بسم الله الرحمن الرحیم

هوش مصنوعی پیشرفته

فصل ۱۷

اتخاذ تصمیم‌های پیچیده

Making Complex Decisions

کاظم فولادی
دانشکده مهندسی برق و کامپیوتر
دانشگاه تهران

http://courses.fouladi.ir/aai

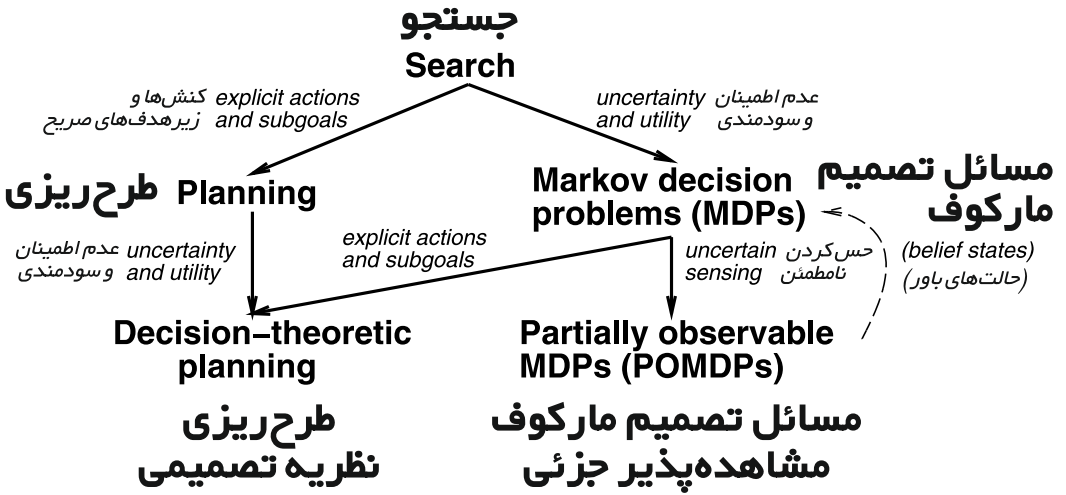
انواع تصمیم

انواع تصمیم	
تصمیم ساده *Simple Decision*	**تصمیم پیچیده** *Complex Decision*
تصمیم‌های تک‌ضرب *One-Shot*	**تصمیم‌های چندمرحله‌ای** *Multi-Stage*
تصمیم بر روی یک کنش	تصمیم بر روی یک دنباله از کنش‌ها

