

MDP-Based Planning



主讲人 朱德龙

Ph.D in Robotics, Perception and AI Lab.
The Chinese University of Hong Kong

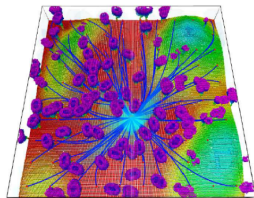
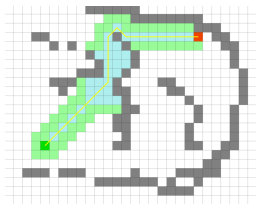




- 1 Uncertainties in Planning
- 2 Planning with Uncertainties
- 3 Markov Decision Process
- 4 Minimax Cost Planning
- 5 Expected Cost Planning
- 6 Real Time Dynamic Programming

Up until now, the planners are assumed no uncertainties with

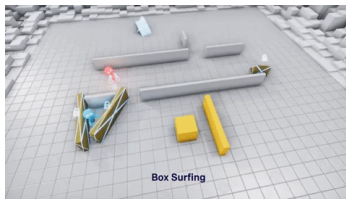
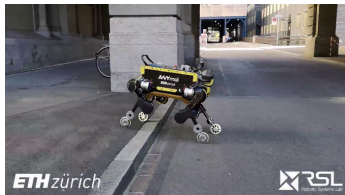
- **perfect action execution,**
- **full knowledge of states.**



Deterministic graph search (left) and planning with perfect map and poses (right).

In real applications, both execution and state estimation are not perfect.

- **Execution Uncertainties:** slippage, rough terrain, wind, air-drag, control errors, etc.
- **State Estimation Uncertainties:** sensor noise, calibration error, imperfect estimation, partial observability, etc.



Big dog with wheeled legs (left) and seek and hide game (right).



Uncertainties can be categorized into two classes from the perspective of robot, which **indicate how much information the robot can use**.

Uncertainty Models

- **Nondeterministic**: the robot have no idea what type of uncertainties or interference will be added to its behaviors.
- **Probabilistic**: the robot has an estimation about uncertainties by observing and gathering statistics.



To formally describe this concept, we first introduce two decision makers to model the generation of uncertainties, then the types of planning with uncertainties.

Decision Makers (Game Player)

- **Robot** is the primary decision maker that performs planning based on fully known states and perfect execution.
- **Nature** adds uncertainties to the execution of plans made by the robot, which is unpredictable to the robot.



Formalization-7.1: A Game Against Nature (Independent Game)

- A nonempty set U called the *robot action space*. Each $u \in U$ is referred to as a *robot action*.
- A nonempty set Θ called the *nature action space*. Each $\theta \in \Theta$ is referred to as a *nature action*.
- A function $L : U \times \Theta \rightarrow \mathbb{R} \cup \{\infty\}$, called the *cost function*, or the negative reward function.



Formalization-7.2: Nature Knows the Robot Action (Dependent Game)

- A nonempty set U called the *robot action space*. Each $u \in U$ is referred to as a *robot action*.
- For each $u \in U$, a nonempty set $\Theta(u)$ called the *nature action space*.
- A function $L : U \times \Theta \rightarrow \mathbb{R} \cup \{\infty\}$, called the *cost function*, or the negative reward function.

What is the best decision for robot, given that it is engaged in a game with nature?



One-step Worst-Case Analysis

- Under *Nondeterministic* model, $P(\theta)$ in *Independent Game* and $P(\theta|u_k)$ in *Dependent Game* are unknown;
- Robot cannot predict the behavior of nature and images it spitefully chooses actions that drive the cost as high as possible;
- Hence, it is reasonable to **make decisions by assuming the worst**.

$$u^* = \operatorname{argmin}_{u \in U} \left\{ \max_{\theta \in \Theta} \{L(u, \theta)\} \right\} \quad (1)$$



One-step Expected-Case Analysis

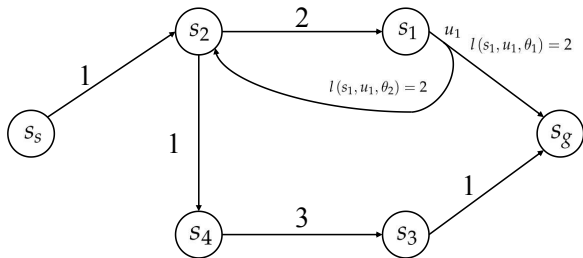
- Under *Probabilistic* model, $P(\theta)$ in *Independent Game* and $P(\theta|u_k)$ in *Dependent Game* are known;
- It is assumed that the applied nature actions have been observed and nature applies a randomized strategy in action selection.
- Hence, we optimize the **average cost to be received**.

$$u^* = \operatorname{argmin}_{u \in U} \{E_{\theta}[L(u, \theta)]\} \quad (2)$$



Formalization-7.2: Discrete Planning with Nature

- 1 A nonempty **state space** X with an **initial state** x_s and a **goal set** $X_F \subset X$.
- 2 For each state $x \in X$, a finite and nonempty **robot action space** $U(x)$. For each $x \in X$ and $u \in U(x)$, a finite and nonempty **nature action space** $\Theta(x, u)$.

