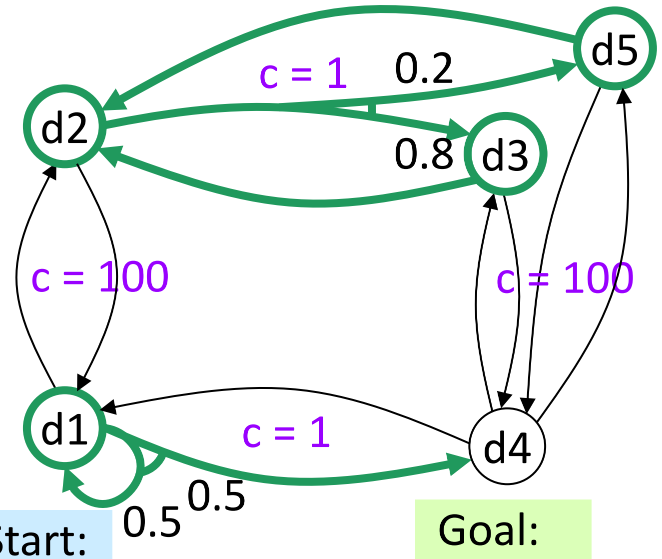
VI(Σ, s_0, S_g, V_0, η)
- global $V \leftarrow V_0$
- global $\pi \leftarrow \emptyset$
- loop
  - $r \leftarrow \max\{$Bellman-Update$(s) \mid s \in S \setminus S_g\}$
  - if $r \le \eta$ then return π

Bellman-Update(s)
- global V, π
- $v_{\text{old}} \leftarrow V(s)$
- for every $a \in$ Applicable(s) do
  - $Q(s,a) \leftarrow \text{cost}(s,a) + \sum_{s' \in S} \Pr(s'|s,a)\, V(s')$
- $V(s) \leftarrow \min_{a \in \text{Applicable}(s)} Q(s,a)$
- $\pi(s) \leftarrow \text{argmin}_{a \in \text{Applicable}(s)} Q(s,a)$
- return $|V(s) - v_{\text{old}}|$

$\eta = 0.2$

from iteration 1

$V(\text{d1}) = 1$
$V(\text{d2}) = 1$
$V(\text{d3}) = 2$
$V(\text{d5}) = 2$
$\pi(\text{d1}) = \text{m14}$
$\pi(\text{d2}) = \text{m23}$
$\pi(\text{d2}) = \text{m32}$
$\pi(\text{d5}) = \text{m52}$



Start:
s_0= d1

Goal:
S_g= {d4}

$Q(\text{d1,m12}) = 100 + 1 = 101$
$Q(\text{d1,m14}) = 1 + ½(1) + ½(0) = 1½$
$V(\text{d1}) = 1½$; $\pi(\text{d1}) = \text{m14}$; $1½ - 1 = ½$

$Q(\text{d2,m21}) = 100 + 1½ = 101½$
$Q(\text{d2,m23}) = 1 + .8(2) + .2(2) = 3$
$V(\text{d2}) = 3$; $\pi(\text{d2}) = \text{m23}$; $3 - 1 = 2$

$Q(\text{d3,m32}) = 1 + 3 = 4$
$Q(\text{d3,m34}) = 100 + 0 = 100$
$V(\text{d3}) = 4$; $\pi(\text{d3}) = \text{m32}$; $4 - 2 = 2$

$Q(\text{d5,m52}) = 1 + 3 = 4$
$Q(\text{d5,m54}) = 100 + 0 = 100$
$V(\text{d5}) = 4$; $\pi(\text{d5}) = \text{m52}$; $4 - 2 = 2$

$r = \max(½, 2, 2, 2) = 2$

Iteration 3

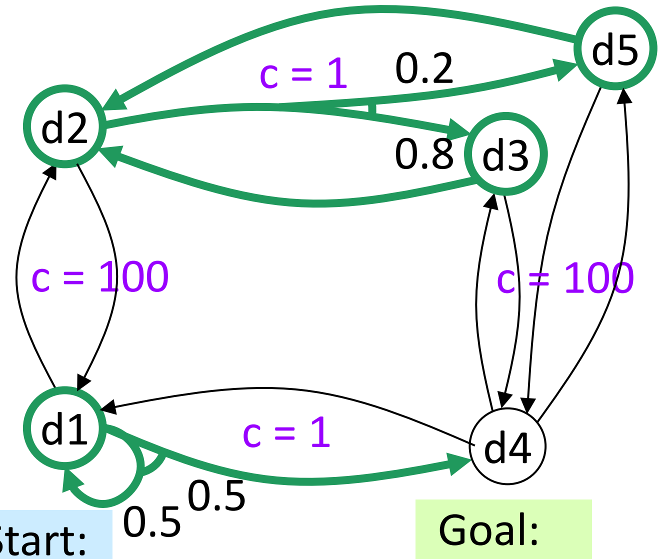
VI(Σ, s_0, S_g, V_0, η)
  global $V \leftarrow V_0$
  global $\pi \leftarrow \emptyset$
  loop
    $r \leftarrow \max\{$Bellman-Update$(s) \mid s \in S \setminus S_g\}$
    if $r \le \eta$ then return π

Bellman-Update(s)
  global V, π
  $v_{old} \leftarrow V(s)$
  for every $a \in$ Applicable(s) do
    $Q(s,a) \leftarrow \text{cost}(s,a) + \sum_{s' \in S} \Pr(s'|s,a)\, V(s')$
  $V(s) \leftarrow \min_{a \in \text{Applicable}(s)} Q(s,a)$
  $\pi(s) \leftarrow \text{argmin}_{a \in \text{Applicable}(s)} Q(s,a)$
  return $|V(s) - v_{old}|$

$\eta = 0.2$

from iteration 2

$V(\text{d1}) = 1\tfrac{1}{2}$
$V(\text{d2}) = 3$
$V(\text{d3}) = 4$
$V(\text{d5}) = 4$
$\pi(\text{d1}) = \text{m14}$
$\pi(\text{d2}) = \text{m23}$
$\pi(\text{d2}) = \text{m32}$
$\pi(\text{d5}) = \text{m52}$



Start:
s_0= d1

Goal:
S_g= {d4}

$Q(\text{d1,m12}) = 100 + 3 = 103$
$Q(\text{d1,m14}) = 1 + \tfrac{1}{2}(1\tfrac{1}{2}) + \tfrac{1}{2}(0) = 1\tfrac{3}{4}$
$V(\text{d1}) = 1\tfrac{3}{4}$; $\pi(\text{d1}) = \text{m14}$; $1\tfrac{3}{4} - 1\tfrac{1}{2} = \tfrac{1}{4}$

$Q(\text{d2,m21}) = 100 + 1\tfrac{3}{4} = 101\tfrac{3}{4}$
$Q(\text{d2,m23}) = 1 + .8(4) + .2(4) = 5$
$V(\text{d2}) = 5$; $\pi(\text{d2}) = \text{m23}$; $5 - 3 = 2$

$Q(\text{d3,m32}) = 1 + 5 = 6$
$Q(\text{d3,m34}) = 100 + 0 = 100$
$V(\text{d3}) = 6$; $\pi(\text{d3}) = \text{m32}$; $6 - 4 = 2$

$Q(\text{d5,m52}) = 1 + 5 = 6$
$Q(\text{d5,m54}) = 100 + 0 = 100$
$V(\text{d5}) = 6$; $\pi(\text{d5}) = \text{m52}$; $6 - 4 = 2$

$r = \max(\tfrac{1}{4}, 2, 2, 2) = 2$

Iteration 4

VI(Σ, s_0, S_g, V_0, η)
  global $V \leftarrow V_0$
  global $\pi \leftarrow \emptyset$
  loop
    $r \leftarrow \max\{$Bellman-Update(s) $\mid s \in S \setminus S_g\}$
    if $r \leq \eta$ then return π

Bellman-Update(s)
  global V, π
  $v_{\text{old}} \leftarrow V(s)$
  for every $a \in$ Applicable(s) do
    $Q(s,a) \leftarrow \text{cost}(s,a) + \sum_{s' \in S} \Pr(s'|s,a)\, V(s')$
  $V(s) \leftarrow \min_{a \in \text{Applicable}(s)} Q(s,a)$
  $\pi(s) \leftarrow \text{argmin}_{a \in \text{Applicable}(s)} Q(s,a)$
  return $|V(s) - v_{\text{old}}|$

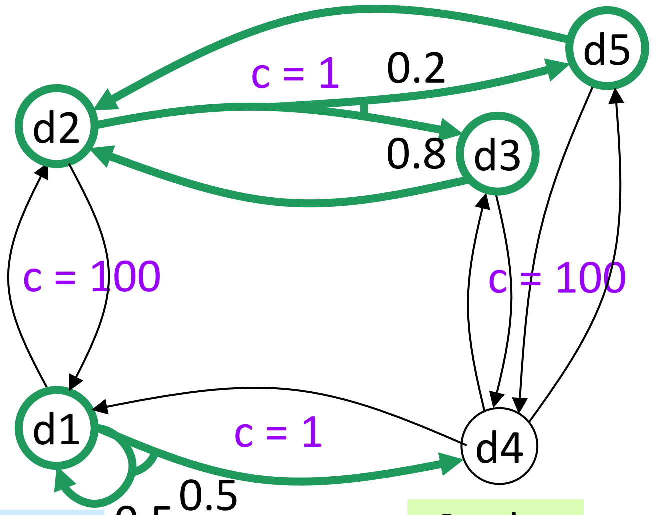
Poll: How many iterations until termination?

A. $i \leq 10$
B. $10 < i \leq 20$
C. $20 < i \leq 40$
D. $40 < i \leq 80$
E. $80 < i \leq 160$
F. $160 < i$

$\eta = 0.2$

from iteration 3

V(d1) = $1^3/_4$
V(d2) = 5
V(d3) = 6
V(d5) = 6
π(d1) = m14
π(d2) = m23
π(d2) = m32
π(d5) = m52



Start:
s_0 = d1

Goal:
S_g = {d4}

Q(d1,m12) = 100 + 5 = 105
Q(d1,m14) = 1 + ½($1^3/_4$) + ½(0) = $1^7/_8$
V(d1) = $1^7/_8$; π(d1) = m14; $1^7/_8 - 1^3/_4 = {}^1/_8$

Q(d2,m21) = 100 + $1^7/_8$ = $101^7/_8$
Q(d2,m23) = 1 + .8(6) + .2(6) = 7
V(d2) = 7; π(d2) = m23; 7 – 5 = 2

Q(d3,m32) = 1 + 7 = 8
Q(d3,m34) = 100 + 0 = 100
V(d3) = 8; π(d3) = m32; 8 – 6 = 2

Q(d5,m52) = 1 + 7 = 8
Q(d5,m54) = 100 + 0 = 100
V(d5) = 8; π(d5) = m52; 8 – 6 = 2

$r = \max({}^1/_8, 2, 2, 2) = 2$

Iteration 1, with a better V_0

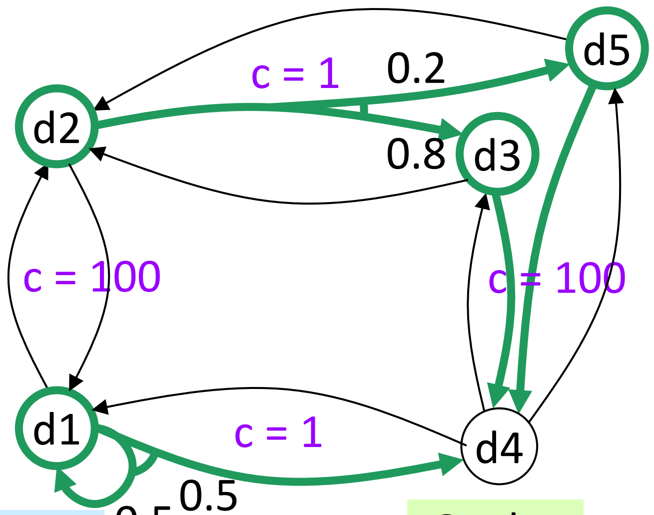
VI(Σ, s_0, S_g, V_0, η)
- global $V \leftarrow V_0$
- global $\pi \leftarrow \emptyset$
- loop
  - $r \leftarrow \max\{$Bellman-Update$(s) \mid s \in S \setminus S_g\}$
  - if $r \le \eta$ then return π

Bellman-Update(s)
- global V, π
- $v_{\text{old}} \leftarrow V(s)$
- for every $a \in$ Applicable(s) do
  - $Q(s,a) \leftarrow \text{cost}(s,a) + \sum_{s' \in S} \Pr(s'|s,a)\, V(s')$
- $V(s) \leftarrow \min_{a \in \text{Applicable}(s)} Q(s,a)$
- $\pi(s) \leftarrow \text{argmin}_{a \in \text{Applicable}(s)} Q(s,a)$
- return $|V(s) - v_{\text{old}}|$

$\eta = 0.2$

initial values

V(d1) = 2
V(d2) = 101
V(d3) = 100
V(d5) = 100



Start:
s_0= d1

Goal:
S_g= {d4}

Q(d1,m12) = 100 + 101 = 201
Q(d1,m14) = 1 + ½(0) + ½(2) = 2
V(d1) = 2; π(d1) = m14; 2 – 2 = 0

Q(d2,m21) = 100 + 2 = 102
Q(d2,m23) = 1 + .8(100) + .2(100) = 101
V(d2) = 101; π(d2) = m23; 101 – 101 = 0

Q(d3,m32) = 1 + 101 = 102
Q(d3,m34) = 100 + 0 = 100
V(d3) = 100; π(d3) = m34; 100 – 100 = 0