انواع تصمیم

تصمیم‌های پیچیده

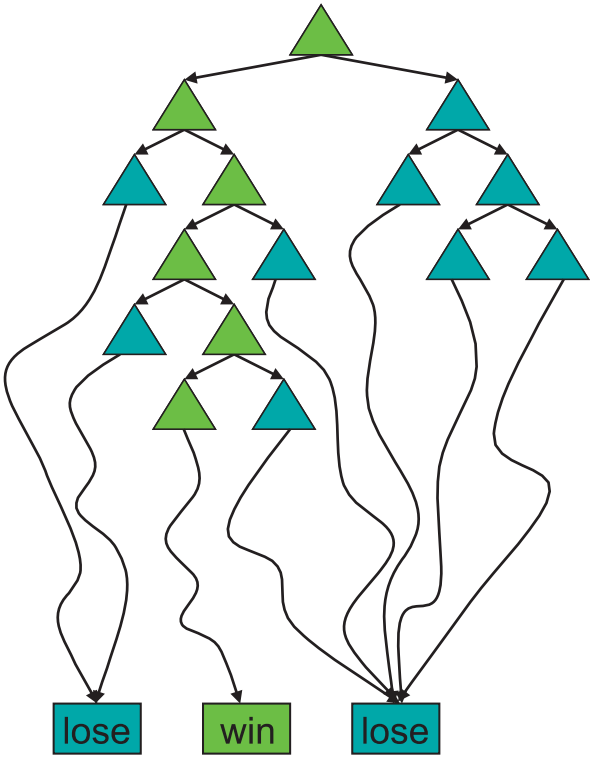
مسائل تصمیم‌گیری



بازی قطعی تک‌بازیکنه

تصمیم‌گیری پیچیده‌ی تک‌عاملی

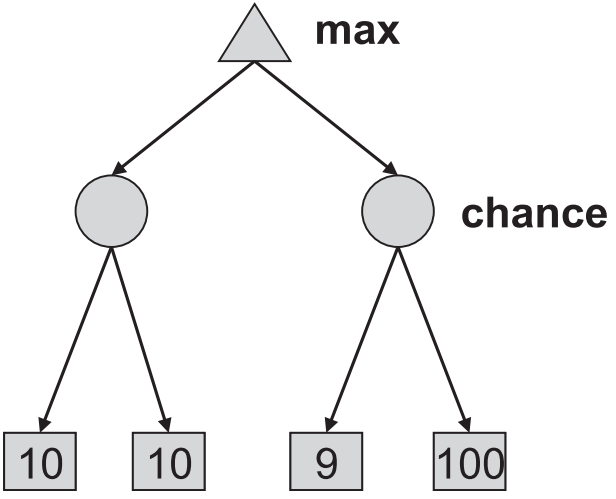
DETERMINISTIC SINGLE-PLAYER



بازی اتفاقی تک‌بازیکنه

امید ماکزیمم

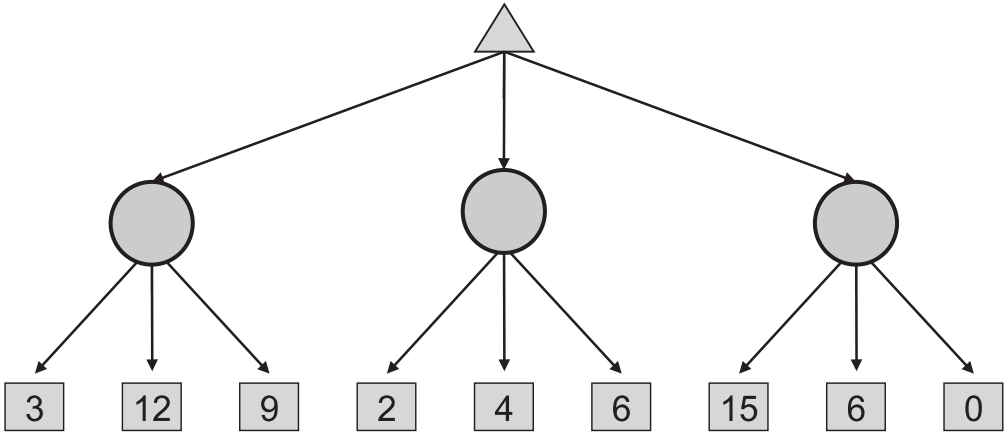
EXPECTIMAX



بازی اتفاقی تک‌بازیکنه

کمیت‌های امید ماکزیمم

STOCHASTIC SINGLE-PLAYER



هوش مصنوعی

اتخاذ تصمیم‌های پیچیده

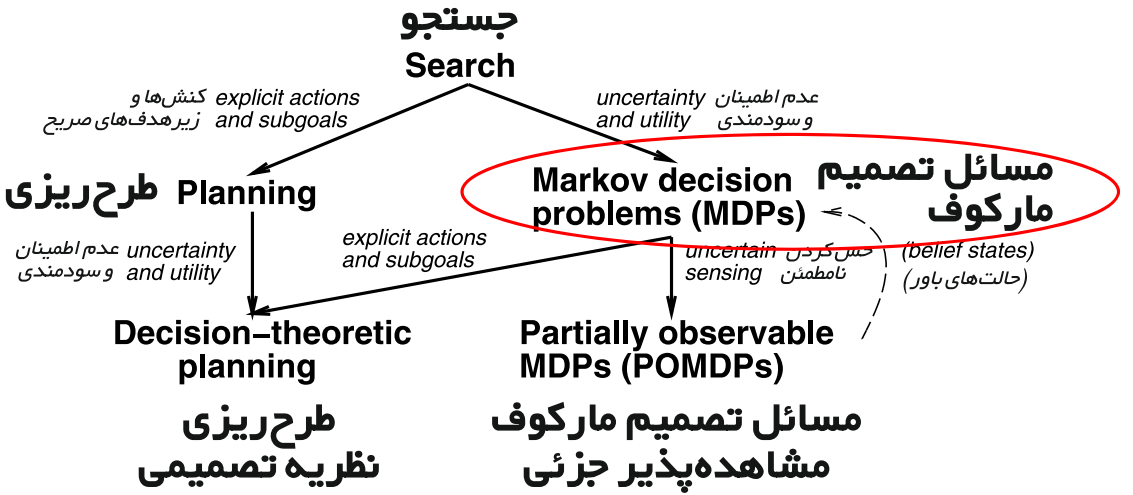
۱

مسائل تصمیم‌گیری ترتیبی

مسائل تصمیم‌گیری

مسائل تصمیم‌گیری مارکوف

MARKOV DECISION PROBLEMS (MDPs)



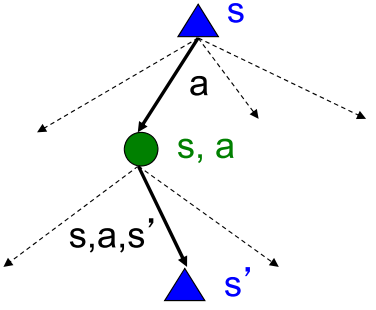
Andrey Markov (1856-1922)

مسائل/فرآیند تصمیم‌گیری مارکوف

تعریف

MARKOV DECISION PROBLEMS / PROCESS (MDPS)

مؤلفه‌های تعریف یک MDP		
تابع پاداش *Reward Function*	**مدل گذر** *Transition Model*	**حالت آغازین** *Initial State*
$R(s)$ $R(s,a)$ $R(s,a,s')$	$T(s,a,s')$	s_0



States $s \in S$, actions $a \in A$

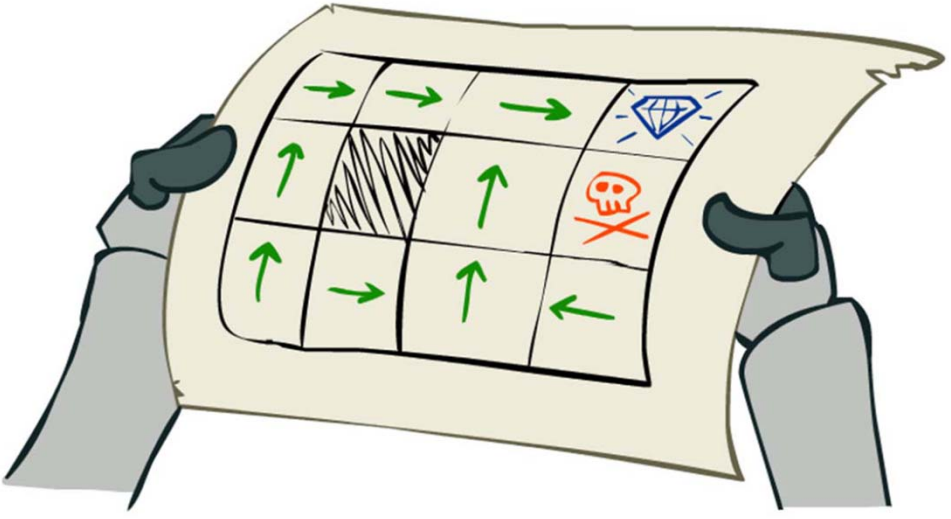
Model $T(s,a,s') \equiv P(s'|s,a)$ = probability that a in s leads to s'

