16-350
Planning Techniques for Robotics

Planning under Uncertainty: Minimax Formulation

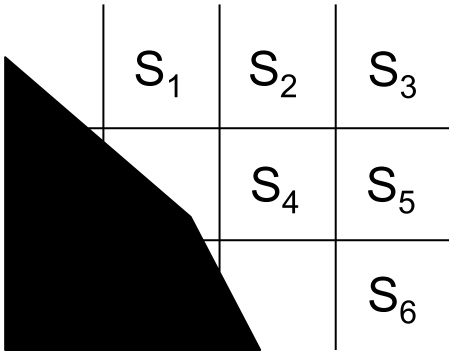
Maxim Likhachev

Robotics Institute

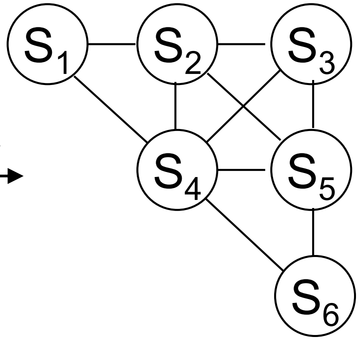
Carnegie Mellon University

Uncertainty in Robotics

- So far our planners assumed no uncertainty
  - execution is perfect



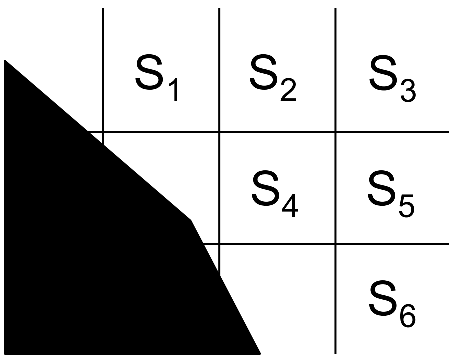
convert into a graph



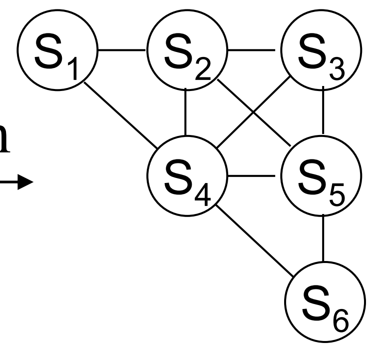
search the graph for a least-cost path from s_{start} to s_{goal}

Uncertainty in Robotics

- So far our planners assumed no uncertainty
  - execution is perfect



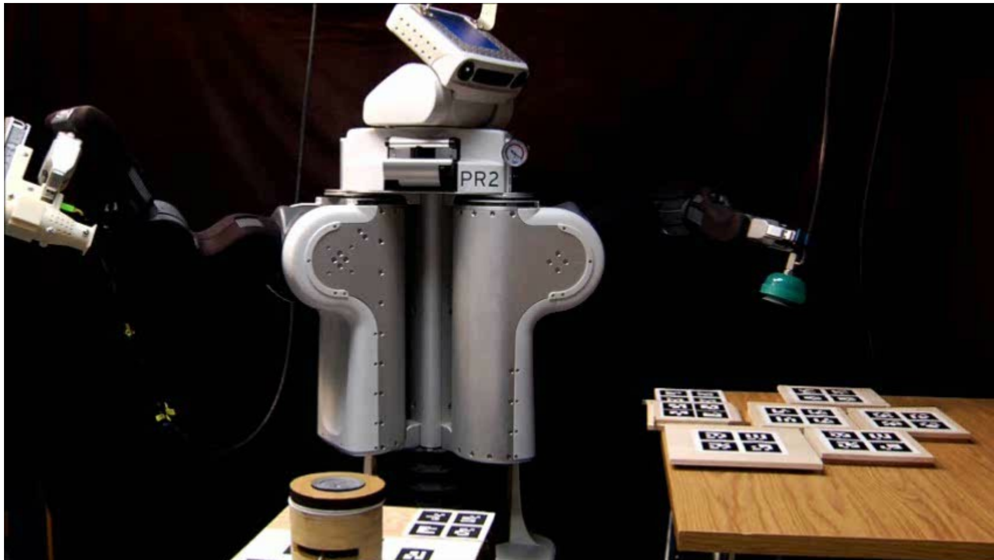
convert into a graph

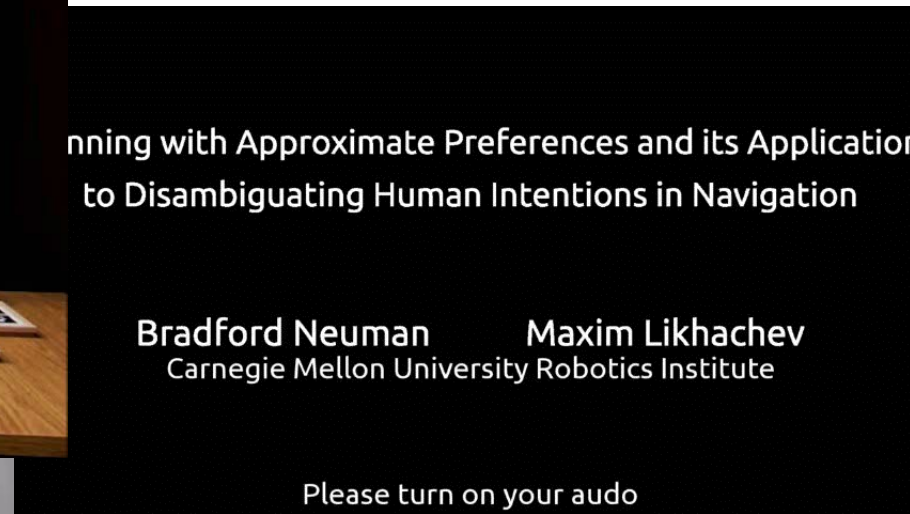


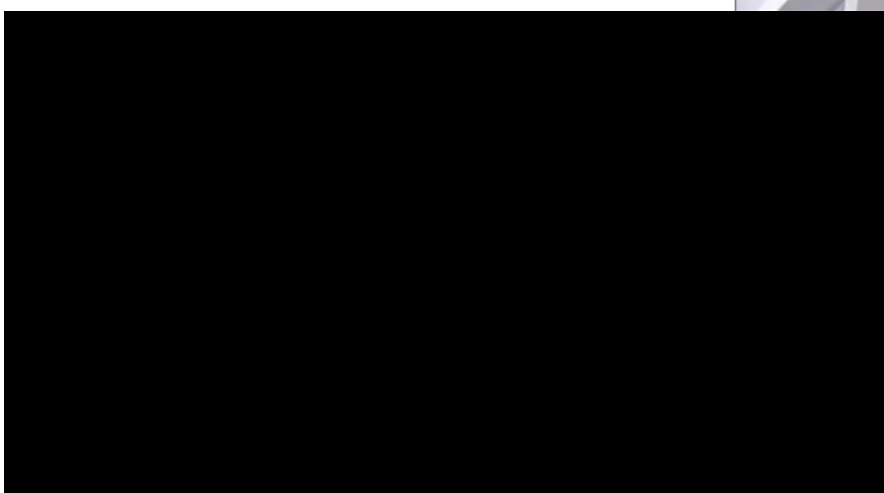
search the graph for a least-cost path from s_{start} to s_{goal}