Q(d5,m52) = 1 + 101 = 102
Q(d5,m54) = 100 + 0 = 100
V(d5) = 100; π(d5) = m54; 100 – 100 = 0

r = max(0, 0, 0, 0) = 0 < η
VI returns π

Discussion

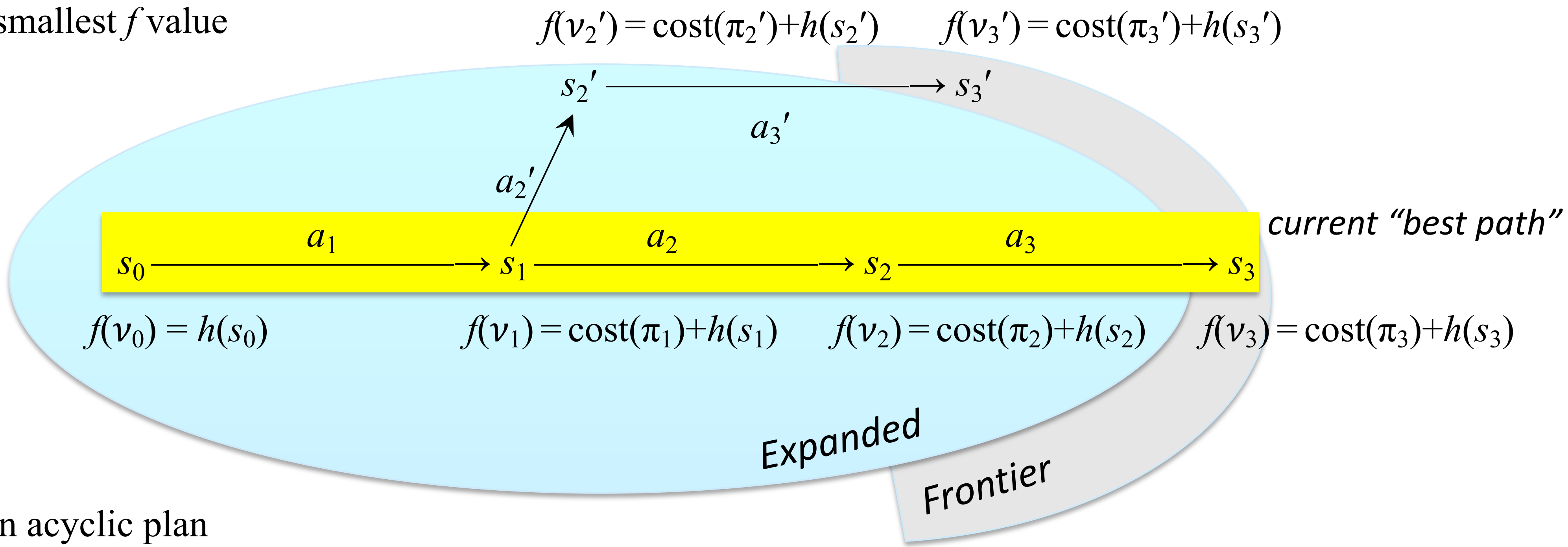
- In each iteration of Policy Iteration
  - Compute V^π for current π
    - $|S|$ linear equations, $|S|$ unknowns
  - Use V^π to choose new π
    - argmin Q value at each state
- More work per iteration than value iteration
  - Needs to solve simultaneous equations
- Usually converges in a smaller number of iterations

- In each iteration of Value Iteration
  - Compute new V: min Q value at each state
  - New V is a revised set of heuristic estimates
    - Doesn't depend on any π
  - Can let π = argmin Q value at each state
    - But V isn't V^π for π or any other policy
  - Less work per iteration: doesn't 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

Digression: A* Search

- All actions are deterministic
- Each node v_i is a pair (π_i, s_i)
- $f(v_i)$ = cost of following π_i from s_0 to $s_i + h(s_i)$
- A* expands the frontier node that has the smallest f value



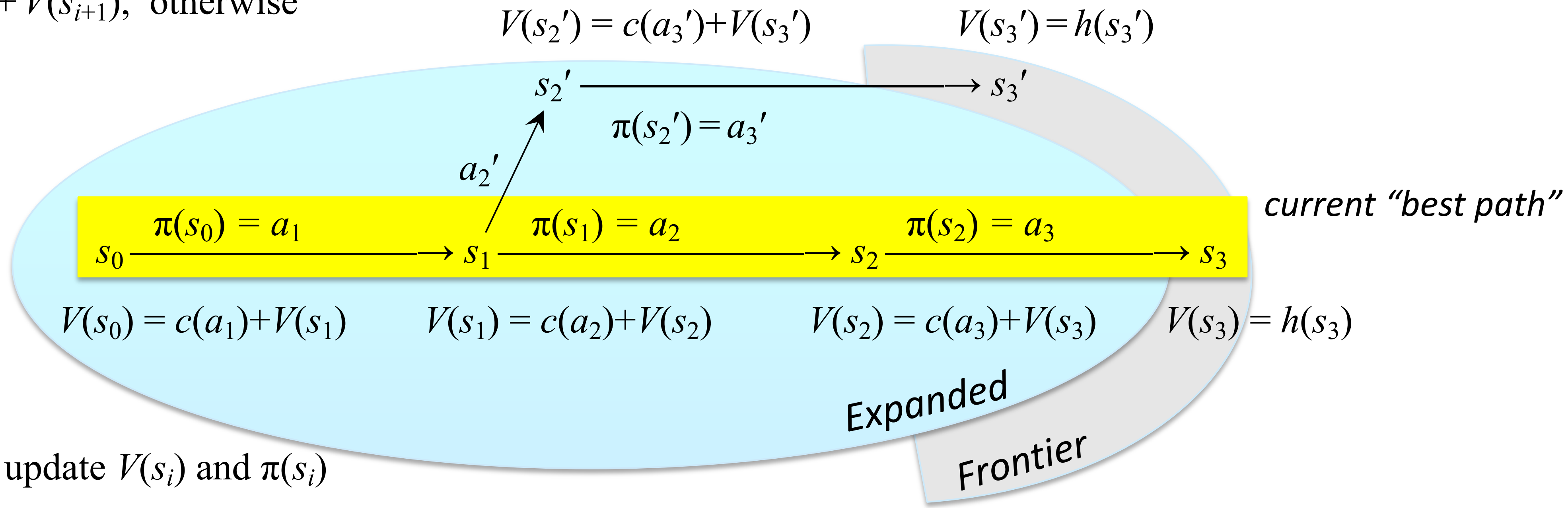
- A* creates an acyclic plan
- In deterministic planning domains, equivalent to an acyclic policy
  - Above, $\langle a_1, a_2, a_3 \rangle$ is equivalent to $\{(s_0,a_1), (s_1,a_2), (s_2,a_3)\}$
- We can rewrite A* to create an acyclic policy …

Equivalent Approach Using Policies

- Global policy $\pi = \{(s_0,a_1), (s_1,a_2), (s_2,a_3), (s_3,a_4), (s_3',a_4')\}$
- $V(s_i)$ = cost(following π **from s_i** to a frontier node) + h(the frontier node)

$$= \begin{cases} h(s_i), & \text{if } s_i \text{ is a frontier node} \\ c(a_i) + V(s_{i+1}), & \text{otherwise} \end{cases}$$

- Expands the same nodes as A*, in the same order

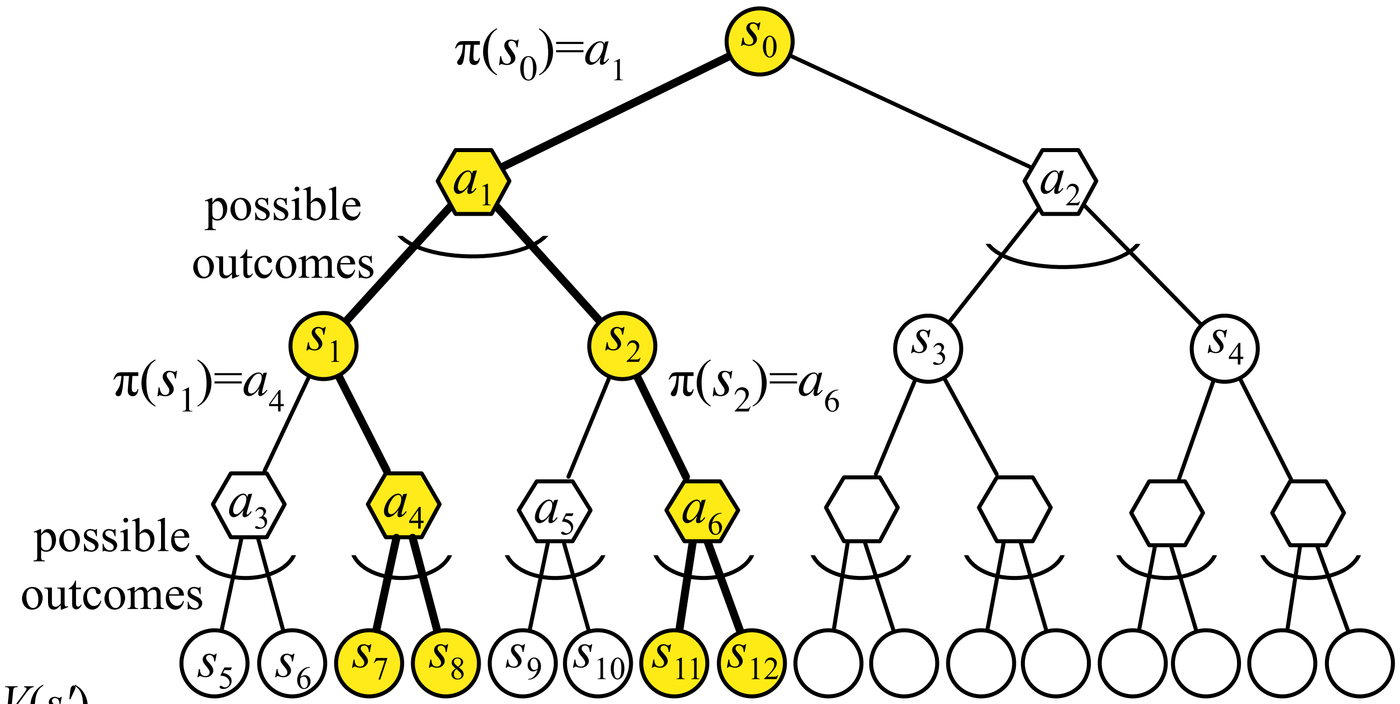


- Expand s_3
- For i = 3,2,1,0, update $V(s_i)$ and $\pi(s_i)$
  - Updating:
    - $V(s) = \min_{a \in \text{Applicable}(s)} c(a) + V(\gamma(s,a))$
    - $\pi(s) = \text{argmin}_{a \in \text{Applicable}(s)} c(a) + V(\gamma(s,a))$
  - E.g., if $c(a_2')+V(s_2') < c(a_2)+V(s_2)$ then $\pi(s_1) \leftarrow a_2'$

Poll: If $V(s)$ doesn't change, do we need to update its parent(s)?
A. yes
B. no

AO* (Basic Idea)

- An SSP can be represented as an AND/OR graph
  - OR nodes: choose an action
  - AND nodes: action's outcomes
- AO*: generalization of A* for *acyclic* SSPs
- $leaves(s_0,\pi) = \{s_7, s_8, s_{11}, s_{12}\}$
  - Expand one of them, e.g., s_{11}
- Going bottom-up, update V and π values for s_{11} and its ancestors
  - $V(s) = \min_{a \in \text{Applicable}(s)} c(a) + \sum_{s' \in \gamma(s,a)} P(s' \mid s,a)\ V(s')$
  - $\pi(s) = \text{argmin}_{a \in \text{Applicable}(s)} c(a) + \sum_{s' \in \gamma(s,a)} P(s' \mid s,a)\ V(s')$
- *e.g.,*
  - $V(s_2) = \min(c(a_5)+V(s_9)+V(s_{10}), c(a_6)+V(s_{11})+V(s_{12}))$

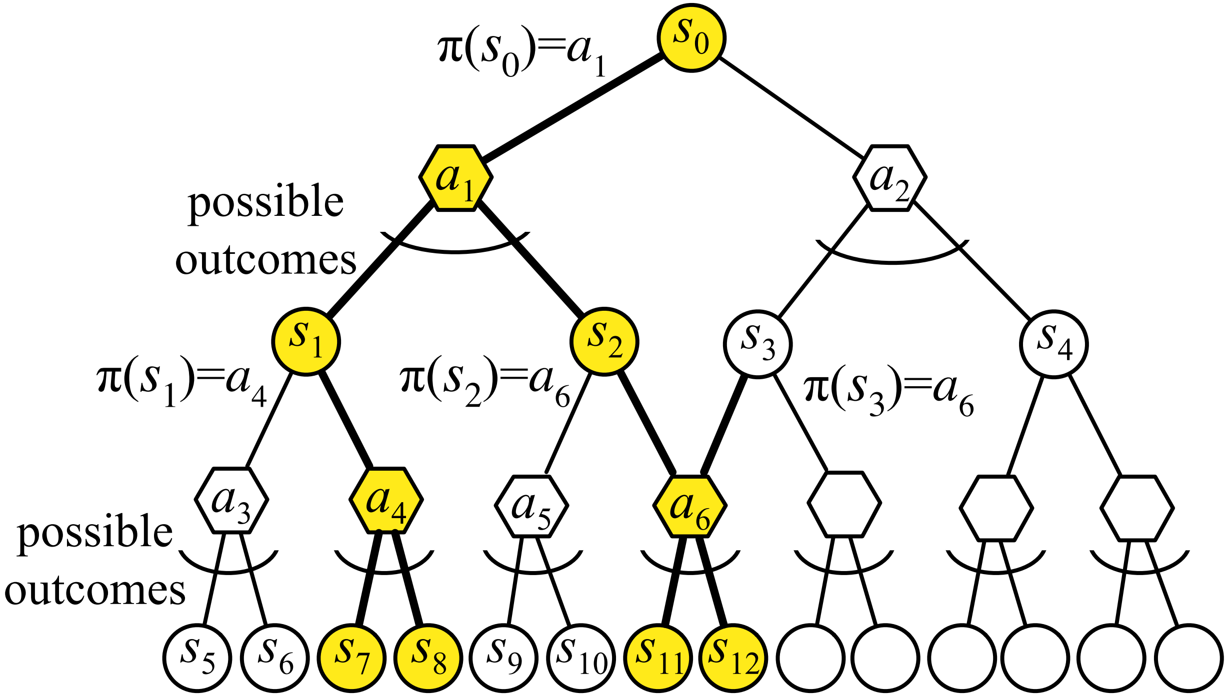


Poll: If $V(s)$ doesn't change, do we need to update its parent(s)?
A. yes
B. no

Poll: Will this work correctly if the acyclic AND/OR graph isn't a tree?
A. yes
B. no

AO* (Basic Idea)

- An SSP can be represented as an AND/OR graph
  - OR nodes: choose an action
  - AND nodes: action's outcomes
- AO*: generalization of A* for *acyclic* SSPs
- $leaves(s_0,\pi) = \{s_7, s_8, s_{11}, s_{12}\}$
  - Expand one of them, e.g., s_{11}
- Going bottom-up, update V and π values for s_{11} **and its ancestors**
  - $V(s) = \min_{a \in \text{Applicable}(s)} c(a) + \sum_{s' \in \gamma(s,a)} V(s')$
  - $\pi(s) = \text{argmin}_{a \in \text{Applicable}(s)} c(a) + \sum_{s' \in \gamma(s,a)} V(s')$
- *e.g.*,
  - $V(s_2) = \min(c(a_5)+V(s_9)+V(s_{10}),\ c(a_6)+V(s_{11})+V(s_{12}))$



Poll: If we change $V(s)$ and s has more than one parent, do we need to update all of them?