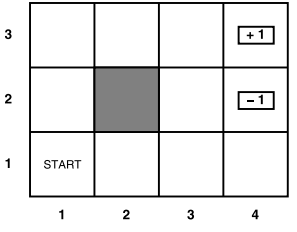
فرض می‌شود که گذرها **مارکوف** باشند، یعنی:
احتمال رسیدن از حالت فعلی به حالت دیگر، فقط به **حالت فعلی** وابسته است و به تاریخچه‌ی حالت‌های اخیر وابسته نیست.

مسائل تصمیم‌گیری مارکوف

مثال

MARKOV DECISION PROBLEMS (MDPS)



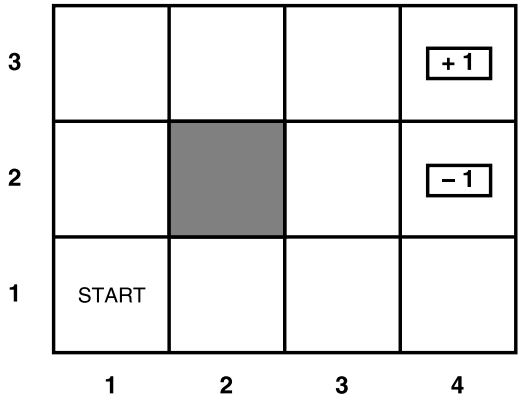


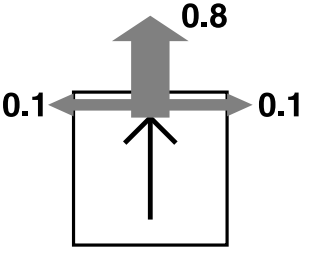
مسائل تصمیم‌گیری مارکوف

مثال

MARKOV DECISION PROBLEMS (MDPS)

حرکت از START و رسیدن به هدف +1





توزیع احتمال نتیجه‌ی کنش حرکت مستقیم

حالت‌ها و کنش‌ها States $s \in S$, actions $a \in A$ = {Right, Left, Down, Up}

مدل Model $T(s,a,s') \equiv P(s'|s,a)$ = probability that a in s leads to s'

تابع پاداش Reward function $R(s)$ (or $R(s,a)$, $R(s,a,s')$)

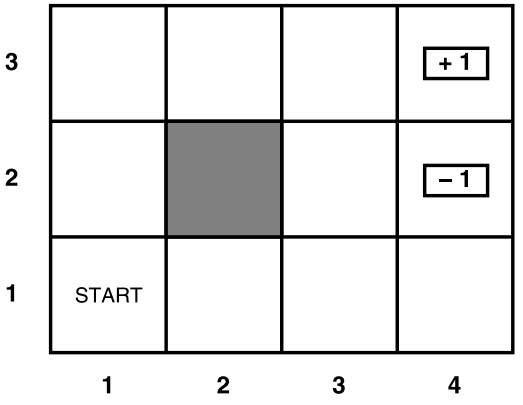
$$= \begin{cases} -0.04 & \text{(جریمه/پنالتی کوچک) برای حالت‌های ناپایانی} \\ \pm 1 & \text{برای حالت‌های پایانی} \end{cases}$$

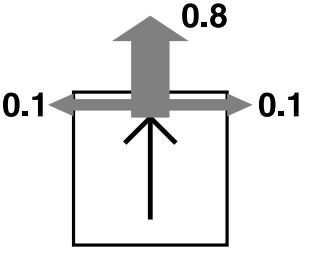
مسائل تصمیم‌گیری مارکوف

مثال

MARKOV DECISION PROBLEMS (MDPS)

حرکت از START و رسیدن به هدف +1





توزیع احتمال نتیجه‌ی کنش حرکت مستقیم

راه‌حل برای محیط قطعی	[Up, Up, Right, Right, Right]
راه‌حل برای محیط اتفاقی	?

وقتی محیط اتفاقی است، اثر کنش‌ها قطعی نخواهد بود.

⇓

یک ترتیب ثابت از کنش‌ها مسئله را حل نمی‌کند؛
(زیرا ممکن است عامل وارد حالتی شود که نتیجه‌ی کنش آن نبوده است)

⇓

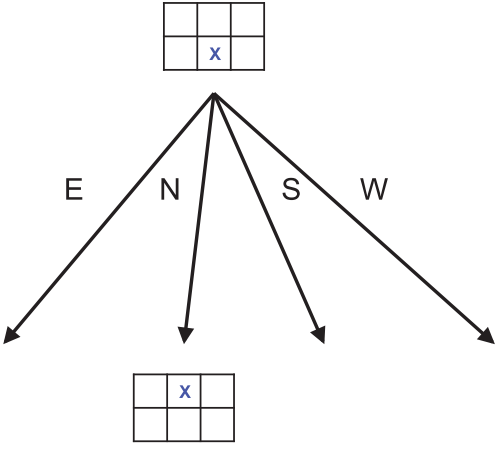
در راه‌حل باید: کنش عامل به‌ازای همه‌ی حالت‌هایی که ممکن است به آنها برسد، تعیین شده باشد (سیاست).