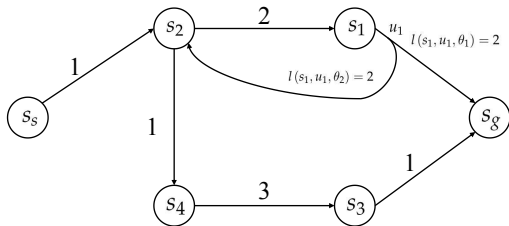


- 1 $X = \{s_s, s_1, s_2, s_3, s_4, s_g\}$
- 2 $X_F = \{s_g\}$
- 3 $U = \{u_s, u_1, u_{21}, u_{24}, u_3, u_4\}$
- 4 $\Theta = \{\theta_0, \theta_1, \theta_2\}$

Formalization-7.2: Multi-Step Discrete Planning with Nature

3 A **state transition function** $f(x, u, \theta)$ for every $x \in X, u \in U$, and $\theta \in \Theta(x, u)$.

$$\begin{aligned} X_{k+1}(x_k, u_k) &= \{x_{k+1} \in X \mid \exists \theta_k \in \Theta(x_k, u_k) \text{ s.t. } x_{k+1} = f(x_k, u_k, \theta_k)\} \\ P(x_{k+1} | x_k, u_k) &= \sum_{\theta_k} P(x_{k+1}, \theta_k | x_k, u_k) \text{ s.t. } \{\theta_k | x_{k+1} = f(x_k, u_k, \theta_k)\} \end{aligned} \quad (3)$$

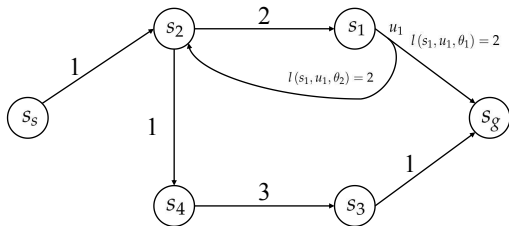


- 1 $X(s_s, u_s) = \{f(s_s, u_s, \theta_0)\}$
- 2 $X(s_1, u_1) = \{f(s_1, u_1, \theta_1), f(s_1, u_1, \theta_2)\}$
- 3 $s_s \rightarrow u_s \rightarrow s_2 \rightarrow u_{21} \rightarrow s_1 \rightarrow u_1 \rightarrow s_g$
 $\underbrace{\hspace{10em}}_k \quad \underbrace{\hspace{10em}}_{F=k+1}$



Formalization-7.2: Multi-Step Discrete Planning with Nature

- 4 A set of **stages**, each denoted by k , that begins at $k = 1$ and continues indefinitely or ends at a maximum stage $k = K + 1 = F$.



- 1 $X(s_s, u_s) = \{f(s_s, u_s, \theta_0)\}$
- 2 $X(s_1, u_1) = \{f(s_1, u_1, \theta_1), f(s_1, u_1, \theta_2)\}$
- 3 $\underbrace{s_s \rightarrow u_s \rightarrow s_2 \rightarrow u_{21} \rightarrow s_1 \rightarrow u_1}_k \rightarrow \underbrace{s_g}_{F=k+1}$

Formalization-7.2: Multi-Step Discrete Planning with Nature

- 5 A stage-additive **cost functional** L . Let $\tilde{x}_F, \tilde{u}_k, \tilde{\theta}_K$ denote the history of states, robot actions, and nature actions up to stage K :

$$\tilde{x}_F = (x_1, x_2, \dots, x_F), \tilde{u}_k = (u_1, u_2, \dots, u_k), \tilde{\theta}_K = (\theta_1, \theta_2, \dots, \theta_K) \quad (4)$$

The cost functional is a metrics that evaluate all possible plans (or paths):

$$L(\tilde{x}_F, \tilde{u}_K, \tilde{\theta}_K) = \sum_{k=1}^K l(x_k, u_k, \theta_k) + l_F(x_F), \quad (5)$$
$$l_F(x_F) = \begin{cases} 0, & \text{if } x_F \in X_G, \\ \infty, & \text{otherwise.} \end{cases}$$



Actually, (1-3) in Formalization-7.2 and the one-step cost l define a standard

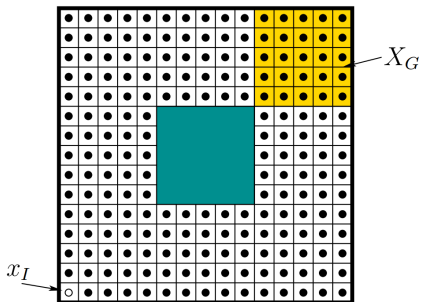
Markov Decision Process (MDP)

A MDP is a 4-tuple (S, A, P, R) in learning field, or (X, U, P, L) in planning field:

- S or X is *state space*,
- A or U is *(robot) action space*,
- $P(x_{k+1}|x_k, u_k)$ is *state transition function* under probabilistic model,
 - which degenerates to a set $X_{k+1}(x_k, u_k)$ under nondeterministic model,
- $R(x_k, x_{k+1})$ is *the immediate reward*, or the negative one-step cost $-l(x_k, u_k, \theta_k)$, after transitioning from x_k to x_{k+1} due to u, θ .

The first difficulty of planning with uncertainties lies on *properly formalizing our problem with a MDP model*.

A grid-based shortest path problem with interference from nature.



- ① $X = \{(i, j) \mid 1 \leq i, j \leq 15\}$
- ② $U = \{u_{stay}, u_{right}, u_{up}, u_{left}, u_{down}\}.$
- ③ $\Theta = \{\theta_1, \theta_2\}$
 $f(x_{k+1}, x_k, u_k) = \mathcal{N}(x_k + u_k, \sigma(\theta_1, \theta_2))$
- ④ $\Theta = \{\theta_0 = [0, 0]^t, \theta_1 = [0, 1]^T, \theta_2 = [0, -1]^T\}$
 $x_{k+1} = x_k + u_k + \theta_k, k \sim \{0, 1, 2\}$
- ⑤ $l(x_k, u_k, \theta_k) = \|x_{k+1} - x_k\|$

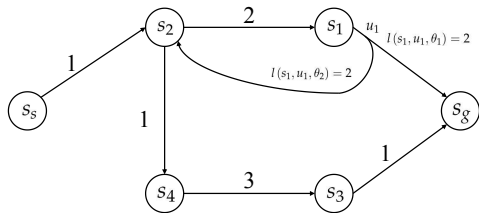


Define a (feedback) plan:

- $\pi : X \rightarrow U$.