A. yes
B. no
C. sometimes

AO*

AO* (Σ, s_0, S_g, V_0)

acyclic

  global $\pi \leftarrow \emptyset$

  global *Envelope* $\leftarrow \{s_0\}$

like *Expanded* ∪ *Frontier* in A*

  global V; $V(s_0) \leftarrow V_0(s_0)$

  while *Fringe* $\neq \emptyset$ do

    select a state $s \in$ *Fringe*

    for all $a \in$ Applicable(s) and $s' \in \gamma(s,a)$ do

      if $s' \notin$ *Envelope* then

        add s' to *Envelope*

        $V(s') \leftarrow V_0(s')$

    AO-Update(s)

  return π

expand s

Fringe = *Envelope* \ (Dom(π) ∪ S_g)

- like non-goal *Frontier* states in A*
- *Fringe* changes when Bellman-Update changes π

Bellman-Update(s)

  global V, π

  $v_{\text{old}} \leftarrow V(s)$

  for every $a \in$ Applicable(s) do

    $Q(s,a) \leftarrow \text{cost}(s,a) + \sum_{s' \in S} \Pr(s'|s,a)\, V(s')$

  $V(s) \leftarrow \min_{a \in \text{Applicable}(s)} Q(s,a)$

  $\pi(s) \leftarrow \text{argmin}_{a \in \text{Applicable}(s)} Q(s,a)$

  return $|V(s) - v_{\text{old}}|$

not needed this time

AO-Update(s)   // update V and π values of s and its ancestors

  $Z \leftarrow \{s\}$ // states that need updating

  while $Z \neq \emptyset$ do

    select $s \in Z$ such that $\hat{\gamma}(s,\pi(s)) \cap Z = \{s\}$

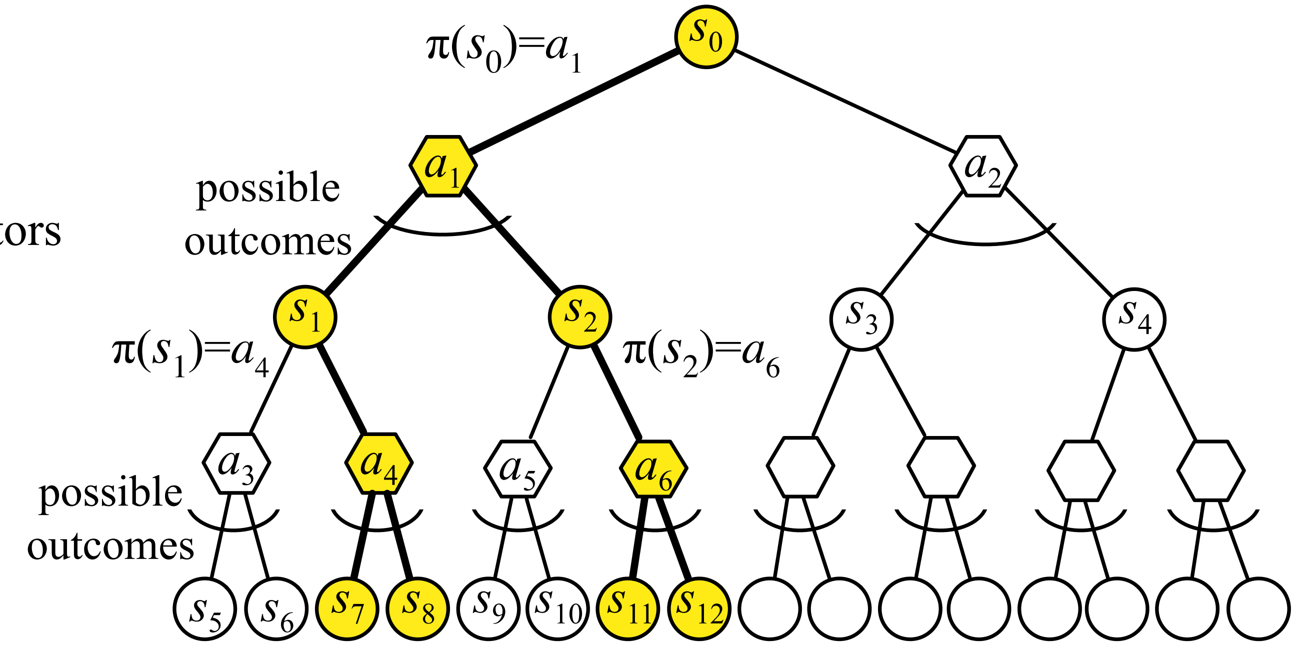
    remove s from Z

    Bellman-Update(s)

    $Z \leftarrow Z \cup \{s' \in \textit{Envelope} \mid s \in \gamma(s',\pi)\}$

a "lowest" state in Z

add s's parents

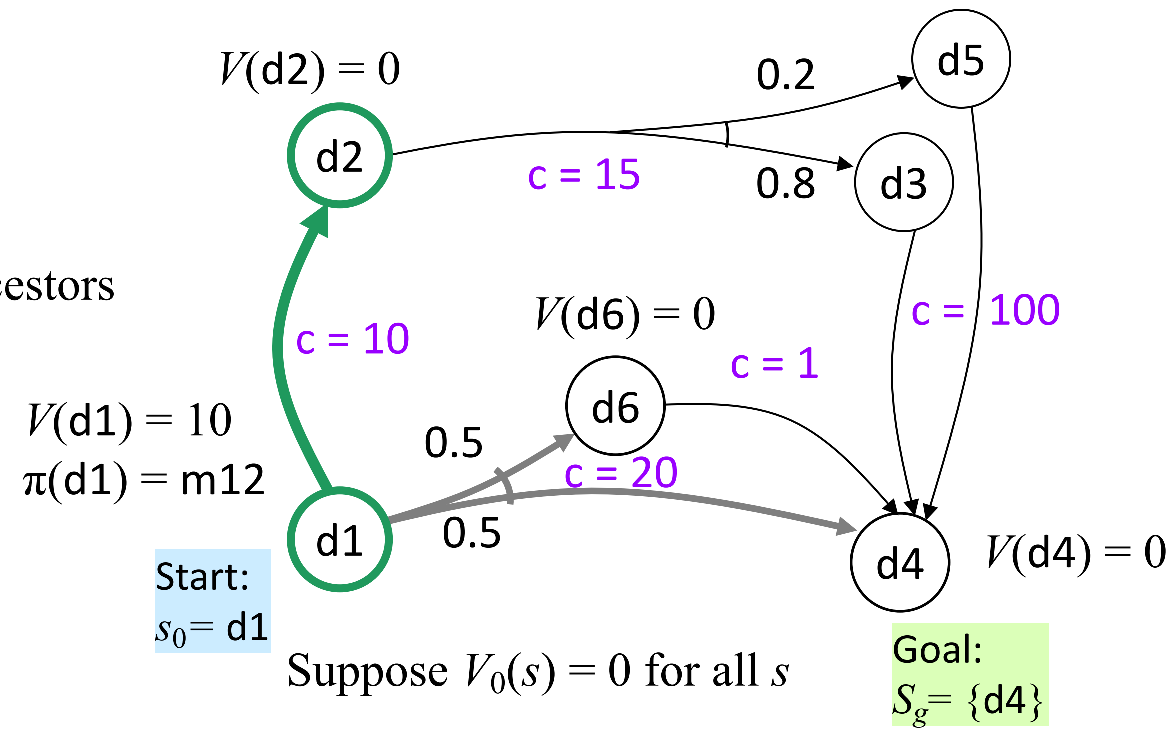


AO*

AO* (Σ, s_0, S_g, V_0)
  global $\pi \leftarrow \emptyset$
  global $Envelope \leftarrow \{s_0\}$
  global V; $V(s_0) \leftarrow V_0(s_0)$
  while $Fringe \neq \emptyset$ do
    select a state $s \in Fringe$
    for all $a \in$ Applicable(s) and $s' \in \gamma(s,a)$ do
      if $s' \notin Envelope$ then
        add s' to $Envelope$
        $V(s') \leftarrow V_0(s')$
    AO-Update(s)
  return π

AO-Update(s) // update V and π values of s and its ancestors
  $Z \leftarrow \{s\}$ // states that need updating
  while $Z \neq \emptyset$ do
    select $s \in Z$ such that $\hat{\gamma}(s,\pi(s)) \cap Z = \{s\}$
    remove s from Z
    Bellman-Update(s)
    $Z \leftarrow Z \cup \{s' \in Envelope \mid s \in \gamma(s',\pi)\}$

Bellman-Update(s)
  global V, π
  $v_{\text{old}} \leftarrow V(s)$
  for every $a \in$ Applicable(s) do
    $Q(s,a) \leftarrow \text{cost}(s,a) + \sum_{s' \in S} \Pr(s'|s,a)\, V(s')$
  $V(s) \leftarrow \min_{a \in \text{Applicable}(s)} Q(s,a)$
  $\pi(s) \leftarrow \text{argmin}_{a \in \text{Applicable}(s)} Q(s,a)$
  return $|V(s) - v_{\text{old}}|$

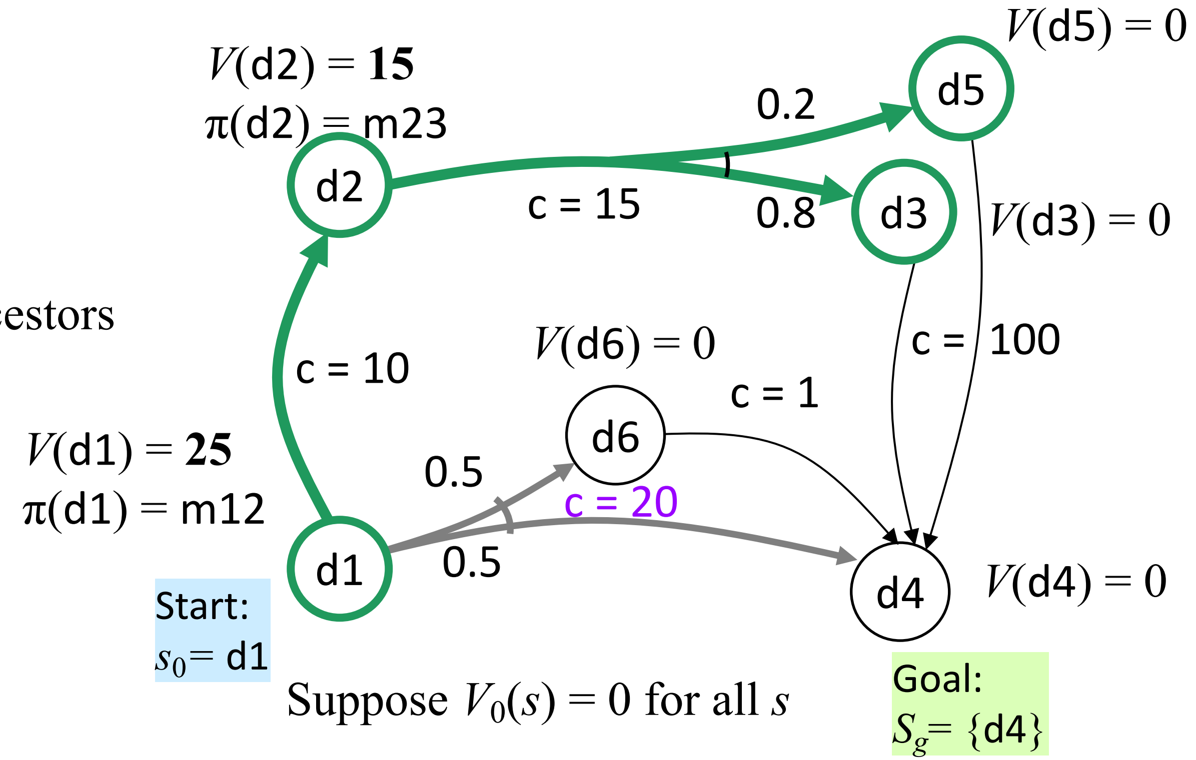


AO*

AO* (Σ, s_0, S_g, V_0)
    global $\pi \leftarrow \emptyset$
    global $Envelope \leftarrow \{s_0\}$
    global V; $V(s_0) \leftarrow V_0(s_0)$
    while $Fringe \neq \emptyset$ do
        select a state $s \in Fringe$
        for all $a \in$ Applicable(s) and $s' \in \gamma(s,a)$ do
            if $s' \notin Envelope$ then
                add s' to $Envelope$
                $V(s') \leftarrow V_0(s')$
        AO-Update(s)
    return π