مسائل تصمیم‌گیری مارکوف

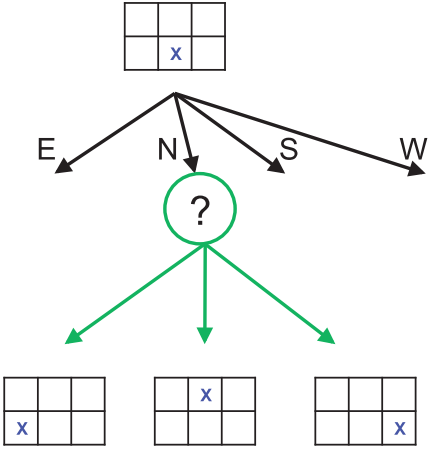
مثال

MARKOV DECISION PROBLEMS (MDPS)

محیط قطعی
Deterministic Environment



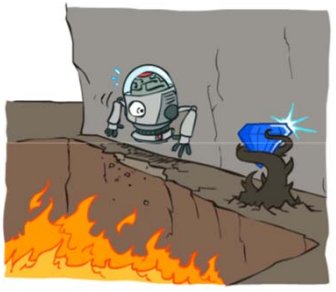
محیط اتفاقی
Stochastic Environment

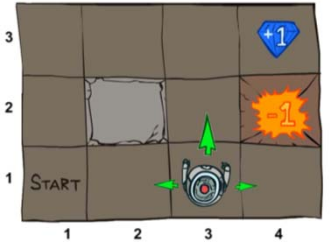


مسائل تصمیم‌گیری مارکوف

مثال

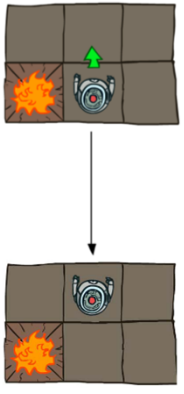
MARKOV DECISION PROBLEMS (MDPS)

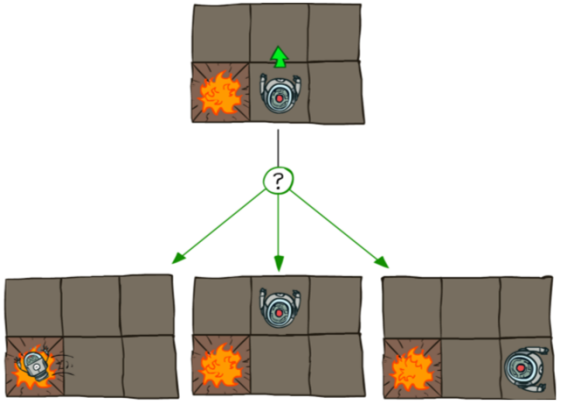




محیط قطعی
Deterministic Environment

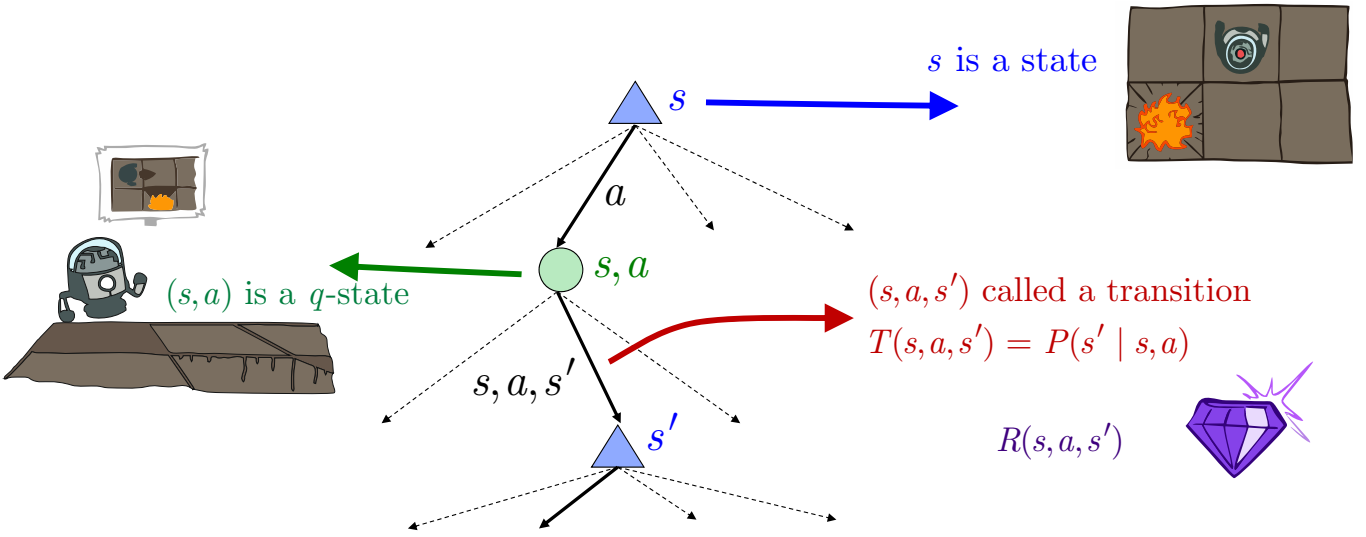
محیط اتفاقی
Stochastic Environment





درخت جستجوی MDP

MDP Search Tree



مسائل تصمیم‌گیری مارکوف

سیاست

POLICY

سیاست
Policy

کنش پیشنهادی برای هر حالت ممکن s

$$\pi(s)$$

وقتی عامل دارای یک سیاست کامل باشد،
بدون توجه به نتیجه‌ی هر کنش، همیشه می‌داند برای مرحله‌ی بعدی باید چه کاری انجام دهد.

