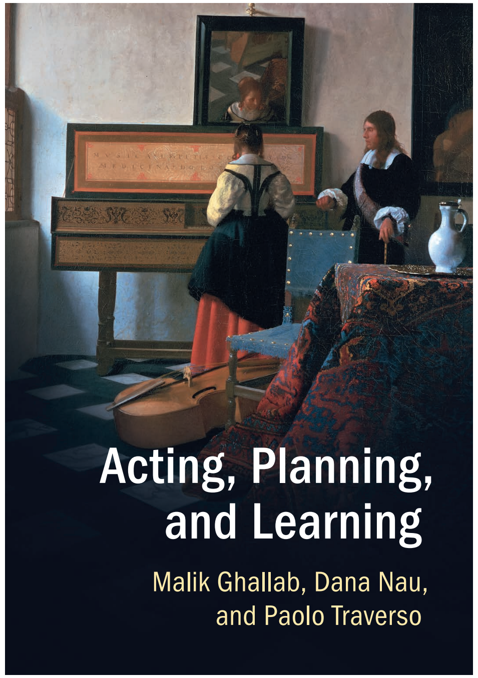
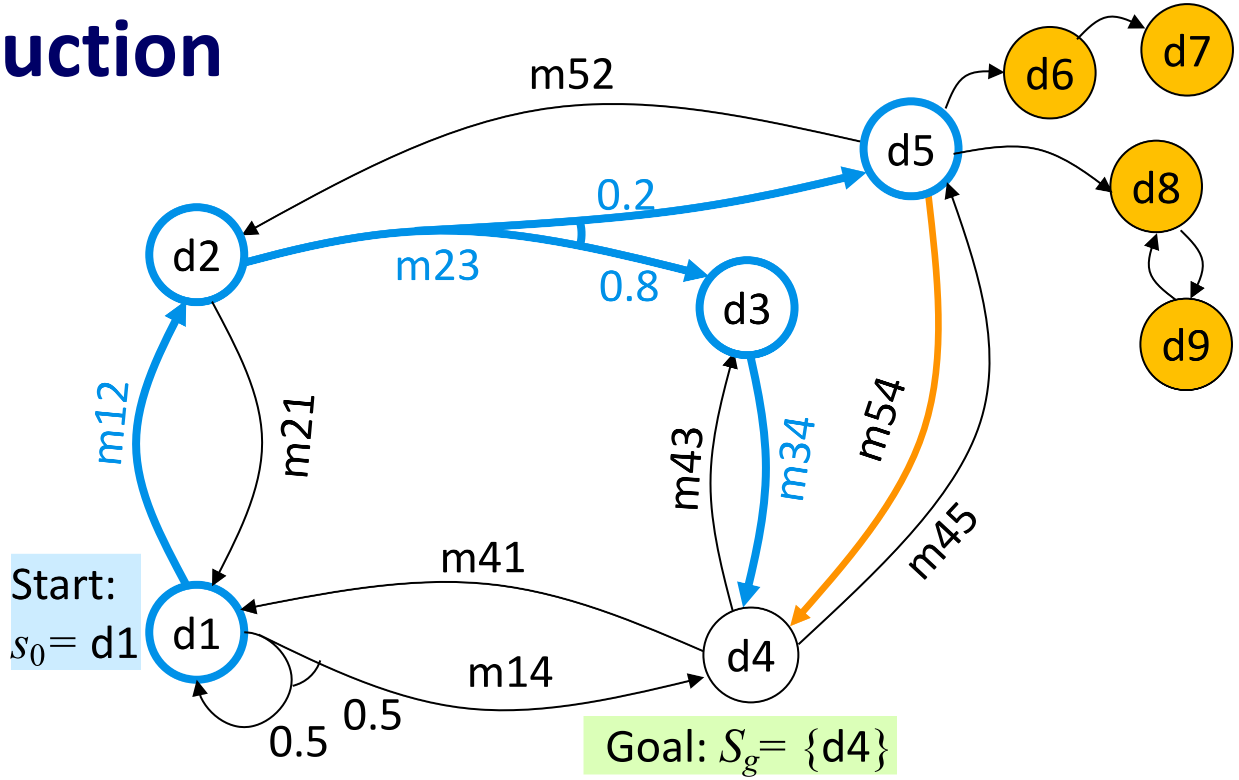
Last update: 11:09 PM, March 7, 2025



Chapter 9
Planning with Probabilistic Models

Dana S. Nau
University of Maryland

Introduction



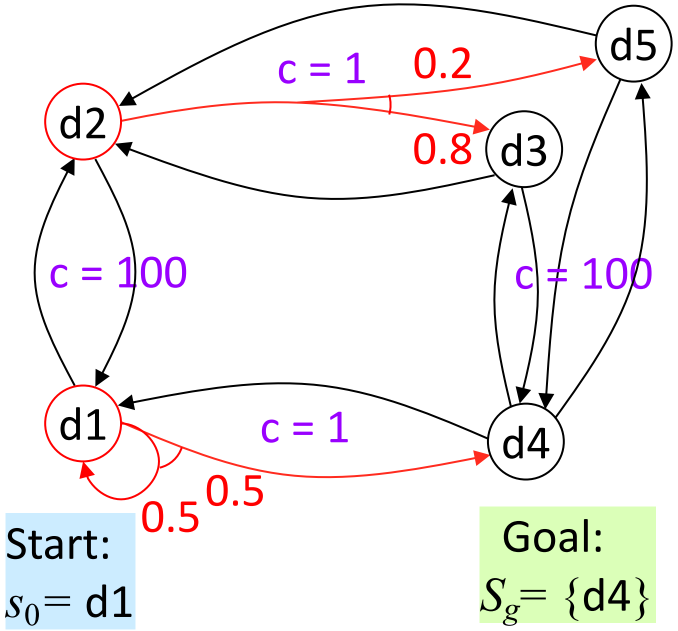
- *Stochastic Shortest Path (SSP)* problem: an MDP problem $P = (\Sigma, s_0, S_g)$ such that
  - there is at least one safe solution
  - cost(s,a,s') is always > 0
- Let $V^*(s)$ = expected cost of an optimal safe solution
- *Optimality principle* (Bellman's theorem)

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

- We'll only consider *safe* SSP's
  - no dead-end states

Safe solution: $\Pr(S_g \mid s_0, \pi) = 1$

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*(Σ, s, S_g)
        if a = failure **then return** failure
        perform action a
        $s \leftarrow$ observe resulting state

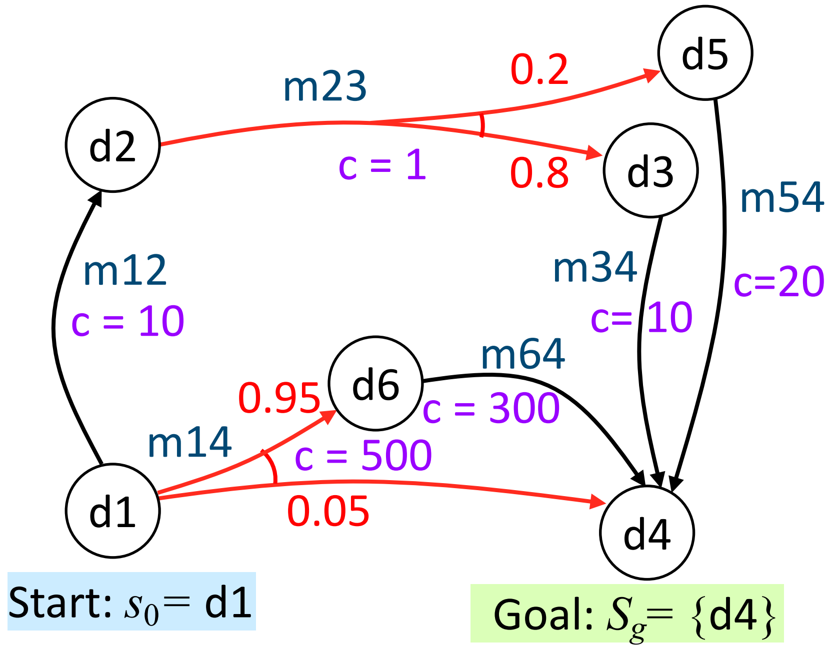
- Like Run-Lookahead from Chapter 2
- Differences:
  - Explicit starting state s_0
    - Not necessary, could observe s_0 instead
  - Doesn't abstract s (to simplify the presentation)
  - *Lookahead* returns an action instead of a plan
- As in Chapter 2, *Lookahead* can be modified to cut off its search before reaching S_g

- What to use for Lookahead?
  - Classical planner on determinized domain
    - next page
  - AO*, LAO*, …
    - Modify to search part of the space
  - Stochastic sampling algorithms

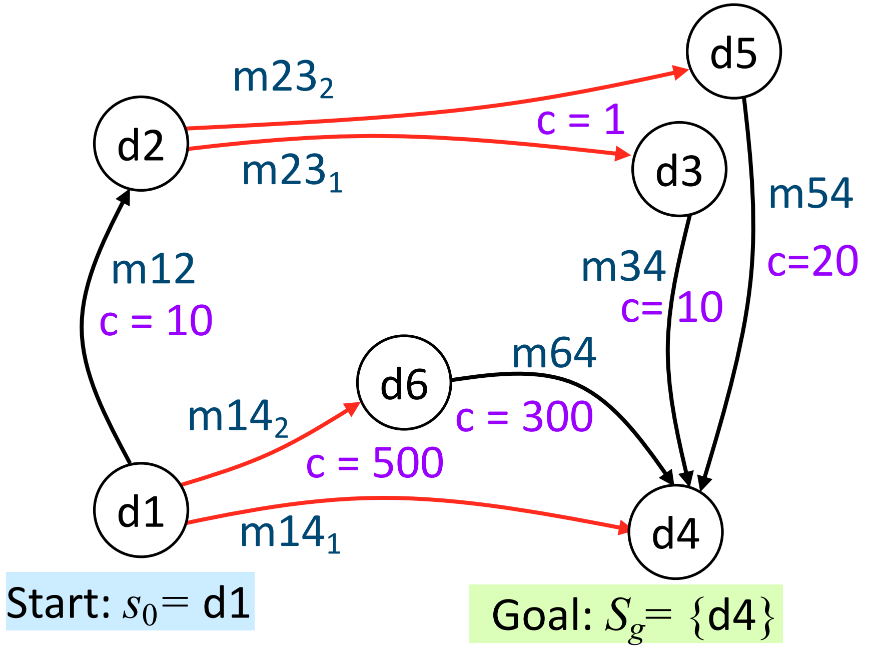
Planning and Acting

Algorithm 5.15 in
Automated Planning and Acting
(see supplemental materials)

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



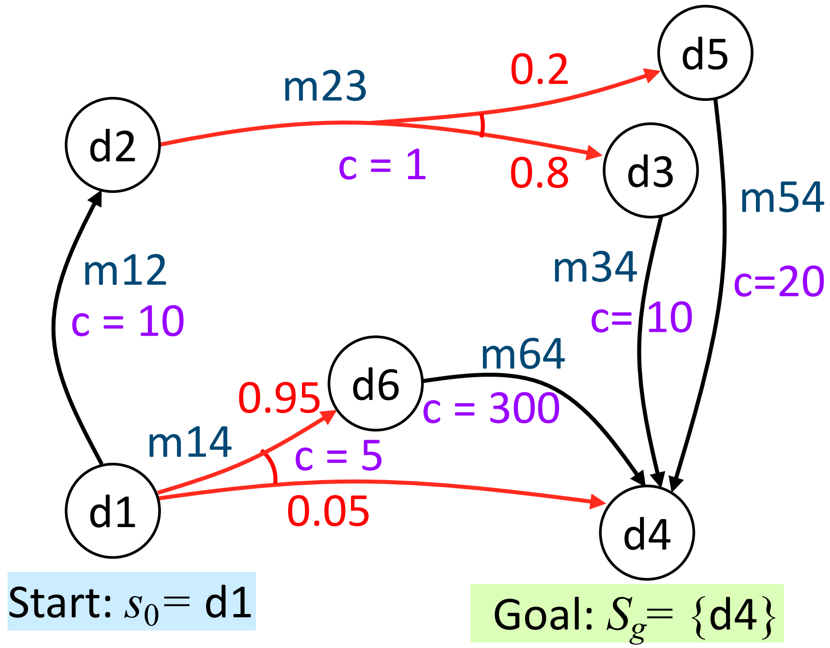
- Generalization of a well-known planner called FF-Replan
- Like Run-Lazy-lookahead from Chapter 2
  - *Lookahead* = classical planner on determinized domain
- Example:
  - Forward-Search returns ⟨m12, $m23_1$, m34⟩
  - Plan2policy returns π = ⟨(d1,m12), (d2,m23), (d3,m34)⟩
  - If m23 goes to d5, then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



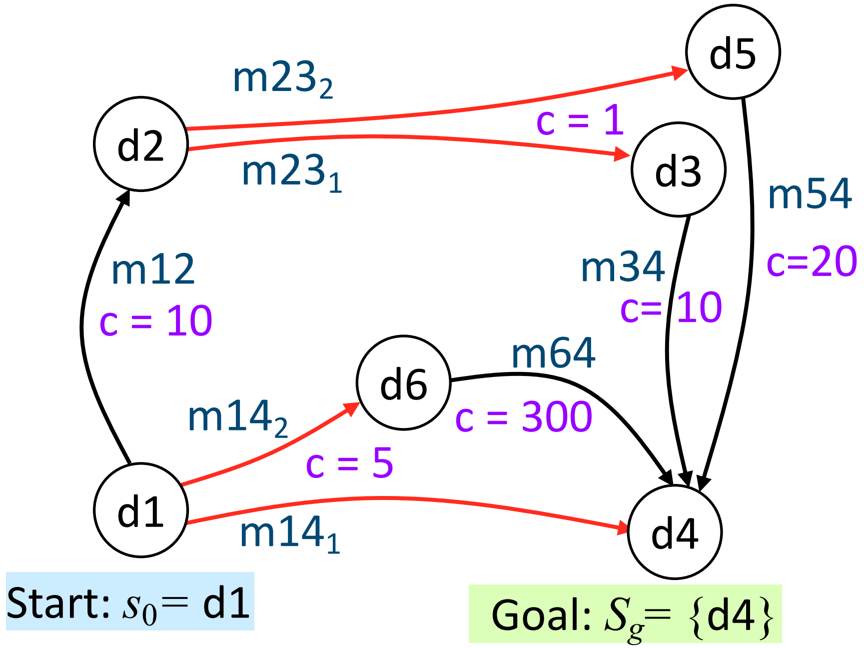
Planning and Acting

Algorithm 5.15 in
Automated Planning and Acting
(see supplemental materials)

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



- Problem: classical planner may choose a plan that depends on low-probability outcome
- Example: Forward-search returns ⟨$m14_1$⟩, cost 5
    - Plan2policy returns π = ⟨(d1,m14)⟩
    - m14 very likely to go to d6 ⇒ cost 305
- RFF algorithm (see book) attempts to alleviate this

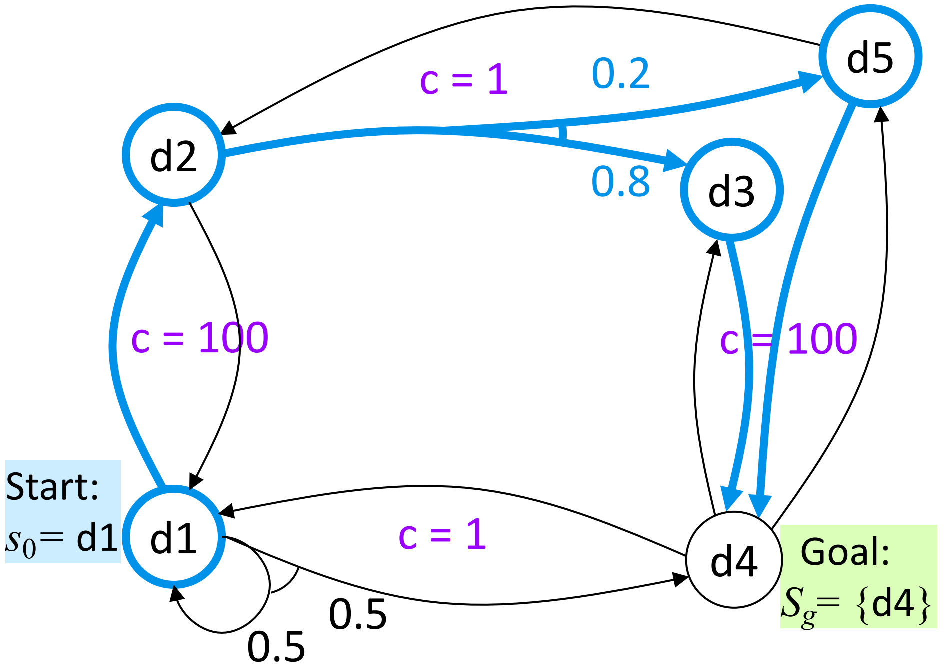


Cost to Go

- Let π be a safe solution that's defined at all non-goal states
  - $\Pr(S_g \mid s_0, \pi) = 1$
  - $Domain(\pi) = S \setminus S_g$
- Compute V^π as $|S|$ equations, $|S|$ unknowns

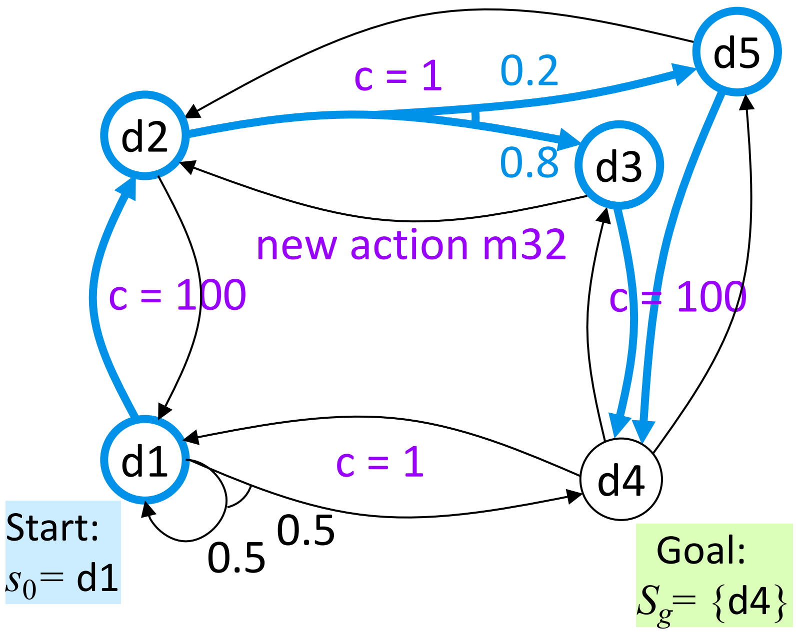
- *Cost-to-go*:
  - expected cost at s if we first use a, then use π afterward
  - $Q^\pi(s,a) = \sum_{s' \in \gamma(s,a')} \Pr(s' \mid s,a') \, [\mathrm{cost}(s,a,s') + V^\pi(s')]$