Define a set of trajectories:

- $\mathcal{H}(\pi, x_s) : \text{induced by } \pi \text{ and started from } x_s$.
- $(\tilde{x}, \tilde{u}, \tilde{\theta}) \in \mathcal{H}(\pi, x_s) : \text{a trajectory or an execution.}$



① $\mathcal{H}(\pi_1, s_s) :$

$$s_s \rightarrow u_s \rightarrow s_2 \rightarrow u_{21} \rightarrow s_1 \rightarrow u_1 \rightarrow s_g$$

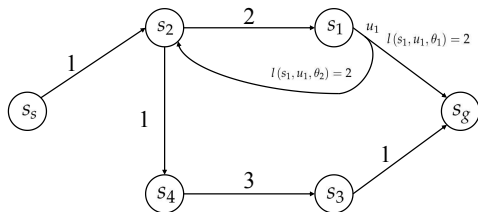
$$s_s \rightarrow u_s \rightarrow s_2 \rightarrow u_{21} \rightarrow s_1 \rightarrow u_1 \rightarrow s_2 \rightarrow \dots$$

② $\mathcal{H}(\pi_2, s_s) :$

$$s_s \rightarrow u_s \rightarrow s_2 \rightarrow u_{24} \rightarrow s_4 \rightarrow u_4 \rightarrow s_3 \rightarrow u_3 \rightarrow s_g$$

Define the cost for a specific plan π (rather than a trajectory):

- $G_\pi(x_s)$: **cost-to-goal**.



- 1 *worst-case analysis* for nondeterministic model:

$$G_\pi(x_s) = \max_{(\tilde{x}, \tilde{u}, \tilde{\theta}) \in \mathcal{H}(\pi, x_s)} \{L(\tilde{x}, \tilde{u}, \tilde{\theta})\} \quad (6)$$

- 2 *expected-case analysis* for probabilistic model:

$$G_\pi(x_s) = E_{\mathcal{H}(\pi, x_s)}[L(\tilde{x}, \tilde{u}, \tilde{\theta})] \quad (7)$$



- ① Nondeterministic model
- ② Worst-case analysis
- ③ Find an optimal plan π^* such that

$$G_{\pi^*}(x_s) = \min_{\pi} \{G_{\pi}(x_s)\} = \min_{\pi} \{ \max_{\mathcal{H}(\pi, x_s)} \{L(\tilde{x}, \tilde{u}, \tilde{\theta})\} \}. \quad (8)$$

- ④ Directly solving the problem is difficult.
- ⑤ Construct optimal solution by **Dynamic Programming**.
 - The recurrence between stage k and $k + 1$ of optimal plan.



Minimax Cost Planning with Nondeterministic Model

- ① The optimal cost-to-goal for final state F can be directly acquired $G_F^* = l_F(x_F)$,
- ② The costs of all optimal one-step plans from stage K to stage $F = K + 1$ is

$$G_K^*(x_K) = \min_{u_K} \max_{\theta_K} \{l(x_K, u_K, \theta_K) + G_F^*(f(x_K, u_K, \theta_K))\} \quad (9)$$

- ③ More generally, G_k^* can be computed once G_{k+1}^* is given

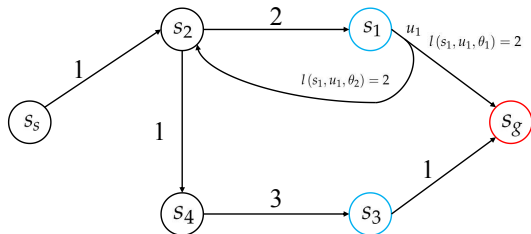
$$G_k^*(x_k) = \min_{u_k} \max_{\theta_k} \left[\min_{u_{k+1}\theta_{k+1}} \cdots \min_{u_K} \max_{\theta_K} (l(x_k, u_k, \theta_k) + \sum_{i=k+1}^K l(x_i, u_i, \theta_i) + l_F(x_F)) \right] \quad (10)$$

$$G_k^*(x_k) = \min_{u_k} \max_{\theta_k} \left[l(x_k, u_k, \theta_k) + \underbrace{\min_{u_{k+1}\theta_{k+1}} \cdots \min_{u_K} \max_{\theta_K} \left(\sum_{i=k+1}^K l(x_i, u_i, \theta_i) + l_F(x_F) \right)}_{G_{k+1}^*(x_{k+1})} \right] \quad (11)$$



Hence, the dynamic programming recurrence for solving minimax cost plan is

$$G_k^*(x_k) = \min_{u_k \in U(x_k)} \left\{ \max_{\theta_k \in \Theta(x_k, u_k)} \{l(x_k, u_k, \theta_k) + G_{k+1}^*(x_{k+1})\} \right\} \quad (12)$$



- 1 suppose $G_{k+1}^*(x_{k+1} = s_g) = 0$
- 2 $G_k^*(x_k = s_3) = \min\{1 + 0\}$
- 3 $G_k^*(x_k = s_1) = \min\{\max\{2 + 0, 2 + \inf\}\}$



```
G(x_F)  $\leftarrow$  0; all other G values are infinite; OPEN = {x_F}; CLOSED =  $\emptyset$ ;  
while  $x_s$  is not expanded do  
  |  $x_{k+1} \leftarrow$  remove  $x$  with the smallest G value from OPEN;  
  | insert  $x_{k+1}$  to CLOSED;  
  | for every  $x_k \notin$  CLOSED s.t.  $x_{k+1} \in X_{k+1}(x_k, u_k)$  do  
  |   | if  $G(x_k) > \max_{\theta_k \in \Theta(x_k, u_k)} \{l(x_k, u_k, \theta_k) + G(x_{k+1})\}$  then  
  |   |   |  $G(x_k) = \max_{\theta_k \in \Theta(x_k, u_k)} \{l(x_k, u_k, \theta_k) + G(x_{k+1})\};$   
  |   |   | insert  $x_k$  into OPEN;  
  |   | end  
  | end  
end
```

Algorithm 1: Nondeterministic Dijkstra



Minimax Cost Planning

$G(x_F) \leftarrow 0$; all other G values are infinite; $OPEN = \{x_F\}$;