سیاست بهینه
Optimal Policy

بهترین کنش برای هر حالت ممکن s

$$\pi^*(s)$$

سیاست بهینه سیاستی است که به **بالاترین مقدار امید سودمندی** برسد.

مسائل تصمیم‌گیری مارکوف

مسائل جستجو در مقابل مسائل تصمیم‌گیری مارکوف

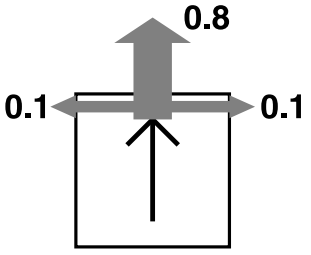
SEARCH PROBLEMS VS. MDPS

مسئله‌ی جستجو *Search Problem*	مسئله‌ی تصمیم‌گیری مارکوف *Markov Decision Problem (MDP)*
هدف: یافتن یک دنباله‌ی بهینه	هدف: یافتن یک سیاست بهینه $\pi(s)$
	یعنی: بهترین کنش برای هر حالت ممکن s (زیرا نمی‌توانیم پیش‌بینی کنیم که هر کنش قطعاً به کدام حالت منجر می‌شود)

مسائل تصمیم‌گیری مارکوف

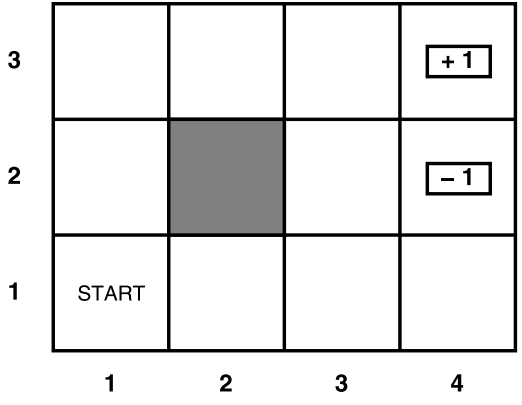
حل کردن مسائل تصمیم‌گیری مارکوف: مثال

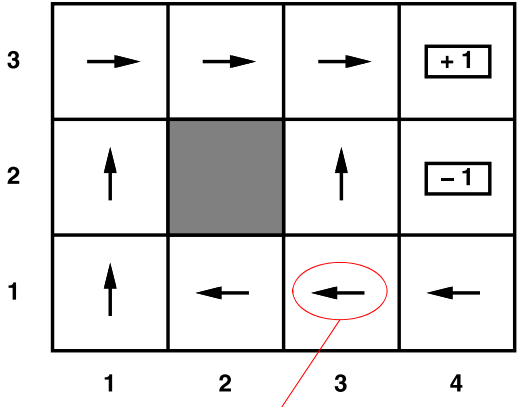
SOLVING MDPS



توزیع احتمال نتیجه‌ی کنش حرکت مستقیم

Optimal policy when state penalty $R(s)$ is –0.04:





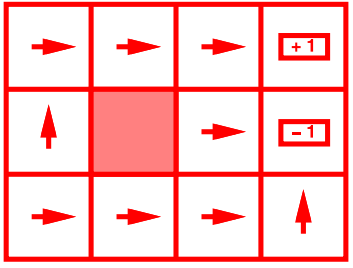
تصمیم محافظه‌کارانه: در مقابل راه میان‌بر و ریسکی ورود به 1- راه طولانی‌تر را انتخاب می‌کند.

مسائل تصمیم‌گیری مارکوف

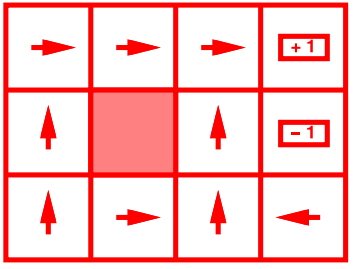
ریسک و پاداش

RISK AND REWARD

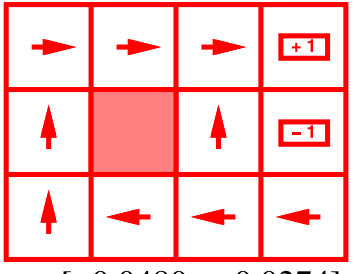
توازن دقیق **ریسک** و **پاداش**، از خصوصیات MDP است.



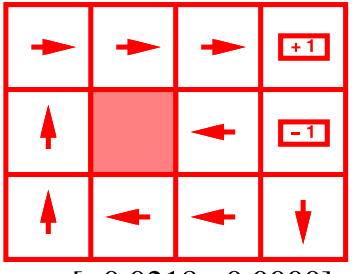
$r = [-\infty : -1.6284]$



$r = [-0.4278 : -0.0850]$



$r = [-0.0480 : -0.0274]$



$r = [-0.0218 : 0.0000]$

سیاست‌های بهینه با بازه‌های مختلف برای پاداش تغییر می‌کند.