- For every $s \in S \setminus S_g$
  - $\{a' \mid a' \in \mathrm{Applicable}(s)\}$ includes $\pi(s)$
  - so $\{Q^\pi(s,a') \mid a' \in \mathrm{Applicable}(s)\}$ includes $V^\pi(s)$
  - so $\min\{Q^\pi(s,a') \mid a' \in \mathrm{Applicable}(s)\} \leq V^\pi(s)$
  - so 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) = 0.8(1+V^π(d3)) + 0.2(1+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

min = 101.5, argmin = m14

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

Q^π(d2,m21) = 100 + 201 = 301

min = 101, argmin = m23

Q^π(d3,m34) = 100 + 0 = 100

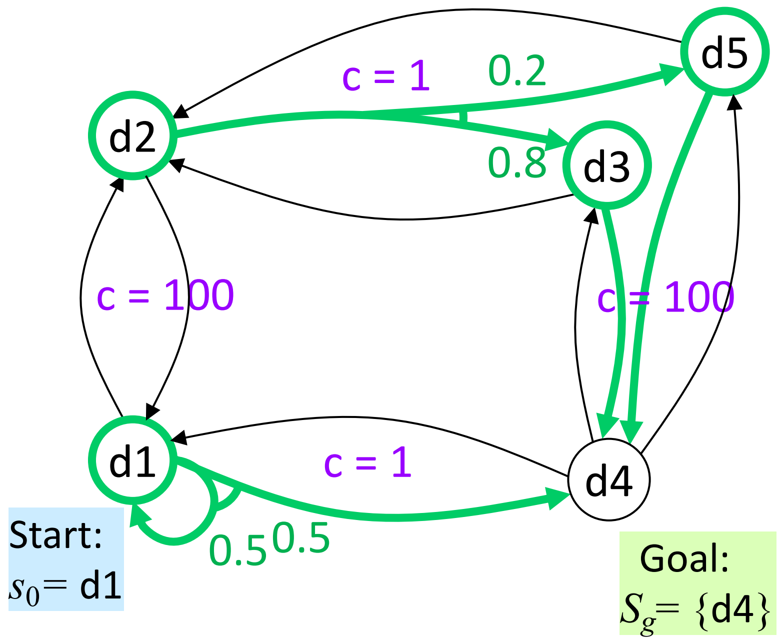
Q^π(d3,m32) = 1 + 101 = 102

min = 100, argmin = m34

Q^π(d5,m54) = 100 + 0 = 100

Q^π(d5,m52) = 1+101 = 102

min = 100, argmin = 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

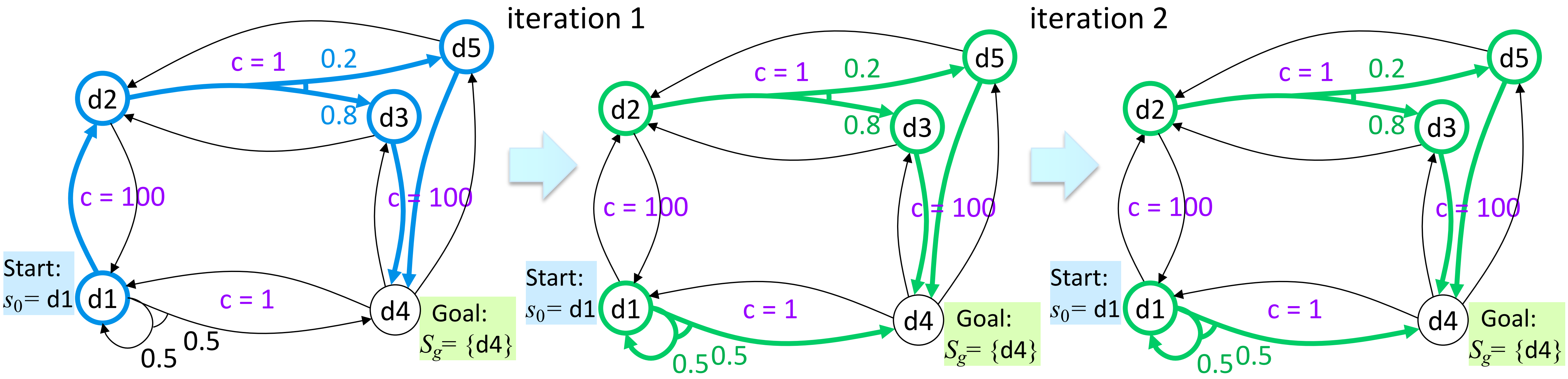
$V^{\pi'}$(d1) = 1 + ½$V^{\pi'}$(d1) + ½$V^{\pi'}$(d4)
$\Rightarrow$ $V^{\pi'}$(d1) = 2

Policy Iteration

PI(Σ, π)
 until π stops changing **do**
  compute $\{V^\pi(s) \mid s \in S\}$ ← $|S|$ equations, $|S|$ unknowns
  for each $s \in S \setminus S_g$ **do**
   $\pi(s) \leftarrow \text{argmin}_{a \in \text{Applicable}(s)}\, Q^\pi(s,a)$

$\mathbb{E}$[cost of using a then π]

- Converges in a finite number of iterations



Value Iteration

VI(Σ, S_g, V_0)
  $V \leftarrow V_0$
  until reaching an approximate fixed point **do**
    for each $s \in S \setminus S_g$ **do**
      Bellman-Update(s)

residual $\leq \eta$

Bellman-Update(s)
  for every $a \in$ Applicable(s) **do**
    $Q(s,a) \leftarrow \sum_{s' \in \gamma(s,a)} \Pr(s'|s,a)\,[\text{cost}(s,a,s') + V(s')]$
  $V(s) \leftarrow \min_a Q(s,a)$
  $\pi(s) \leftarrow \text{argmin}_a Q(s,a)$

- V_0: a heuristic function
  - e.g., adapt an h heuristic from Chapter 2
  - $V_0(s)$ = estimated cost of getting from s to S_g
  - Require $V_0(s) = 0$ for every $s \in S_g$

- V and π are global arrays
- Initially $V(s) = V_0(s)\ \forall s$
- In each iteration, $V \leftarrow$ improved estimate
- *approximate fixed point*: V stops changing very much
  - usually when the *residual* (max change to V) is $\leq \eta$
  - The book gives several other possibilities

- V doesn't depend on π
  - Could wait and compute π at the very end

Iteration 1

VI(Σ, S_g, V_0)
  $V \leftarrow V_0$
  until reaching an approximate fixed point **do**
    for each $s \in S \setminus S_g$ **do**
      Bellman-Update(s)

$\eta = 0.25$

Bellman-Update(s)
  for every $a \in$ Applicable(s) **do**
    $Q(s,a) \leftarrow \sum_{s' \in \gamma(s,a)} \Pr(s'|s,a)\, [\text{cost}(s,a,s') + V(s')]$
  $V(s) \leftarrow \min_a Q(s,a)$
  $\pi(s) \leftarrow \text{argmin}_a Q(s,a)$

initial values

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

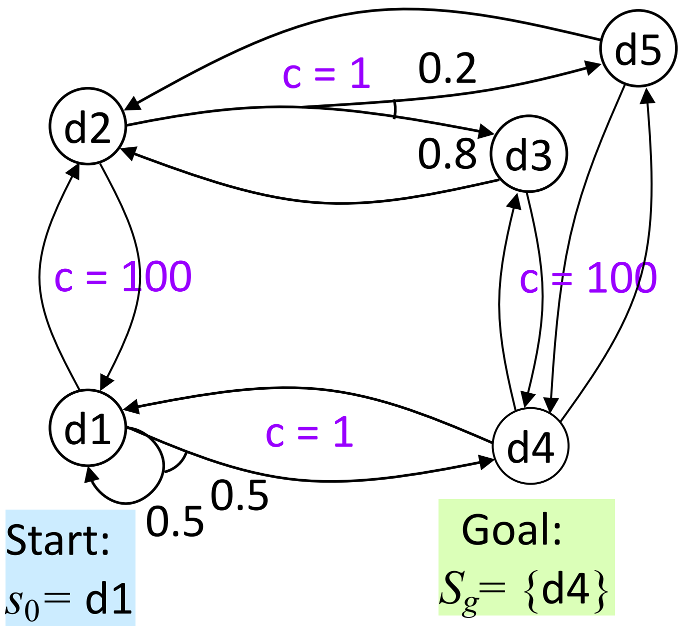
$Q(\text{d1,m12}) = 100 + 0 = 100$
$Q(\text{d1,m14}) = ½(1 + 0) + ½(1 + 0) = 1$
$V(\text{d1}) = 1;\ \pi(\text{d1}) = \text{m14};\quad \Delta V(\text{d1}): 1 – 0 = 1$

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

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

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

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



Iteration 2

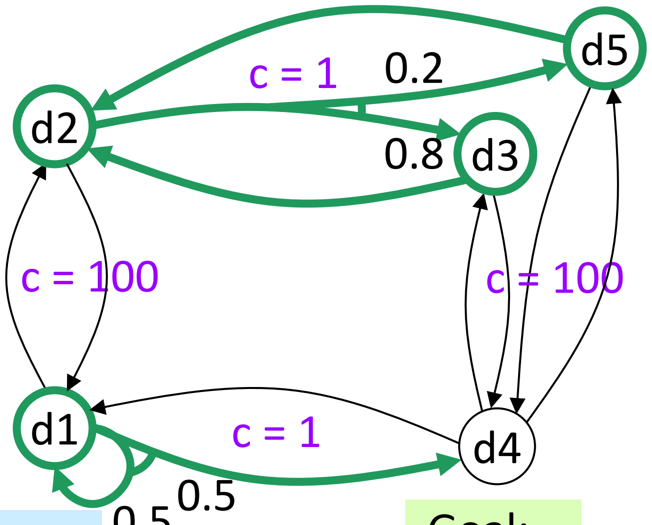
VI(Σ, S_g, V_0)
  $V \leftarrow V_0$
  until reaching an approximate fixed point **do**
    for each $s \in S \setminus S_g$ **do**
      Bellman-Update(s)

$\eta = 0.25$

Bellman-Update(s)
  for every $a \in$ Applicable(s) **do**
    $Q(s,a) \leftarrow \sum_{s' \in \gamma(s,a)} \Pr(s'|s,a)\,[\text{cost}(s,a,s') + V(s')]$
  $V(s) \leftarrow \min_a Q(s,a)$
  $\pi(s) \leftarrow \text{argmin}_a Q(s,a)$

from iteration 1

V(d1) = 1
V(d2) = 1
V(d3) = 2
V(d5) = 2
π(d1) = m14
π(d2) = m23
π(d2) = m32
π(d5) = m52



Q(d1,m12) = 100 + 1 = 101
Q(d1,m14) = 1 + ½(1) + ½(0) = 1½
V(d1) = 1½; π(d1) = m14; $\Delta V(d1)$: 1½ – 1 = ½

Q(d2,m21) = 100 + 1½ = 101½
Q(d2,m23) = 1 + .8(2) + .2(2) = 3
V(d2) = 3; π(d2) = m23; $\Delta V(d2)$: 3 – 1 = 2

Q(d3,m32) = 1 + 3 = 4
Q(d3,m34) = 100 + 0 = 100
V(d3) = 4; π(d3) = m32; $\Delta V(d3)$: 4 – 2 = 2

Q(d5,m52) = 1 + 3 = 4
Q(d5,m54) = 100 + 0 = 100
V(d5) = 4; π(d5) = m52; $\Delta V(d5)$: 4 – 2 = 2

r = max(½, 2, 2, 2) = 2

Iteration 3

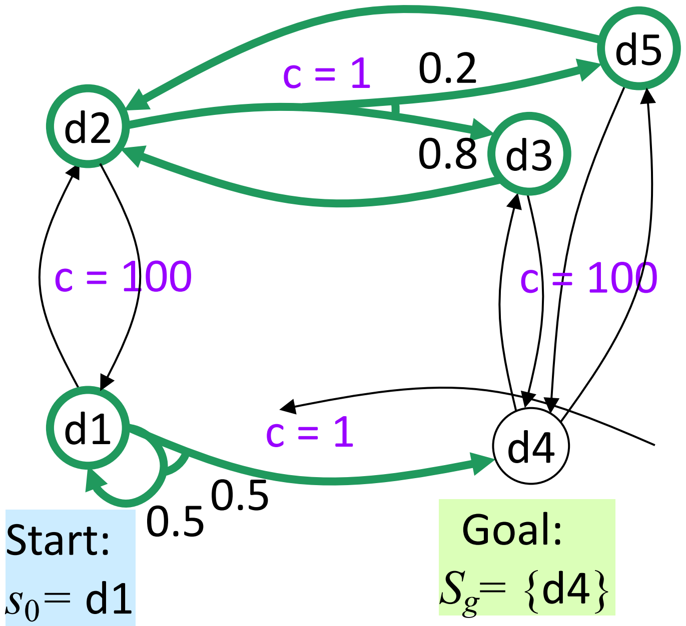
VI(Σ, S_g, V_0)
  $V \leftarrow V_0$
  until reaching an approximate fixed point **do**
    for each $s \in S \setminus S_g$ **do**
      Bellman-Update(s)

Bellman-Update(s)
  for every $a \in$ Applicable(s) **do**
    $Q(s,a) \leftarrow \sum_{s' \in \gamma(s,a)} \Pr(s'|s,a)\,[\mathrm{cost}(s,a,s') + V(s')]$
  $V(s) \leftarrow \min_a Q(s,a)$
  $\pi(s) \leftarrow \mathrm{argmin}_a\, Q(s,a)$

$\eta = 0.25$

from iteration 2

$V(\mathrm{d1}) = 1½$
$V(\mathrm{d2}) = 3$
$V(\mathrm{d3}) = 4$
$V(\mathrm{d5}) = 4$
$\pi(\mathrm{d1}) = \mathrm{m14}$
$\pi(\mathrm{d2}) = \mathrm{m23}$
$\pi(\mathrm{d2}) = \mathrm{m32}$
$\pi(\mathrm{d5}) = \mathrm{m52}$



$Q(\mathrm{d1},\mathrm{m12}) = 100 + 3 = 103$
$Q(\mathrm{d1},\mathrm{m14}) = 1 + ½(1½) + ½(0) = 1\,^3\!/_4$
$V(\mathrm{d1}) = 1\,^3\!/_4$; $\pi(\mathrm{d1}) = \mathrm{m14}$; $\Delta V(\mathrm{d1})$: $1\,^3\!/_4 - 1½ = ¼$

$Q(\mathrm{d2},\mathrm{m21}) = 100 + 1\,^3\!/_4 = 101\,^3\!/_4$
$Q(\mathrm{d2},\mathrm{m23}) = 1 + .8(4) + .2(4) = 5$
$V(\mathrm{d2}) = 5$; $\pi(\mathrm{d2}) = \mathrm{m23}$; $\Delta V(\mathrm{d2})$: $5 - 3 = 2$

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

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

$r = \max(¼, 2, 2, 2) = 2$

Iteration 4

VI(Σ, S_g, V_0)
  $V \leftarrow V_0$
  until reaching an approximate fixed point **do**
    for each $s \in S \setminus S_g$ **do**
      Bellman-Update(s)

$\eta = 0.25$

Bellman-Update(s)
  for every $a \in$ Applicable(s) **do**
    $Q(s,a) \leftarrow \sum_{s' \in \gamma(s,a)} \Pr(s'|s,a)\,[\text{cost}(s,a,s') + V(s')]$
  $V(s) \leftarrow \min_a Q(s,a)$
  $\pi(s) \leftarrow \text{argmin}_a Q(s,a)$

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

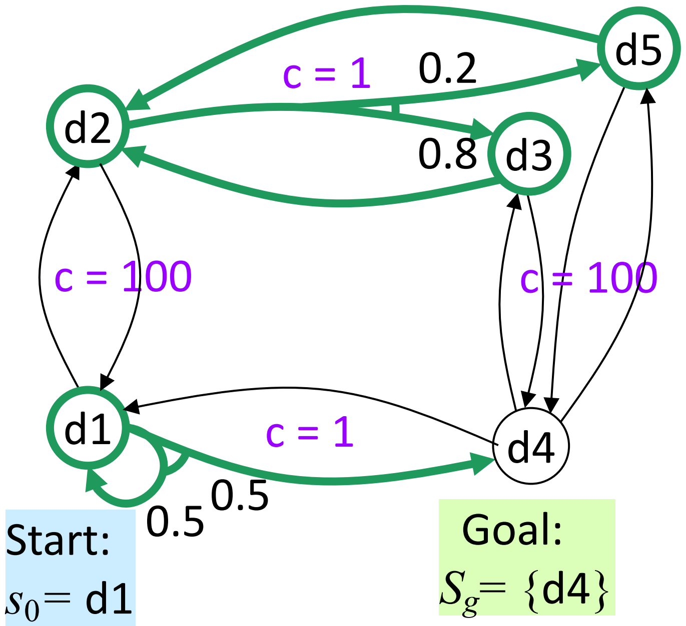
Q(d1,m12) = 100 + 5 = 105
Q(d1,m14) = 1 + ½($1^3/_4$) + ½(0) = $1^7/_8$
V(d1) = $1^7/_8$; π(d1) = m14; $1^7/_8 - 1^3/_4 = {}^1/_8$

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

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



Poll: Total number of iterations?

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

Iteration 1, with a better V_0

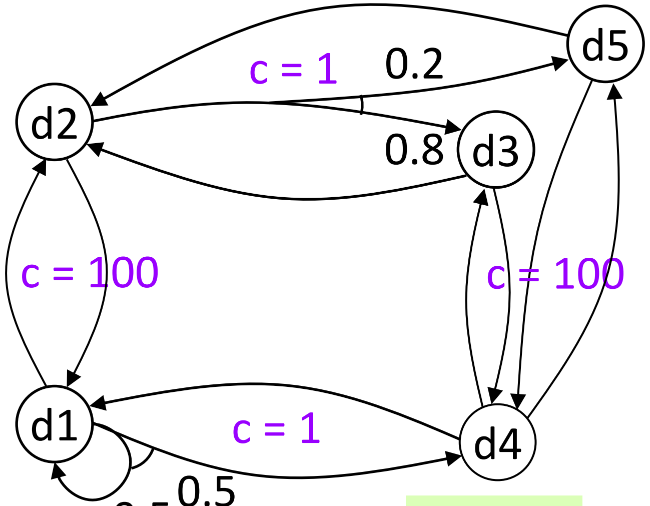
VI(Σ, S_g, V_0)
  $V \leftarrow V_0$
  until reaching an approximate fixed point **do**
    for each $s \in S \setminus S_g$ **do**
      Bellman-Update(s)

$\eta = 0.25$

Bellman-Update(s)
  for every $a \in$ Applicable(s) **do**
    $Q(s,a) \leftarrow \sum_{s' \in \gamma(s,a)} \Pr(s'|s,a)\,[\text{cost}(s,a,s') + V(s')]$
  $V(s) \leftarrow \min_a Q(s,a)$
  $\pi(s) \leftarrow \text{argmin}_a Q(s,a)$

$V_0(s)$ = min cost of path to d4

V(d1) = 1
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(1) = 1.5
V(d1) = 1.5; π(d1) = m14; ΔV(d1): 1.5 – 1 = 0.5

Q(d2,m21) = 100 + 1.5 = 101.5
Q(d2,m23) = 1 + .8(100) + .2(100) = 101
V(d2) = 101; π(d2) = m23; ΔV(d2): 101 – 101 = 0

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

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

r = max(0.5, 0, 0, 0) = 0.5

Iteration 2

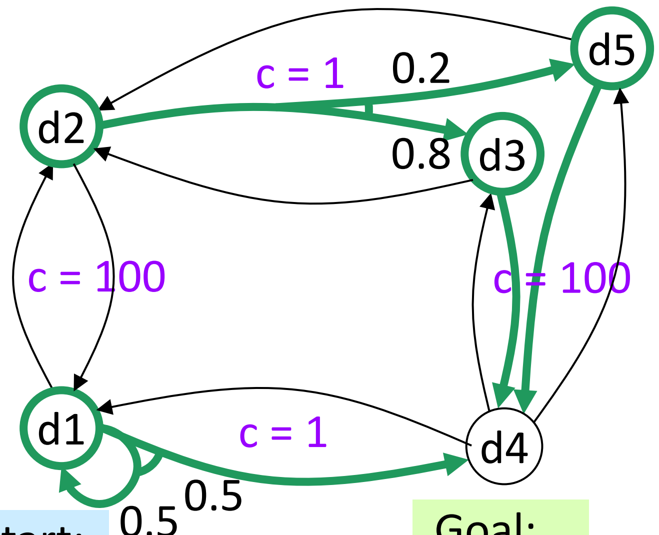
VI(Σ, S_g, V_0)
  $V \leftarrow V_0$
  until reaching an approximate fixed point **do**
    for each $s \in S \setminus S_g$ **do**
      Bellman-Update(s)