AO-Update(s)   // update V and π values of s and its ancestors
    $Z \leftarrow \{s\}$ // states that need updating
    while $Z \neq \emptyset$ do
        select $s \in Z$ such that $\hat{\gamma}(s,\pi(s)) \cap Z = \{s\}$
        remove s from Z
        Bellman-Update(s)
        $Z \leftarrow Z \cup \{s' \in Envelope \mid s \in \gamma(s',\pi)\}$

Bellman-Update(s)
    global V, π
    $v_{\text{old}} \leftarrow V(s)$
    for every $a \in$ Applicable(s) do
        $Q(s,a) \leftarrow \text{cost}(s,a) + \sum_{s' \in S} \Pr(s'|s,a)\, V(s')$
    $V(s) \leftarrow \min_{a \in \text{Applicable}(s)} Q(s,a)$
    $\pi(s) \leftarrow \text{argmin}_{a \in \text{Applicable}(s)} Q(s,a)$
    return $|V(s) - v_{\text{old}}|$



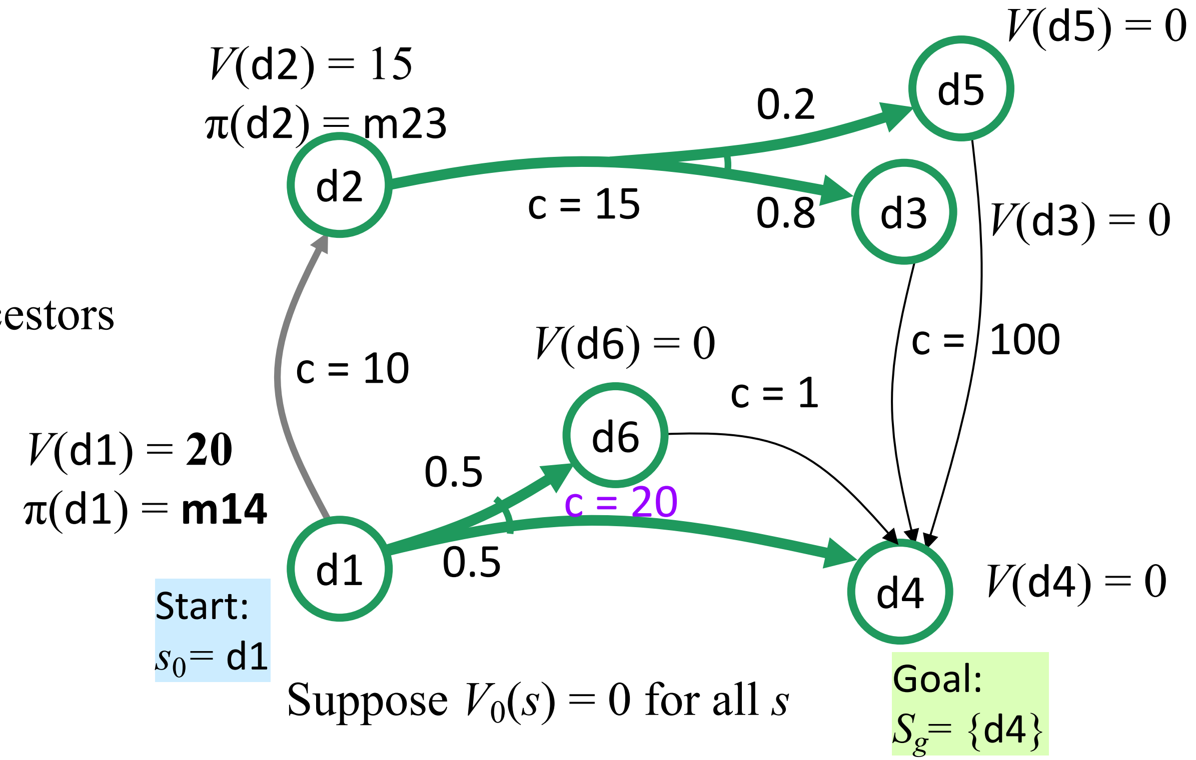
Suppose $V_0(s) = 0$ for all s

AO*

AO* (Σ, s_0, S_g, V_0)
- global $\pi \leftarrow \emptyset$
- global $Envelope \leftarrow \{s_0\}$
- global V; $V(s_0) \leftarrow V_0(s_0)$
- while $Fringe \neq \emptyset$ do
  - select a state $s \in Fringe$
  - for all $a \in$ Applicable(s) and $s' \in \gamma(s,a)$ do
    - if $s' \notin Envelope$ then
      - add s' to $Envelope$
      - $V(s') \leftarrow V_0(s')$
  - AO-Update(s)
- return π

AO-Update(s) // update V and π values of s and its ancestors
- $Z \leftarrow \{s\}$ // states that need updating
- while $Z \neq \emptyset$ do
  - select $s \in Z$ such that $\hat{\gamma}(s,\pi(s)) \cap Z = \{s\}$
  - remove s from Z
  - Bellman-Update(s)
  - $Z \leftarrow Z \cup \{s' \in Envelope \mid s \in \gamma(s',\pi)\}$

Bellman-Update(s)
- global V, π
- $v_{\text{old}} \leftarrow V(s)$
- for every $a \in$ Applicable(s) do
  - $Q(s,a) \leftarrow \text{cost}(s,a) + \sum_{s' \in S} \Pr(s'|s,a)\, V(s')$
- $V(s) \leftarrow \min_{a \in \text{Applicable}(s)} Q(s,a)$
- $\pi(s) \leftarrow \text{argmin}_{a \in \text{Applicable}(s)} Q(s,a)$
- return $|V(s) - v_{\text{old}}|$



AO*

AO* (Σ, s_0, S_g, V_0)

  global $\pi \leftarrow \emptyset$

  global $Envelope \leftarrow \{s_0\}$

  global V; $V(s_0) \leftarrow V_0(s_0)$

  while $Fringe \neq \emptyset$ do

    select a state $s \in Fringe$

    for all $a \in$ Applicable(s) and $s' \in \gamma(s,a)$ do

      if $s' \notin Envelope$ then

        add s' to $Envelope$

        $V(s') \leftarrow V_0(s')$

    AO-Update(s)

  return π

AO-Update(s) // update V and π values of s and its ancestors

  $Z \leftarrow \{s\}$ // states that need updating

  while $Z \neq \emptyset$ do

    select $s \in Z$ such that $\hat{\gamma}(s,\pi(s)) \cap Z = \{s\}$

    remove s from Z

    Bellman-Update(s)

    $Z \leftarrow Z \cup \{s' \in Envelope \mid s \in \gamma(s',\pi)\}$

Bellman-Update(s)

  global V, π

  $v_{\text{old}} \leftarrow V(s)$

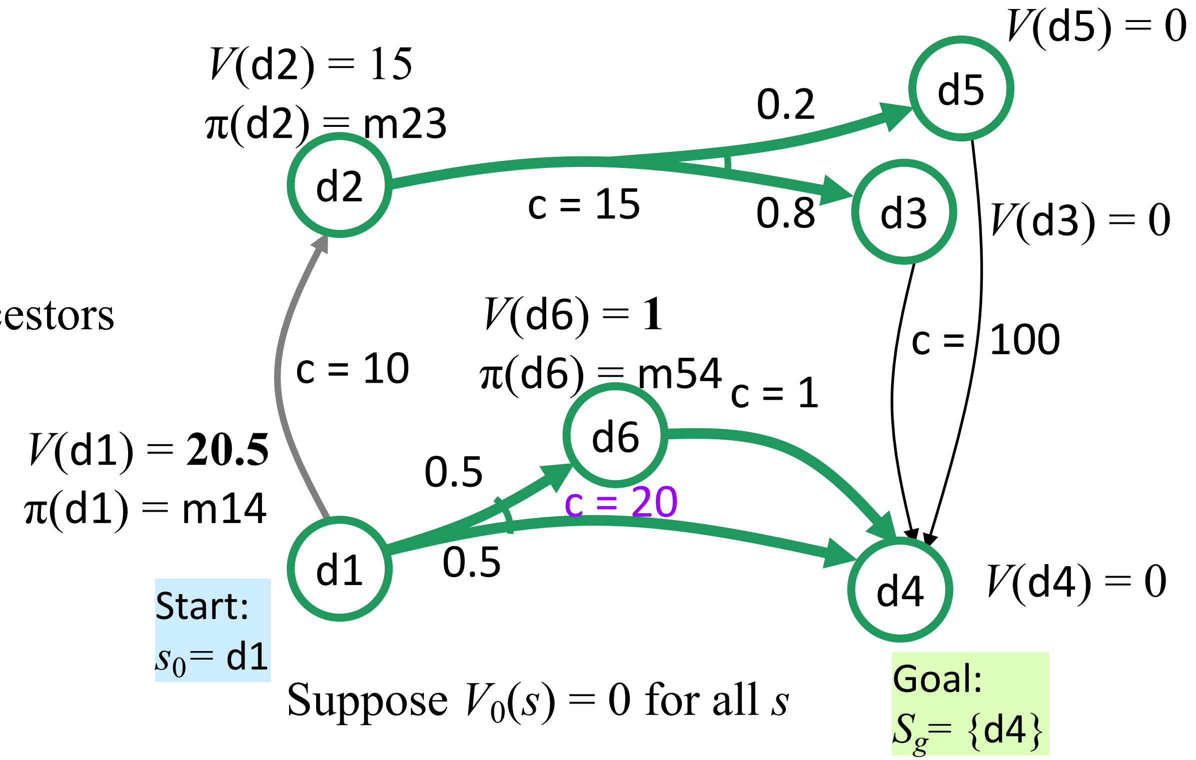
  for every $a \in$ Applicable(s) do

    $Q(s,a) \leftarrow \text{cost}(s,a) + \sum_{s' \in S} \Pr(s'|s,a)\, V(s')$

  $V(s) \leftarrow \min_{a \in \text{Applicable}(s)} Q(s,a)$

  $\pi(s) \leftarrow \text{argmin}_{a \in \text{Applicable}(s)} Q(s,a)$

  return $|V(s) - v_{\text{old}}|$

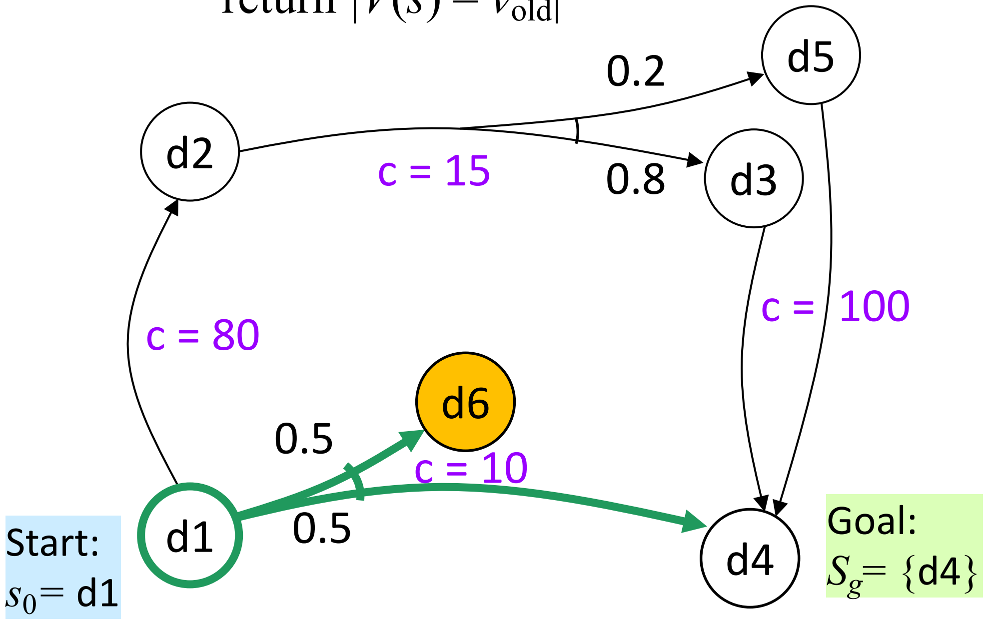


What to do about dead ends?

AO* (Σ, s_0, S_g, V_0)
  global $\pi \leftarrow \emptyset$
  global $Envelope \leftarrow \{s_0\}$
  global V; $V(s_0) \leftarrow V_0(s_0)$
  while $Fringe \neq \emptyset$ do
    select a state $s \in Fringe$
    for all $a \in$ Applicable(s) and $s' \in \gamma(s,a)$ do
      if $s' \notin Envelope$ then
        add s' to $Envelope$
        $V(s') \leftarrow V_0(s')$
    AO-Update(s)
  return π

AO-Update(s)   // update V and π values of s and its ancestors
  $Z \leftarrow \{s\}$ // states that need updating
  while $Z \neq \emptyset$ do
    select $s \in Z$ such that $\hat{\gamma}(s,\pi(s)) \cap Z = \{s\}$
    remove s from Z
    Bellman-Update(s)
    $Z \leftarrow Z \cup \{s' \in Envelope \mid s \in \gamma(s',\pi)\}$

Bellman-Update(s)
  global V, π
  $v_{\text{old}} \leftarrow V(s)$
  for every $a \in$ Applicable(s) do
    $Q(s,a) \leftarrow \text{cost}(s,a) + \sum_{s' \in S} \Pr(s'|s,a)\, V(s')$
  $V(s) \leftarrow \min_{a \in \text{Applicable}(s)} Q(s,a)$
  $\pi(s) \leftarrow \text{argmin}_{a \in \text{Applicable}(s)} Q(s,a)$
  return $|V(s) - v_{\text{old}}|$

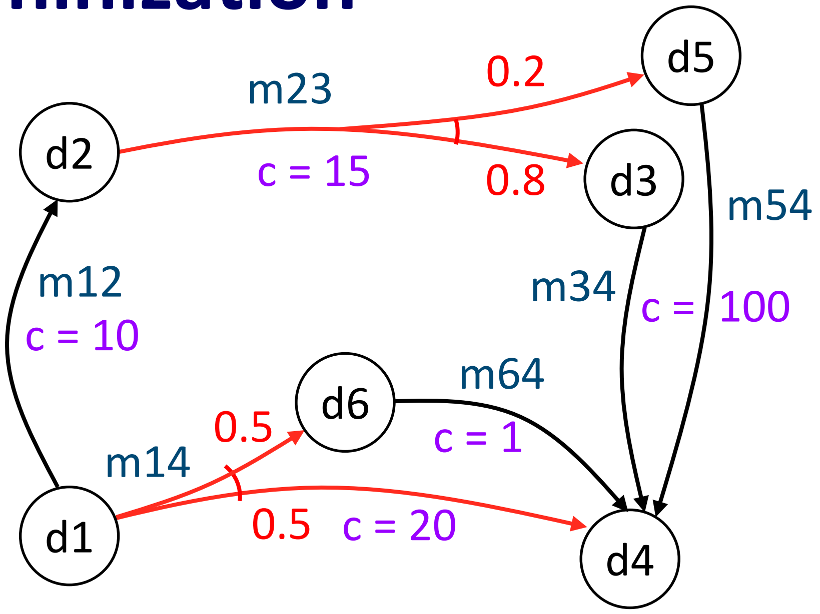


If action a in state s goes to a dead end, Guided-Find-Safe-Solution (Chap. 5) makes a inapplicable in s. AO* can be modified to do this.