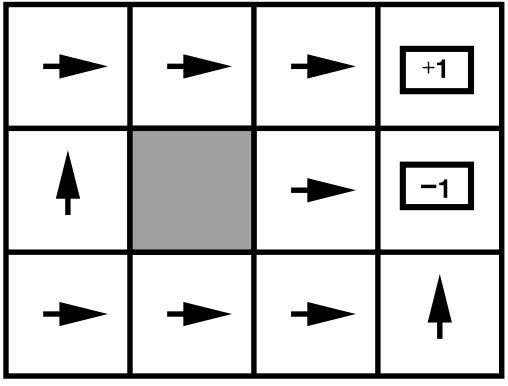
توازن بین تغییرات پاداش و ریسک، به مقدار پاداش حالت‌های غیرپایانی بستگی دارد.

مسائل تصمیم‌گیری مارکوف

توازن ریسک و پاداش: مثال ۱

RISK AND REWARD

سیاست بهینه برای پاداش‌های منفی (جریمه) بزرگ



$$R(s) < -1.6284$$

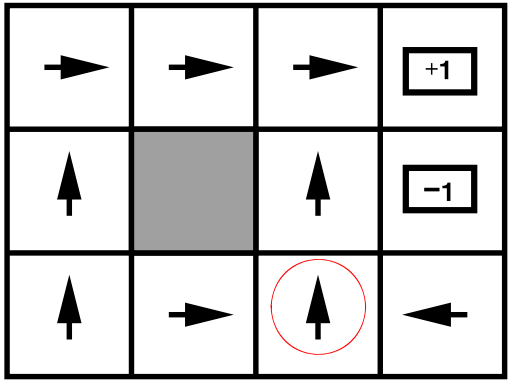
شرایط آن‌قدر سخت می‌شود که عامل برای یافتن نزدیک‌ترین خروجی تلاش می‌کند.
حتی اگر خروجی دارای مقدار 1- باشد.

مسائل تصمیم‌گیری مارکوف

توازن ریسک و پاداش: مثال ۲

RISK AND REWARD

سیاست بهینه برای پاداش‌های منفی (جریمه) متوسط



$$-0.4278 < R(s) < -0.0850$$

زندگی برای عامل ناخوشایند است و عامل کوتاه‌ترین مسیر برای رسیدن به 1+ را انتخاب می‌کند؛
و ریسک (خطر) افتادن ناگهانی در 1- را نیز می‌پذیرد.

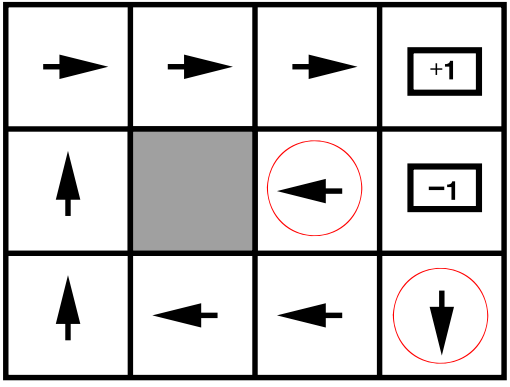
(به‌طور خاص برای زمانی که عامل راه میان‌بری را از (3,1) انتخاب کند.)

مسائل تصمیم‌گیری مارکوف

توازن ریسک و پاداش: مثال ۳

RISK AND REWARD

سیاست بهینه برای پاداش‌های منفی (جریمه) کوچک



$$-0.0221 < R(s) < 0$$

زندگی عامل با کمی افسردگی همراه است:
در این حالت سیاست بهینه هیچ ریسکی را نمی‌پذیرد.

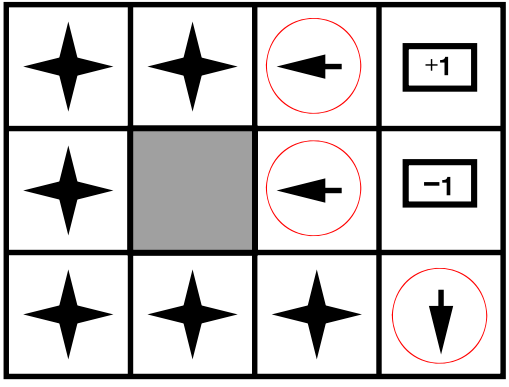
(در (4,1) و (3,2) عامل سعی می‌کند با دور شدن از حالت 1- به‌طور ناگهانی در حالت 1- گرفتار نشود؛ حتی اگر به دیوار بخورد.)

مسائل تصمیم‌گیری مارکوف

توازن ریسک و پاداش: مثال ۴

RISK AND REWARD

سیاست بهینه برای پاداش‌های مثبت



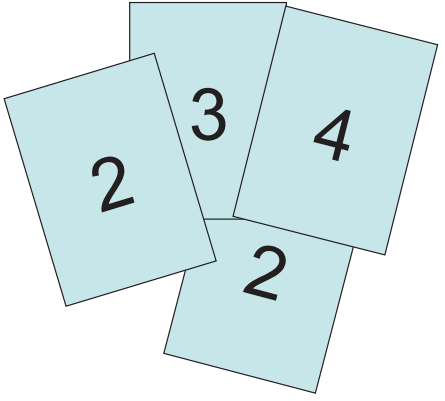
$$R(s) > 0$$

زندگی عامل لذت‌بخش است:
عامل از هر دو حالت خروج امتناع می‌کند.