$\eta = 0.25$

Bellman-Update(s)
  for every $a \in$ Applicable(s) **do**
    $Q(s,a) \leftarrow \sum_{s' \in \gamma(s,a)} \Pr(s'|s,a)\,[\text{cost}(s,a,s') + V(s')]$
  $V(s) \leftarrow \min_a Q(s,a)$
  $\pi(s) \leftarrow \text{argmin}_a Q(s,a)$

from iteration 1

V(d1) = 1.5
V(d2) = 101
V(d3) = 100
V(d5) = 100
π(d1) = m14
π(d2) = m23
π(d2) = m34
π(d5) = m54



Start:
s_0= d1

Goal:
S_g= {d4}

Q(d1,m12) = 100 + 101 = 201
Q(d1,m14) = 1 + ½(0) + ½(1.5) = 1.75
V(d1) = 1.75; π(d1) = m14; Δ1.75 – 1.5 = 0.25

Q(d2,m21) = 100 + 2 = 102
Q(d2,m23) = 1 + .8(100) + .2(100) = 101
V(d2) = 101; π(d2) = m23; Δ101 – 101 = 0

Q(d3,m32) = 1 + 101 = 102
Q(d3,m34) = 100 + 0 = 100
V(d3) = 100; π(d3) = m34; Δ100 – 100 = 0

Q(d5,m52) = 1 + 101 = 102
Q(d5,m54) = 100 + 0 = 100
V(d5) = 100; π(d5) = m54; Δ100 – 100 = 0

r = max(0.25, 0, 0, 0) = 0.25
VI returns π

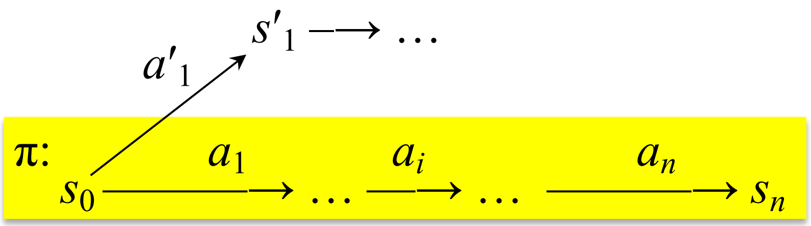
Discussion

- 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 $Q^\pi(s,a)$ at each state
- More work per iteration than value iteration
  - Needs to solve simultaneous equations

- In each iteration of Value Iteration
  - Compute new V: min $Q(s,a)$ at each state
  - New V is a revised set of heuristic estimates
    - Doesn't depend on any π
  - new π = argmin $Q(s,a)$ at each state
    - But $V \neq V^\pi$
  - Less work per iteration: doesn't need to solve a set of equations
- More iterations to converge unless V_0 is good

- 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 as Value Iteration



$f(s_0) = h(s_0)$ $\qquad$ $f(\nu_n) = \text{cost}(\langle a_1,\ldots,a_n\rangle) + h(s_n)$

$V(s_0) = \text{cost}(\langle a_1,\ldots,a_n\rangle) + h(s_n)$ $\qquad$ $V(s_n) = h(s_n)$

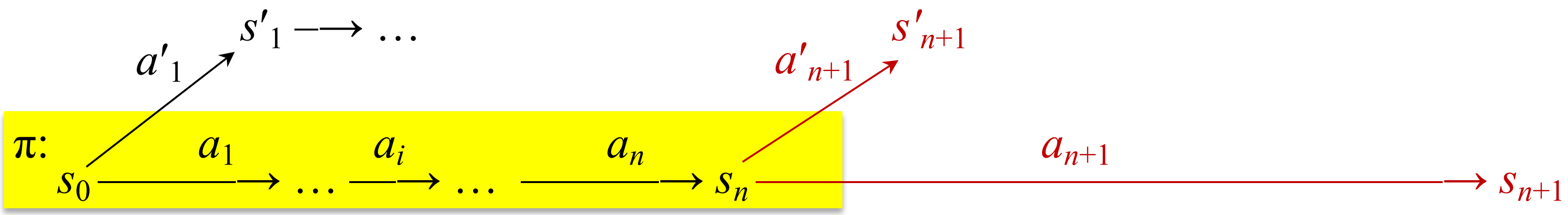
- At each iteration, A* chooses frontier node with smallest f value
  = frontier node with min $\text{cost}(\langle a_1,\ldots,a_n\rangle) + h(s_n)$
- For $i = 1, \ldots, n$, let
  $V(s_i) = \min[(\text{cost of path from } s_i \text{ to frontier}) + h(\text{frontier state})]$
  $= \text{cost}(\langle a_{i+1},\ldots,a_n\rangle) + h(s_i)$
- Can do A* search with either f or V
- Make $\langle a_1,\ldots,a_n\rangle$ a policy:
  - $\pi(s_0) = a_1,\ \pi(s_1) = a_2,\ \ldots,\ \pi(s_{n-1}) = a_n$

frontier state with smallest f

- **while** True **do**
  - starting at s_0, follow π to a frontier state s
  - **if** s is a goal **then return** π
  - expand s
  - do A*'s usual pruning
  - **for each** s' on the path from s back to s_0 **do**
    - **for each** $a \in \mathit{Applicable}(s')$ **do**
      - $Q(s', a) = \text{cost}(a) + V(\gamma(s', a))$
      - $V(s') = \min_a Q(s', a)$
      - $\pi(s') = \text{argmin}_a Q(s', a)$

Bellman update

Digression: A* Search as Value Iteration



$f(s_0) = h(s_0)$

$V(s_0) = \text{cost}(\langle a_1,\ldots,a_n\rangle) + h(s_n)$

$f(v_n) = \text{cost}(\langle a_1,\ldots,a_n\rangle) + h(s_n)$

$V(s_n) = h(s_n)$

$f(v_{n+1}) = \text{cost}(\langle a_1,\ldots,a_n,a_{n+1}\rangle) + h(s_{n+1})$

$V(s_{n+1}) = h(s_{n+1})$

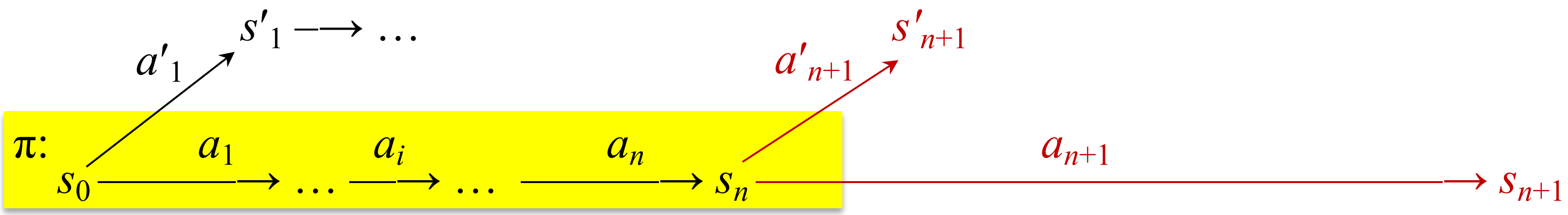
- At each iteration, A* chooses frontier node with smallest f value
  = frontier node with min $\text{cost}(\langle a_1,\ldots,a_n\rangle) + h(s_n)$
- For $i = 1, \ldots, n$, let
  $V(s_i) = \min[(\text{cost of path from } s_i \text{ to frontier}) + h(\text{frontier state})]$
  $= \text{cost}(\langle a_{i+1},\ldots,a_n\rangle) + h(s_i)$
- Can do A* search with either f or V
- Make $\langle a_1,\ldots,a_n\rangle$ a policy:
  - $\pi(s_0) = a_1,\ \pi(s_1) = a_2,\ \ldots,\ \pi(s_{n-1}) = a_n$

frontier state with smallest f

- **while** True **do**
  - starting at s_0, follow π to a frontier state s
  - **if** s is a goal **then return** π
  - expand s
  - do A*'s usual pruning
  - **for each** s' on the path from s back to s_0 **do**
    - **for each** $a \in Applicable(s')$ **do**
      - $Q(s', a) = \text{cost}(a) + V(\gamma(s', a))$
      - $V(s') = \min_a Q(s', a)$
      - $\pi(s') = \text{argmin}_a Q(s', a)$

Bellman update

Digression: A* Search as Value Iteration



$f(s_0) = h(s_0)$

$V(s_0) = \min\{c_1 + V(s_1),\ c'_1 + V(s'_1)\}$
$\quad = \text{cost}(a_1) + V(s_1)$
$\quad = \text{cost}(\langle a_1,\ldots,a_n,a_{n+1}\rangle) + h(s_{n+1})$

$f(\nu_n) = \text{cost}(\langle a_1,\ldots,a_n\rangle) + h(s_n)$

$V(s_n) = \min\{c_{n+1} + V(s_{n+1}),\ c'_{n+1} + V(s'_{n+1})\}$
$\quad = \text{cost}(a_{n+1}) + V(s_{n+1})$
$\quad = \text{cost}(\langle a_{n+1}\rangle) + h(s_{n+1})$

$f(\nu_{n+1}) = \text{cost}(\langle a_1,\ldots,a_n,a_{n+1}\rangle) + h(s_{n+1})$

$V(s_{n+1}) = h(s_{n+1})$

- At each iteration, A* chooses frontier node with smallest f value
  = frontier node with min $\text{cost}(\langle a_1,\ldots,a_n\rangle) + h(s_n)$
- For $i = 1, \ldots, n$, let
  $V(s_i) = \min[(\text{cost of path from } s_i \text{ to frontier}) + h(\text{frontier state})]$
  $= \text{cost}(\langle a_{i+1},\ldots,a_n\rangle) + h(s_i)$
- Can do A* search with either f or V
- Make $\langle a_1,\ldots,a_n\rangle$ a policy:
  - $\pi(s_0) = a_1,\ \pi(s_1) = a_2,\ \ldots,\ \pi(s_{n-1}) = a_n$

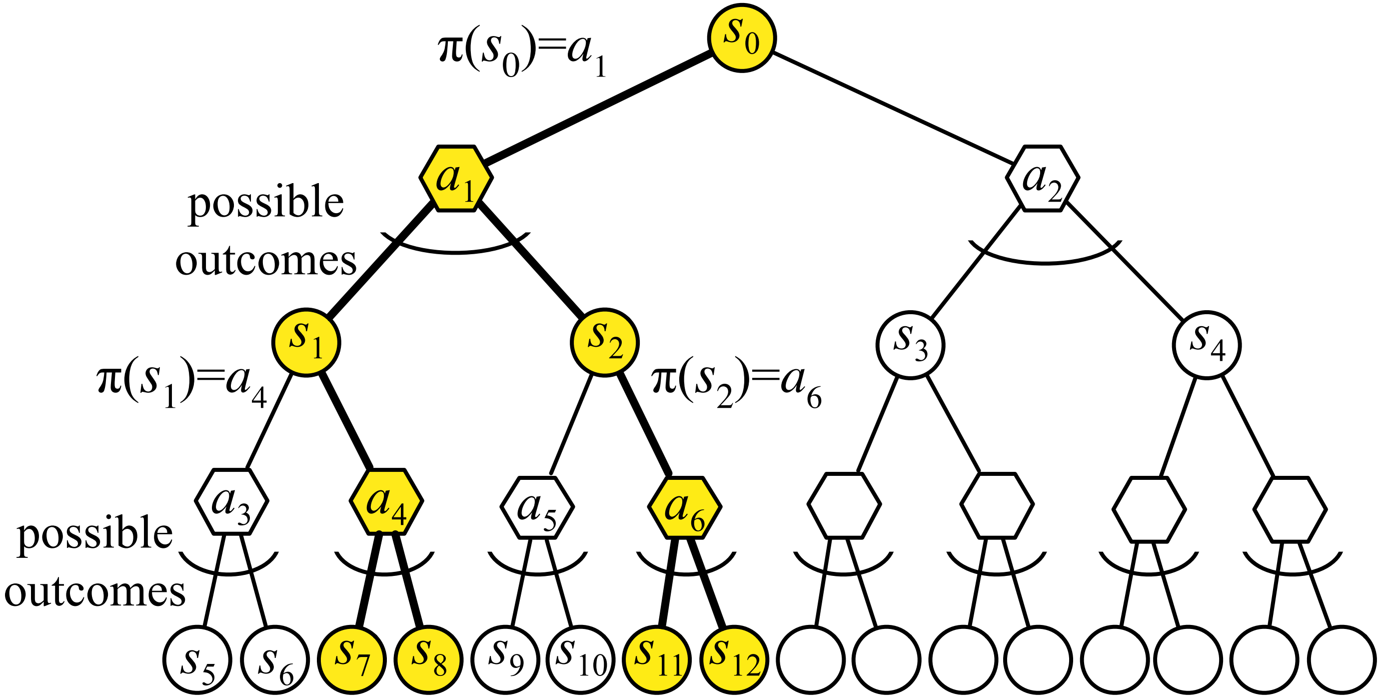
frontier state with smallest f

- **while** True **do**
  - starting at s_0, follow π to a frontier state s
  - **if** s is a goal **then return** π
  - expand s
  - do A*'s usual pruning
  - **for each** s' on the path from s back to s_0 **do**
    - **for each** $a \in Applicable(s')$ **do**
      - $Q(s', a) = \text{cost}(a) + V(\gamma(s', a))$
      - $V(s') = \min_a Q(s', a)$
      - $\pi(s') = \text{argmin}_a Q(s', a)$

Bellman update

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* MDPs
- $leaves(s_0,\pi) = \{s_7, s_8, s_{11}, s_{12}\}$
  - Expand one of them, e.g., s_{11}
- Going bottom-up, update V and π values for s_{11} and its ancestors
  - $V(s) = \min_{a \in \text{Applicable}(s)} \sum_{s' \in \gamma(s,a)} P(s' | s,a)\ [\text{cost}(s,a,s') + V(s')\}$
  - $\pi(s) = \text{argmin}_{a \in \text{Applicable}(s)} \sum_{s' \in \gamma(s,a)} P(s' | s,a)\ [\text{cost}(s,a,s') + V(s')\}$
- *e.g.,*
  - $V(s_2) = \min(Q(s_2,a_5), Q(s_2,a_6))$



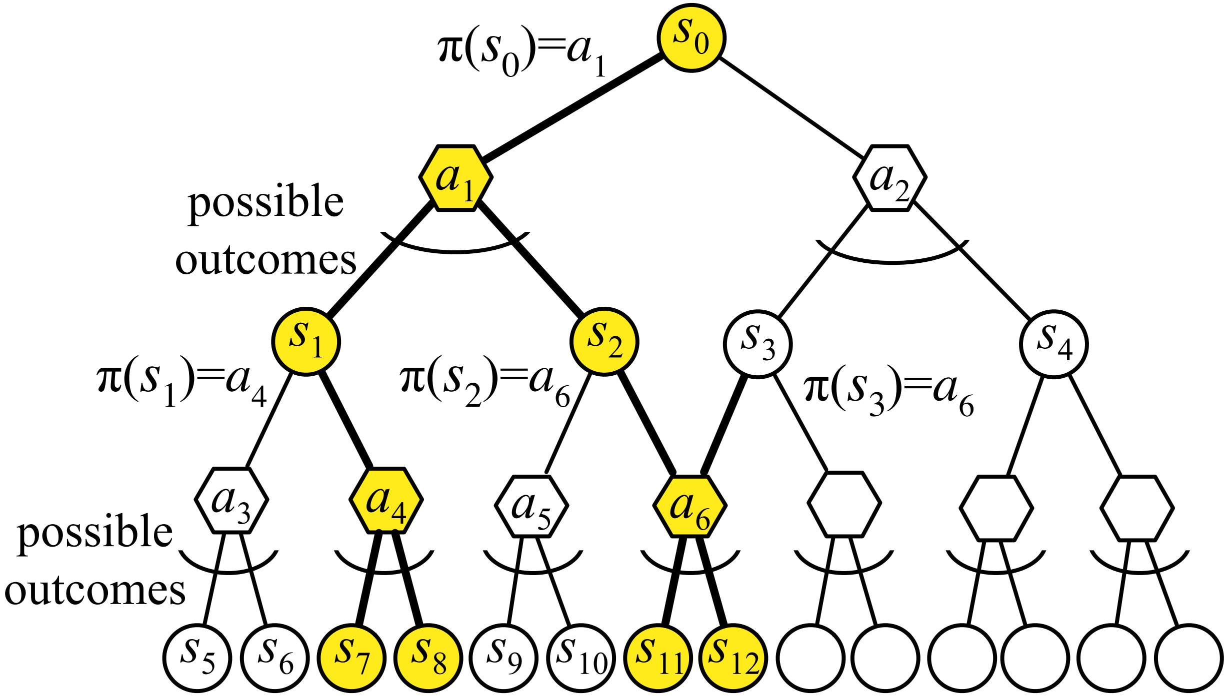
Poll: Can this work correctly on an acyclic AND/OR graph that isn't a tree?
A. yes
B. no
C. don't know

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** MDPs

- *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**
  - $V(s) = \min_{a \in \text{Applicable}(s)} \sum_{s' \in \gamma(s,a)} P(s' | s,a)$ [cost$(s,a,s') + V(s')$}
  - $\pi(s) = \text{argmin}_{a \in \text{Applicable}(s)} \sum_{s' \in \gamma(s,a)} P(s' | s,a)$ [cost$(s,a,s') + V(s')$}
- *e.g.,*
  - $V(s_2) = \min(Q(s_2,a_5), Q(s_2,a_6))$



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

A. yes
B. no
C. sometimes
D. don't know

AO*

AO* (Σ, s_0, S_g, V_0)

global:
- $Envelope \leftarrow \{s_0\}$ — like *Expanded* ∪ *Frontier* in A*
- $\pi \leftarrow \emptyset$
- $V(s_0) \leftarrow V_0(s_0)$

while $Fringe(s_0,\pi) \neq \emptyset$ **do** — $Fringe(s_0,\pi) = leaves(s_0,\pi) \setminus S_g$

  select a state $s \in Fringe(s_0,\pi)$

  expand s:

  for each $a \in$ 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)

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 $\hat{\gamma}(s,\pi(s)) \cap Z = \{s\}$ — s has no descendants in Z

    remove s from Z

    Bellman-Update(s)

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

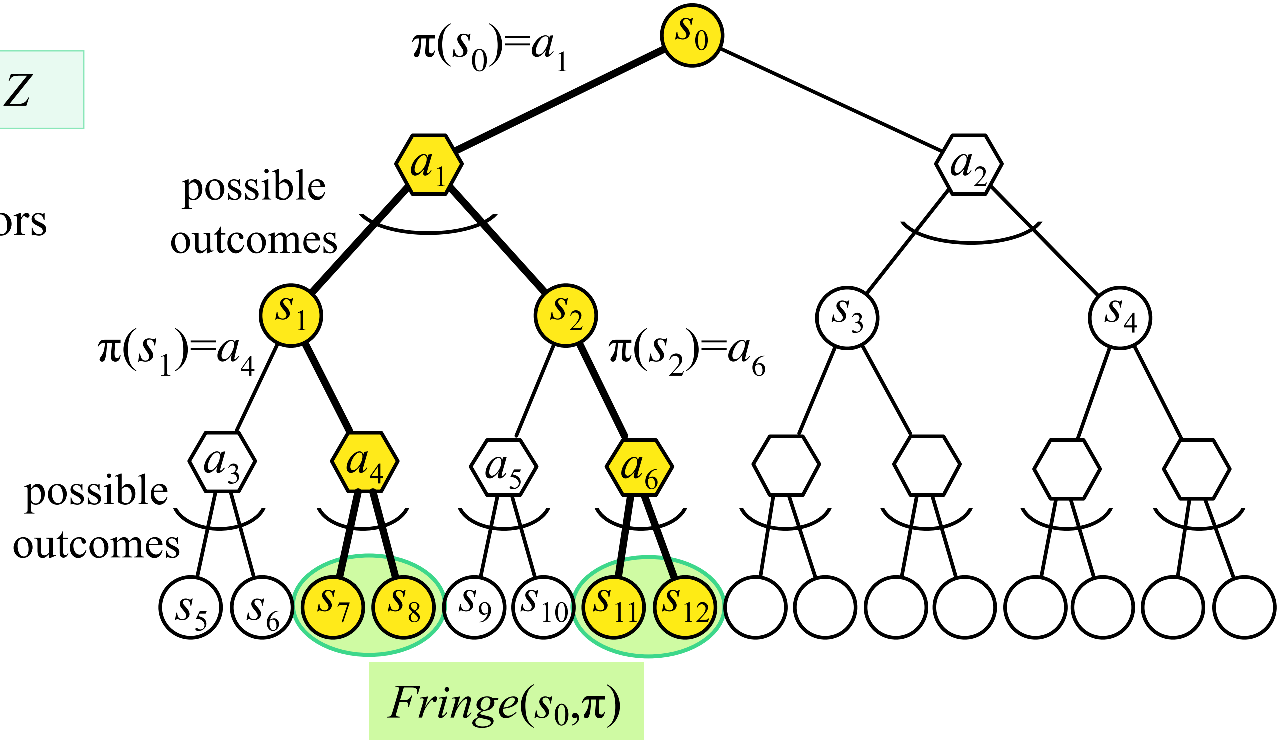
Bellman-Update(s)