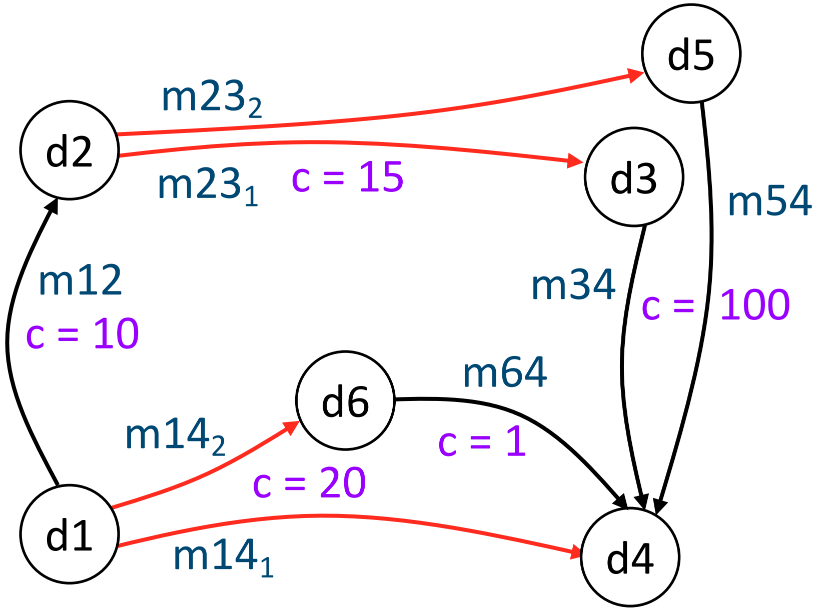
Heuristics through Determinization

What to use for $V_0(s)$?

- One possibility: use a heuristic function for the *determinized* domain Σ_d
- For each action a of Σ
  - If a has k possible outcomes, then
    - Σ_d contains k deterministic actions $a_1, \ldots, a_k$
    - one for each of a's outcome



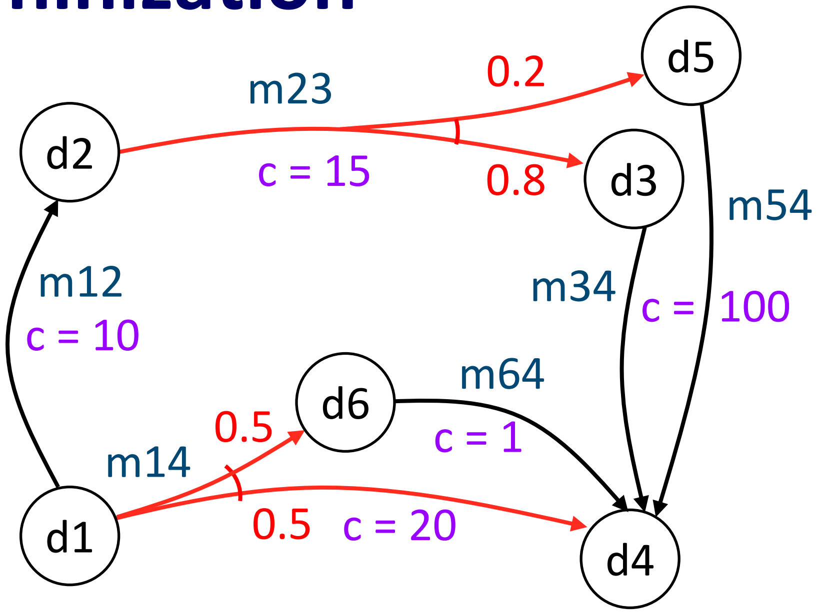
- Let h be a classical heuristic function for Σ_d
- If h is admissible for Σ_d then also admissible for Σ
- Why:
  - Let π be any optimal solution for Σ
  - Let p be any acyclic execution of π
    - p is a plan in Σ_d
    - $h(s_0) \leq \text{cost}(p) \leq \text{cost}(\pi)$

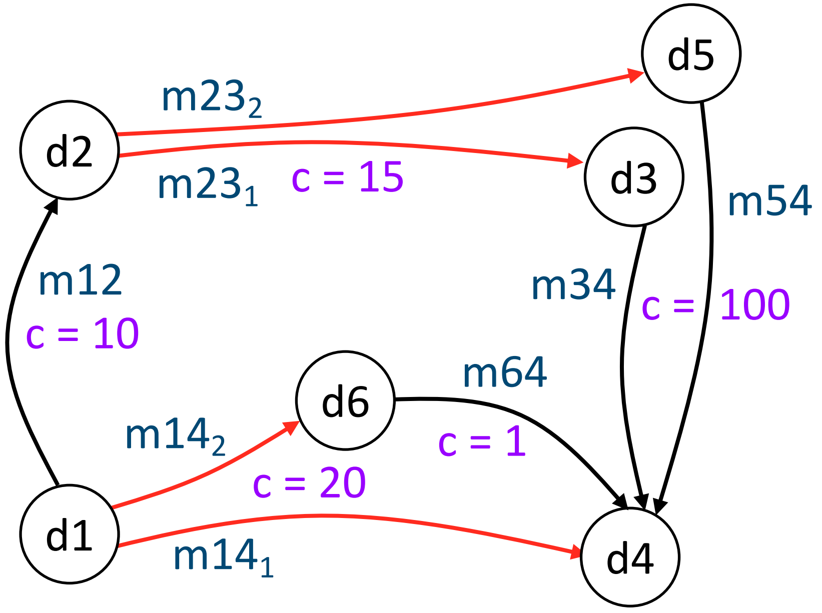


Heuristics through Determinization

What to use for $V_0(s)$?

- Another possibility: call a classical planner on the determinized problem (Σ_d, s, g)
    - Get plan $p = \langle a_1, a_2, \ldots, a_n \rangle$
    - Return $V_0(s) = \text{cost}(p)$
- **Theorem.** If the classical planner always returns optimal plans, then V_0 is admissible
- Outline of proof:
    - Let π^* be an optimal solution for (Σ, s, g)
    - Then $V^*(s) = E(\text{cost}(\pi^*))$
      $= \text{weighted avg. of all executions of } \pi$
    - Let p^* be the least costly execution of π
        - p^* is a solution for (Σ_d, s, g), so the classical planner returns a plan p of $\leq$ cost
        - Thus $V_0(s) = \text{cost}(p) \leq \text{cost}(p^*) \leq E(\text{cost}(\pi))$





LAO*

may be cyclic

LAO* (Σ, s_0, S_g, V_0)
  global $\pi \leftarrow \emptyset$
  global $Envelope \leftarrow \{s_0\}$
  global V; $V(s_0) \leftarrow V_0(s_0)$
  while $Fringe \neq \emptyset$ do
    select a state $s \in Fringe$
    for all $a \in$ Applicable(s) and $s' \in \gamma(s,a)$ do
      if $s' \notin Envelope$ then
        add s' to $Envelope$
        $V(s') \leftarrow V_0(s')$
    LAO-Update(s)
  return π

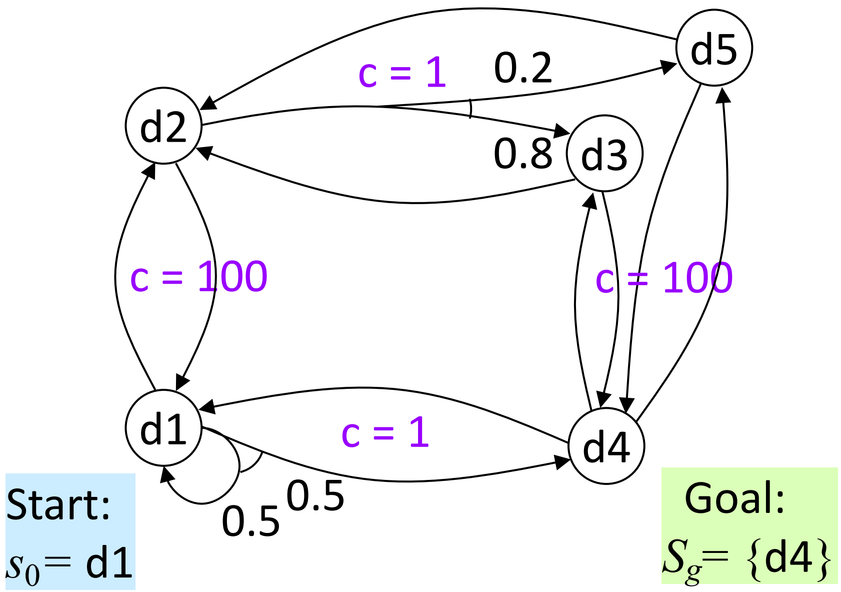
Same as AO* except here

All π-ancestors of s in $Envelope$

LAO-Update(s)
  $Z \leftarrow \{s\} \cup \{s' \in Envelope \mid s \in \hat{\gamma}(s',\pi)\}$
  loop until $r \leq \eta$ or new states added to $Fringe$
    $r \leftarrow \max\{$Bellman-Update(s) $\mid s \in Z\}$

Value iteration, restricted to Z

Bellman-Update(s)
  global V, π
  $v_{\text{old}} \leftarrow V(s)$
  for every $a \in$ Applicable(s) do
    $Q(s,a) \leftarrow \text{cost}(s,a) + \sum_{s' \in S} \Pr(s'|s,a)\, V(s')$
  $V(s) \leftarrow \min_{a \in \text{Applicable}(s)} Q(s,a)$
  $\pi(s) \leftarrow \text{argmin}_{a \in \text{Applicable}(s)} Q(s,a)$
  return $|V(s) - v_{\text{old}}|$



Example: $V_0(s) = 0$ for all s

Example 1

LAO* (Σ, s_0, S_g, V_0)
    global $\pi \leftarrow \emptyset$
    global $Envelope \leftarrow \{s_0\}$
    global V; $V(s_0) \leftarrow V_0(s_0)$
    while $Fringe \neq \emptyset$ do
        select a state $s \in Fringe$
        for all $a \in$ Applicable(s) and $s' \in \gamma(s,a)$ do
            if $s' \in \gamma(s,a) \notin Envelope$ then
                add s' to $Envelope$
                $V(s') \leftarrow V_0(s')$
        LAO-Update(s)
    return π

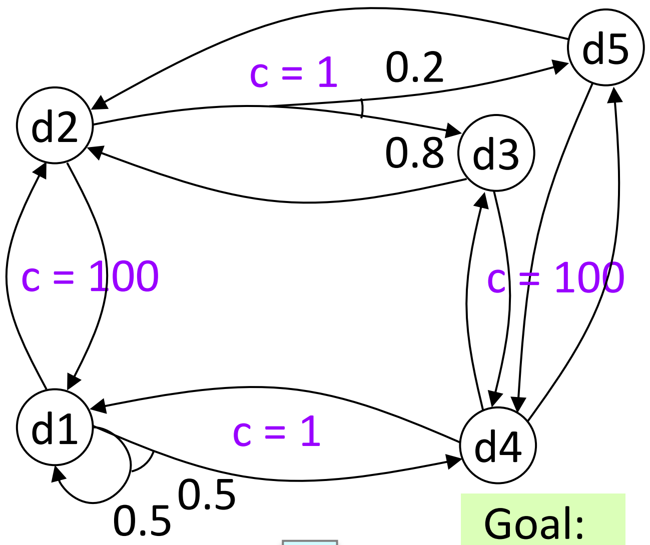
LAO-Update(s)
    $Z \leftarrow \{s\} \cup \{s' \in Envelope \mid s \in \hat{\gamma}(s',\pi)\}$
    loop until $r \leq \eta$ or new states added to $Fringe$
        $r \leftarrow \max\{$Bellman-Update(s) $\mid s \in Z\}$

Bellman-Update(s)
    global V, π; $v_{\text{old}} \leftarrow V(s)$
    for every $a \in$ Applicable(s) do
        $Q(s,a) \leftarrow \text{cost}(s,a) + \sum_{s' \in S} \Pr(s'|s,a)\, V(s')$
    $V(s) \leftarrow \min_{a \in \text{Applicable}(s)} Q(s,a)$
    $\pi(s) \leftarrow \text{argmin}_{a \in \text{Applicable}(s)} Q(s,a)$
    return $|V(s) - v_{\text{old}}|$