(تازمانی که کنش‌ها در (4,1) و (3,2) و (3,3) مطابق شکل فوق باشد، هر سیاستی بهینه است
و عامل به‌خاطر عدم ورود به حالت‌های پایانی، در کل پاداشی نامتناهی به‌دست می‌آورد.)

مسائل تصمیم‌گیری مارکوف

مثال: بازی High-Low



- سه نوع کارت وجود دارد: 2, 3, 4
- بی‌نهایت دسته کارت (deck) داریم، تعداد 2ها دو برابر است.
- با نشان دادن 3 شروع می‌کنید.
- پس از برداشتن کارت جدید، کلمه‌ی high یا low را می‌گویید.
- کارت جدید برگردانده می‌شود.
- اگر درست گفته باشید، امتیازی به میزان عدد روی کارت جدید می‌برید.
- در صورت برخورد با گره (Tie) کاری انجام نمی‌شود (عدد مساوی).
- اگر اشتباه گفته باشید، بازی پایان می‌یابد.

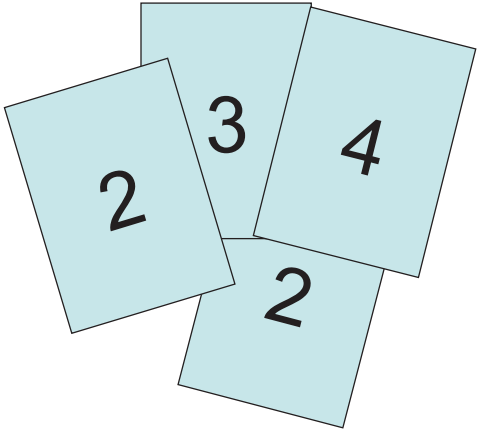
تفاوت با ExpectiMax

- #۱: در هر حرکت، پاداشی دریافت می‌شود (نه فقط در پایان بازی!). ❋
- #۲: ممکن بازی برای همیشه ادامه یابد.

مسائل تصمیم‌گیری مارکوف

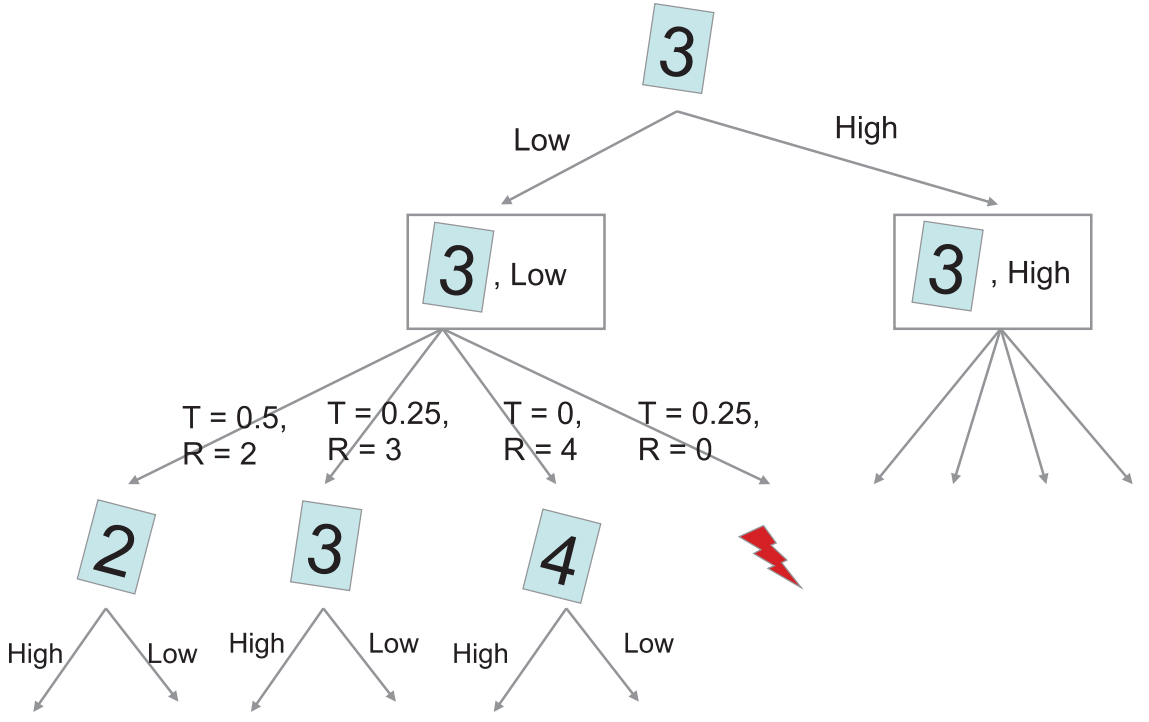
مثال: بازی High-Low

- **States:** 2, 3, 4, done
- **Actions:** High, Low
- **Model:** $T(s, a, s')$:
  - $P(s' = 4 \mid 4, \text{Low}) = 1/4$
  - $P(s' = 3 \mid 4, \text{Low}) = 1/4$
  - $P(s' = 2 \mid 4, \text{Low}) = 1/2$
  - $P(s' = \text{done} \mid 4, \text{Low}) = 0$
  - $P(s' = 4 \mid 4, \text{High}) = 1/4$
  - $P(s' = 3 \mid 4, \text{High}) = 0$
  - $P(s' = 2 \mid 4, \text{High}) = 0$
  - $P(s' = \text{done} \mid 4, \text{High}) = 3/4$
  - …
- **Rewards:** $R(s, a, s')$:
  - Number shown on s' if $s \neq s'$
  - 0 otherwise
- **Start:** 3