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

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

  $V(s) \leftarrow \min_a Q(s,a)$

  $\pi(s) \leftarrow \text{argmin}_a Q(s,a)$



AO* Example

AO* (Σ, s_0, S_g, V_0)

  $Envelope \leftarrow \{s_0\}$

  $\pi \leftarrow \emptyset$

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

  while $Fringe(s_0,\pi) \neq \emptyset$ **do**

    select a state $s \in Fringe(s_0,\pi)$

    for each $a \in$ 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)

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)

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

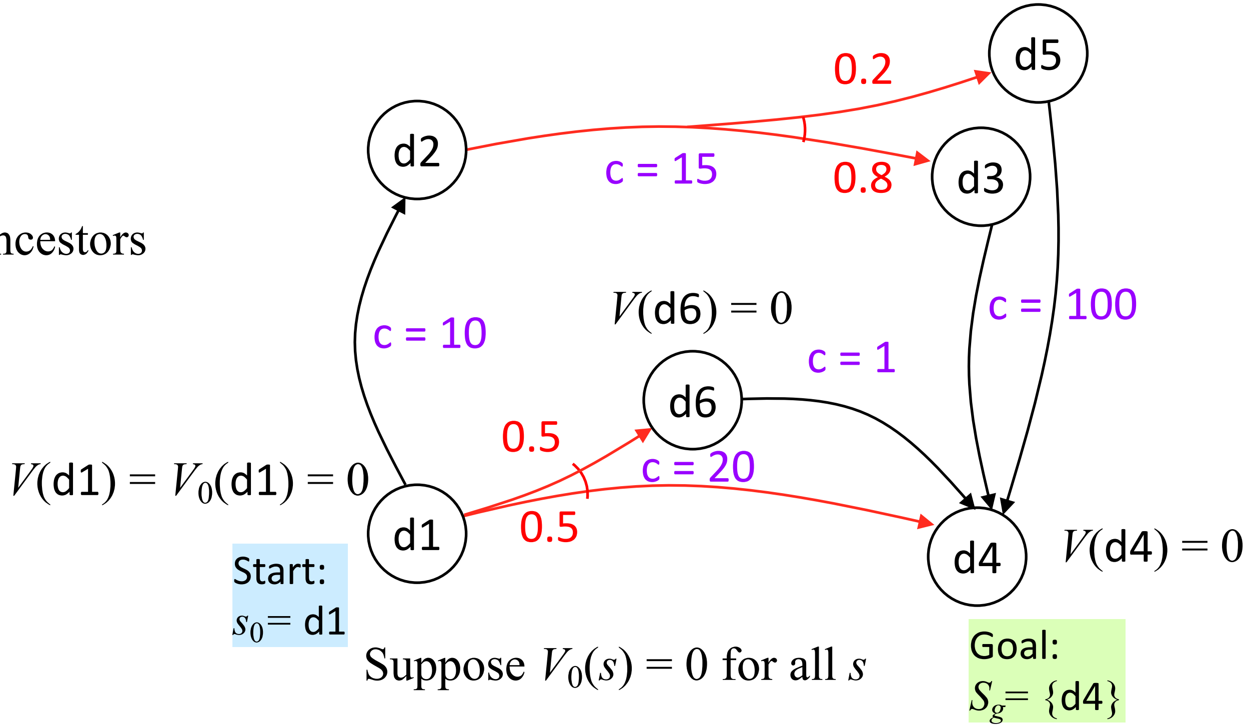
Bellman-Update(s)

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

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

  $V(s) \leftarrow \min_a Q(s,a)$

  $\pi(s) \leftarrow \text{argmin}_a\, Q(s,a)$



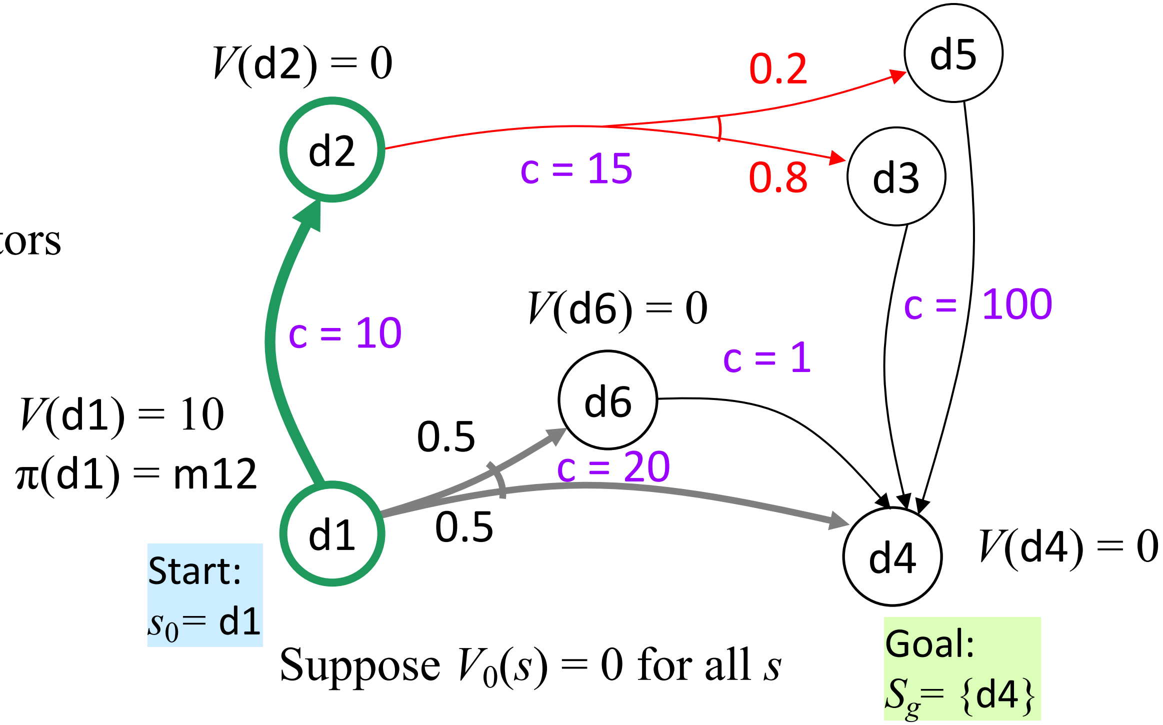
Iteration 1

AO* (Σ, s_0, S_g, V_0)
  Envelope $\leftarrow \{s_0\}$
  $\pi \leftarrow \emptyset$
  $V(s_0) \leftarrow V_0(s_0)$
  while $Fringe(s_0,\pi) \neq \emptyset$ **do**
    select a state $s \in Fringe(s_0,\pi)$
    for each $a \in$ 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)

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

Bellman-Update(s)
  for every $a \in$ Applicable(s) **do**
    $Q(s,a) \leftarrow \sum_{s' \in \gamma(s,a)} \Pr(s'|s,a)\,[\text{cost}(s,a,s') + V(s')]$
  $V(s) \leftarrow \min_a Q(s,a)$
  $\pi(s) \leftarrow \text{argmin}_a Q(s,a)$

Poll: Which states are currently in *Envelope*? (choose 1 or more)
A. d1 B. d2 C. d3 D. d4 E. d5



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

Iteration 2

AO* (Σ, s_0, S_g, V_0)

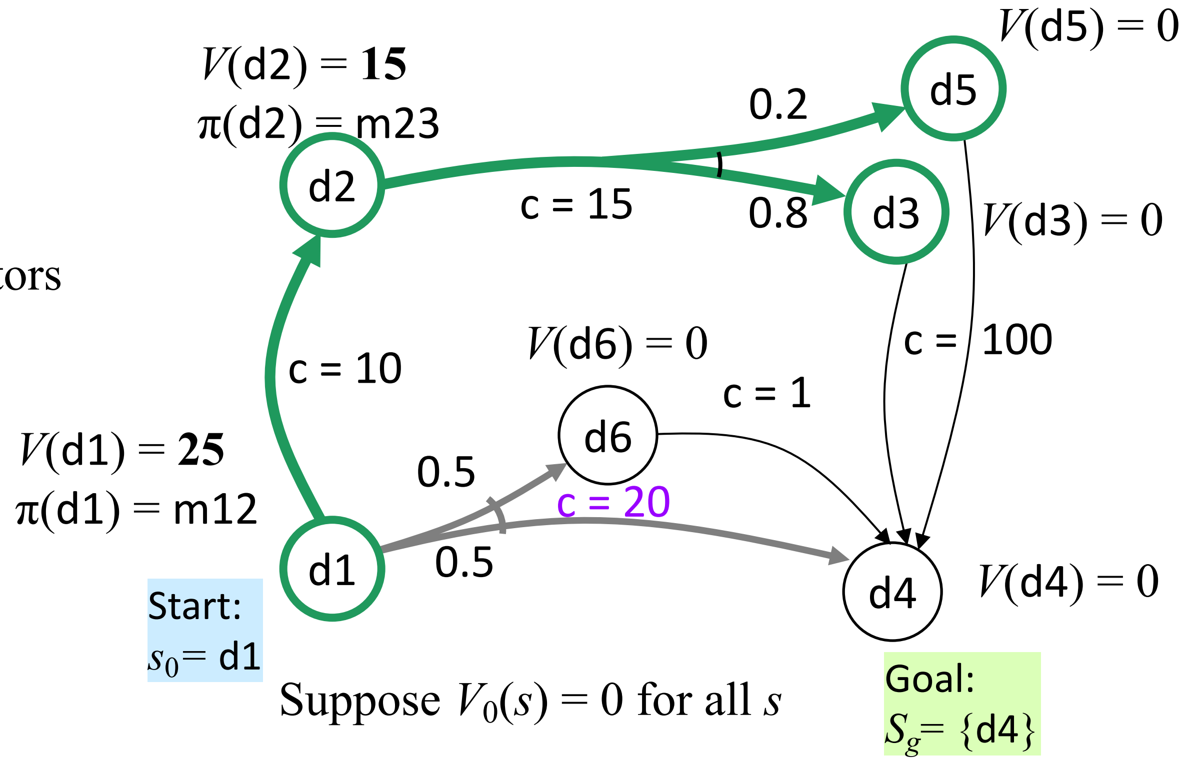
- $Envelope \leftarrow \{s_0\}$
- $\pi \leftarrow \emptyset$
- $V(s_0) \leftarrow V_0(s_0)$
- **while** $Fringe(s_0,\pi) \neq \emptyset$ **do**
  - select a state $s \in Fringe(s_0,\pi)$
  - **for each** $a \in$ Applicable(s) and $s' \in \gamma(s,a)$ **do**
    - **if** $s' \notin Envelope$ **then**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

Bellman-Update(s)

- **for every** $a \in$ Applicable(s) **do**
  - $Q(s,a) \leftarrow \sum_{s' \in \gamma(s,a)} \Pr(s'|s,a)\,[\text{cost}(s,a,s') + V(s')]$
- $V(s) \leftarrow \min_a Q(s,a)$
- $\pi(s) \leftarrow \text{argmin}_a Q(s,a)$

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



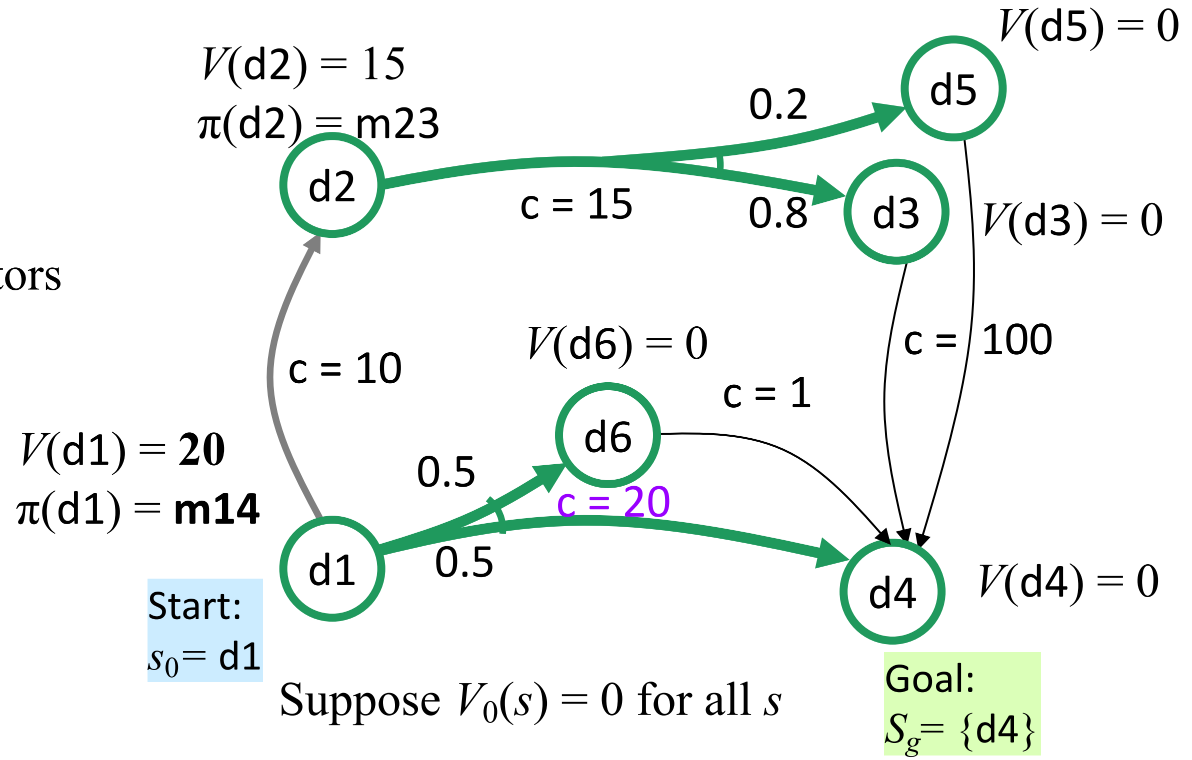
Suppose $V_0(s) = 0$ for all s

Iteration 3

AO* (Σ, s_0, S_g, V_0)
  Envelope $\leftarrow \{s_0\}$
  $\pi \leftarrow \emptyset$
  $V(s_0) \leftarrow V_0(s_0)$
  while $Fringe(s_0,\pi) \neq \emptyset$ **do**
    select a state $s \in Fringe(s_0,\pi)$
    for each $a \in$ 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)

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

Bellman-Update(s)
  for every $a \in$ Applicable(s) **do**
    $Q(s,a) \leftarrow \sum_{s' \in \gamma(s,a)} \Pr(s'|s,a)\,[\text{cost}(s,a,s') + V(s')]$
  $V(s) \leftarrow \min_a Q(s,a)$
  $\pi(s) \leftarrow \text{argmin}_a Q(s,a)$



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

Iteration 4

AO* (Σ, s_0, S_g, V_0)

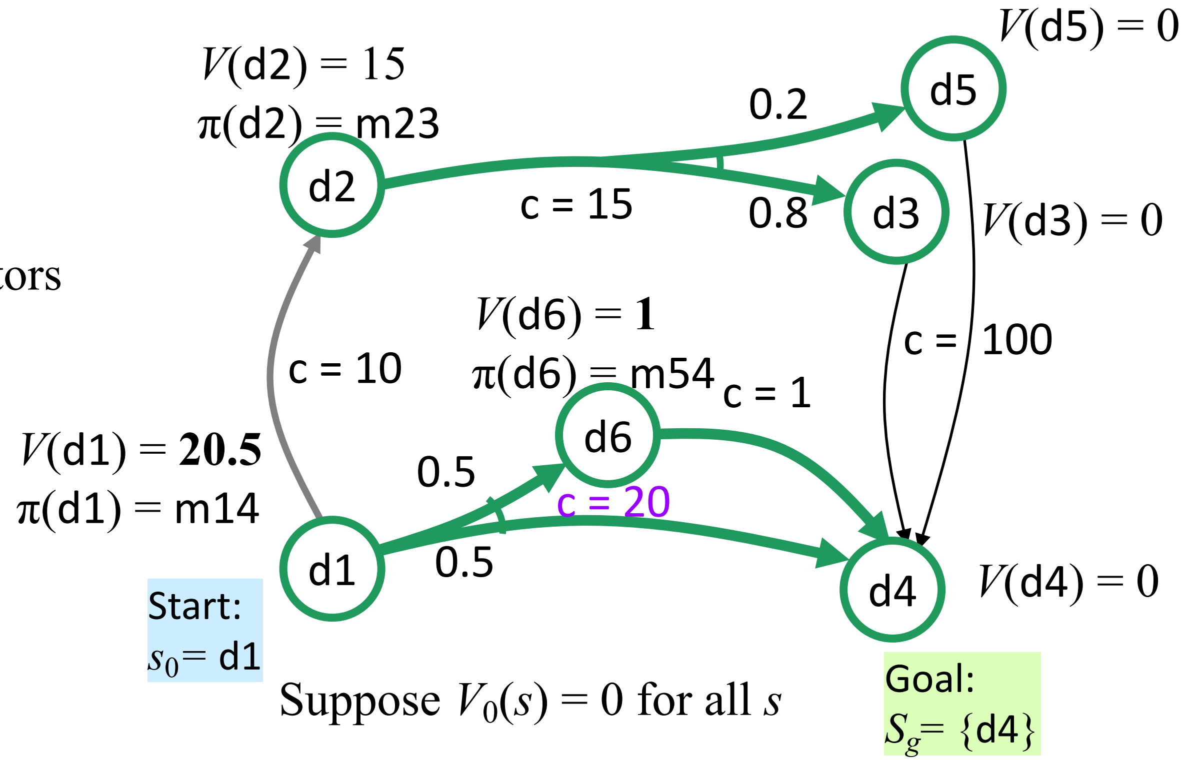
- *Envelope* $\leftarrow \{s_0\}$
- $\pi \leftarrow \emptyset$
- $V(s_0) \leftarrow V_0(s_0)$
- **while** *Fringe*$(s_0,\pi) \neq \emptyset$ **do**
  - select a state $s \in$ *Fringe*(s_0,π)
  - **for each** $a \in$ Applicable(s) and $s' \in \gamma(s,a)$ **do**
    - **if** $s' \notin$ *Envelope* **then**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

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

Bellman-Update(s)

- **for every** $a \in$ Applicable(s) **do**
  - $Q(s,a) \leftarrow \sum_{s' \in \gamma(s,a)} \Pr(s'|s,a)\,[\mathrm{cost}(s,a,s') + V(s')]$
- $V(s) \leftarrow \min_a Q(s,a)$
- $\pi(s) \leftarrow \mathrm{argmin}_a Q(s,a)$