$CLOSED = \emptyset$;

while x_s is not expanded **do**

$x_{k+1} \leftarrow$ remove x with the smallest G value from
 $OPEN$;

 insert x_{k+1} to $CLOSED$;

for every $x_k \notin CLOSED$ s.t. $x_{k+1} \in X_{k+1}(x_k, u_k)$ **do**

if $G(x_k) > \max_{\theta_k} \{l(x_k, u_k, \theta_k) + G(x_{k+1})\}$ **then**

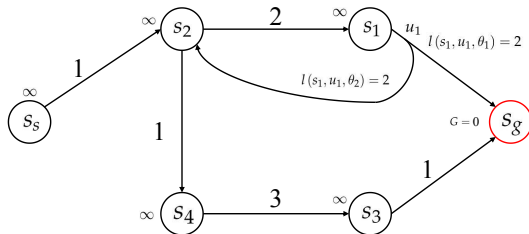
$G(x_k) \leftarrow \max_{\theta_k} \{l(x_k, u_k, \theta_k) + G(x_{k+1})\}$;

 insert x_k into $OPEN$;

end

end

end



① $CLOSED = \{\}$

② $OPEN = \{s_g\}$

③ next state to expand: s_g

④ $G(x_{k+1} = s_g) = 0$



Minimax Cost Planning

$G(x_F) \leftarrow 0$; all other G values are infinite; $OPEN = \{x_F\}$;
 $CLOSED = \emptyset$;

while x_s is not expanded **do**

$x_{k+1} \leftarrow$ remove x with the smallest G value from
 $OPEN$;

 insert x_{k+1} to $CLOSED$;

for every $x_k \notin CLOSED$ s.t. $x_{k+1} \in X_{k+1}(x_k, u_k)$ **do**

if $G(x_k) > \max_{\theta_k} \{l(x_k, u_k, \theta_k) + G(x_{k+1})\}$ **then**

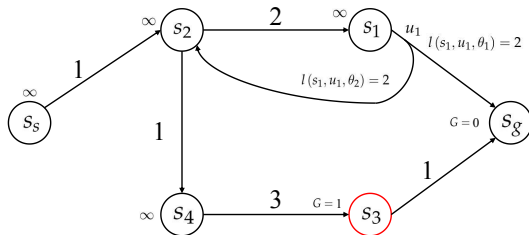
$G(x_k) \leftarrow \max_{\theta_k} \{l(x_k, u_k, \theta_k) + G(x_{k+1})\}$;

 insert x_k into $OPEN$;

end

end

end



- ① $CLOSED = \{s_g\}$
- ② $G(x_k = s_3) = 1 + G(x_{k+1} = s_g)$
- ③ $OPEN = \{s_3\}$
- ④ next state to expand: -

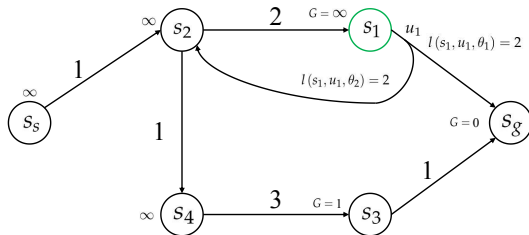


Minimax Cost Planning

```

 $G(x_F) \leftarrow 0$ ; all other  $G$  values are infinite;  $OPEN = \{x_F\}$ ;
 $CLOSED = \emptyset$ ;
while  $x_s$  is not expanded do
     $x_{k+1} \leftarrow$  remove  $x$  with the smallest  $G$  value from
         $OPEN$ ;
    insert  $x_{k+1}$  to  $CLOSED$ ;
    for every  $x_k \notin CLOSED$  s.t.  $x_{k+1} \in X_{k+1}(x_k, u_k)$  do
        if  $G(x_k) > \max_{\theta_k} \{l(x_k, u_k, \theta_k) + G(x_{k+1})\}$  then
             $G(x_k) \leftarrow \max_{\theta_k} \{l(x_k, u_k, \theta_k) + G(x_{k+1})\}$ ;
            insert  $x_k$  into  $OPEN$ ;
        end
    end
end

```



- ① $CLOSED = \{s_g\}$
- ② $G(x_k = s_1)$ cannot be reduced
- ③ $OPEN = \{s_3\}$
- ④ next state to expand: s_3



Minimax Cost Planning

$G(x_F) \leftarrow 0$; all other G values are infinite; $OPEN = \{x_F\}$;
 $CLOSED = \emptyset$;

while x_s is not expanded **do**

$x_{k+1} \leftarrow$ remove x with the smallest G value from
 $OPEN$;

 insert x_{k+1} to $CLOSED$;

for every $x_k \notin CLOSED$ s.t. $x_{k+1} \in X_{k+1}(x_k, u_k)$ **do**

if $G(x_k) > \max_{\theta_k} \{l(x_k, u_k, \theta_k) + G(x_{k+1})\}$ **then**

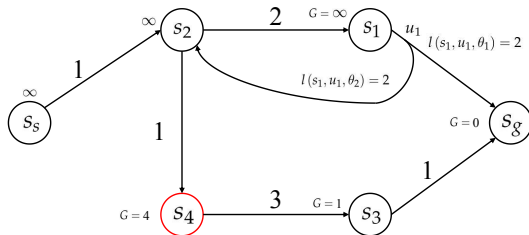
$G(x_k) \leftarrow \max_{\theta_k} \{l(x_k, u_k, \theta_k) + G(x_{k+1})\}$;

 insert x_k into $OPEN$;

end

end

end



① $CLOSED = \{s_g, s_3\}$

② $G(x_k = s_4) = 3 + G(x_{k+1} = s_3)$

③ $OPEN = \{s_4\}$

④ next state to expand: s_4



Minimax Cost Planning

$G(x_F) \leftarrow 0$; all other G values are infinite; $OPEN = \{x_F\}$;
 $CLOSED = \emptyset$;

while x_s is not expanded **do**

$x_{k+1} \leftarrow$ remove x with the smallest G value from
 $OPEN$;

 insert x_{k+1} to $CLOSED$;

for every $x_k \notin CLOSED$ s.t. $x_{k+1} \in X_{k+1}(x_k, u_k)$ **do**

if $G(x_k) > \max_{\theta_k} \{l(x_k, u_k, \theta_k) + G(x_{k+1})\}$ **then**

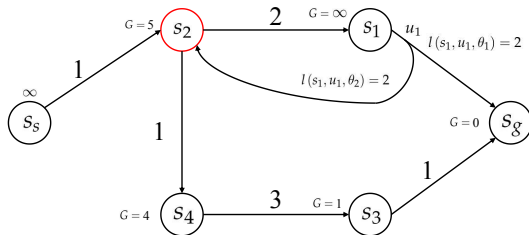
$G(x_k) \leftarrow \max_{\theta_k} \{l(x_k, u_k, \theta_k) + G(x_{k+1})\}$;

 insert x_k into $OPEN$;

end

end

end



- ① $CLOSED = \{s_g, s_3, s_4\}$
- ② $G(x_k = s_2) = 1 + G(x_{k+1} = s_4)$
- ③ $OPEN = \{s_2\}$
- ④ next state to expand: s_2