$\eta = 0.2$
$V_0(s) = 0$ for all s

$Envelope = \{$d1$\}$
$\pi = \emptyset$

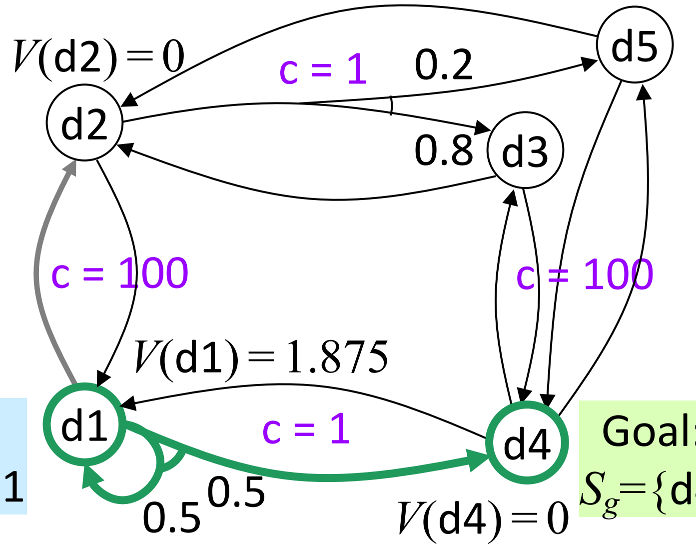
Start: $s_0 =$ d1



Goal: $S_g = \{$d4$\}$

Iteration 1

Start: $s_0 =$ d1



Goal: $S_g = \{$d4$\}$

LAO* *iteration 1:*
    select $s =$ d1
    Applicable(d1) = {m12, m14}
    add d2 to $Envelope$; V(d2) $\leftarrow 0$
    add d4 to $Envelope$; V(d4) $\leftarrow 0$
    Call LAO-Update(d1)
        π is empty, so $Z = \{$d1$\}$
        LAO-Update *iteration 1:*
            Call Bellman-update(d1):
                Q(d1,m12) = 100 + 0 = 100
                Q(d1,m14) = 1 + (½(0) + ½(0)) = 1
                V(d1) = 1; π(d1) = m14
            $r = V(\text{d1}) - 0 = 1$
        Keep iterating until $r \leq 0.2$
            V(d1) = 1.875; $r = 0.0625$
LAO* *iteration 2:*
    $Fringe = \emptyset$, so return π = {(d1,m14)}

Example 2

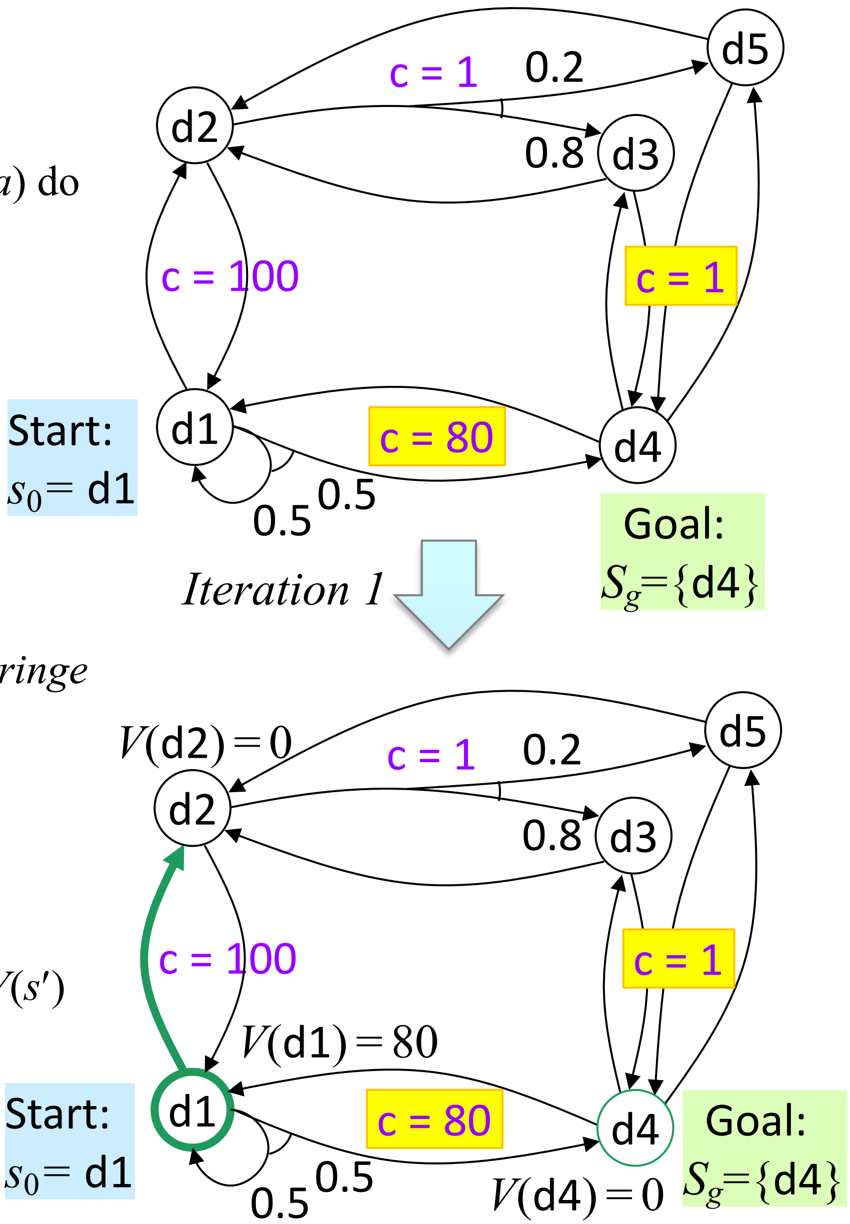
η=0.2; $V_0(s) = 0$ for all s

Envelope = {d1}; π = ∅

LAO* (Σ, s_0, S_g, V_0)
    global $\pi \leftarrow \emptyset$
    global *Envelope* $\leftarrow \{s_0\}$
    global V; $V(s_0) \leftarrow V_0(s_0)$
    while *Fringe* $\neq \emptyset$ do
        select a state $s \in$ *Fringe*
        for all $a \in$ Applicable(s) and $s' \in \gamma(s,a)$ do
            if $s' \in \gamma(s,a) \notin$ *Envelope* then
                add s' to *Envelope*
                $V(s') \leftarrow V_0(s')$
        LAO-Update(s)
    return π

LAO-Update(s)
    $Z \leftarrow \{s\} \cup \{s' \in Envelope \mid s \in \hat{\gamma}(s',\pi)\}$
    loop until $r \leq \eta$ or new states added to *Fringe*
        $r \leftarrow \max\{$Bellman-Update(s) $\mid s \in Z\}$

Bellman-Update(s)
    global V, π; $v_{\text{old}} \leftarrow V(s)$
    for every $a \in$ Applicable(s) do
        $Q(s,a) \leftarrow \text{cost}(s,a) + \sum_{s' \in S} \Pr(s'|s,a)\, V(s')$
    $V(s) \leftarrow \min_{a \in \text{Applicable}(s)} Q(s,a)$
    $\pi(s) \leftarrow \text{argmin}_{a \in \text{Applicable}(s)} Q(s,a)$
    return $|V(s) - v_{\text{old}}|$



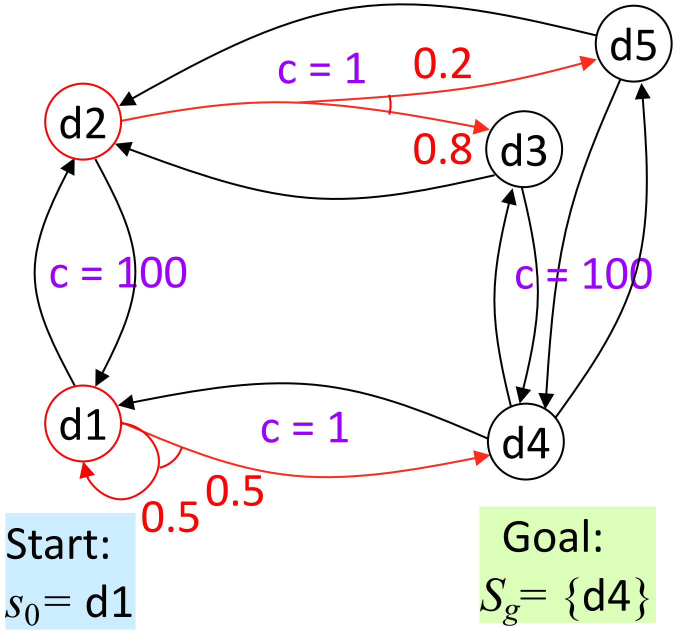
LAO* *iteration 1:*
    select s = d1; Applicable(d1) = {m12, m14}
    V(d2) ← 0; add d2 to *Envelope*
    V(d4) ← 0; add d4 to *Envelope*
    Call LAO-Update(d1)
        π is empty, so Z = {d1}
        LAO-Update *iteration 1:*
            Call Bellman-update(d1):
                Q(d1,m12) = 100 + 0 = 100
                Q(d1,m14) = 80 + (½(0) + ½(0)) = 80
                V(d1) = 80; π(d1) = m14
            r = V(d1) – 0 = 80
        LAO-Update *iteration 2:*
            Call Bellman-update(d1):
                Q(d1,m12) = 100 + 0 = 100
                Q(d1,m14) = 80 + (½(80) + ½(0)) = 120
                V(d1) = 100; π(d1) = m12
            r = V(d1) – 80 = 20
            new state d2 in *Fringe*, so return
        LAO-Update returns

After more iterations, LAO* *eventually returns*
    π = {(d1,m12),(d2,m23),(d3,m34),(d5,m54)}

Skipping Ahead

- Skipping ILAO*, HDP, LDFS_a, LRTDP, SLATE
  - I'll come back to these if there's time

Planning and Acting



Run-Lookahead(Σ, s_0, S_g)
  $s \leftarrow s_0$
  while $s \notin S_g$ and Applicable(s) $\neq \emptyset$ do
    $a \leftarrow$Lookahead(s,θ)
    perform action a
    $s \leftarrow$ observe resulting state

- Like Run-Lookahead from Chapter 2
- Differences:
  - Explicit starting state s_0
    - Not necessary, could observe s_0 instead
  - Doesn't abstract s (to simplify the presentation)
  - Lookahead returns an action instead of a plan
- What to use for Lookahead?
  - AO*, LAO*, …
    - Modify to search part of the space
  - Classical planner on determinized domain
    - next page
  - Stochastic sampling algorithms

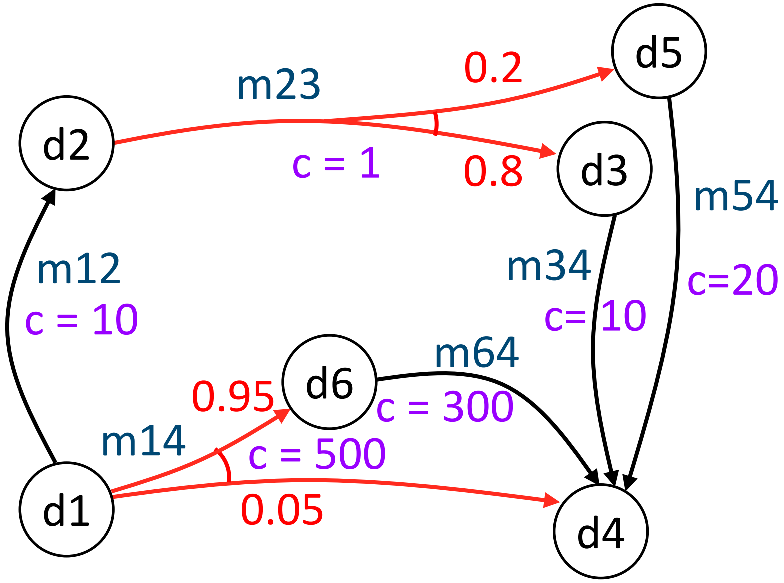
Planning and Acting

From Chapter 5

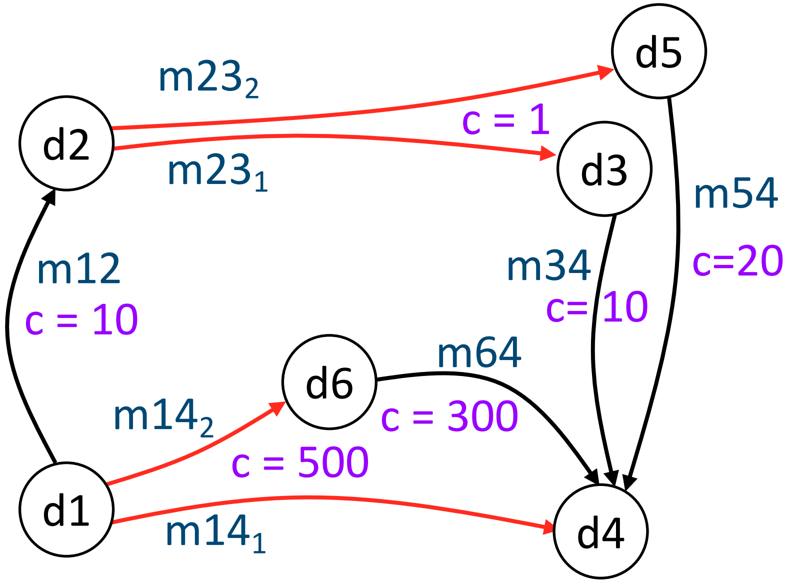
FS-Replan(Σ, s, S_g)
  $\pi_d \leftarrow \emptyset$
  while $s \notin S_g$ and Applicable(s) $\neq \emptyset$ do
    if $\pi_d(s)$ is undefined then do
      $\pi_d \leftarrow$ Plan2policy(Forward-search (Σ_d, s, S_g))
      if π_d = failure then return failure
    perform action $\pi_d(s)$
    $s \leftarrow$ observe resulting state