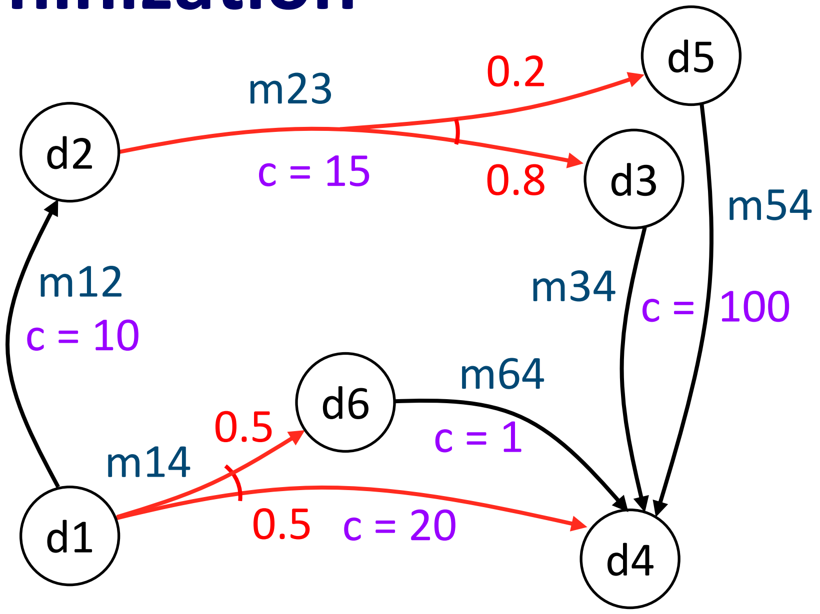


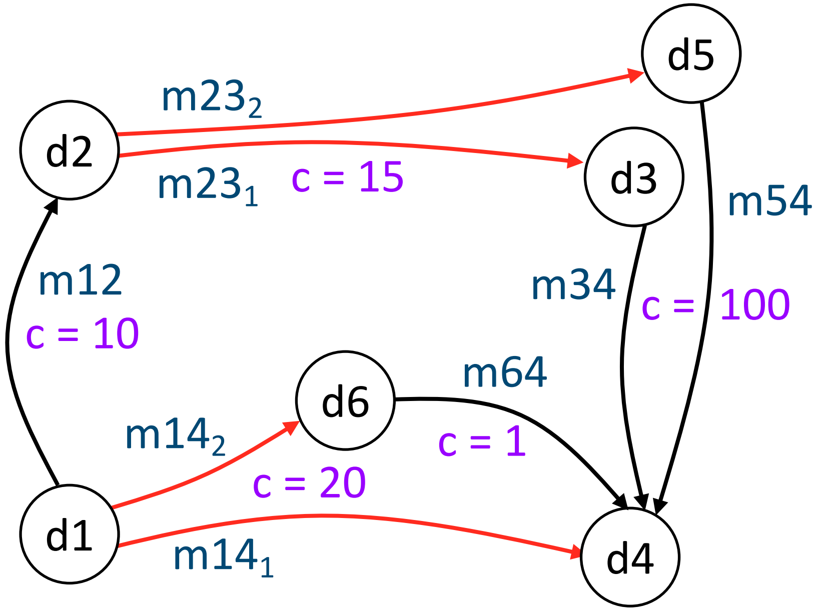
Heuristics through Determinization

What to use for $V_0(s)$?

- One possibility: a heuristic 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
- Proof:
  - Let π^* be any 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 $h(s) \leq \text{cost}(p^*) \leq E(\text{cost}(\pi))$

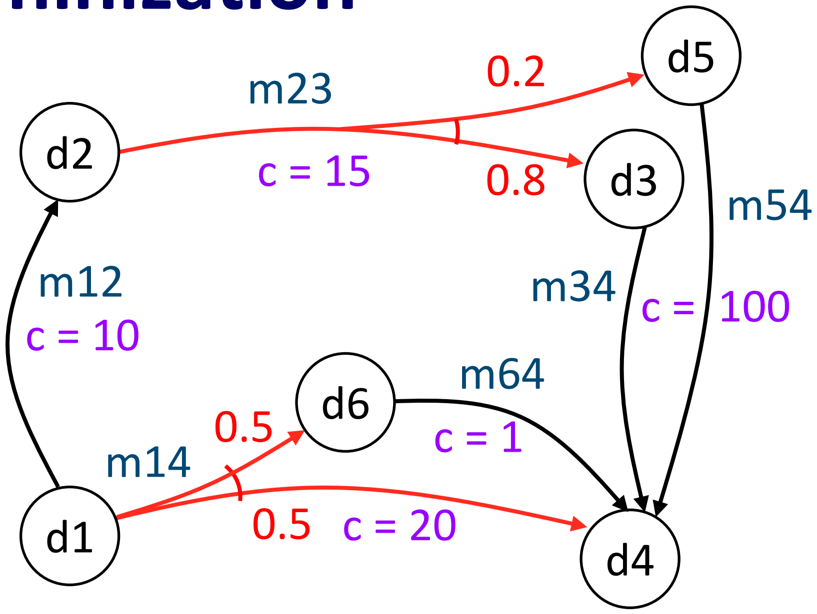


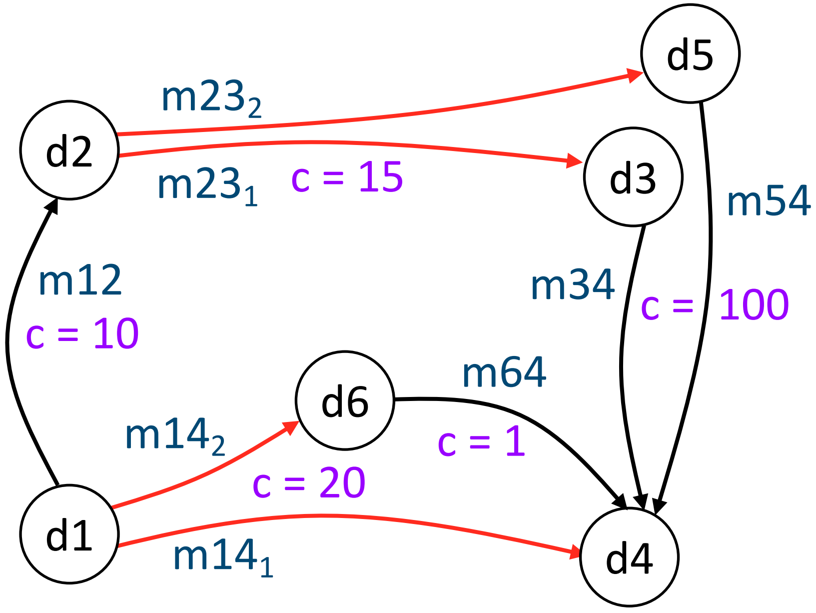


Heuristics through Determinization

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 $V_0(s) = \text{cost}(p)$
- If the classical planner always returns optimal solutions, then V_0 is admissible for Σ
- Proof:
  - Let s be any state, π^* be any optimal solution for (Σ, s, g)
  - Let p^* = least costly execution of π^*
  - Then
  - p^* is an optimal solution for (Σ_d, s, g),
    - so $V_0(s) = \text{cost}(p) = \text{cost}(p^*) \leq E(\text{cost}(\pi))$

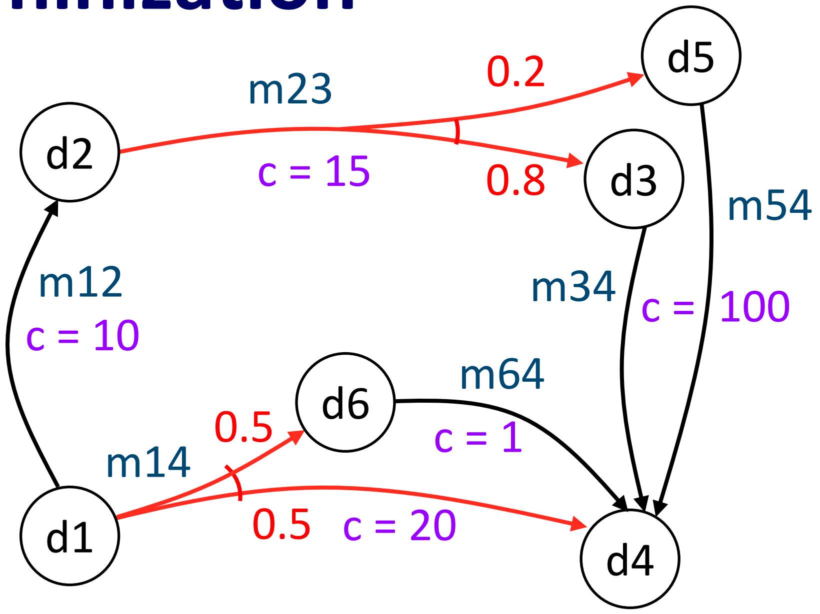


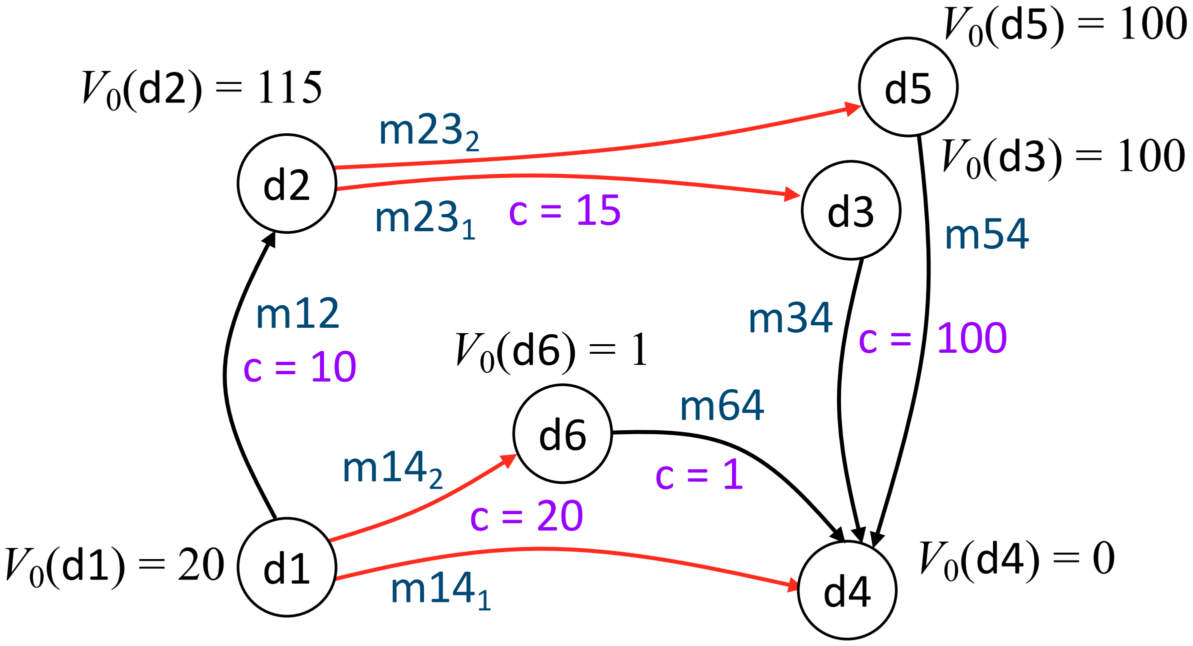


Heuristics through Determinization

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 solutions, then V_0 is admissible
- Outline of proof:
    - Let π^* be any optimal solution for (Σ, s, g)
    - Then $V^*(s) = E(\text{cost}(\pi^*))$
      = weighted avg. of all executions of π
    - 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 ≤ cost
        - So $V_0(s) = \text{cost}(p) \leq \text{cost}(p^*) \leq E(\text{cost}(\pi))$





Example

AO* (Σ, s_0, S_g, V_0)

$Envelope \leftarrow \{s_0\}$

$\pi \leftarrow \emptyset$

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

while $Fringe(s_0,\pi) \neq \emptyset$ **do**

  select a state $s \in Fringe(s_0,\pi)$

  for each $a \in$ 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)

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)

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

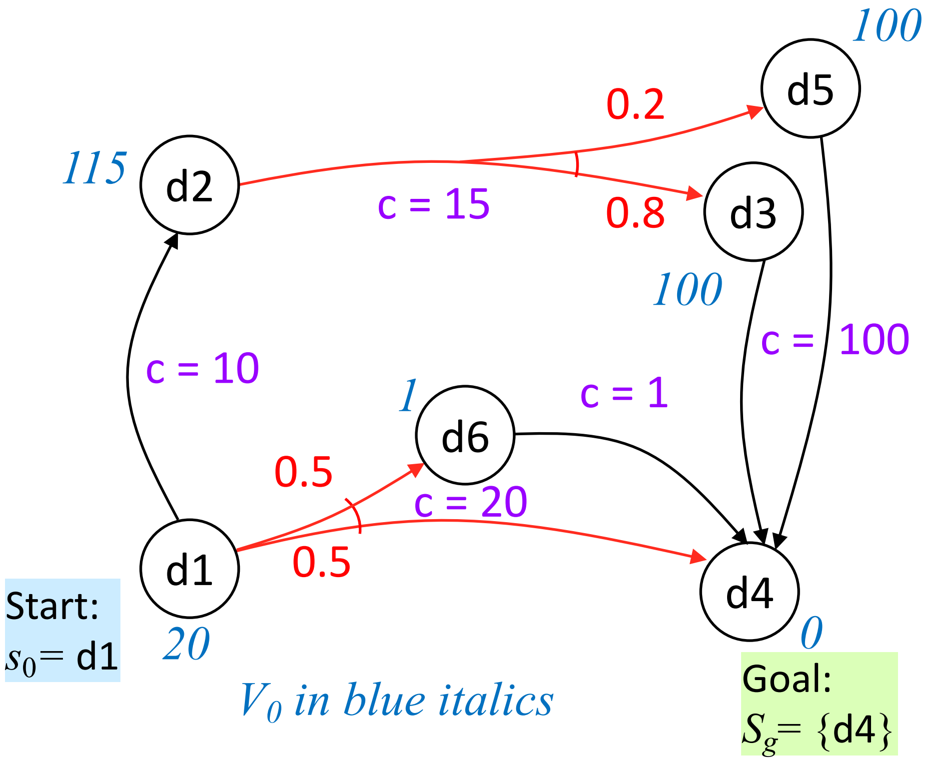
Bellman-Update(s)

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

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

$V(s) \leftarrow \min_a Q(s,a)$

$\pi(s) \leftarrow \text{argmin}_a Q(s,a)$



Iteration 1

AO* (Σ, s_0, S_g, V_0)

    $Envelope \leftarrow \{s_0\}$

    $\pi \leftarrow \emptyset$

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

    while $Fringe(s_0,\pi) \neq \emptyset$ **do**

        select a state $s \in Fringe(s_0,\pi)$

        for each $a \in$ 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)

Bellman-Update(s)

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

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

    $V(s) \leftarrow \min_a Q(s,a)$

    $\pi(s) \leftarrow \text{argmin}_a\, Q(s,a)$

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

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

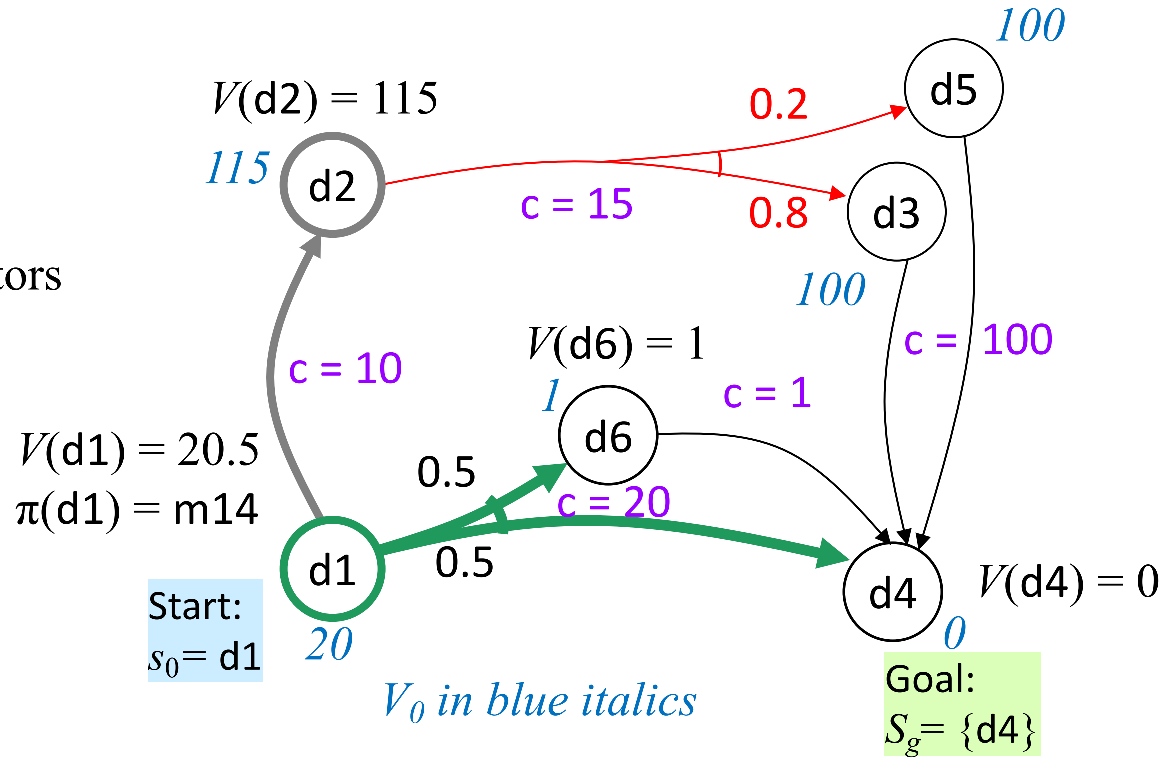
    while $Z \neq \emptyset$ **do**

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

        remove s from Z

        Bellman-Update(s)

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



Iterations 2, 3

AO* (Σ, s_0, S_g, V_0)