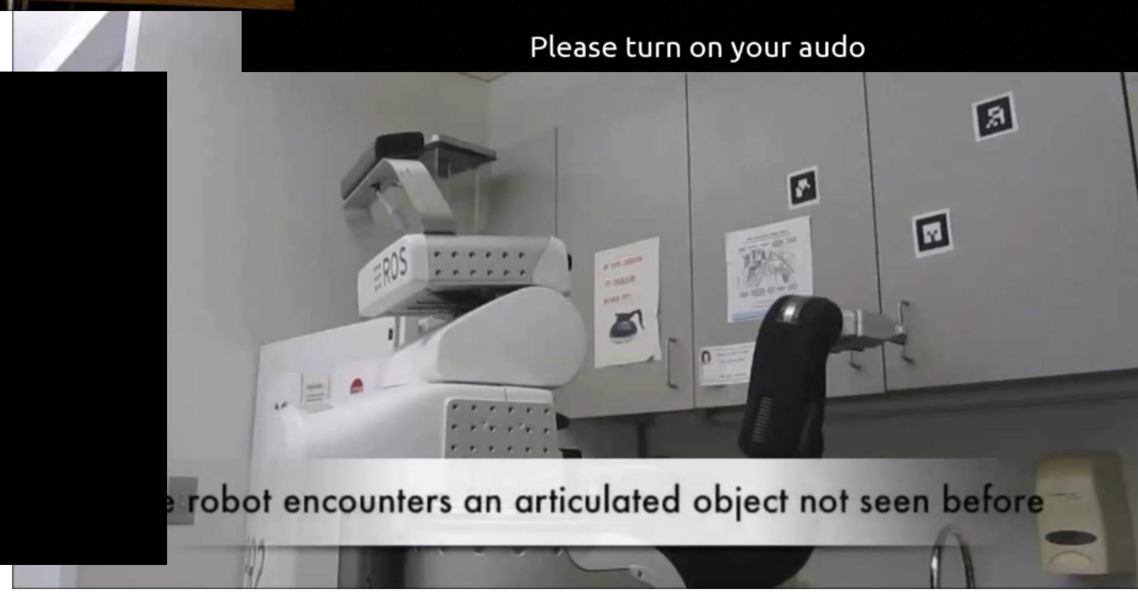
- Any deviations from the plan are dealt by re-planning
- Could be quite suboptimal and sometimes dangerous
  - planning a path along cliff does not take into account slippage
  - others examples???

Uncertainty in Robotics





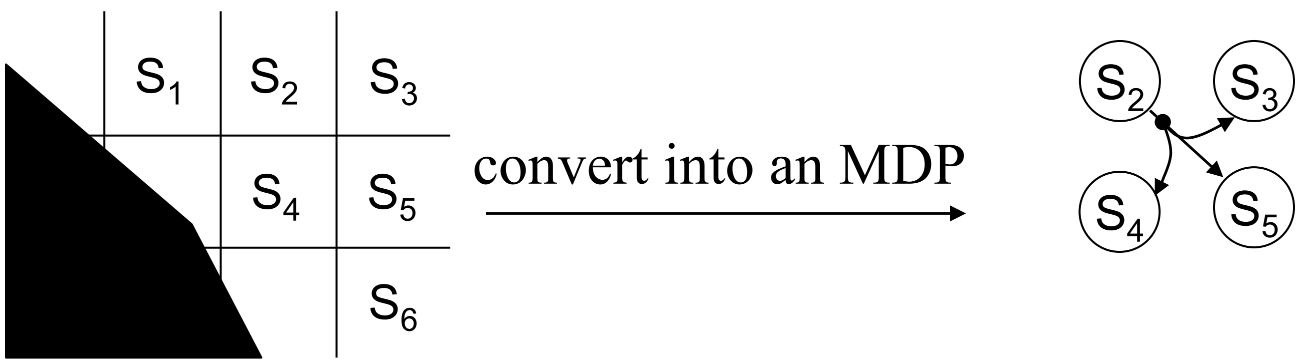




Uncertainty in Robotics

- Modeling uncertainty in execution <u>during planning</u>

Markov Decision Processes (MDP)

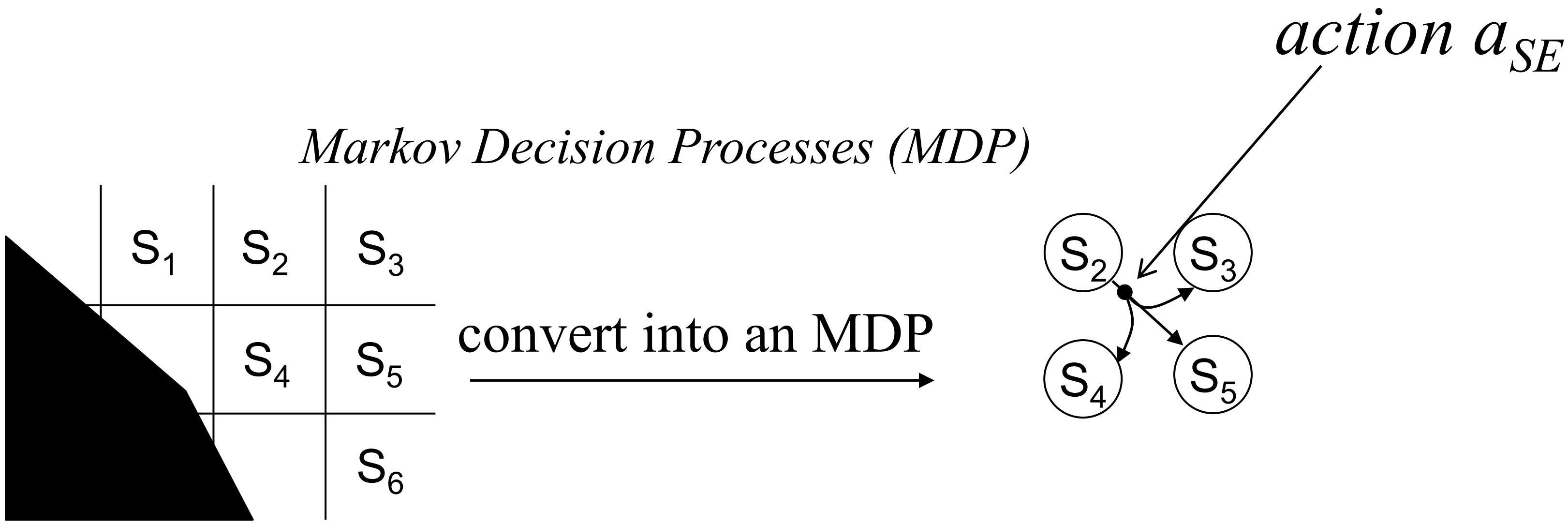


- at least one action in the graph has more than one outcome
- each outcome is associated with probability and cost

Uncertainty in Robotics

- Modeling uncertainty in execution <u>during planning</u>

Markov Decision Processes (MDP)



- at least one action in the graph has more than one outcome

- each outcome is associated with probability and cost

example: $s_3, s_4, s_5 \in succ(s_2, a_{SE})$,

$$P(s_5|a_{se},s_2) = 0.9, \quad c(s_2,a_{se},s_5) = 1.4$$
$$P(s_3|a_{se},s_2) = 0.05, \quad c(s_2,a_{se},s_3) = 1.0$$
$$P(s_4|a_{se},s_2) = 0.05, \quad c(s_2,a_{se},s_4) = 1.0$$