From Chapter 2

Determinized domain



- Generalization of a well-known planner called FF-Replan
- Like Run-Lazy-lookahead from Chapter 2
  - Lookahead = classical planner on determinized domain
- Example:
  - Forward-search returns ⟨m12, $m23_1$, m34⟩
  - Plan2policy returns π_d = ⟨(d1,m12), (d2,m23), (d3,m34)⟩
  - If m23 goes to d5, then call Forward-search again



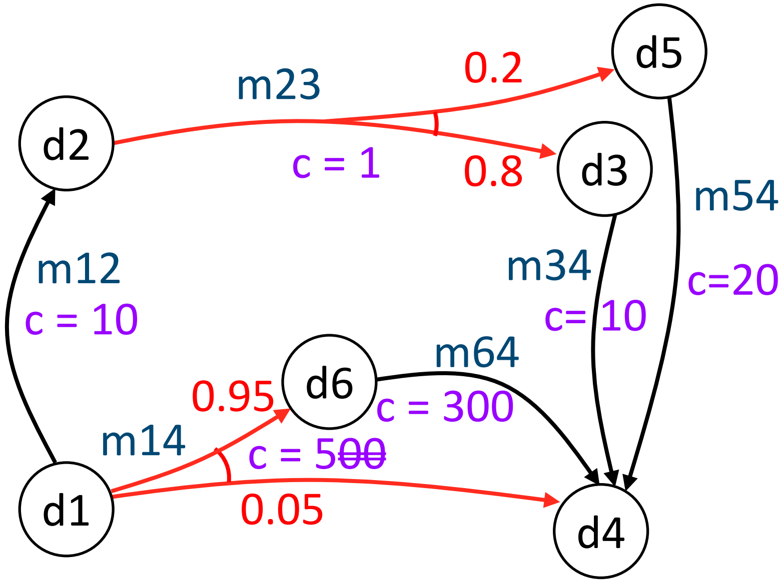
Planning and Acting

From Chapter 5

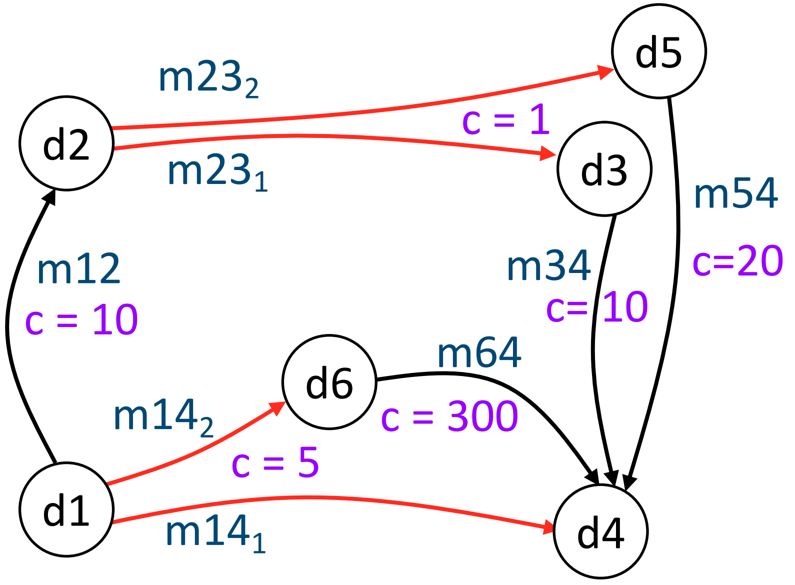
FS-Replan(Σ, s, S_g)
  $\pi_d \leftarrow \emptyset$
  while $s \notin S_g$ and Applicable(s) $\neq \emptyset$ do
    if $\pi_d(s)$ is undefined then do
      $\pi_d \leftarrow$ Plan2policy(Forward-search (Σ_d, s, S_g))
      if π_d = failure then return failure
    perform action $\pi_d(s)$
    $s \leftarrow$ observe resulting state

From Chapter 2

Determinized domain

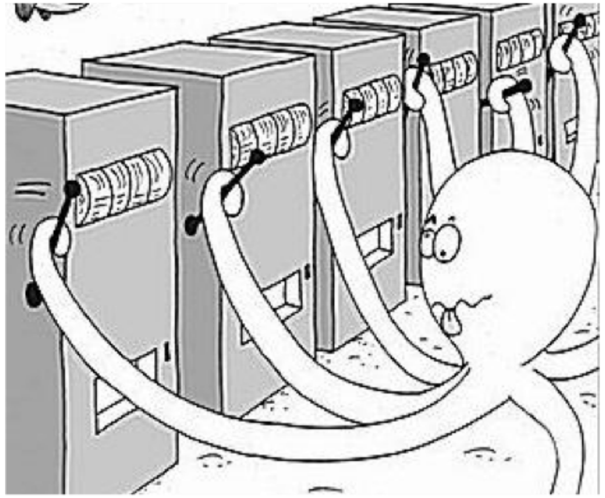


- Problem: classical planner may choose a plan that depends on low-probability outcome
- Example: Forward-search returns ⟨$m14_1$⟩
  - m14 very likely to go to d6 ⇒ large cost
- RFF algorithm (see book) attempts to alleviate this



Digression: Monte Carlo rollouts

- Multi-arm bandit problem
  - Statistical model of sequential experiments
  - Name derived from *one-armed bandit* (slot machine)
- Multiple actions $a_1, a_2, \ldots, a_n$
  - Each a_i provides a reward from an unknown probability distribution p_i
  - Assume each p_i is *stationary*
    - Same every time, regardless of history
  - Objective: maximize expected utility of a sequence of actions
- Exploitation vs exploration dilemma:
  - ***Exploitation***: choose an action that has given you high rewards in the past
  - ***Exploration***: choose an action that's less familiar, in hopes that it might produce a higher reward



UCB (Upper Confidence Bound) Algorithm

- Assume all rewards are between 0 and 1
  - If they aren't, normalize them
- For each action a, let
  - $r(a)$ = average reward you've gotten from a
  - $n(a)$ = number of times you've tried a
  - $n_t = \sum_a n(a)$
  - $Q(a) = r(a) + \sqrt{2(\ln n_t)/n(a)}$

UCB:

if there are any untried actions:

$\quad \tilde{a} \leftarrow$ any untried action

else:

$\quad \tilde{a} \leftarrow \text{argmax}_a\, Q(a)$

perform $\tilde{a}$

update $r(\tilde{a})$, $n(\tilde{a})$, n_t, $Q(\tilde{a})$

- Theorem (given some assumptions):

  As the number of calls to UCB $\to \infty$,

  UCB's choice at each call $\to$ optimal choice

Actions:	**a1**		**a2**		**a3**			
No. of tries:	n(a1) =	5	n(a2) =	3	n(a3) =	2	t =	10
Rewards:	r(a1) =	0.4		0.3333		0		
Q values:	Q(a1) =	1.35971		1.5723		1.5174		
Payoffs:								
1st		0						
2nd				0				
3rd						0		
4th		1						
5th		0						
6th				0				
7th						0		
8th		1						
9th		0						
10th				1				

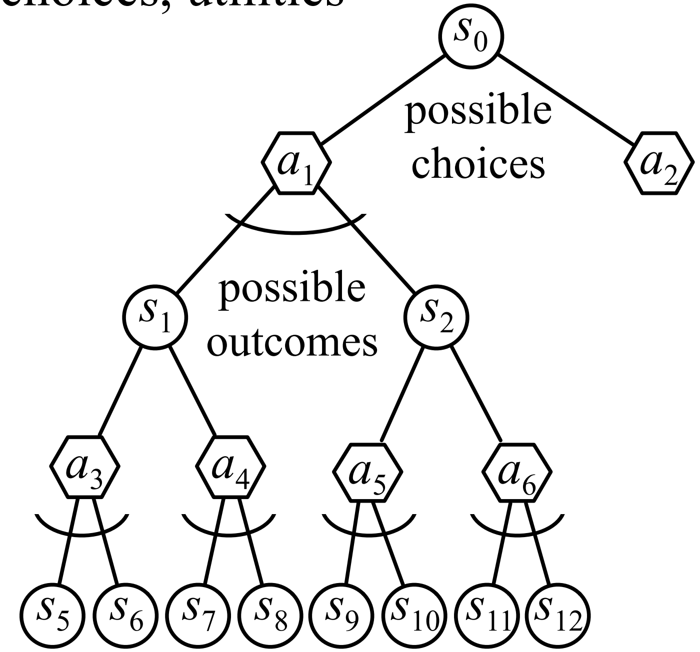
UCT Algorithm

- UCT(s,h): recursive UCB computation on an SSP
  - Adapted for minimization rather than maximization
- *Monte Carlo rollout*:
  - At s, choose action $\tilde{a}$ using UCB computation
    - Perform $\tilde{a}$, get state s'
    - Do the same thing recursively at s'
    - Continue until reaching a goal, dead end, or depth h
  - At each state visited, keep statistics on choices, utilities

max search depth

$\mathsf{UCT}(s,h)$
  if $s \in S_g$ then return 0
  if $h = 0$ then return $V_0(s)$
  if $s \notin \mathit{Envelope}$ then do
    add s to $\mathit{Envelope}$
    $n(s) \leftarrow 0$ like n_t on previous slide
    for all $a \in \text{Applicable}(s)$ do
      $Q(s,a) \leftarrow 0;\ \ n(s,a) \leftarrow 0$ like $Q(a)$, $n(a)$ on previous slide
  $\mathit{Untried} \leftarrow \{a \in \text{Applicable}(s) \mid n(s,a) = 0\}$
  if $\mathit{Untried} \neq \varnothing$ then $\tilde{a} \leftarrow \mathsf{Choose}(\mathit{Untried})$
  ln, I think
  else $\tilde{a} \leftarrow \text{argmin}_{a \in \text{Applicable}(s)} \{Q(s,a) - C \times [\log(n(s))/n(s,a)]^{\frac{1}{2}}\}$
  $s' \leftarrow \mathsf{Sample}(\Sigma, s, \tilde{a})$
  $\sqrt{2}$ on previous slide
  $\mathit{cost\text{-}rollout} \leftarrow \text{cost}(s,\tilde{a}) + \mathsf{UCT}(s', h-1)$
  $Q(s,\tilde{a}) \leftarrow [n(s,\tilde{a}) \times Q(s,\tilde{a}) + \mathit{cost\text{-}rollout}]/(1 + n(s,\tilde{a}))$
  $n(s) \leftarrow n(s) + 1$
  $n(s,\tilde{a}) \leftarrow n(s,\tilde{a}) + 1$
  return $\mathit{cost\text{-}rollout}$

choose s' with probability $\Pr(s' \mid s,\tilde{a})$

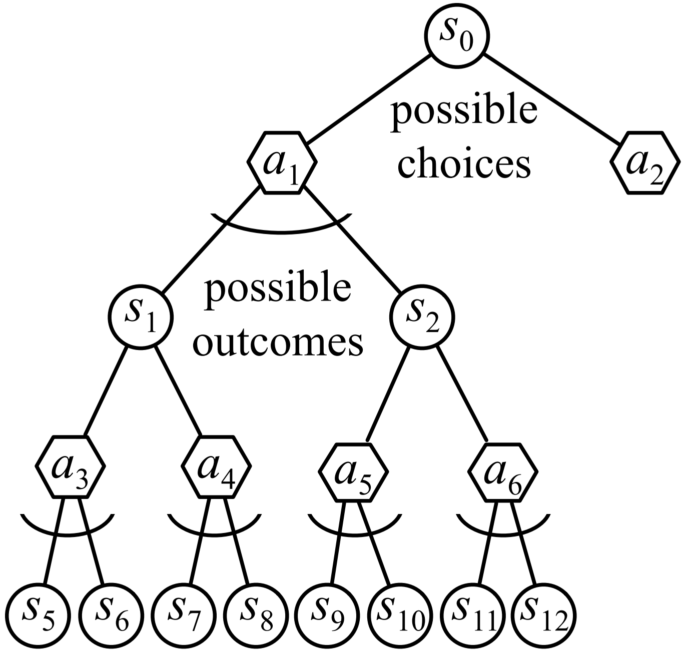


Using UCT Offline

for i – 1 to n
  call UCT(s_0,∞)
$\forall s,\ \pi(s) \leftarrow \operatorname{argmin}\{Q(s,a) \mid a \in \text{Applicable}(s)\}$

- As $n \to \infty$, π converges to optimal
  - Problem: finding optimal π may take many iterations

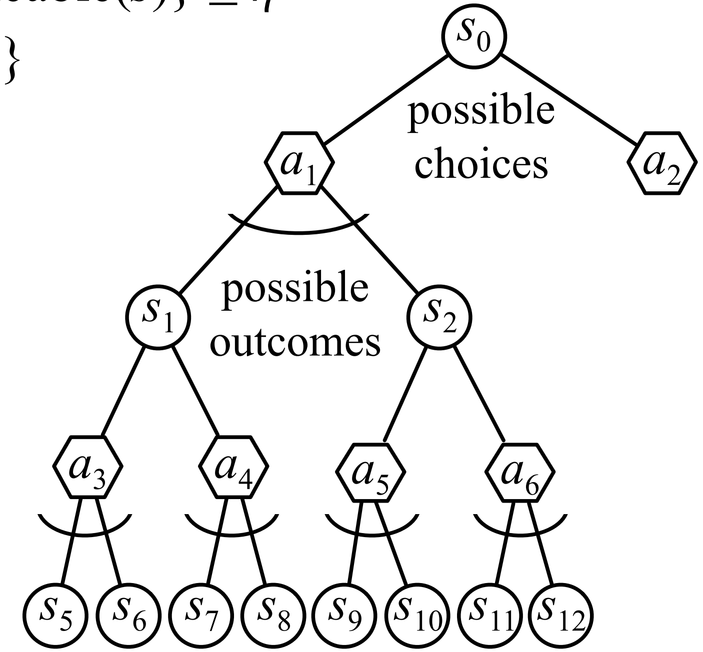
```
UCT(s, h)
  if s ∈ S_g then return 0
  if h = 0 then return V_0(s)
  if s ∉ Envelope then do
     add s to Envelope
     n(s) ← 0
     for all a ∈ Applicable(s) do
        Q(s, a) ← 0;  n(s, a) ← 0
  Untried ← {a ∈ Applicable(s) | n(s, a) = 0}
  if Untried ≠ ∅ then ã ← Choose(Untried)
  else ã ← argmin_{a∈Applicable(s)} {Q(s, a) − C × [log(n(s))/n(s, a)]^(1/2)}
  s′ ← Sample(Σ, s, ã)
  cost-rollout ← cost(s, ã) + UCT(s′, h − 1)
  Q(s, ã) ← [n(s, ã) × Q(s, ã) + cost-rollout]/(1 + n(s, ã))
  n(s) ← n(s) + 1
  n(s, ã) ← n(s, ã) + 1
  return cost-rollout
```