CLOSED = \emptyset ;

$$x_{k+1} \leftarrow \text{remove } x \text{ with the smallest } G \text{ value from OPEN};$$

insert x_{k+1} to CLOSED;

for every $x_k \notin \text{CLOSED}$ s.t. $x_{k+1} \in X_{k+1}(x_k, u_k)$ **do**

if $G(x_k) > \max_{\theta_k} \{l(x_k, u_k, \theta_k) + G(x_{k+1})\}$ **then**

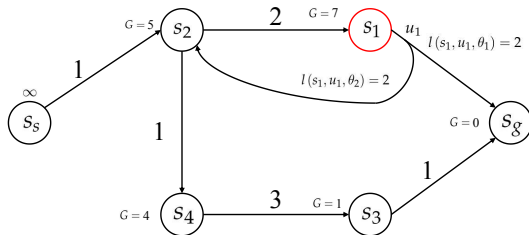
$$G(x_k) \leftarrow \max_{\theta_k} \{l(x_k, u_k, \theta_k) + G(x_{k+1})\};$$

insert x_k into OPEN;

end

end

end



- 1 CLOSED = $\{s_g, s_3, s_4, s_2\}$
- 2 $G(x_k = s_1) = 2 + G(x_{k+1} = s_2)$
- 3 OPEN = $\{s_1\}$
- 4 next state to expand: -



Minimax Cost Planning

$G(x_F) \leftarrow 0$; all other G values are infinite; $OPEN = \{x_F\}$;
 $CLOSED = \emptyset$;

while x_s is not expanded **do**

$x_{k+1} \leftarrow$ remove x with the smallest G value from
 $OPEN$;

 insert x_{k+1} to $CLOSED$;

for every $x_k \notin CLOSED$ s.t. $x_{k+1} \in X_{k+1}(x_k, u_k)$ **do**

if $G(x_k) > \max_{\theta_k} \{l(x_k, u_k, \theta_k) + G(x_{k+1})\}$ **then**

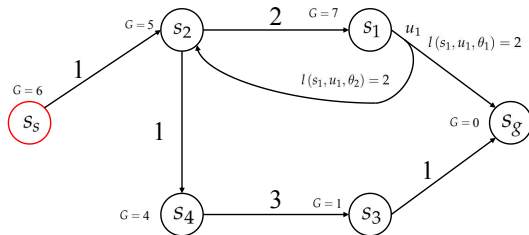
$G(x_k) \leftarrow \max_{\theta_k} \{l(x_k, u_k, \theta_k) + G(x_{k+1})\}$;

 insert x_k into $OPEN$;

end

end

end



① $CLOSED = \{s_g, s_3, s_4, s_2\}$

② $G(x_k = s_s) = 1 + G(x_{k+1} = s_2)$

③ $OPEN = \{s_1, s_s\}$

④ next state to expand: s_s

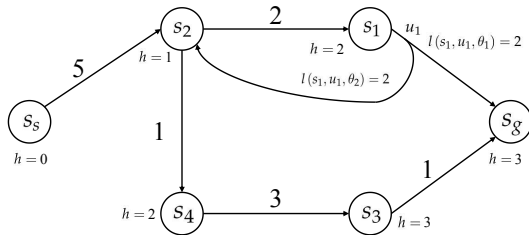


Minimax Cost Planning

```

 $G(x_F) \leftarrow 0$ ; all other  $G$  values are infinite;  $OPEN = \{x_F\}$ ;
 $CLOSED = \emptyset$ ;
while  $x_s$  is not expanded do
     $x_{k+1} \leftarrow$  remove  $x$  with the smallest  $G$  value from
         $OPEN$ ;
    insert  $x_{k+1}$  to  $CLOSED$ ;
    for every  $x_k \notin CLOSED$  s.t.  $x_{k+1} \in X_{k+1}(x_k, u_k)$  do
        if  $G(x_k) > \max_{\theta_k} \{l(x_k, u_k, \theta_k) + G(x_{k+1})\}$  then
             $G(x_k) \leftarrow \max_{\theta_k} \{l(x_k, u_k, \theta_k) + G(x_{k+1})\}$ ;
            insert  $x_k$  into  $OPEN$ ;
        end
    end
end

```



- ① How to improve the left algorithm to A* ?
- ② Can this algorithm deal with failure cases ?
- ③ How to track the optimal plan π^* ?



Pros/Cons of minimax cost planning

- robust to uncertainty
- overly pessimistic
- harder to compute than normal paths
 - especially if nondeterministic Dijkstra or A^* does not apply
 - even if nondeterministic A^* does apply, still more expensive than computing a single path with A^* . – **Why ?**



- ① Probabilistic model
- ② expected-case analysis
- ③ Find an optimal plan π^* such that

$$G_{\pi^*}(x_s) = \min_{\pi} \{G_{\pi}(x_s)\} = \min_{\pi} \{E_{\mathcal{H}(\pi, x_s)}[L(\tilde{x}, \tilde{u}, \tilde{\theta})]\}. \quad (13)$$

- ④ Directly solving the problem is difficult.
- ⑤ Construct optimal solution by **Dynamic Programming**.
 - The recurrence between stage k and $k + 1$ of optimal plan.