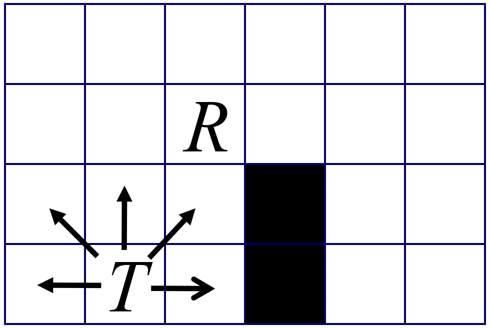
Moving along Cliff Example

- Example on the board

Moving-Target Search Example

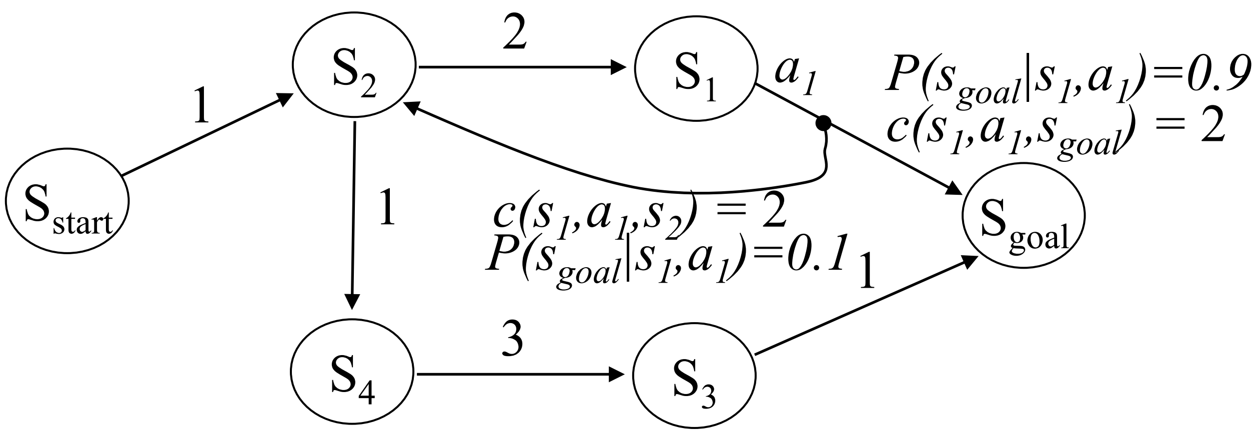
- Uncertainty in the target moves

- What is a state-space and action space?



Planning in MDPs

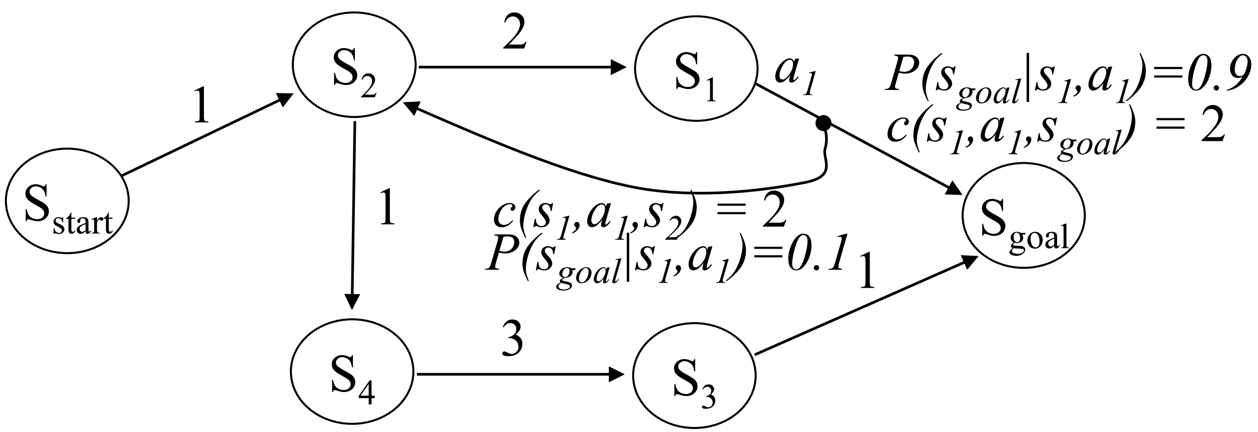
- What plan to compute?
  - Plan that minimizes the worst-case scenario (minimax plan)
  - Plan that minimizes the expected cost



- Without uncertainty, plan is a single path:
  a sequence of states (a sequence of actions)
- In MDPs, plan is a policy π:
  mapping from a state onto an action

Planning in MDPs

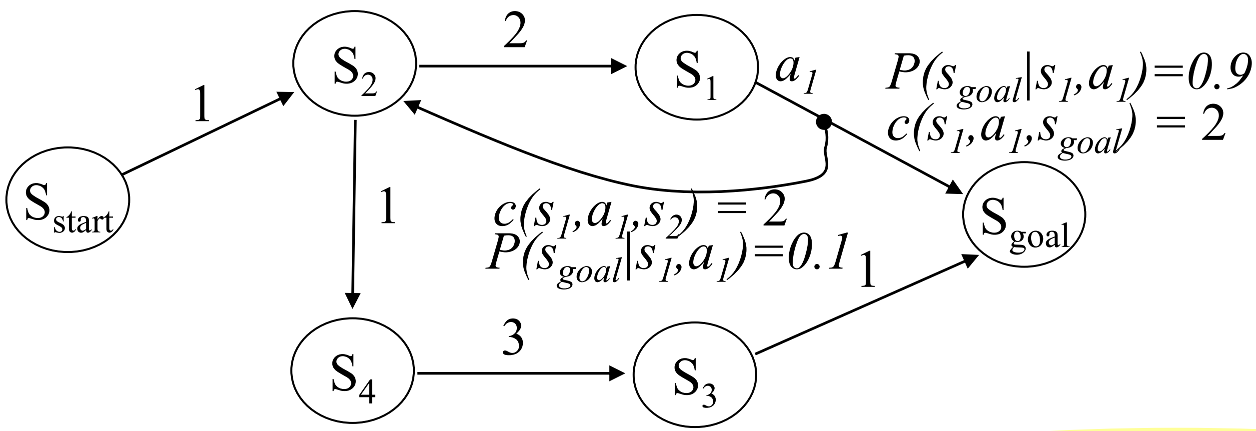
- What plan to compute?
  - Plan that minimizes the worst-case scenario (minimax plan)
  - Plan that minimizes the expected cost



- Without uncertainty, plan is a single path: a sequence of states (a sequence of actions)
- In MDPs, plan is a policy π: mapping from a state onto an action

Why?

Minimax Formulation



What is the best plan?