Using UCT Online

- Works better for online planning
  - $\pi(s_0)$ approaches optimal must faster than the rest of π
- Lookahead procedure for Run-Lookahead:
  - call UCT(s,h) multiple times at current state s
  - e.g., until
    - allotted time runs out, or
    - $\max\{\text{last change in } Q(s,a) \mid a \in \text{Applicable}(s)\} \leq \eta$
  - return $\text{argmax}\{Q(s,a) \mid a \in \text{Applicable}(s)\}$

$$
\begin{array}{l}
\mathsf{UCT}(s,h) \\
\quad \text{if } s \in S_g \text{ then return } 0 \\
\quad \text{if } h = 0 \text{ then return } V_0(s) \\
\quad \text{if } s \notin \mathit{Envelope} \text{ then do} \\
\quad\quad \text{add } s \text{ to } \mathit{Envelope} \\
\quad\quad n(s) \leftarrow 0 \\
\quad\quad \text{for all } a \in \text{Applicable}(s) \text{ do} \\
\quad\quad\quad Q(s,a) \leftarrow 0;\ \ n(s,a) \leftarrow 0 \\
\quad \mathit{Untried} \leftarrow \{a \in \text{Applicable}(s) \mid n(s,a) = 0\} \\
\quad \text{if } \mathit{Untried} \neq \varnothing \text{ then } \tilde{a} \leftarrow \mathsf{Choose}(\mathit{Untried}) \\
\quad \text{else } \tilde{a} \leftarrow \text{argmin}_{a \in \text{Applicable}(s)}\{Q(s,a) - C \times [\log(n(s))/n(s,a)]^{\frac{1}{2}}\} \\
\quad s' \leftarrow \mathsf{Sample}(\Sigma, s, \tilde{a}) \\
\quad \mathit{cost\text{-}rollout} \leftarrow \text{cost}(s,\tilde{a}) + \mathsf{UCT}(s', h-1) \\
\quad Q(s,\tilde{a}) \leftarrow [n(s,\tilde{a}) \times Q(s,\tilde{a}) + \mathit{cost\text{-}rollout}]/(1 + n(s,\tilde{a})) \\
\quad n(s) \leftarrow n(s) + 1 \\
\quad n(s,\tilde{a}) \leftarrow n(s,\tilde{a}) + 1 \\
\quad \text{return } \mathit{cost\text{-}rollout}
\end{array}
$$

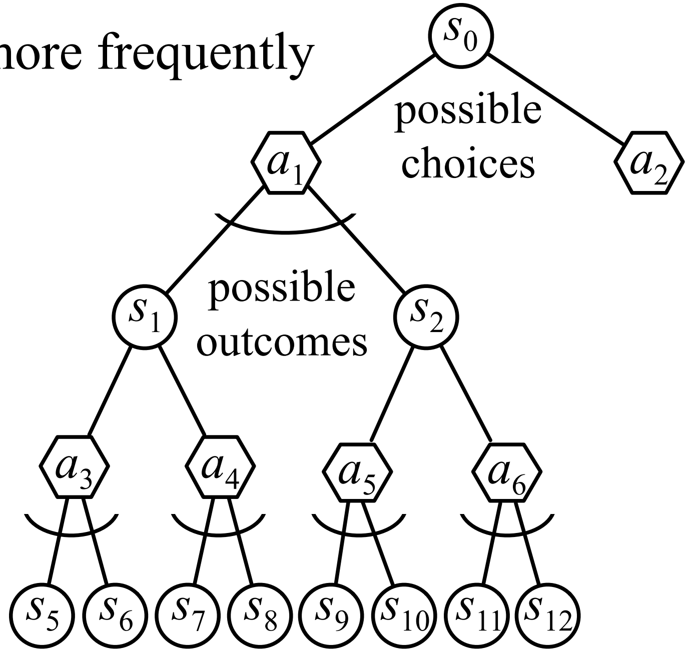


Using UCT Online

- Lookahead procedure for Run-Lazy-Lookahead:
  - loop:
    - call UCT many times at current state
    - At state s visited, $\pi(s) \leftarrow$ action with highest expected utility

- Problem: the farther you follow π, the less likely that $\pi(s)$ is optimal
  - Near the bottom of the tree, $\pi(s)$ might be $\approx$ random choice
- Possible workaround
  - Modify Run-Lazy-Lookahead to call UCT more frequently

UCT(s, h)
    if $s \in S_g$ then return 0
    if $h = 0$ then return $V_0(s)$
    if $s \notin \mathit{Envelope}$ then do
        add s to $\mathit{Envelope}$
        $n(s) \leftarrow 0$
        for all $a \in \text{Applicable}(s)$ do
            $Q(s,a) \leftarrow 0;\ \ n(s,a) \leftarrow 0$
    $\mathit{Untried} \leftarrow \{a \in \text{Applicable}(s) \mid n(s,a) = 0\}$
    if $\mathit{Untried} \neq \varnothing$ then $\tilde{a} \leftarrow \text{Choose}(\mathit{Untried})$
    else $\tilde{a} \leftarrow \text{argmin}_{a \in \text{Applicable}(s)} \{Q(s,a) - C \times [\log(n(s))/n(s,a)]^{\frac{1}{2}}\}$
    $s' \leftarrow \text{Sample}(\Sigma, s, \tilde{a})$
    $\mathit{cost\text{-}rollout} \leftarrow \text{cost}(s, \tilde{a}) + \text{UCT}(s', h-1)$
    $Q(s,\tilde{a}) \leftarrow [n(s,\tilde{a}) \times Q(s,\tilde{a}) + \mathit{cost\text{-}rollout}]/(1 + n(s,\tilde{a}))$
    $n(s) \leftarrow n(s) + 1$
    $n(s,\tilde{a}) \leftarrow n(s,\tilde{a}) + 1$
    return $\mathit{cost\text{-}rollout}$

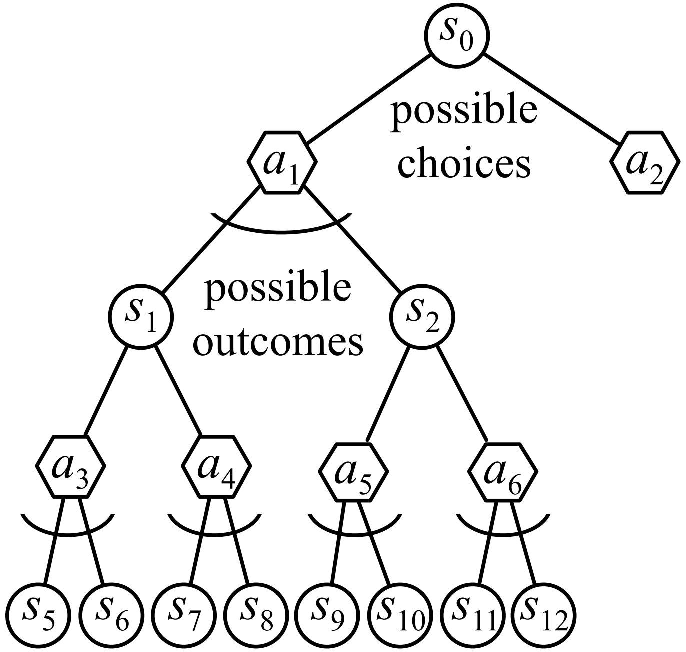


Using UCT with a Simulator

- Suppose you don't know the probabilities and costs
  - But you have a fast, accurate simulator for the environment
- Lookahead procedure (see previous slides)
  - Run UCT multiple times in the simulated environment
  - Learn state-transition probabilities, expected utilities

$\mathsf{UCT}(s, h)$
  if $s \in S_g$ then return 0
  if $h = 0$ then return $V_0(s)$
  if $s \notin \mathit{Envelope}$ then do
    add s to $\mathit{Envelope}$
    $n(s) \leftarrow 0$
    for all $a \in \text{Applicable}(s)$ do
      $Q(s,a) \leftarrow 0;\ \ n(s,a) \leftarrow 0$
  $\mathit{Untried} \leftarrow \{a \in \text{Applicable}(s) \mid n(s,a) = 0\}$
  if $\mathit{Untried} \neq \varnothing$ then $\tilde{a} \leftarrow \mathsf{Choose}(\mathit{Untried})$
  else $\tilde{a} \leftarrow \text{argmin}_{a \in \text{Applicable}(s)} \{Q(s,a) - C \times [\log(n(s))/n(s,a)]^{\frac{1}{2}}\}$
  $s' \leftarrow \boxed{\mathsf{Sample}(\Sigma, s, \tilde{a})}$
  $\mathit{cost\text{-}rollout} \leftarrow \text{cost}(s, \tilde{a}) + \mathsf{UCT}(s', h-1)$
  $Q(s,\tilde{a}) \leftarrow [n(s,\tilde{a}) \times Q(s,\tilde{a}) + \mathit{cost\text{-}rollout}]/(1 + n(s,\tilde{a}))$
  $n(s) \leftarrow n(s) + 1$
  $n(s,\tilde{a}) \leftarrow n(s,\tilde{a}) + 1$
  return $\mathit{cost\text{-}rollout}$

simulate a; observe s'

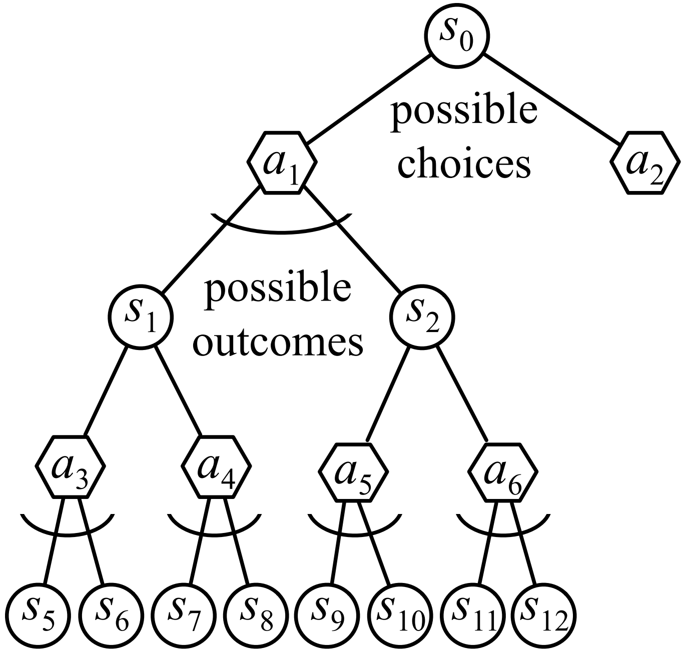


Using UCT for Exploration

- Suppose you don't know the probabilities and costs
- Suppose you can restart your actor as many times as you want
  - Caveat: usually not very feasible in real environments
- Can modify UCT to be an acting procedure,
  use it to explore the environment

$\mathsf{UCT}(s,h)$
    if $s \in S_g$ then return 0
    if $h = 0$ then return $V_0(s)$
    if $s \notin \mathit{Envelope}$ then do
        add s to $\mathit{Envelope}$
        $n(s) \leftarrow 0$
        for all $a \in \text{Applicable}(s)$ do
            $Q(s,a) \leftarrow 0;\ \ n(s,a) \leftarrow 0$
    $\mathit{Untried} \leftarrow \{a \in \text{Applicable}(s) \mid n(s,a) = 0\}$
    if $\mathit{Untried} \neq \varnothing$ then $\tilde{a} \leftarrow \mathsf{Choose}(\mathit{Untried})$
    else $\tilde{a} \leftarrow \text{argmin}_{a \in \text{Applicable}(s)}\{Q(s,a) - C \times [\log(n(s))/n(s,a)]^{\frac{1}{2}}\}$
    $s' \leftarrow \boxed{\mathsf{Sample}(\Sigma, s, \tilde{a})}$
    $\mathit{cost\text{-}rollout} \leftarrow \text{cost}(s,\tilde{a}) + \mathsf{UCT}(s', h-1)$
    $Q(s,\tilde{a}) \leftarrow [n(s,\tilde{a}) \times Q(s,\tilde{a}) + \mathit{cost\text{-}rollout}]/(1 + n(s,\tilde{a}))$
    $n(s) \leftarrow n(s) + 1$
    $n(s,\tilde{a}) \leftarrow n(s,\tilde{a}) + 1$
    return $\mathit{cost\text{-}rollout}$

perform a; observe s'



UCT in Two-Player Games

- Generate Monte Carlo rollouts using a modified version of UCT
- Main differences:
  - Instead of choosing actions that minimize accumulated cost, choose actions that maximize payoff at the end of the game
  - UCT for player 1 recursively calls UCT for player 2
    - Choose opponent's action
  - UCT for player 2 recursively calls UCT for player 1

- First competent computer programs for go
  - ≈ 2008–2012
- Monte Carlo rollout techniques similar to UCT were used to train AlphaGo

Summary

- SSPs
- solutions, closed solutions, histories
- unsafe solutions, acyclic safe solutions, cyclic safe solutions
- expected cost, planning as optimization
- policy iteration
- value iteration (asynchronous version)
  - Bellman-update
- AO*, LAO*
- Planning and Acting
  - Run-Lookahead
  - FS-Replan
- UCB, UCT