- ① The optimal cost-to-goal for final state F can be directly acquired $G_F^* = l_F(x_F)$,
- ② The costs of all optimal one-step plans from stage K to stage $F = K + 1$ is

$$G_K^*(x_K) = \min_{u_K} \{E_{\theta_K} [l(x_K, u_K, \theta_K) + G_F^*(f(x_K, u_K, \theta_K))]\} \quad (14)$$

- ③ More generally, G_k^* can be computed once G_{k+1}^* is given

$$G_k^*(x_k) = \min_{u_k} \{E_{\theta_k} [\min_{u_{k+1}, \dots, u_K} \{E_{\theta_{k+1}, \dots, \theta_K} [l(x_k, u_k, \theta_k) + \sum_{i=k+1}^K l(x_i, u_i, \theta_i) + l_F(x_F)]]\}] \} \quad (15)$$

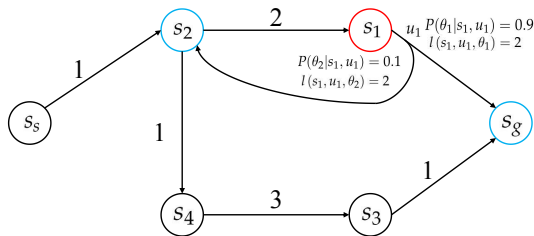
$$G_k^*(x_k) = \min_{u_k} \{E_{\theta_k} [l(x_k, u_k, \theta_k) + \underbrace{\min_{u_{k+1}, \dots, u_K} \{E_{\theta_{k+1}, \dots, \theta_K} [\sum_{i=k+1}^K l(x_i, u_i, \theta_i) + l_F(x_F)]\}}_{G_{k+1}^*(x_{k+1})}]\}] \} \quad (16)$$



Hence, the dynamic programming recurrence for solving expected cost plan is

$$G_k^*(x_k) = \min_{u_k \in U(x_k)} \{ E_{\theta_k} [l(x_k, u_k, \theta_k) + G_{k+1}^*(x_{k+1})] \}, \quad (17)$$

which has a special name called **Bellman Optimality Equation**.



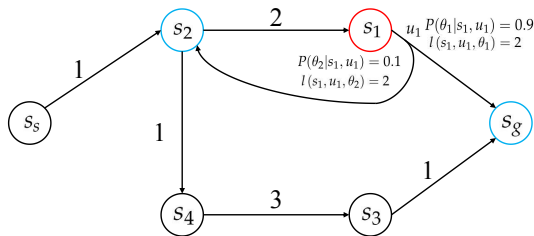
- 1 suppose $G_{k+1}^*(x_{k+1} = s_g) = 0$
- 2 suppose $G_{k+1}^*(x_{k+1} = s_2) = 2$
- 3 $G_k^*(x_k = s_1) = \min\{(2 + 0) * 0.9 + (2 + 2) * 0.1\}$



Hence, the dynamic programming recurrence for solving expected cost plan is

$$G_k^*(x_k) = \min_{u_k \in U(x_k)} \{ E_{\theta_k} [l(x_k, u_k, \theta_k) + G_{k+1}^*(x_{k+1})] \} \quad (18)$$

which has a special name called **Bellman Optimality Equation**.



- ① suppose $G_{k+1}^*(x_{k+1} = s_g) = 0$
- ② suppose $G_{k+1}^*(x_{k+1} = s_2) = 2$
- ③ how can we know $G_{k+1}^*(x_{k+1} = s_2)$?



Initialize G values of all states to finite values;

while *not converge* **do**

for *all the states* x **do**

$$G(x_F) = 0$$

$$G_k(x_k) = \min_{u_k \in U(x_k)} \{ \underbrace{E_{\theta_k} [l(x_k, u_k, \theta_k) + G_{k+1}(x_{k+1})]}_{\text{Bellman Update Equation}} \}, x_k \neq x_F$$

end

end

Algorithm 2: Value Iteration (VI)

- ① Optimal values is achieved by conducting value iteration.
 - optimality is not related to iteration order.
 - convergence speed depends on iteration order.
- ② Bellman update equation is a method of achieving Bellman optimal equation.



Expected Cost Planning

Initialize G values of all states to finite values;

while not converge **do**

for all the states x **do**

$G(x_F) = 0$;

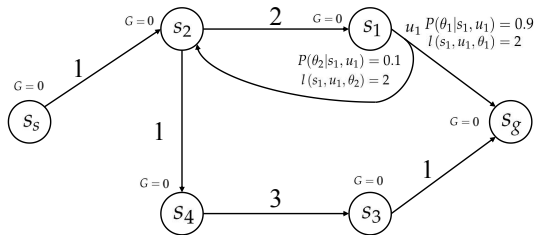
$G_k(x_k) =$

$\min_{u_k \in U(x_k)} \{E_{\theta_k} [l(x_k, u_k, \theta_k) + G_{k+1}(x_{k+1})]\},$

 if $x_k \neq x_F$;

end

end



① initialize all G value with zero.

② iteration order:

$s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow s_4 \rightarrow s_s$



Expected Cost Planning

Initialize G values of all states to finite values;

while not converge **do**

for all the states x **do**

$G(x_F) = 0$;

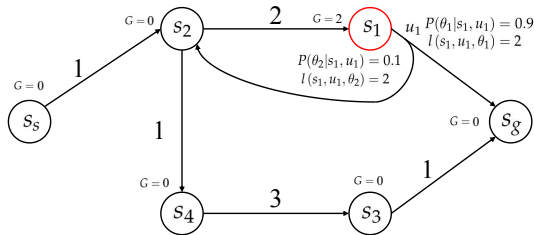
$G_k(x_k) =$

$\min_{u_k \in U(x_k)} \{E_{\theta_k} [l(x_k, u_k, \theta_k) + G_{k+1}(x_{k+1})]\},$

 if $x_k \neq x_F$;

end

end



$$\textcircled{1} G_k^*(x_k = s_1) = \min\{(2 + 0) * 0.9 + (2 + 0) * 0.1\}$$



Expected Cost Planning

Initialize G values of all states to finite values;

while not converge **do**

for all the states x **do**

$G(x_F) = 0$;