- Optimal policy π^*:
  minimizes the *worst* cost-to-goal
  $\pi^* = argmin_\pi \, max_{outcomes\ of\ \pi}\{cost\text{-}to\text{-}goal\}$

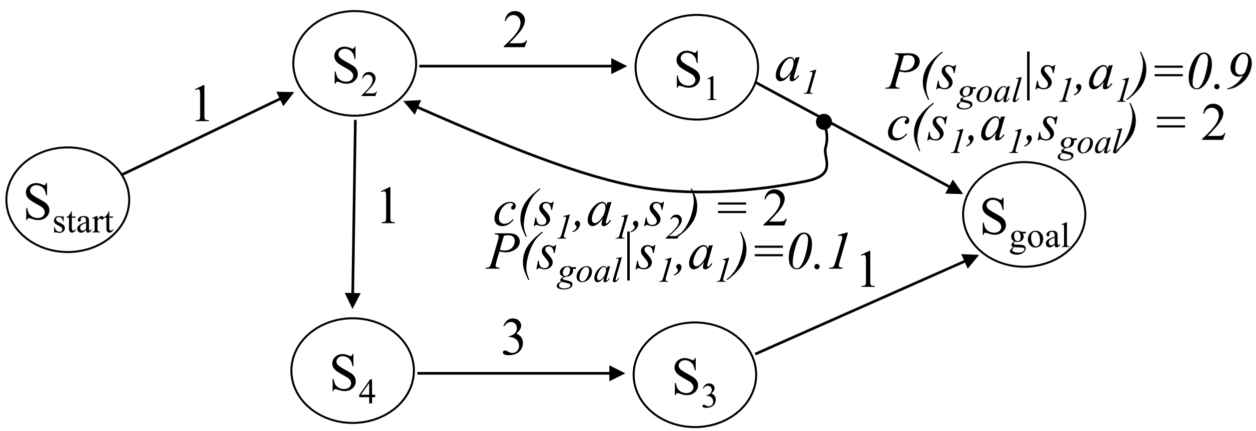
- worst cost-to-goal for π_1=(go through s_4) is:

  1+1+3+1 = 6

- worst cost-to-goal for π_2=(try to go through s_1) is:

  1+2+2+2+2+2+2 + … = ∞

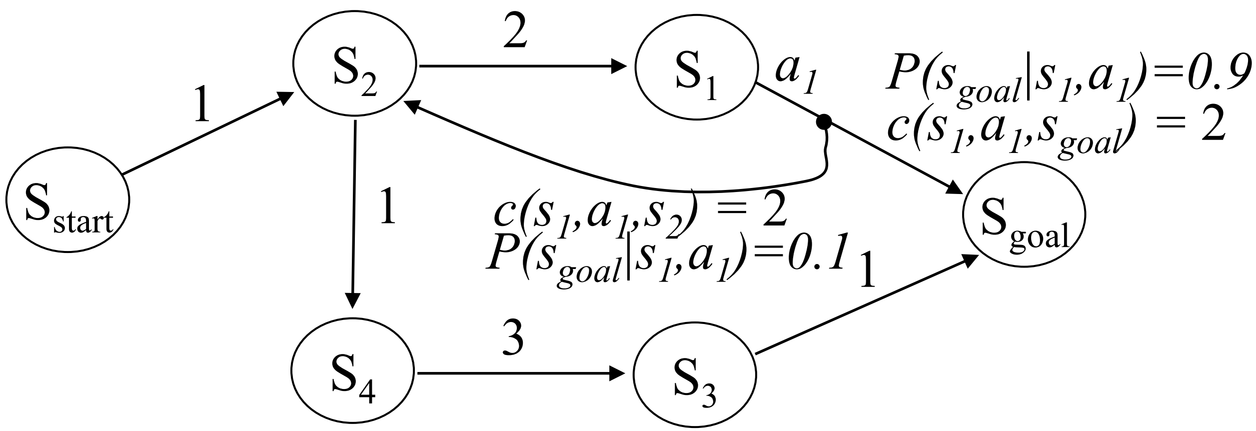
Minimax Formulation



- Optimal policy π^*:
  minimizes the *worst* cost-to-goal
  $\pi^* = argmin_\pi \, max_{outcomes\ of\ \pi}\{cost\text{-}to\text{-}goal\}$

- Optimal minimax policy π^* = (go through s_4) = $[\{s_{start}, a_{ne}\}, \{s_2, a_{south}\}, \{s_4, a_{east}\}, \{s_3, a_{ne}\}, \{s_{goal}, null\}]$

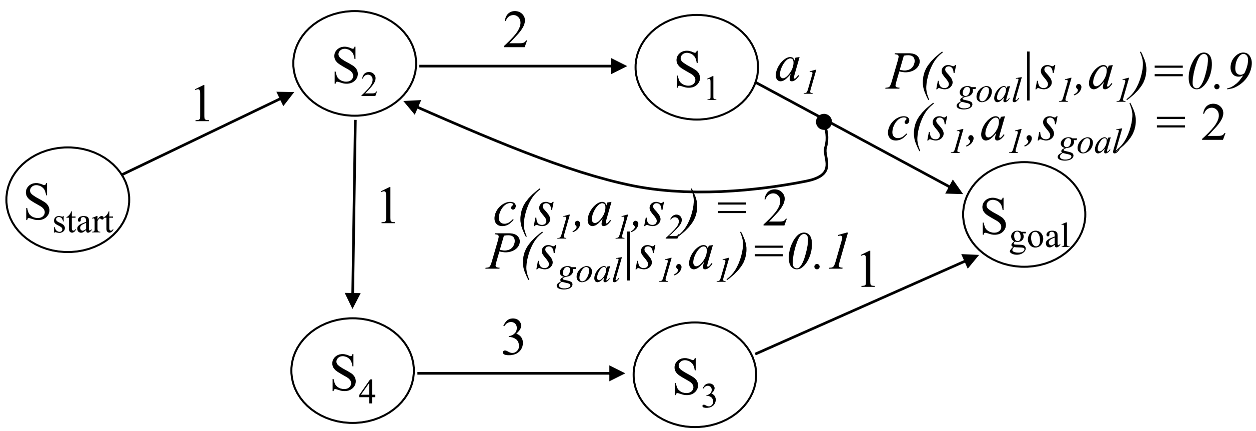
Computing Minimax Plans



- Minimax backward A*:

$g(s_{goal}) = 0;$ all other *g*-values are infinite; $OPEN = \{s_{goal}\};$
while$(s_{start}$ *not expanded)*
remove *s* with the smallest $[f(s) = g(s)+h(s)]$ from *OPEN*;
insert *s* into *CLOSED*;
for every *s'* s.t $s \in succ(s', a)$ for some *a* and *s'* not in *CLOSED*
if $g(s') > \boldsymbol{max}_{u \in succ(s', a)}\ c(s',u) + g(u)$
$g(s') = \boldsymbol{max}_{u \in succ(s', a)}\ c(s',u) + g(u);$
insert *s'* into *OPEN*;