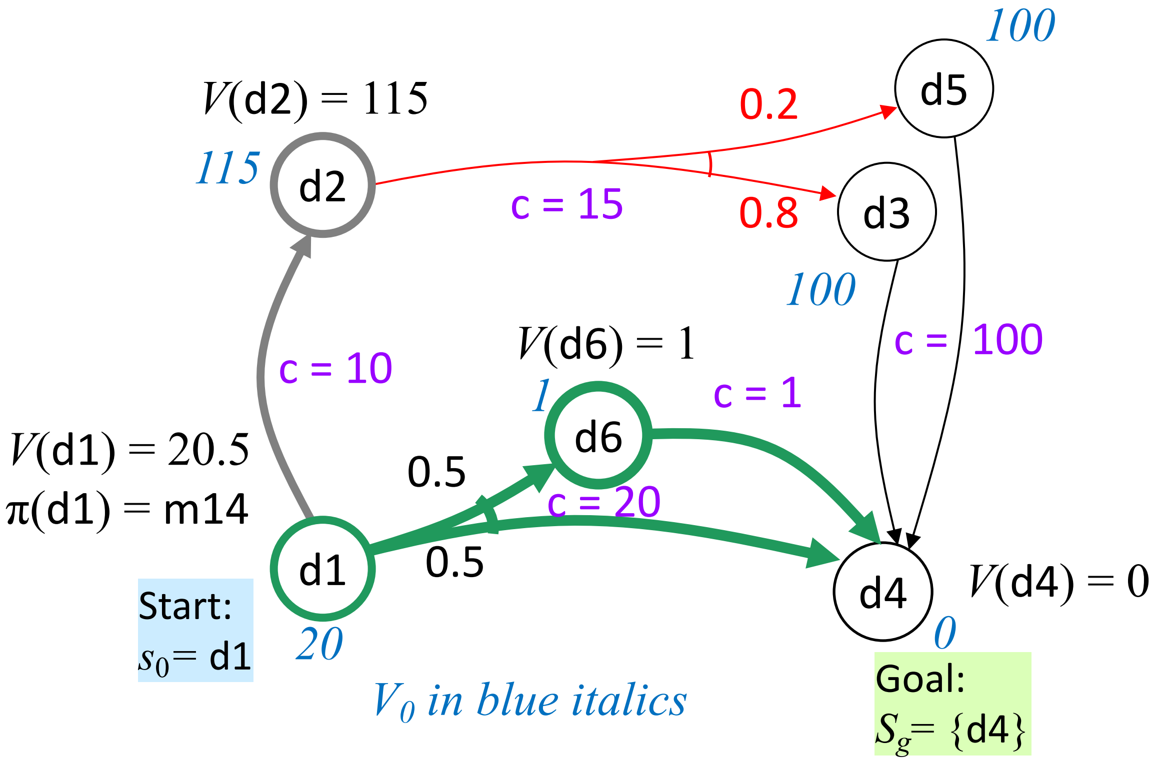
- $Envelope \leftarrow \{s_0\}$
- $\pi \leftarrow \emptyset$
- $V(s_0) \leftarrow V_0(s_0)$
- **while** $Fringe(s_0,\pi) \neq \emptyset$ **do**
  - select a state $s \in Fringe(s_0,\pi)$
  - **for each** $a \in$ Applicable(s) and $s' \in \gamma(s,a)$ **do**
    - **if** $s' \notin Envelope$ **then**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

AO-Update(s) // update 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)
  - $Z \leftarrow Z \cup \{s' \in Envelope \mid s \in \gamma(s',\pi(s'))\}$

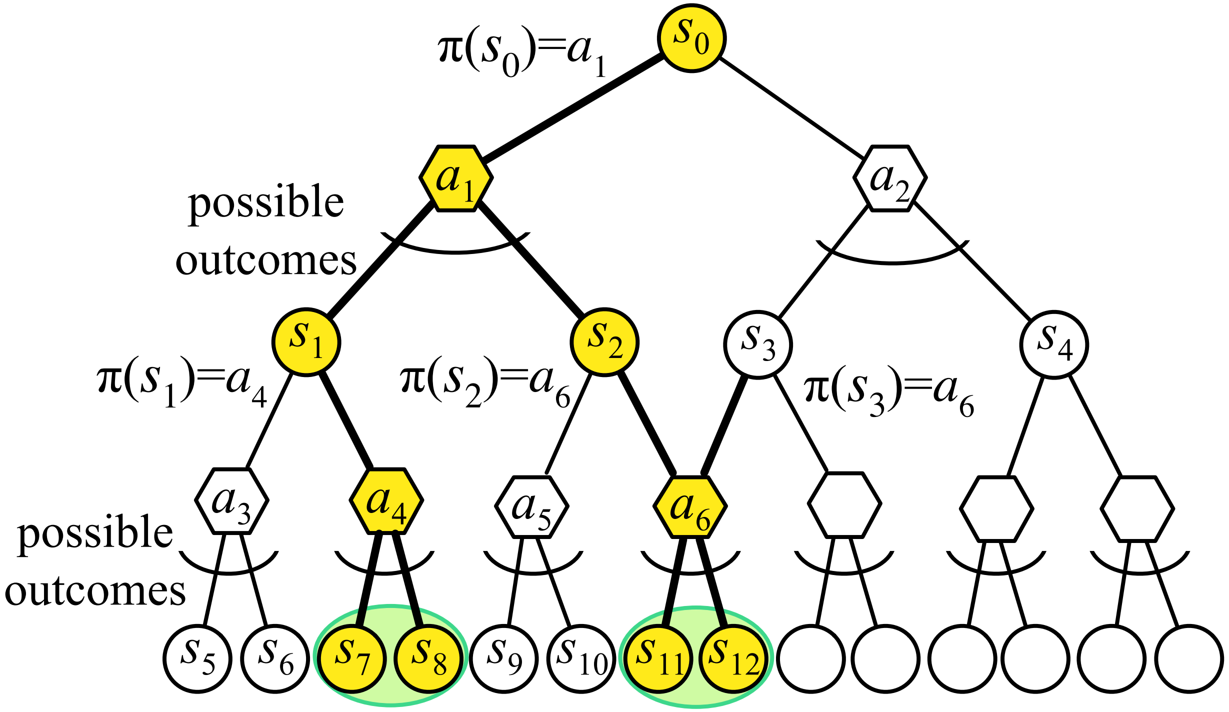
Bellman-Update(s)

- **for every** $a \in$ Applicable(s) **do**
  - $Q(s,a) \leftarrow \sum_{s' \in \gamma(s,a)} \Pr(s'|s,a)\,[\text{cost}(s,a,s') + V(s')]$
- $V(s) \leftarrow \min_a Q(s,a)$
- $\pi(s) \leftarrow \text{argmin}_a Q(s,a)$



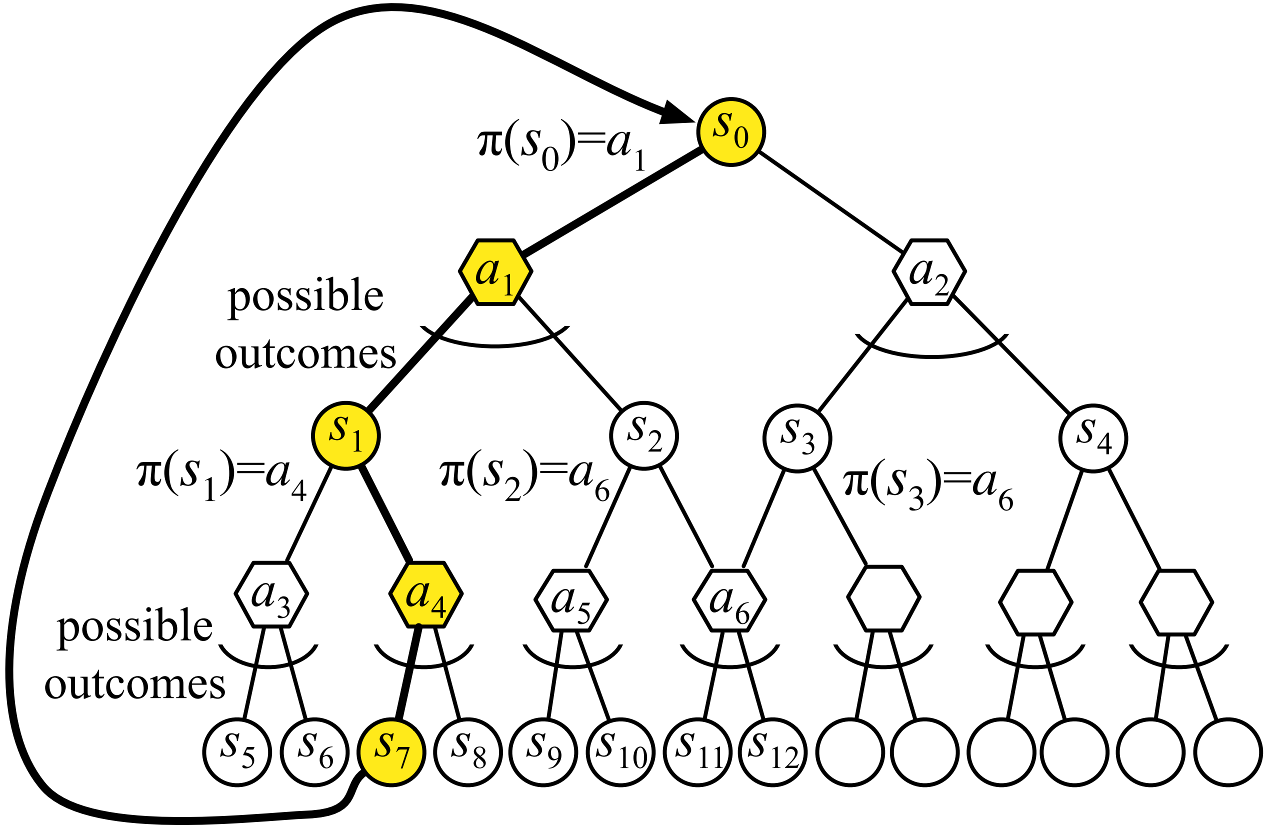
LAO* (Basic Idea)

- LAO*: generalization of AO* for cyclic MDPs
- $\mathit{Fringe}(s_0,\pi) = \mathit{leaves}(s_0,\pi) \setminus S_g = \{s_7, s_8, s_{11}, s_{12}\}$
  - Expand one of them, e.g., s_7



LAO* (Basic Idea)

- LAO*: generalization of AO* for cyclic MDPs
- *Fringe*(s_0,π) = *leaves*$(s_0,\pi) \setminus S_g = \{s_7, s_8, s_{11}, s_{12}\}$
  - Expand one of them, e.g., s_7
- Do value iteration on the cyclic part
  - Stop if either
    - V stops changing very much
    - for some s, $\pi(s)$ changes to go to other states



LAO*

may be cyclic

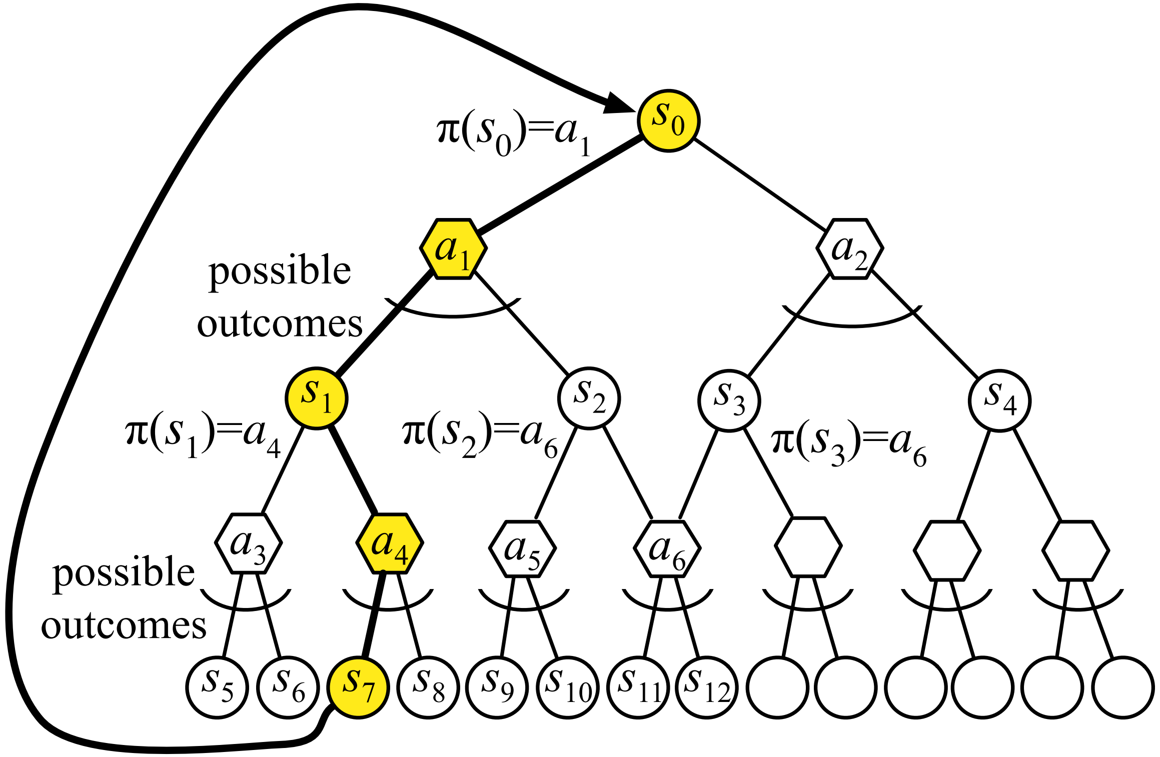
LAO* (Σ, s_0, S_g, V_0)

Same as AO*

- $\pi \leftarrow \emptyset$
- $Envelope \leftarrow \{s_0\}$
- $V(s_0) \leftarrow V_0(s_0)$
- **while** $Fringe(s_0,\pi) \neq \emptyset$ **do**
  - select a state $s \in Fringe(s_0,\pi)$
  - 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

Bellman-Update(s)

- **for every** $a \in$ Applicable(s) **do**
  - $Q(s,a) \leftarrow \sum_{s' \in \gamma(s,a)} \Pr(s'|s,a)\,[\text{cost}(s,a,s') + V(s')]$
- $V(s) \leftarrow \min_a Q(s,a)$
- $\pi(s) \leftarrow \text{argmin}_a Q(s,a)$



LAO-Update(s)

s's ancestors in π

- $Z \leftarrow \{s\} \cup \{s' \in Envelope \mid s \in \hat{\gamma}(s',\pi)\}$
- **until** new states are added to $Fringe(s_0,\pi)$ or $residual \leq \eta$ **do**
  - **for each** $s \in Z$ **do**
    - Bellman-Update(s)

Value iteration, restricted to Z

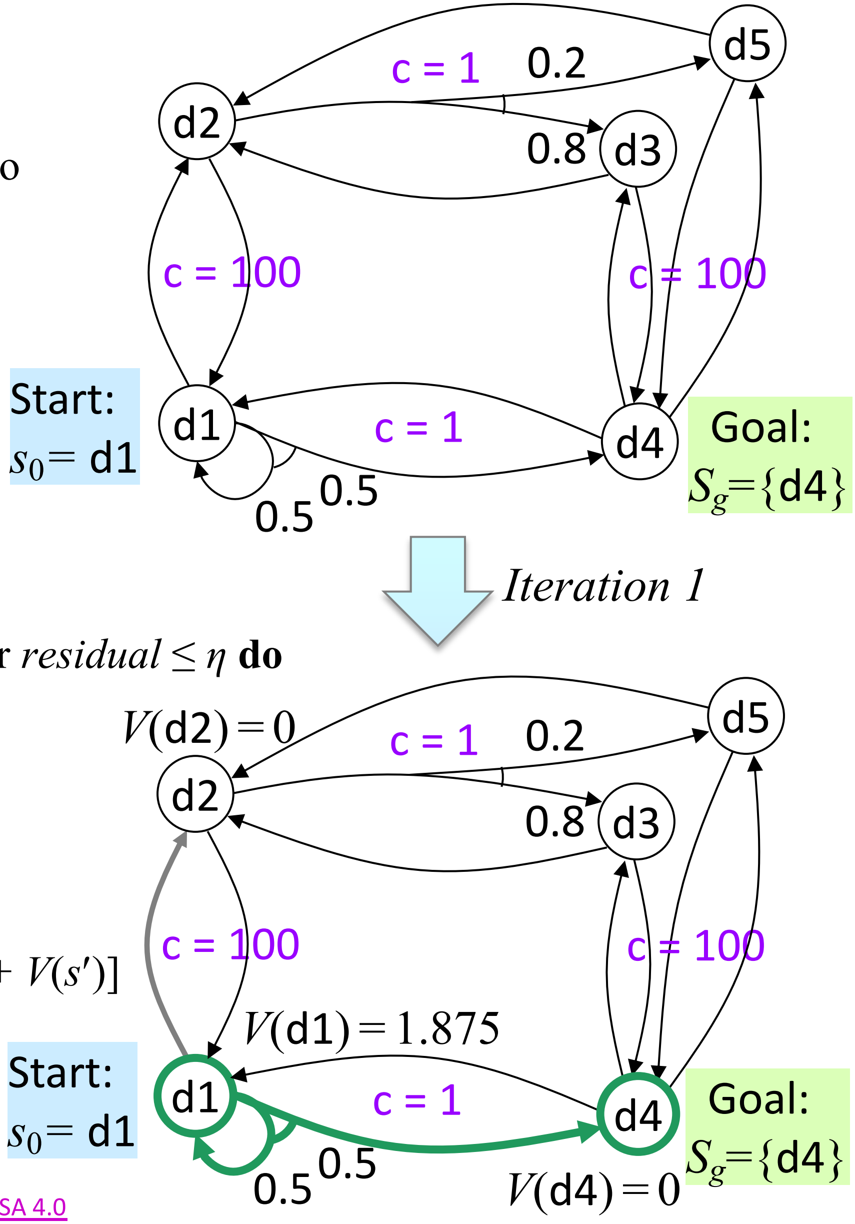
Example 1

LAO* (Σ, s_0, S_g, V_0)
- $\pi \leftarrow \emptyset$
- $Envelope \leftarrow \{s_0\}$
- $V(s_0) \leftarrow V_0(s_0)$
- **while** $Fringe(s_0,\pi) \neq \emptyset$ **do**
  - select a state $s \in Fringe(s_0,\pi)$
  - 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

LAO-Update(s)
- $Z \leftarrow \{s\} \cup \{s' \in Envelope \mid s \in \hat{\gamma}(s',\pi)\}$
- **until** new states are added to $Fringe(s_0,\pi)$ or $residual \leq \eta$ **do**
  - **for each** $s \in Z$ **do**
    - Bellman-Update(s)

Bellman-Update(s)
- **for every** $a \in$ Applicable(s) **do**
  - $Q(s,a) \leftarrow \sum_{s' \in \gamma(s,a)} \Pr(s'|s,a)\,[\mathrm{cost}(s,a,s') + V(s')]$
- $V(s) \leftarrow \min_a Q(s,a)$
- $\pi(s) \leftarrow \mathrm{argmin}_a Q(s,a)$

$\eta = 0.25$; $V_0(s) = 0$ for all s



$\pi = \emptyset$; $Envelope = \{$d1$\}$; $V(s_0) = 0$

Iteration 1:
- $\pi = \emptyset$, so $Fringe(s_0,\pi) = \{s_0\} = \{$d1$\}$
- select s = d1
- Applicable(d1) = {m12, m14}
- add d2 to $Envelope$; V(d2) ← 0
- add d4 to $Envelope$; V(d4) ← 0
- Call LAO-Update(d1)
  - π is empty, so $Z = \{$d1$\}$
  - Call Bellman-update(d1):
    - $v_{old} = V$(d1) = 0
    - Q(d1,m12) = 100 + 0 = 100
    - Q(d1,m14) = 1 + (½(0) + ½(0)) = 1
    - V(d1) = 1; π(d1) = m14
  - $r = |V$(d1) – 0$| = 1$
  - Keep iterating until $r \leq 0.2$
    - V(d1) = 1.875; $r = 0.0625$

Iteration 2:
- $Fringe(s_0,\pi) = \emptyset$, so return π = {(d1,m14)}

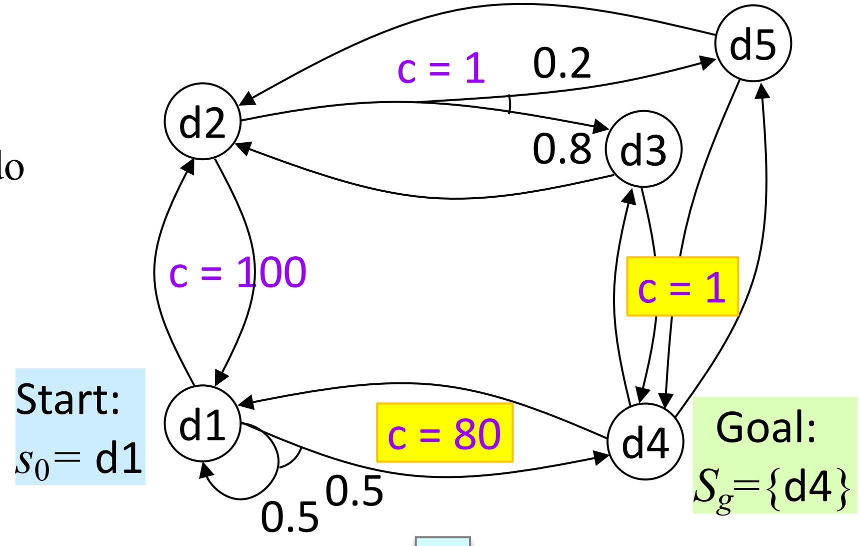
Example 2

LAO* (Σ, s_0, S_g, V_0)
    $\pi \leftarrow \emptyset$
    $Envelope \leftarrow \{s_0\}$
    $V(s_0) \leftarrow V_0(s_0)$
    while $Fringe(s_0,\pi) \neq \emptyset$ **do**
        select a state $s \in Fringe(s_0,\pi)$
        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