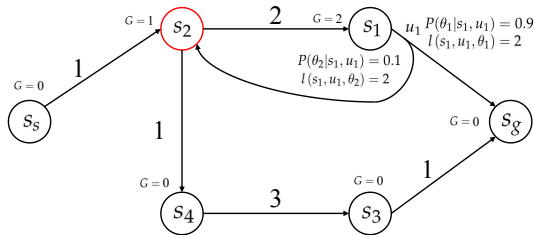
$G_k(x_k) =$

$\min_{u_k \in U(x_k)} \{E_{\theta_k} [l(x_k, u_k, \theta_k) + G_{k+1}(x_{k+1})]\}$,

 if $x_k \neq x_F$;

end

end



① $G_k^*(x_k = s_2) =$
 $\min\{(1 + 0) * 1.0, (2 + 2) * 1.0\}$



Expected Cost Planning

Initialize G values of all states to finite values;

while not converge **do**

for all the states x **do**

$G(x_F) = 0$;

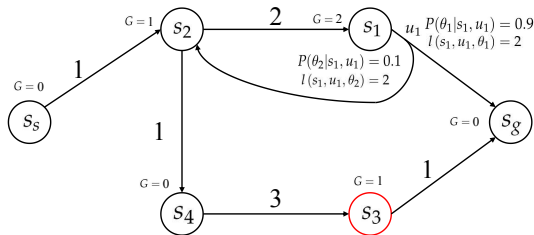
$G_k(x_k) =$

$\min_{u_k \in U(x_k)} \{E_{\theta_k} [l(x_k, u_k, \theta_k) + G_{k+1}(x_{k+1})]\}$,

 if $x_k \neq x_F$;

end

end



$$\textcircled{1} G_k^*(x_k = s_3) = \min\{(1 + 0) * 1.0\}$$



Expected Cost Planning

Initialize G values of all states to finite values;

while not converge **do**

for all the states x **do**

$G(x_F) = 0$;

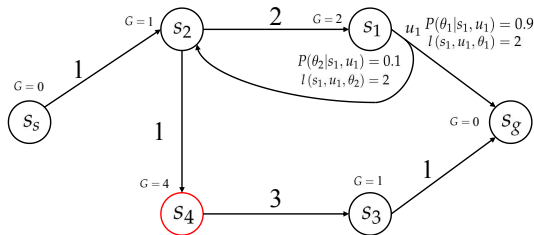
$G_k(x_k) =$

$\min_{u_k \in U(x_k)} \{E_{\theta_k} [l(x_k, u_k, \theta_k) + G_{k+1}(x_{k+1})]\}$,

 if $x_k \neq x_F$;

end

end



$$\textcircled{1} \quad G_k^*(x_k = s_4) = \min\{(3 + 1) * 1.0\}$$



Expected Cost Planning

Initialize G values of all states to finite values;

while not converge **do**

for all the states x **do**

$G(x_F) = 0$;

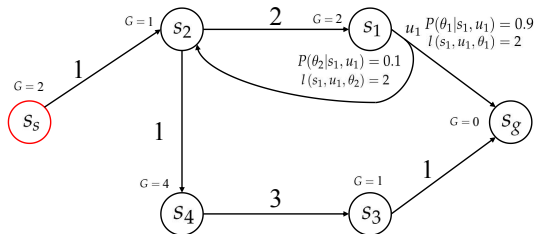
$G_k(x_k) =$

$\min_{u_k \in U(x_k)} \{E_{\theta_k} [l(x_k, u_k, \theta_k) + G_{k+1}(x_{k+1})]\}$,

 if $x_k \neq x_F$;

end

end



$$\textcircled{1} G_k^*(x_k = s_s) = \min\{(1 + 1) * 1.0\}$$



Expected Cost Planning

Initialize G values of all states to finite values;

while not converge **do**

for all the states x **do**

$G(x_F) = 0$;

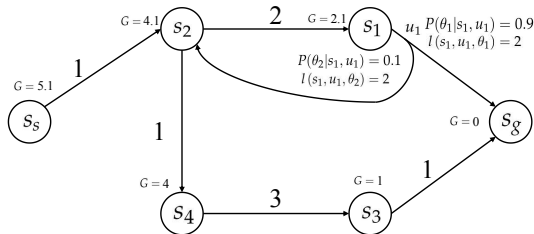
$G_k(x_k) =$

$\min_{u_k \in U(x_k)} \{E_{\theta_k} [l(x_k, u_k, \theta_k) + G_{k+1}(x_{k+1})]\}$,

 if $x_k \neq x_F$;

end

end

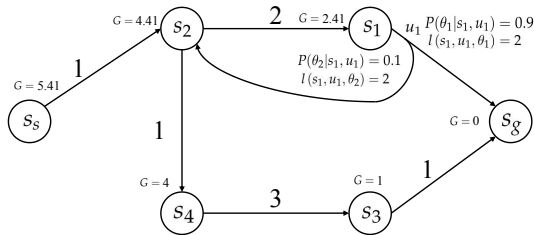


① after second iteration



Expected Cost Planning

```
Initialize G values of all states to finite values;  
while not converge do  
  for all the states  $x$  do  
     $G(x_F) = 0$ ;  
     $G_k(x_k) =$   
       $\min_{u_k \in U(x_k)} \{E_{\theta_k} [l(x_k, u_k, \theta_k) + G_{k+1}(x_{k+1})]\},$   
    if  $x_k \neq x_F$ ;  
  end  
end
```



- ① after third iteration
- ② when will the algorithm converge ?
- ③ how to improve the left algorithm ?
- ④ how to prove the convergence ?
- ⑤ how to get the best plan π^* ?