Computing Minimax Plans

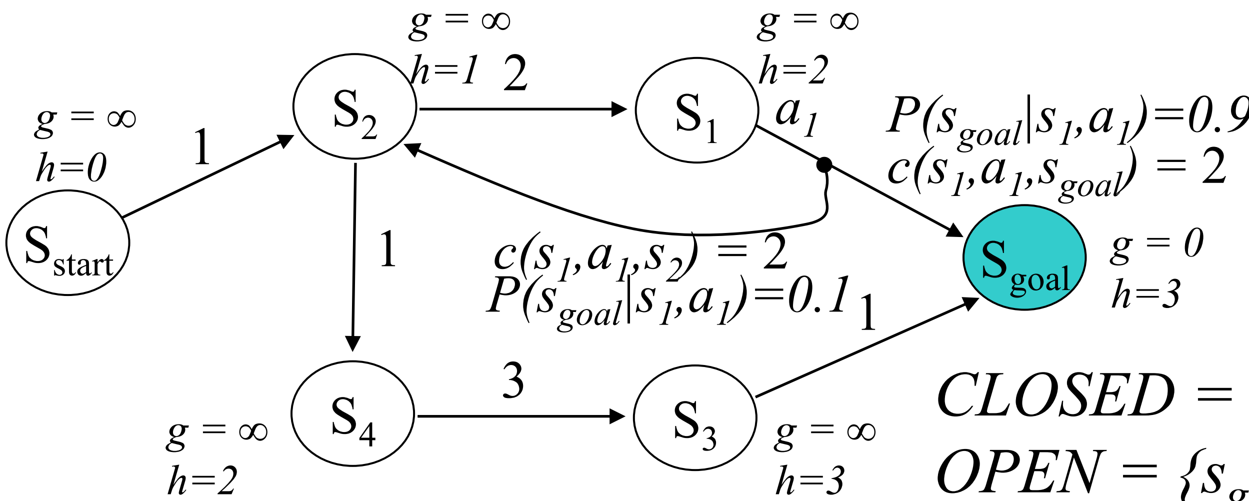


- Minimax backward A*:

$g(s_{goal}) = 0$; all other g-values are infinite; $OPEN = \{s_{goal}\}$;
while(s_{start} *not expanded*)
  remove s with the smallest $[f(s) = g(s)+h(s)]$ from *OPEN*;
  insert s into *CLOSED*;
  for every s' s.t $s \in succ(s', a)$ for some a and s' not in *CLOSED*
    if $g(s') > \mathbf{max}_{u \in succ(s', a)} c(s',u) + g(u)$
      $g(s') = \mathbf{max}_{u \in succ(s', a)} c(s',u) + g(u)$;
      insert s' into *OPEN*;

reduces to usual backward A if no uncertainty in outcomes*

Computing Minimax Plans



$CLOSED = \{\}$

$OPEN = \{s_{goal}\}$

next state to expand: s_{goal}

- Minimax backward A*:

$g(s_{goal}) = 0$; all other g-values are infinite; $OPEN = \{s_{goal}\}$;

while(s_{start} *not expanded*)

  remove s with the smallest $[f(s) = g(s)+h(s)]$ from $OPEN$;

  insert s into $CLOSED$;

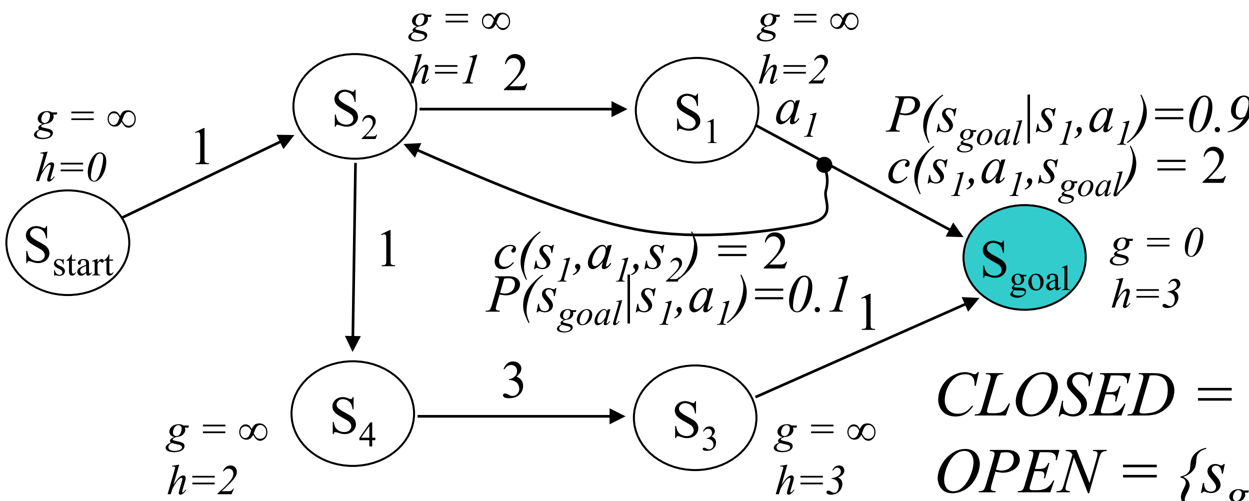
  for every s' s.t $s \in succ(s', a)$ for some a and s' not in $CLOSED$

    if $g(s') > max_{u \in succ(s', a)}\ c(s',u) + g(u)$

      $g(s') = max_{u \in succ(s', a)}\ c(s',u) + g(u)$;

      insert s' into $OPEN$;

Computing Minimax Plans



$CLOSED = \{\}$

$OPEN = \{s_{goal}\}$

next state to expand: s_{goal}

- Minimax backward A*:

$g(s_{goal}) = 0$; all other g-values are infinite; $OPEN = \{s_{goal}\}$;

while(s_{start} *not expanded*)

remove s with the smallest $[f(s) = g(s)+h(s)]$ from $OPEN$;

insert s into $CLOSED$;

for every s' s.t $s \in succ(s', a)$ for some a and s' not in $CLOSED$

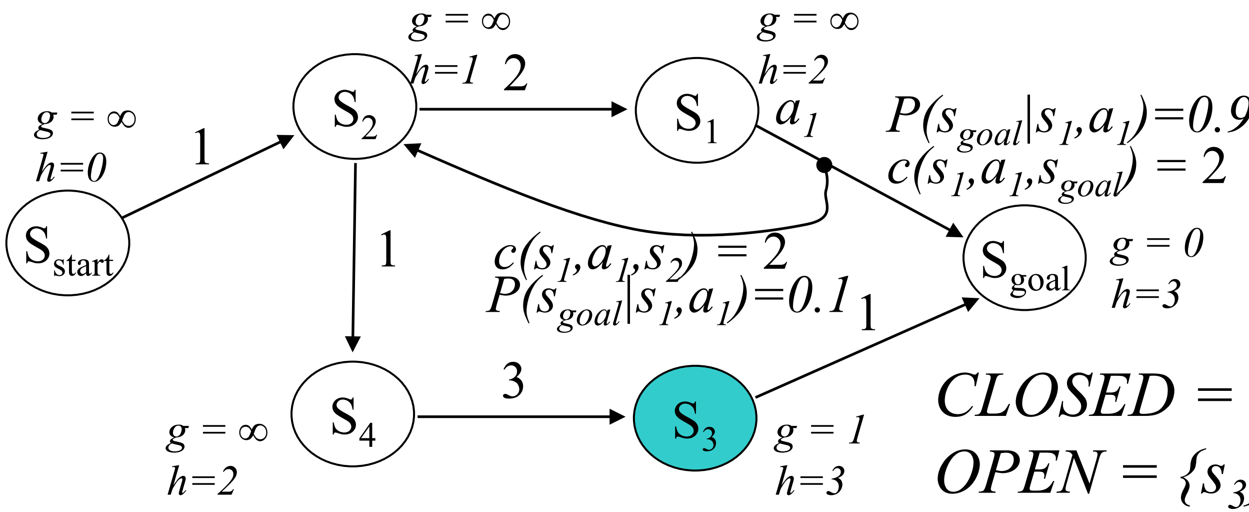
if $g(s') > max_{u \in succ(s', a)}\, c(s',u) + g(u)$

$g(s') = max_{u \in succ(s', a)}\, c(s',u) + g(u)$;

insert s' into $OPEN$;