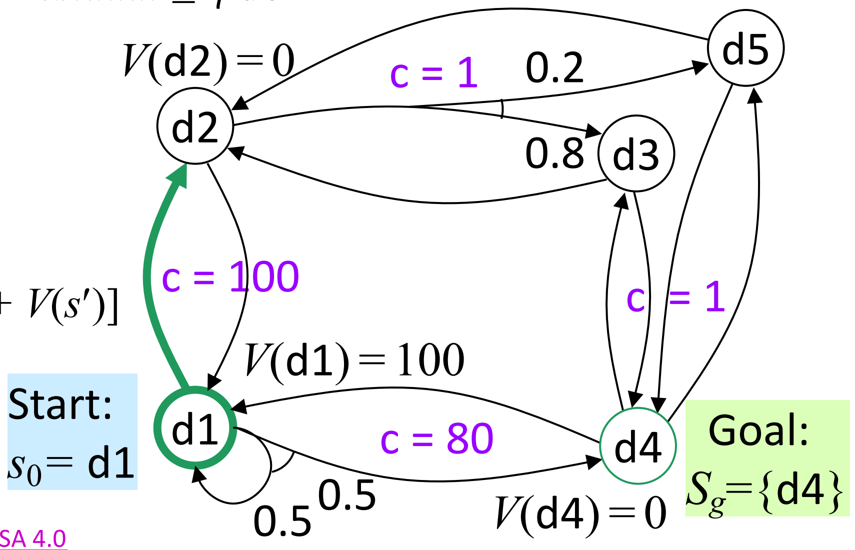
LAO-Update(s)
    $Z \leftarrow \{s\} \cup \{s' \in Envelope \mid s \in \hat{\gamma}(s',\pi)\}$
    until new states are added to $Fringe(s_0,\pi)$ or $residual \leq \eta$ **do**
        for each $s \in Z$ **do**
            Bellman-Update(s)

Bellman-Update(s)
    for every $a \in$ Applicable(s) **do**
        $Q(s,a) \leftarrow \sum_{s' \in \gamma(s,a)} \Pr(s'|s,a)\,[\mathrm{cost}(s,a,s') + V(s')]$
    $V(s) \leftarrow \min_a Q(s,a)$
    $\pi(s) \leftarrow \mathrm{argmin}_a Q(s,a)$

$\eta = 0.25$; $V_0(s) = 0$ for all s



Iteration 1



$\pi = \emptyset$; $Envelope = \{\text{d1}\}$; $V(s_0) = 0$

Iteration 1:

- $\pi = \emptyset$, so $Fringe(s_0,\pi) = \{s_0\} = \{\text{d1}\}$
- select s = d1; Applicable(d1) = {m12, m14}
- add d2 to $Envelope$; $V(\text{d2}) \leftarrow 0$
- add d4 to $Envelope$; $V(\text{d4}) \leftarrow 0$
- Call LAO-Update(d1)
    - $\pi = \emptyset$, so $Z = \{\text{d1}\}$
    - *loop iteration 1:*
        - call Bellman-update(d1):
            - $v_{old} = V(\text{d1}) = 0$
            - $Q(\text{d1,m12}) = 100 + 0 = 100$
            - $Q(\text{d1,m14}) = 80 + (½(0) + ½(0)) = 80$
            - $V(\text{d1}) = 80$; $\pi(\text{d1}) = \text{m14}$
            - *return* $|80 - 0| = 80$
        - call Bellman-update(d1) again:
            - $v_{old} = V(\text{d1}) = 80$
            - $Q(\text{d1,m12}) = 100 + 0 = 100$
            - $Q(\text{d1,m14}) = 80 + (½(80) + ½(0)) = 120$
            - $V(\text{d1}) = 100$; $\pi(\text{d1}) = \text{m12}$
            - *return* $|100 - 80| = 20$
    - new state d2 in $Fringe(s_0,\pi)$
- LAO-Update returns; *more on next page*

Example 2 continued

LAO* (Σ, s_0, S_g, V_0)

$\pi \leftarrow \emptyset$

Envelope $\leftarrow \{s_0\}$

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

while *Fringe*$(s_0,\pi) \neq \emptyset$ **do**

select a state $s \in$ *Fringe*(s_0,π)

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

LAO-Update(s)

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

until new states are added to *Fringe*(s_0,π) or *residual* $\leq \eta$ **do**

for each $s \in Z$ **do**

Bellman-Update(s)

Bellman-Update(s)

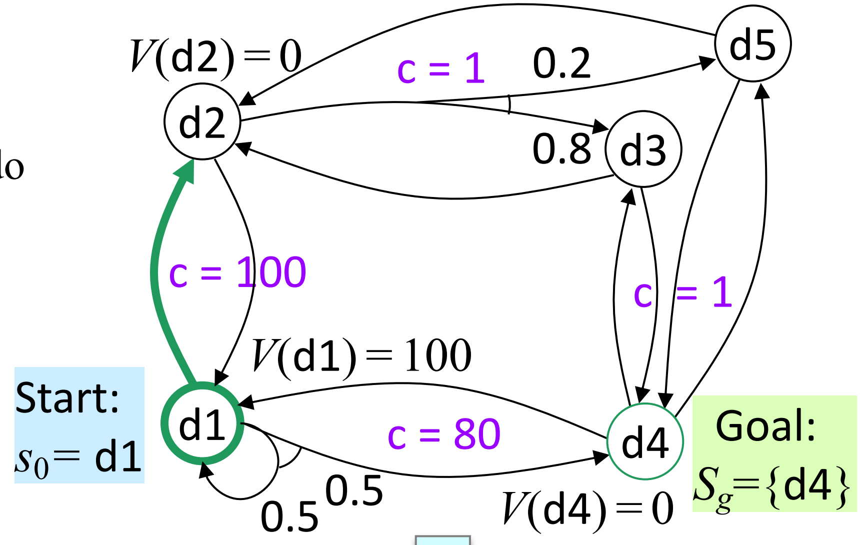
for every $a \in$ Applicable(s) **do**

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

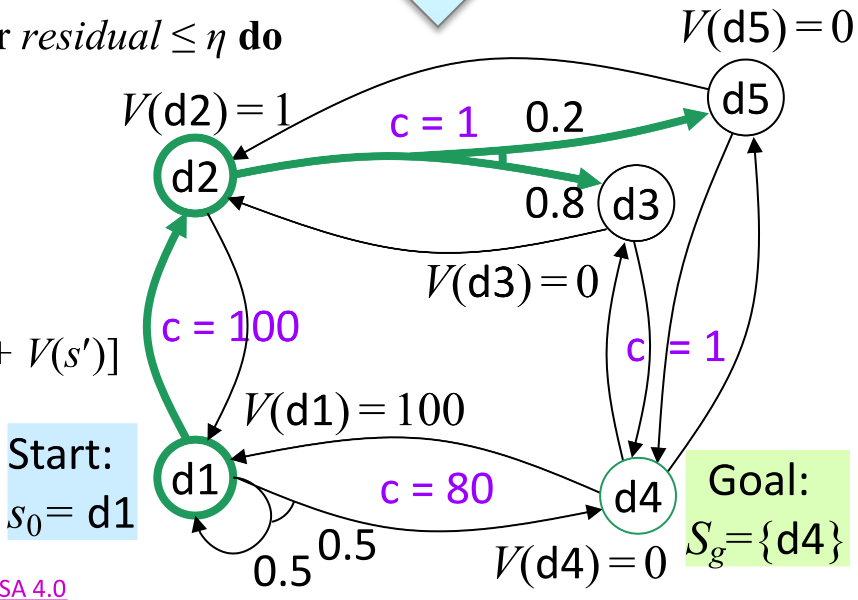
$V(s) \leftarrow \min_a Q(s,a)$

$\pi(s) \leftarrow \text{argmin}_a Q(s,a)$

$\eta = 0.25$; $V_0(s) = 0$ for all s



Iteration 2



Iteration 2:

$\pi = \{(\text{d1},\text{m12})\}$, so *Fringe*$(s_0,\pi) = \{\text{d2}\}$

select s = d2; Applicable(d2) = {m21, m23}

add d3, d5 to *Envelope*; $V(\text{d3}) = V(\text{d5}) = 0$

Call LAO-Update(d2)

$Z = \{\text{d2}\} \cup \{\text{d1}\}$

loop iteration 1:

call Bellman-update(d1):

$v_{old} = 100$

$Q(\text{d1},\text{m12}) = 100 + 0 = 100$

$Q(\text{d1},\text{m14}) = 80 + (½(80) + ½(0)) = 120$

$V(\text{d1}) = 100$; $\pi(\text{d1}) = \text{m12}$

return $|100 - 100| = 0$

call Bellman-update(d2):

$v_{old} = 0$

$Q(\text{d2},\text{m21}) = 100 + 100 = 200$

$Q(\text{d1},\text{m23}) = 1 + (.8(0) + .2(0)) = 1$

$V(\text{d2}) = 1$; $\pi(\text{d2}) = \text{m23}$

$r = \max(|1 - 0|) = 1$

new states d3,d5 in *Fringe*(s_0,π)

LAO-Update returns

After more iterations, LAO* *eventually returns*

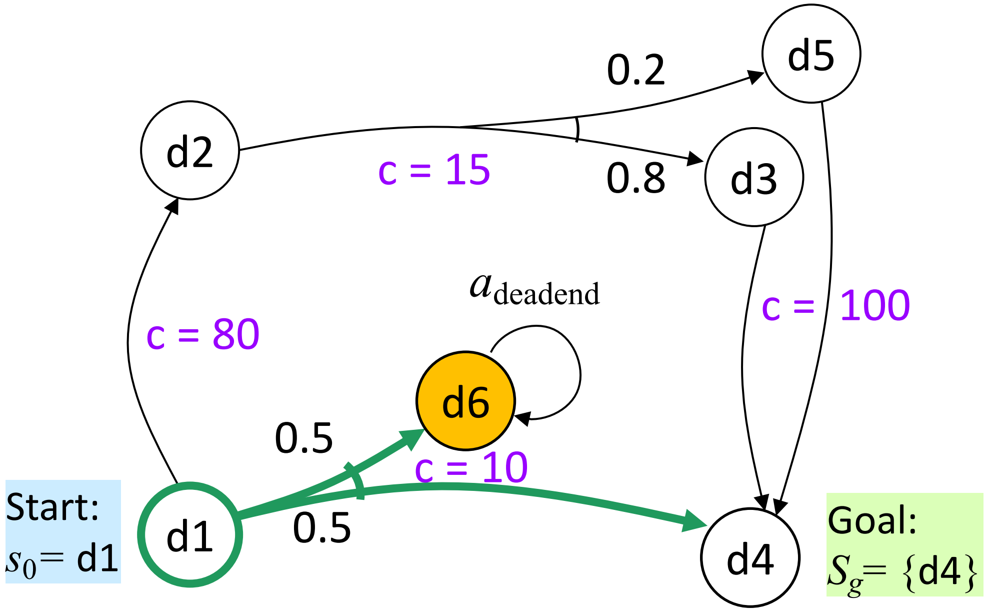
$\pi = \{(\text{d1},\text{m12}),(\text{d2},\text{m23}),(\text{d3},\text{m34}),(\text{d5},\text{m54})\}$

Dead Ends

LAO* (Σ, s_0, S_g, V_0)
  $\pi \leftarrow \emptyset$
  $Envelope \leftarrow \{s_0\}$
  $V(s_0) \leftarrow V_0(s_0)$
  while $Fringe(s_0,\pi) \neq \emptyset$ **do**
    select a state $s \in Fringe(s_0,\pi)$
    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

LAO-Update(s)
  $Z \leftarrow \{s\} \cup \{s' \in Envelope \mid s \in \hat{\gamma}(s',\pi)\}$
  until new states are added to $Fringe(s_0,\pi)$ or $residual \leq \eta$ **do**
    for each $s \in Z$ **do**
      Bellman-Update(s)

Bellman-Update(s)
  for every $a \in$ Applicable(s) **do**
    $Q(s,a) \leftarrow \sum_{s' \in \gamma(s,a)} \Pr(s'|s,a)\,[\text{cost}(s,a,s') + V(s')]$
  $V(s) \leftarrow \min_a Q(s,a)$
  $\pi(s) \leftarrow \text{argmin}_a Q(s,a)$



At dead-end states, add a dummy action a_{deadend}

- $\gamma(s, a_{\text{deadend}}) = s$
- cost = a constant > 0

Digression: Monte Carlo rollouts

- Multi-arm bandit problem: statistical model of sequential experiments
  - Name derived from *one-armed bandit* (slot machine)
- Multiple actions $a_1, a_2, \ldots, a_n$, e.g., a_n = play machine i
- Each a_i provides a reward from an unknown probability distribution p_i
  - Assume every p_i is *stationary*
    - same every time, regardless of history (not true for real slot machines)
  - 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 a less-familiar action in hopes that it might produce a higher reward

Credit: Billy Hicks, CC BY-SA 4.0

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
  - $t = \sum_a n(a)$
  - $Q(a) = r(a) + \sqrt{2(\ln t)/n(a)}$

> UCB algorithm
>
> if there are any untried actions:
>
> $\tilde{a} \leftarrow$ any untried action
>
> else: $\tilde{a} \leftarrow \text{argmax}_a\, Q(a)$
>
> perform $\tilde{a}$ and get reward
>
> update $r(\tilde{a})$, $n(\tilde{a})$, t, $Q(\tilde{a})$

- Theorem (given some assumptions):

  As $t \to \infty$, $\text{argmax}_{\tilde{a} \in A}\, Q(a) \to$ optimal choice

Credit: Billy Hicks, CC BY-SA 4.0

UCB (Upper Confidence Bound) Algorithm

- For each action a, let
  - $r(a)$ = average reward you've gotten from a
  - $n(a)$ = number of times you've tried a
  - $t = \sum_a n(a)$
  - $Q(a) = r(a) + \sqrt{2(\ln t)/n(a)}$

> UCB:
>
> if there are any untried actions:
>
> $\tilde{a} \leftarrow$ any untried action
>
> else:
>
> $\tilde{a} \leftarrow \text{argmax}_a\, Q(a)$
>
> perform $\tilde{a}$ and get reward
>
> update $r(\tilde{a})$, $n(\tilde{a})$, t, $Q(\tilde{a})$

- Example: actions a1, a2, a3
  - a1 pays every other time
  - a2 pays every third time
  - a3 never pays

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				

Poll: what action will UCB choose the 11th time?

Monte Carlo Tree Search

root

max search depth

number of rollouts of $\tilde{a}$

MCTS(s_r, h)

until *termination condition* **do**

Select an open state $s \in \widehat{\gamma}(s_r, \pi)$

Choose an action $\tilde{a} \in \mathit{Applicable}(s)$

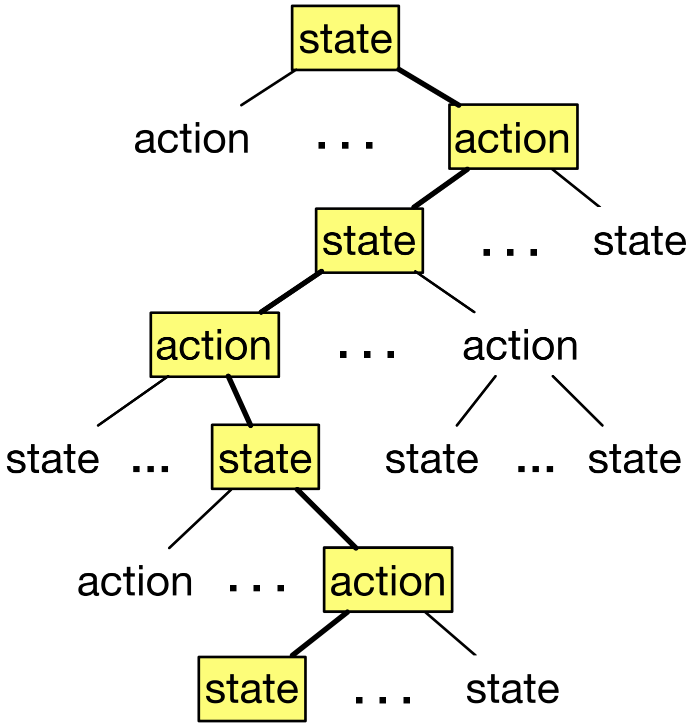
$Q(s, \tilde{a}) \leftarrow [n(s, \tilde{a}) \times Q(s, \tilde{a}) + [\text{Rollout}(s, \tilde{a}, \pi_r, h)]/(1 + n(s, \tilde{a}))$

$n(s, \tilde{a}) \leftarrow n(s, \tilde{a}) + 1$

$V(s) \leftarrow \min_a \{Q(s, a)\}$

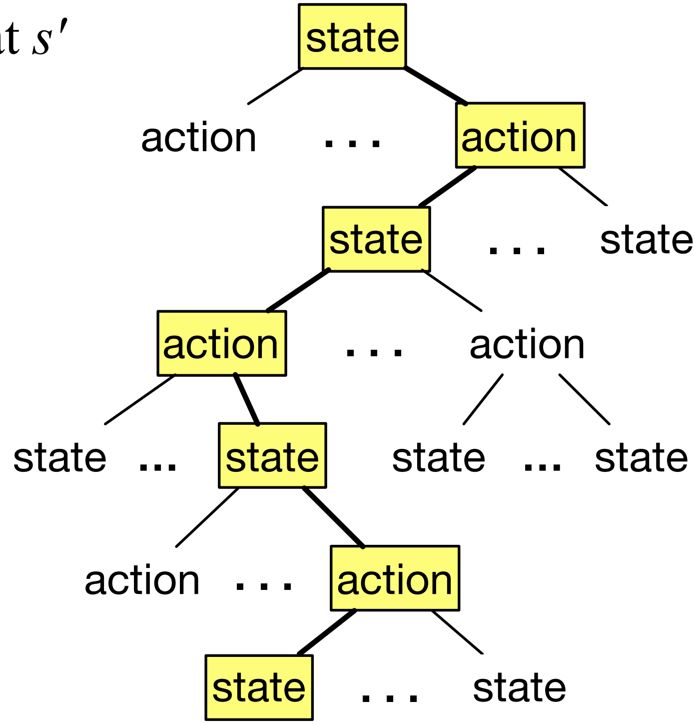
$\pi(s) \leftarrow \operatorname{argmin}_a \{Q(s, a)\}$

Update all ancestors of s in $\widehat{\gamma}(s', \pi)$



UCT Algorithm

- 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



UCB:

if there are untried actions:

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

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

perform $\tilde{a}$ and get reward

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

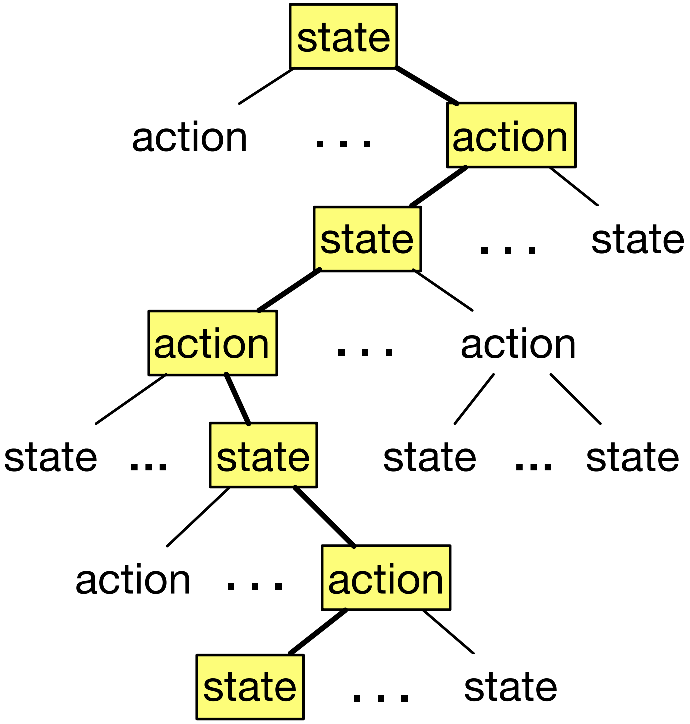
- Statistics for each action a:
  - $r(a)$ = average reward
  - $n(a)$ = number of times you've tried a
  - $t = \sum_a n(a)$
  - $Q(a) = r(a) + \sqrt{2(\ln t)/n(a)}$

UCT Algorithm

max search depth

UCT(s, h)
  until *termination condition* **do**
    UCT-Rollout(s, h)

UCT-Rollout(s, h)
  if $s \in S_g$ **then** return 0
  if $h = 0$ **then** return $V_0(s)$
  if $s \notin Envelope$ **then**
    add s to *Envelope*
    $n(s) \leftarrow 0$ — like t in UCB
    foreach $a \in Applicable(s)$ **do**
      $Q(s,a) \leftarrow 0;\ n(s,a) \leftarrow 0$ — like $Q(a)$, $n(a)$ in UCB
  $\tilde{a} \leftarrow$ Select(s)
  $s' \leftarrow$ Sample$(s, \tilde{a})$ — choose s' with probability $\Pr(s' \mid s, \tilde{a})$
  $cost\text{-}rollout \leftarrow \text{cost}(s, \tilde{a}, s') + \text{UCT-Rollout}(s', h-1)$
  $Q(s,\tilde{a}) \leftarrow [n(s,\tilde{a}) \times Q(s,\tilde{a}) + cost\text{-}rollout]/(1 + n(s,\tilde{a}))$
  $\pi(s) \leftarrow \text{argmax}\{Q(s,a) \mid a \in Applicable(s)\}$
  $n(s) \leftarrow n(s) + 1$
  $n(s,\tilde{a}) \leftarrow n(s,\tilde{a}) + 1$
  return *cost-rollout*



UCB:
  if there are untried actions:
    $\tilde{a} \leftarrow$ any untried action
  else: $\tilde{a} \leftarrow \text{argmax}_a\, Q(a)$
  perform $\tilde{a}$ and get reward
  update $r(\tilde{a})$, $n(\tilde{a})$, t, $Q(\tilde{a})$

- Statistics for each action a:
  - $r(a)$ = average reward
  - $n(a)$ = number of times you've tried a
  - $t = \sum_a n(a)$
  - $Q(a) = r(a) + \sqrt{2(\ln t)/n(a)}$

ln, I think

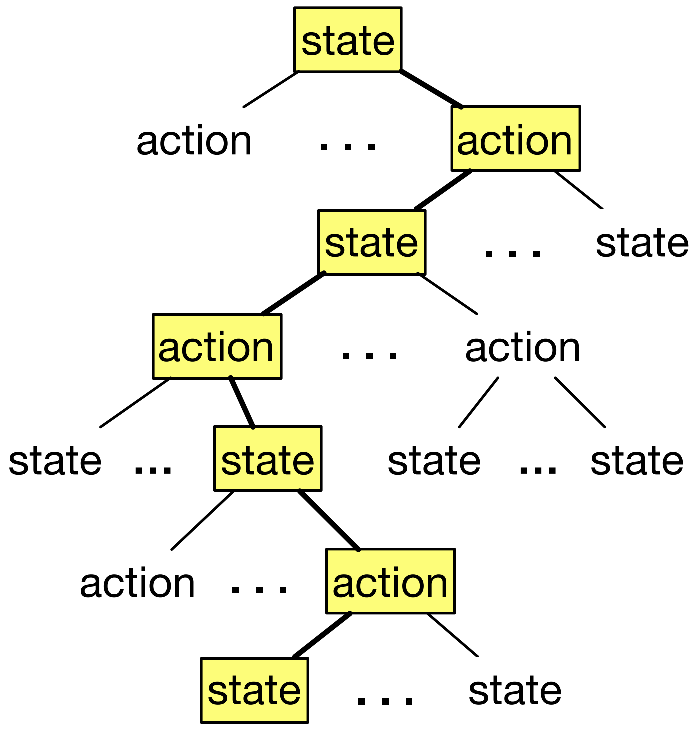
$$\text{Select}(s) = \begin{cases} \text{any } a \in Untried(s) & \text{if } Untried(s) \neq \varnothing \\ \text{argmin}_a\{Q(s,a) - C \times [\log(n(s))/n(s,a)]^{1/2}\} & \text{if not,} \end{cases}$$

Using UCT Offline

UCT(s, h)
  until *termination condition* **do**
    UCT-Rollout(s, h)

UCT-Rollout(s, h)
  if $s \in S_g$ **then** return 0
  if $h = 0$ **then** return $V_0(s)$
  if $s \notin Envelope$ **then**
    add s to *Envelope*
    $n(s) \leftarrow 0$
    foreach $a \in Applicable(s)$ **do**
      $Q(s, a) \leftarrow 0;\ n(s, a) \leftarrow 0$
  $\tilde{a} \leftarrow$ Select(s)
  $s' \leftarrow$ Sample$(s, \tilde{a})$
  $cost\text{-}rollout \leftarrow \text{cost}(s, \tilde{a}, s') + \text{UCT-Rollout}(s', h-1)$
  $Q(s, \tilde{a}) \leftarrow [n(s, \tilde{a}) \times Q(s, \tilde{a}) + cost\text{-}rollout]/(1 + n(s, \tilde{a}))$
  $\pi(s) \leftarrow \text{argmax}\{Q(s, a) \mid a \in Applicable(s)\}$
  $n(s) \leftarrow n(s) + 1$
  $n(s, \tilde{a}) \leftarrow n(s, \tilde{a}) + 1$
  return *cost-rollout*

- Suppose we do n calls to UCT-Rollout
- As $n \to \infty$, π converges to optimal
  - Problem: finding optimal π may take many iterations



$$\text{Select}(s) = \begin{cases} \text{any } a \in Untried(s) & \text{if } Untried(s) \neq \varnothing \\ \text{argmin}_a \{Q(s, a) - C \times [\log(n(s))/n(s, a)]^{1/2}\} & \text{if not,} \end{cases}$$

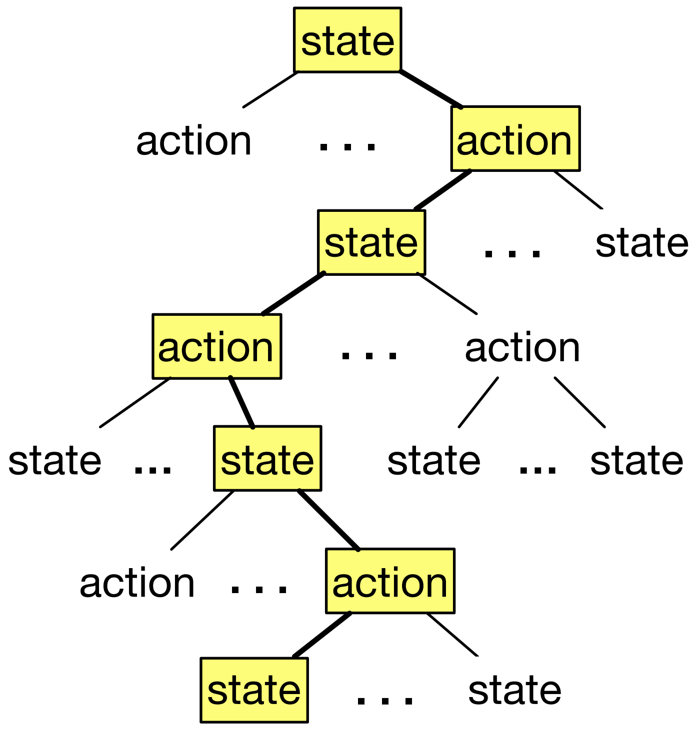
Using UCT Online

UCT(s, h)
until *termination condition* **do**
UCT-Rollout(s, h)

UCT-Rollout(s, h)
if $s \in S_g$ **then** return 0
if $h = 0$ **then** return $V_0(s)$
if $s \notin Envelope$ **then**
add s to *Envelope*
$n(s) \leftarrow 0$
foreach $a \in Applicable(s)$ **do**
$Q(s,a) \leftarrow 0;\ n(s,a) \leftarrow 0$

$\tilde{a} \leftarrow$ Select(s)
$s' \leftarrow$ Sample$(s, \tilde{a})$
$cost\text{-}rollout \leftarrow \text{cost}(s, \tilde{a}, s') + \text{UCT-Rollout}(s', h-1)$
$Q(s,\tilde{a}) \leftarrow [n(s,\tilde{a}) \times Q(s,\tilde{a}) + cost\text{-}rollout]/(1 + n(s,\tilde{a}))$
$\pi(s) \leftarrow \text{argmax}\{Q(s,a) \mid a \in Applicable(s)\}$
$n(s) \leftarrow n(s) + 1$
$n(s,\tilde{a}) \leftarrow n(s,\tilde{a}) + 1$
return *cost-rollout*

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



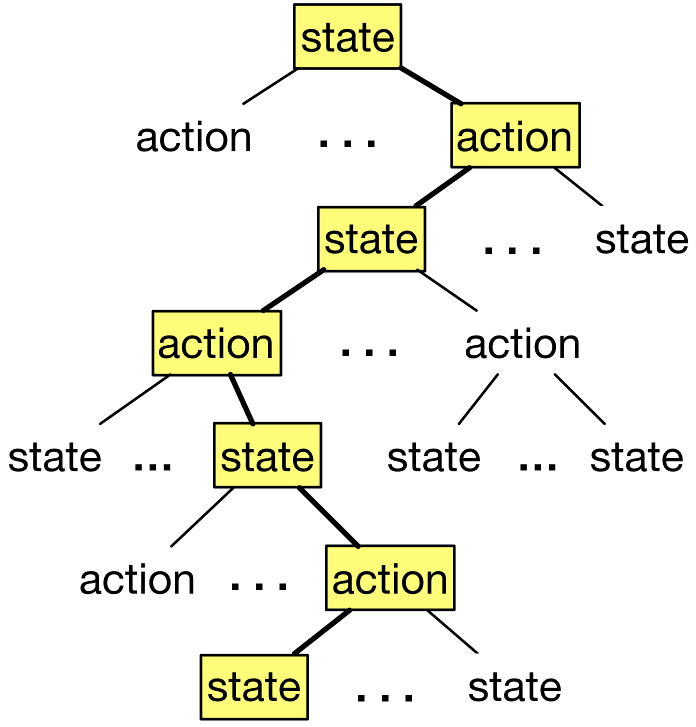
$$\text{Select}(s) = \begin{cases} \text{any } a \in Untried(s) & \text{if } Untried(s) \neq \varnothing \\ \text{argmin}_a\{Q(s,a) - C \times [\log(n(s))/n(s,a)]^{1/2}\} & \text{if not,} \end{cases}$$

Using UCT Online

UCT(s, h)
  until *termination condition* **do**
    UCT-Rollout(s, h)

UCT-Rollout(s, h)
  if $s \in S_g$ **then** return 0
  if $h = 0$ **then** return $V_0(s)$
  if $s \notin Envelope$ **then**
    add s to *Envelope*
    $n(s) \leftarrow 0$
    foreach $a \in Applicable(s)$ **do**
      $Q(s,a) \leftarrow 0;\ n(s,a) \leftarrow 0$
  $\tilde{a} \leftarrow$ Select(s)
  $s' \leftarrow$ Sample$(s, \tilde{a})$
  $cost\text{-}rollout \leftarrow \text{cost}(s, \tilde{a}, s') + \text{UCT-Rollout}(s', h-1)$
  $Q(s,\tilde{a}) \leftarrow [n(s,\tilde{a}) \times Q(s,\tilde{a}) + cost\text{-}rollout]/(1 + n(s,\tilde{a}))$
  $\pi(s) \leftarrow \text{argmax}\{Q(s,a) \mid a \in Applicable(s)\}$
  $n(s) \leftarrow n(s) + 1$
  $n(s,\tilde{a}) \leftarrow n(s,\tilde{a}) + 1$
  return *cost-rollout*

- Suppose Run-Lazy-Lookahead uses the same Lookahead procedure:
  - call UCT-Rollout(s,h) multiple times at current state s
  - return $\pi(s)$
- 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
  - Make Run-Lazy-Lookahead call UCT more frequently



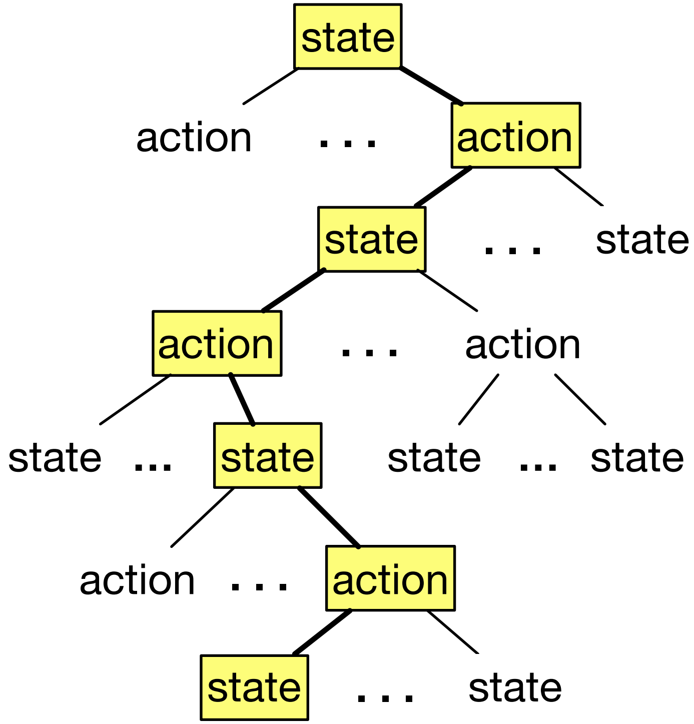
$$\text{Select}(s) = \begin{cases} \text{any } a \in Untried(s) & \text{if } Untried(s) \neq \varnothing \\ \text{argmin}_a\{Q(s,a) - C \times [\log(n(s))/n(s,a)]^{1/2}\} & \text{if not,} \end{cases}$$

Using UCT with a Simulator

UCT(s, h)
  until *termination condition* **do**
    UCT-Rollout(s, h)

UCT-Rollout(s, h)
  if $s \in S_g$ **then** return 0
  if $h = 0$ **then** return $V_0(s)$
  if $s \notin Envelope$ **then**
    add s to *Envelope*
    $n(s) \leftarrow 0$
    foreach $a \in Applicable(s)$ **do**
      $Q(s, a) \leftarrow 0;\ n(s, a) \leftarrow 0$
  $\tilde{a} \leftarrow$ Select(s)
  $s' \leftarrow$ Sample$(s, \tilde{a})$ simulate a; observe s'
  $cost\text{-}rollout \leftarrow \text{cost}(s, \tilde{a}, s') + \text{UCT-Rollout}(s', h-1)$
  $Q(s, \tilde{a}) \leftarrow [n(s, \tilde{a}) \times Q(s, \tilde{a}) + cost\text{-}rollout]/(1 + n(s, \tilde{a}))$
  $\pi(s) \leftarrow \text{argmax}\{Q(s, a) \mid a \in Applicable(s)\}$
  $n(s) \leftarrow n(s) + 1$
  $n(s, \tilde{a}) \leftarrow n(s, \tilde{a}) + 1$
  return *cost-rollout*

- Suppose you don't know the probabilities and costs
  - But you have a fast, accurate simulator for the environment
- Run UCT many times in the simulated environment
  - Learn state-transition probabilities, expected utilities



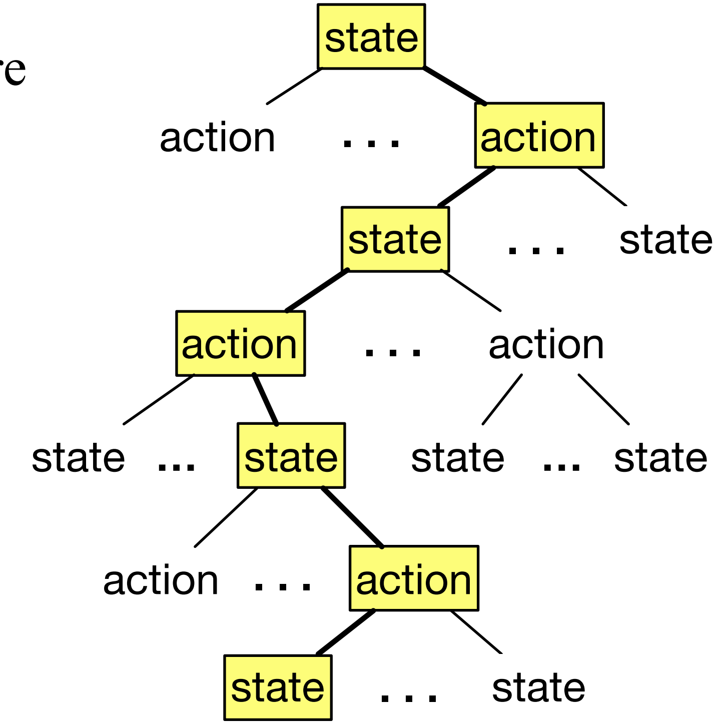
$$\text{Select}(s) = \begin{cases} \text{any } a \in Untried(s) & \text{if } Untried(s) \neq \varnothing \\ \text{argmin}_a\{Q(s,a) - C \times [\log(n(s))/n(s,a)]^{1/2}\} & \text{if not,} \end{cases}$$

Using UCT for Exploration

UCT(s, h)
  until *termination condition* **do**
    UCT-Rollout(s, h)

UCT-Rollout(s, h)
  if $s \in S_g$ **then** return 0
  if $h = 0$ **then** return $V_0(s)$
  if $s \notin Envelope$ **then**
    add s to *Envelope*
    $n(s) \leftarrow 0$
    foreach $a \in Applicable(s)$ **do**
      $Q(s, a) \leftarrow 0;\ n(s, a) \leftarrow 0$
  $\tilde{a} \leftarrow$ Select(s)
  $s' \leftarrow$ Sample$(s, \tilde{a})$ perform a; observe s'
  $cost\text{-}rollout \leftarrow \text{cost}(s, \tilde{a}, s') + \text{UCT-Rollout}(s', h-1)$
  $Q(s, \tilde{a}) \leftarrow [n(s, \tilde{a}) \times Q(s, \tilde{a}) + cost\text{-}rollout]/(1 + n(s, \tilde{a}))$
  $\pi(s) \leftarrow \text{argmax}\{Q(s, a) \mid a \in Applicable(s)\}$
  $n(s) \leftarrow n(s) + 1$
  $n(s, \tilde{a}) \leftarrow n(s, \tilde{a}) + 1$
  return *cost-rollout*

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



$$\text{Select}(s) = \begin{cases} \text{any } a \in Untried(s) & \text{if } Untried(s) \neq \varnothing \\ \text{argmin}_a\{Q(s,a) - C \times [\log(n(s))/n(s,a)]^{1/2}\} & \text{if not,} \end{cases}$$

UCT in Two-Player Games

- Generate Monte Carlo rollouts using a modified version of UCT
- Main differences:
  - Instead of accumulated cost, use heuristic evaluation function values
  - 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

Credit: Brian Jeffery Beggerly, CC BY 2.0

Summary

- MDPs and 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