Pros/Cons of expected cost planning

- probabilistic optimal
 - reflect average performance
 - a specific execution maybe not optimal
- require a distribution of uncertainties
- harder to compute than normal paths
 - iterate over the entire map (states)
 - affected by initialization and iteration order



RTDP algorithm:

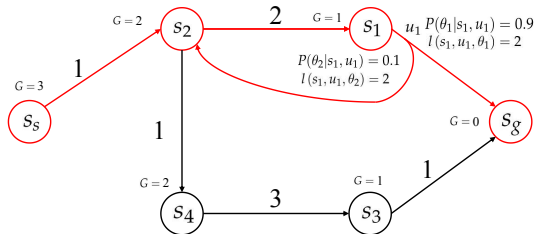
- ① Initialize G values of all states to admissible values;
- ② Follow greedy policy picking outcomes at random until goal is reached;
- ③ Backup all states visited on the way;
- ④ Reset to x_s and repeat 2-4 until all states on the current greedy policy have Bellman errors $< \Delta$, where $\Delta(x_k) = \|G(x_k) - G(x_{k+1})\|$;

Advantages:

- ① very efficient alternative to Value Iteration
- ② does NOT compute values of all states
- ③ focuses computations on states that are relevant



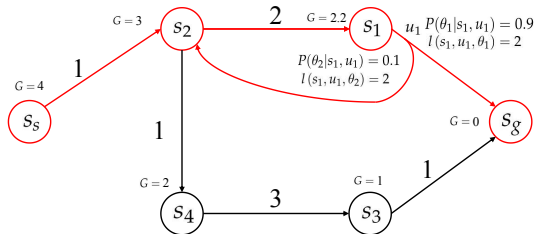
- 1 Initialize G values of all states to admissible values;
- 2 Follow greedy policy picking outcomes at random until goal is reached;
- 3 Backup all states visited on the way;
- 4 Reset to x_s and repeat 2-4 until all states on the current greedy policy have Bellman errors $< \Delta$, where $\Delta(x_k) = \|G(x_k) - G(x_{k+1})\|$;



- initial trajectory with a heuristic



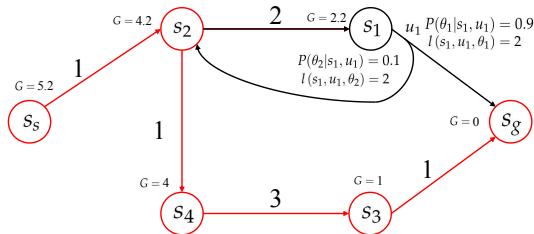
- 1 Initialize G values of all states to admissible values;
- 2 Follow greedy policy picking outcomes at random until goal is reached;
- 3 Backup all states visited on the way;
- 4 Reset to x_s and repeat 2-4 until all states on the current greedy policy have Bellman errors $< \Delta$, where $\Delta(x_k) = \|G(x_k) - G(x_{k+1})\|$;



- backup all states visited on the way



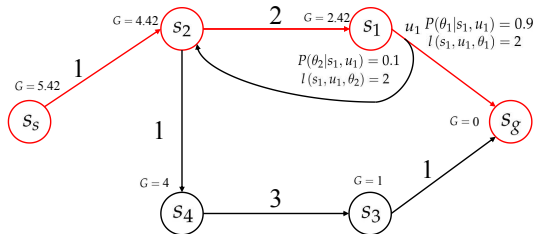
- 1 Initialize G values of all states to admissible values;
- 2 Follow greedy policy picking outcomes at random until goal is reached;
- 3 Backup all states visited on the way;
- 4 Reset to x_s and repeat 2-4 until all states on the current greedy policy have Bellman errors $< \Delta$, where $\Delta(x_k) = \|G(x_k) - G(x_{k+1})\|$;



- greedy policy and backup all states



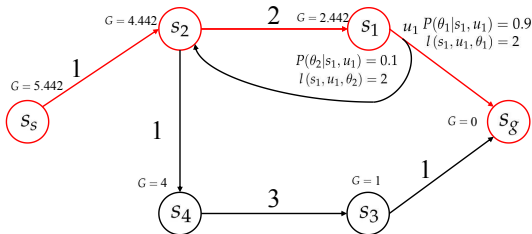
- 1 Initialize G values of all states to admissible values;
- 2 Follow greedy policy picking outcomes at random until goal is reached;
- 3 Backup all states visited on the way;
- 4 Reset to x_s and repeat 2-4 until all states on the current greedy policy have Bellman errors $< \Delta$, where $\Delta(x_k) = \|G(x_k) - G(x_{k+1})\|$;



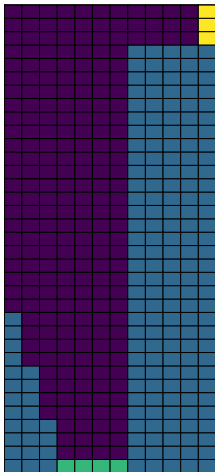
- greedy policy and backup all states



- 1 Initialize G values of all states to admissible values;
- 2 Follow greedy policy picking outcomes at random until goal is reached;
- 3 Backup all states visited on the way;
- 4 Reset to x_s and repeat 2-4 until all states on the current greedy policy have Bellman errors $< \Delta$, where $\Delta(x_k) = \|G(x_k) - G(x_{k+1})\|$;



- greedy policy and backup all states



Grid Map

- ① $X = \{(x, y) \mid 0 \leq x \leq 11, 0 \leq y \leq 34\}$
- ② $X_I = \{\text{green grids}\}$
 $X_F = \{\text{yellow grids}\}$
- ③ $U = \{(\ddot{x}, \ddot{y}) \mid \ddot{x} \in \{0, \pm 1\}, \ddot{y} \in \{0, \pm 1\}\}$
- ④ $\Theta = \{\theta_1, \theta_2\}$
 - $\theta_1: f(\mathbf{x}_{k+1}, \mathbf{x}_k, \mathbf{u}_k) = \mathbf{x}_k \quad p_1 = 0.1$
 - $\theta_2: f(\mathbf{x}_{k+1}, \mathbf{x}_k, \mathbf{u}_k) = \mathbf{x}_{k+1} \quad p_1 = 0.9$
- ⑤ $l(\mathbf{x}_k, \mathbf{x}_k, \theta_k) = -1$
- ⑥ **Find an optimal plan from X_I to X_F**



Thanks For Your Attention!