After s_{goal} *expanded, what are* $g(s_3)$ *and* $g(s_1)$*?*

Computing Minimax Plans

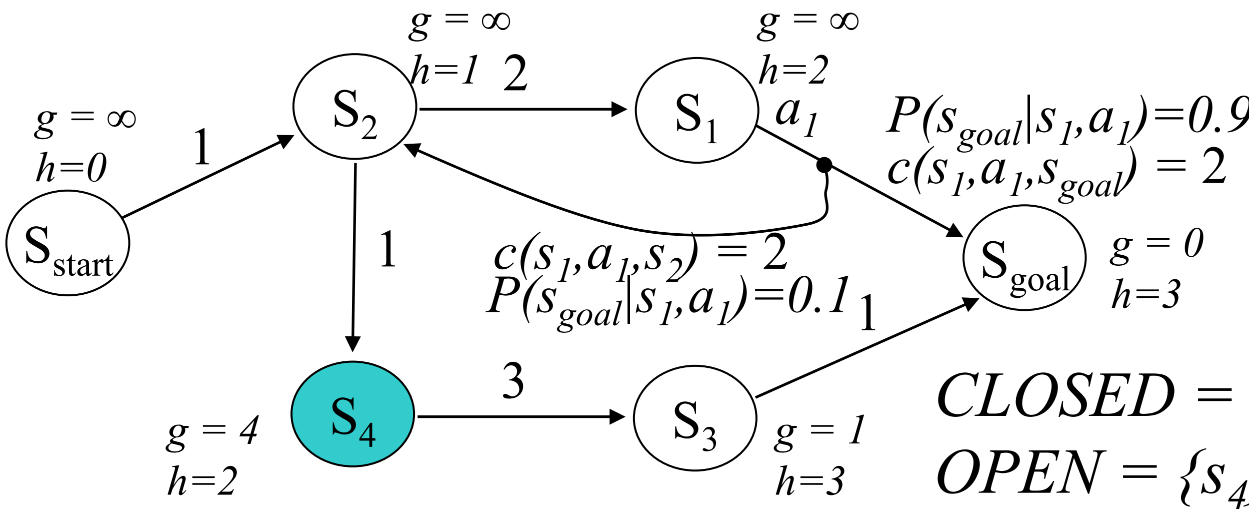


$CLOSED = \{s_{goal}\}$
$OPEN = \{s_3\}$
next state to expand: s_3

- Minimax backward A*:

$g(s_{goal}) = 0$; all other g-values are infinite; $OPEN = \{s_{goal}\}$;
while(s_{start} *not expanded*)
  remove s with the smallest $[f(s) = g(s)+h(s)]$ from *OPEN*;
  insert s into *CLOSED*;
  for every s' s.t $s \in succ(s', a)$ for some a and s' not in *CLOSED*
    if $g(s') > \max_{u \in succ(s', a)} c(s',u) + g(u)$
      $g(s') = \max_{u \in succ(s', a)} c(s',u) + g(u)$;
      insert s' into *OPEN*;

Computing Minimax Plans

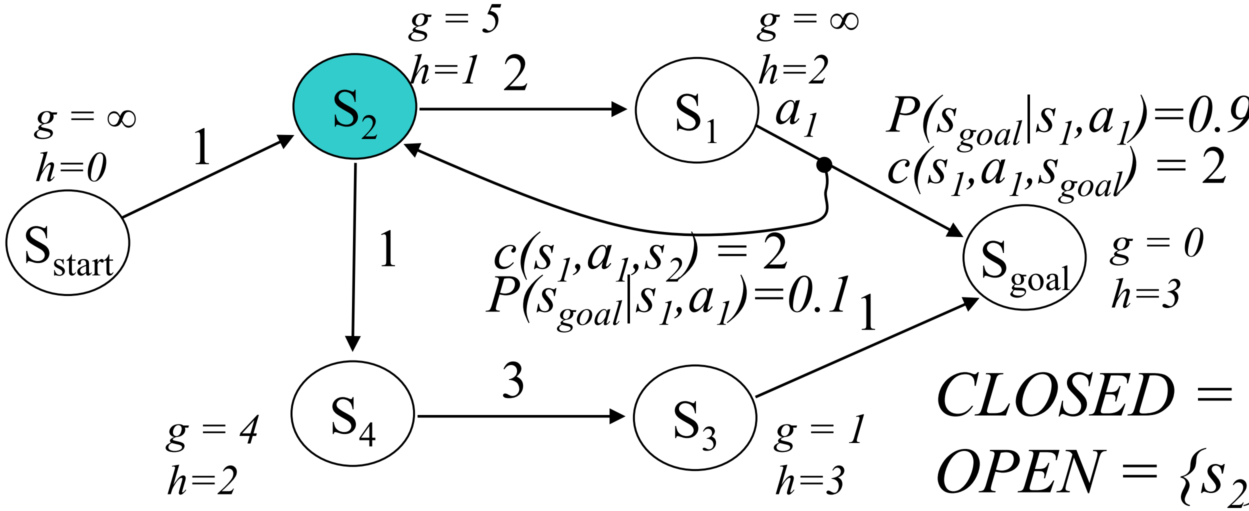


$CLOSED = \{s_{goal}, s_3\}$
$OPEN = \{s_4\}$
next state to expand: s_4

- Minimax backward A*:

$g(s_{goal}) = 0$; all other g-values are infinite; $OPEN = \{s_{goal}\}$;
while$(s_{start}$ *not expanded)*
  remove s with the smallest $[f(s) = g(s)+h(s)]$ from $OPEN$;
  insert s into $CLOSED$;
  for every s' s.t $s \in succ(s', a)$ for some a and s' not in $CLOSED$
    if $g(s') > \max_{u \in succ(s', a)} c(s',u) + g(u)$
      $g(s') = \max_{u \in succ(s', a)} c(s',u) + g(u)$;
      insert s' into $OPEN$;

Computing Minimax Plans



$CLOSED = \{s_{goal}, s_3, s_4\}$

$OPEN = \{s_2\}$

next state to expand: s_2

- Minimax backward A*:

$g(s_{goal}) = 0$; all other g-values are infinite; $OPEN = \{s_{goal}\}$;

while(s_{start} *not expanded*)

remove s with the smallest $[f(s) = g(s)+h(s)]$ from *OPEN*;

insert s into *CLOSED*;

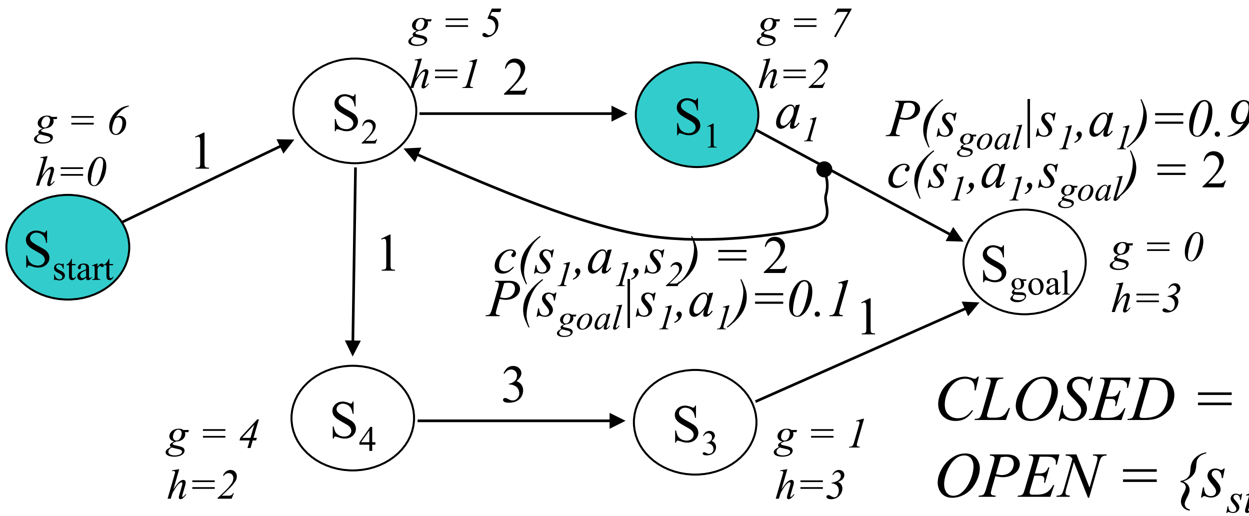
for every s' s.t $s \in succ(s', a)$ for some a and s' not in *CLOSED*

if $g(s') > \max_{u \in succ(s', a)} c(s',u) + g(u)$

$g(s') = \max_{u \in succ(s', a)} c(s',u) + g(u)$;

insert s' into *OPEN*;

Computing Minimax Plans



$CLOSED = \{s_{goal}, s_3, s_4, s_2\}$

$OPEN = \{s_{start}, s_1\}$

next state to expand: s_{start}

- Minimax backward A*:

$g(s_{goal}) = 0$; all other g-values are infinite; $OPEN = \{s_{goal}\}$;

while(s_{start} *not expanded*)

remove s with the smallest $[f(s) = g(s)+h(s)]$ from *OPEN*;

insert s into *CLOSED*;

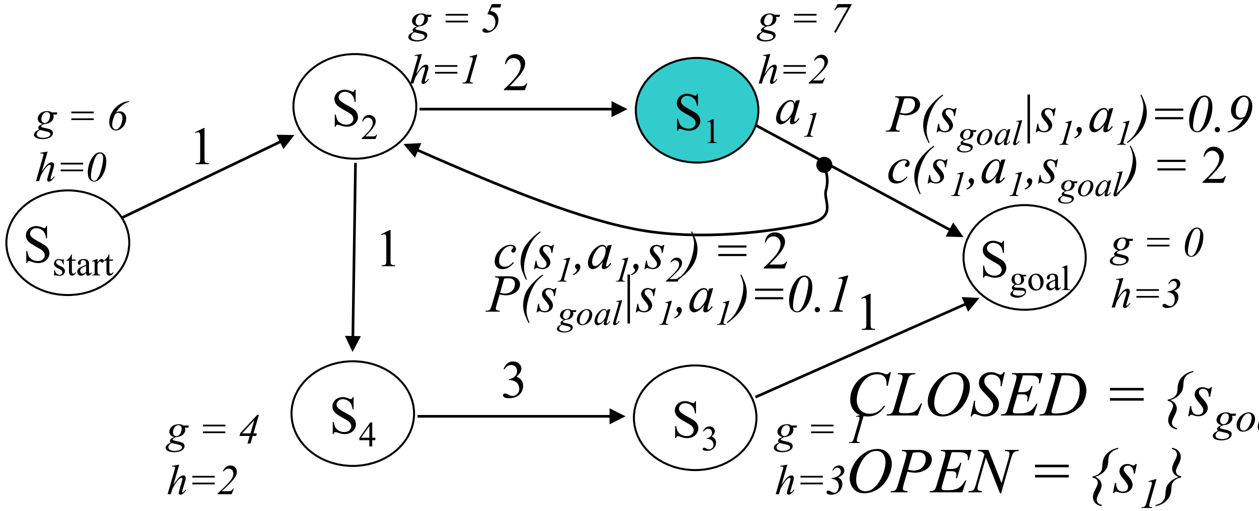
for every s' s.t $s \in succ(s', a)$ for some a and s' not in *CLOSED*

if $g(s') > \max_{u \in succ(s', a)} c(s',u) + g(u)$

$g(s') = \max_{u \in succ(s', a)} c(s',u) + g(u)$;

insert s' into *OPEN*;

Computing Minimax Plans



$CLOSED = \{s_{goal}, s_3, s_4, s_2, s_{start}\}$

$OPEN = \{s_1\}$

DONE

- Minimax backward A*:

  $g(s_{goal}) = 0;$ all other g-values are infinite; $OPEN = \{s_{goal}\};$

  while(s_{start} *not expanded*)

  remove s with the smallest $[f(s) = g(s)+h(s)]$ from *OPEN*;

  insert s into *CLOSED*;

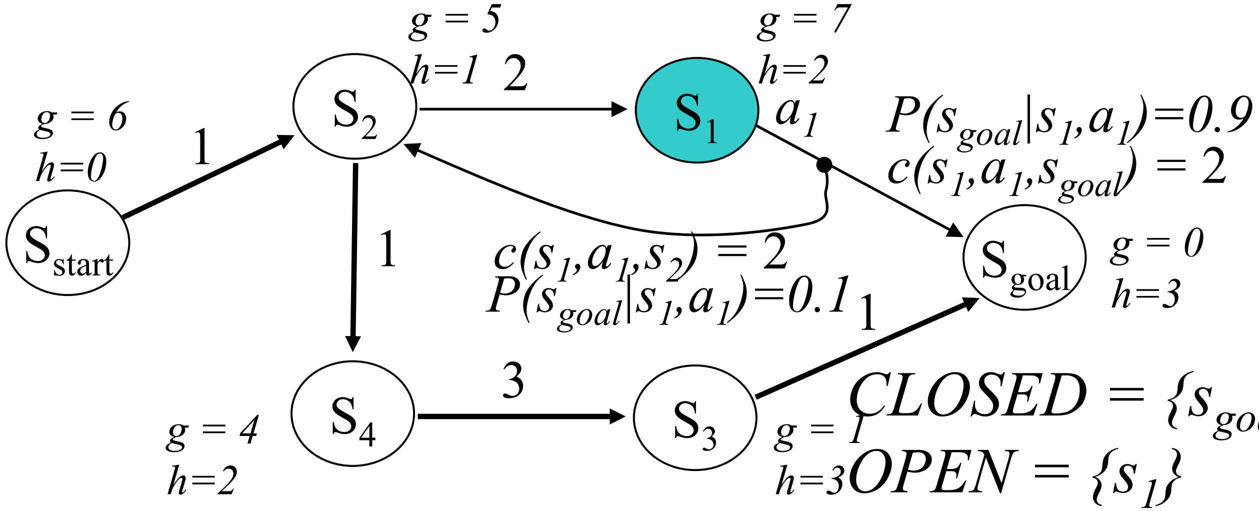
  for every s' s.t $s \in succ(s', a)$ for some a and s' not in *CLOSED*

  if $g(s') > \max_{u \in succ(s', a)} c(s',u) + g(u)$

  $g(s') = \max_{u \in succ(s', a)} c(s',u) + g(u);$

  insert s' into *OPEN*;

Computing Minimax Plans



$CLOSED = \{s_{goal}, s_3, s_4, s_2, s_{start}\}$

$OPEN = \{s_1\}$

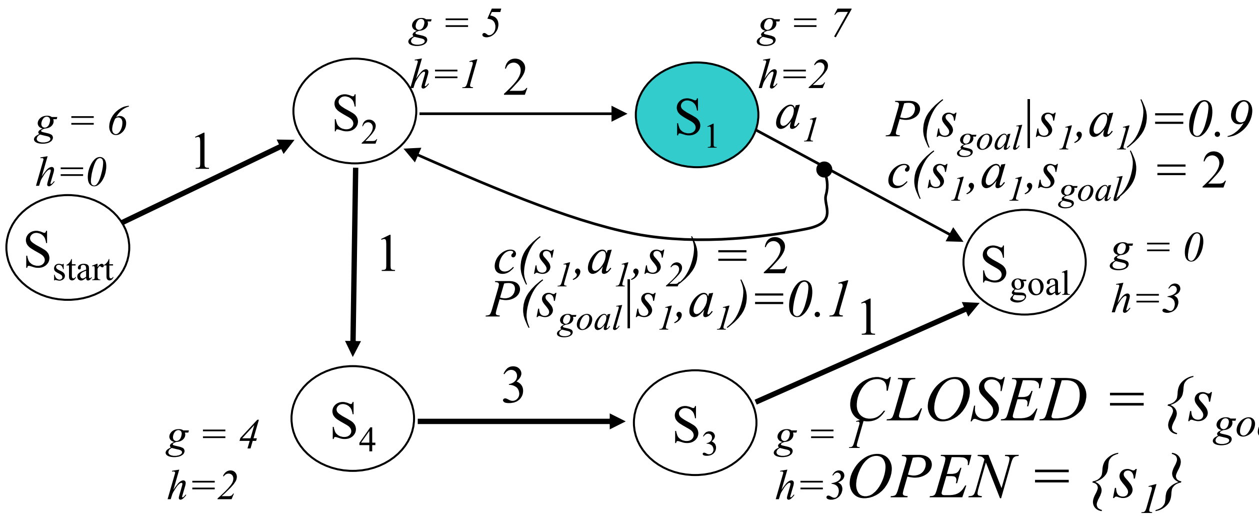
DONE

- Minimax backward A*:

  $g(s_{goal}) = 0$; all other g-values are infinite; $OPEN = \{s_{goal}\}$;
  while(s_{start} *not expanded*)
    remove s with the smallest $[f(s) = g(s)+h(s)]$ from *OPEN*;
    insert s into *CLOSED*;
    for every s' s.t $s \in succ(s', a)$ for some a and s' not in *CLOSED*
      if $g(s') > \max_{u \in succ(s', a)} c(s',u) + g(u)$
        $g(s') = \max_{u \in succ(s', a)} c(s',u) + g(u)$;
        insert s' into *OPEN*;

in this example, the computed policy is a path, but in general it is a DAG (directed acyclic graph)

Computing Minimax Plans



$CLOSED = \{s_{goal}, s_3, s_4, s_2, s_{start}\}$

$OPEN = \{s_1\}$

DONE

- Minimax backward A*:

  $g(s_{goal}) = 0$; all other g-values are infinite; $OPEN = \{s_{goal}\}$;

  while(s_{start} *not expanded*)

  remove s with the smallest $[f(s) = g(s)+h(s)]$ from OPEN

  insert s into *CLOSED*;

  for every s' s.t $s \in succ(s', a)$ for some

  if $g(s') > max_{u \in succ(s', a)}\ c(s',u) + g(u)$

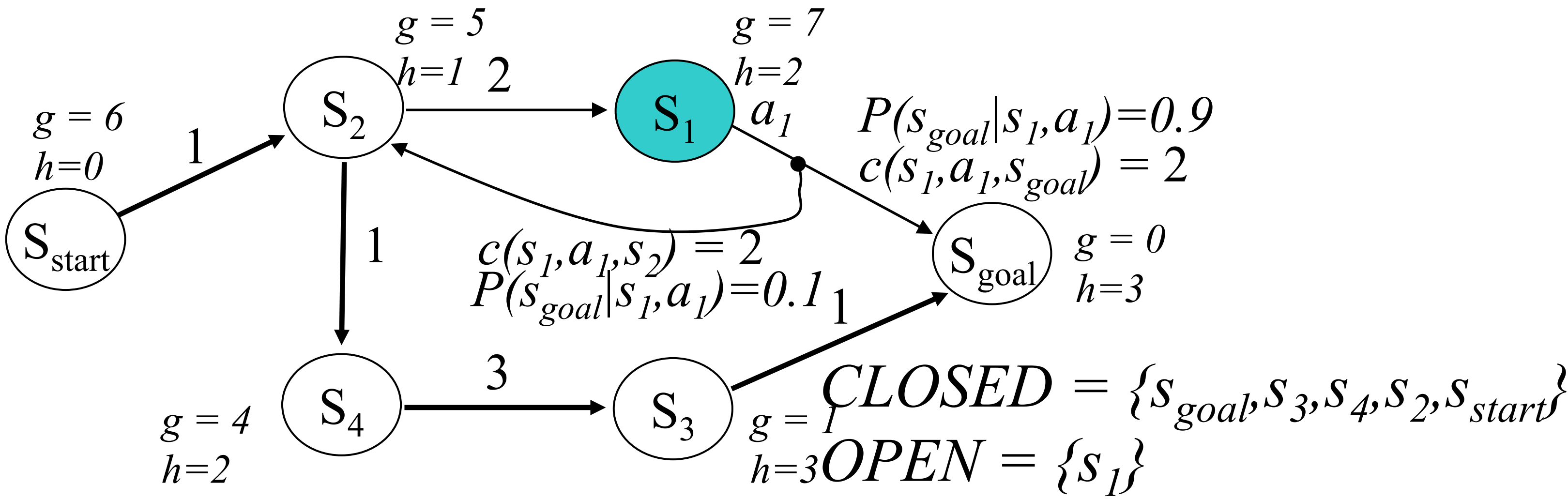
  $g(s') = max_{u \in succ(s', a)}\ c(s',u) + g(u)$;

  insert s' into *OPEN*;

What are its branches?
Why DAG?

in this example, the computed policy is a path, but in general it is a DAG (directed acyclic graph)

Computing Minimax Plans



- Minimax backward A*:

  $g(s_{goal}) = 0$; all other g-values are infinite; $OPEN = \{s_{goal}\}$;

  while(s_{start} *not expanded*)

  remove s with the smallest $[f(s) = g(s)+h(s)]$ from *OPEN*;

  insert s into *CLOSED*;

  for every s' s.t $s \in succ(s', a)$ for some a and s' not in *CLOSED*

  if $g(s') > \max_{u \in succ(s', a)} c(s',u) + g(u)$

  $g(s') = \max_{u \in succ(s', a)} c($

  insert s' into *OPEN*;

Minimax A guarantees to find an optimal plan, and never expands a state more than once, provided heuristics are consistent (just like A*)*

Computing Minimax Plans

- Pros/cons of minimax plans
  - robust to uncertainty
  - overly pessimistic
  - harder to compute than normal paths
    - especially if backwards minimax A* does not apply
    - even if backwards minimax A* does apply, still more expensive than computing a single path with A* (heuristics are not guiding well)

Why?

What You Should Know..

- What is and MDP (Markov Decision Process) and how it differs from normal Graphs
- What is Minimax solution to MDPs
- Pros and cons of Minimax solutions
- Operation of Minimax backward A*