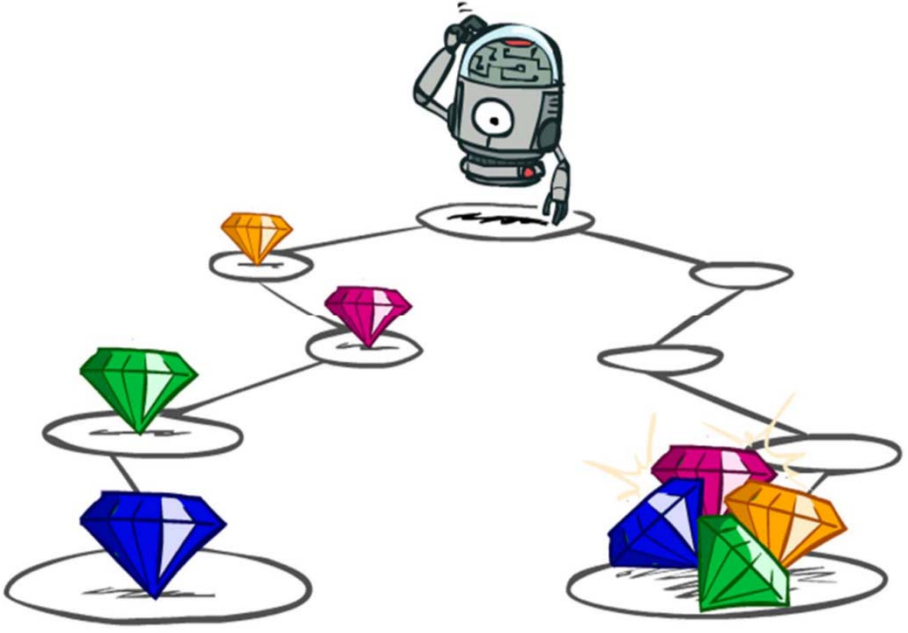
مسائل تصمیم‌گیری مارکوف

مثال: بازی High-Low



سودمندی دنباله‌های حالت

UTILITY OF STATE SEQUENCES



سودمندی دنباله‌های حالت

UTILITY OF STATE SEQUENCES

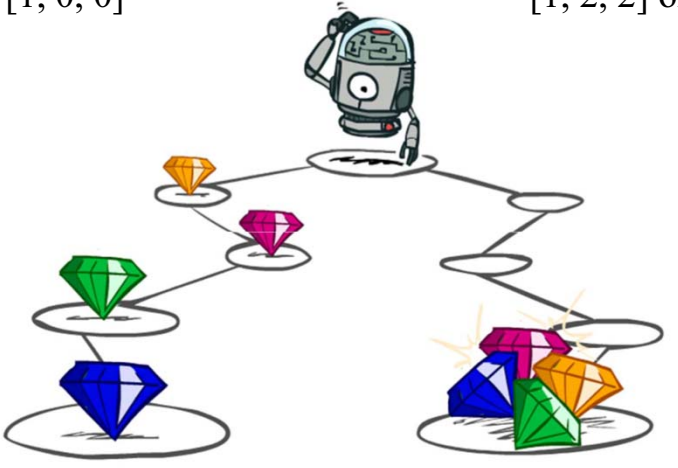
یک عامل باید کدام ترجیحات را بر روی دنباله‌های پاداش در نظر بگیرد؟

بیشتر یا کمتر؟
More or Less?

[1, 2, 2] or [2, 3, 4]

اکنون یا دیرتر؟
Now or Later?

[0, 0, 1] or [1, 0, 0]



سودمندی دنباله‌های حالت

UTILITY OF STATE SEQUENCES

برای تعیین ترجیح‌ها بین دنباله‌های حالت‌ها

ترجیح‌های ایستان را بر روی دنباله‌های پاداش در نظر می‌گیریم:

stationary preferences

$$[r, r_0, r_1, r_2, \ldots] \succ [r, r'_0, r'_1, r'_2, \ldots] \Leftrightarrow [r_0, r_1, r_2, \ldots] \succ [r'_0, r'_1, r'_2, \ldots]$$

قضیه:
با فرض ایستان بودن
برای انتساب سودمندی به دنباله‌های حالت
(ترکیب پاداش‌ها در طول زمان) فقط دو راه وجود دارد:

تابع سودمندی جمعی

1) *Additive* utility function:

$$U([s_0, s_1, s_2, \ldots]) = R(s_0) + R(s_1) + R(s_2) + \cdots$$

تابع سودمندی تخفیف‌یافته

2) *Discounted* utility function:

$$U([s_0, s_1, s_2, \ldots]) = R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \cdots$$

where γ is the discount factor فاکتور تخفیف

سودمندی دنباله‌های حالت

تخفیف

DISCOUNTING

- مستدل است که: مجموع پاداش‌ها را ماکزیمم کنیم.
- مستدل است که پاداش‌های کنونی را نسبت به پاداش‌های آینده ترجیح بدهیم.

⇐ **یک راه‌حل**: ارزش پاداش‌ها به‌صورت نمایی کاهش یابد (مفهوم تخفیف).

1

ثروت فعلی

γ

ثروت در گام بعدی

γ^2

ثروت در دو گام بعد

سودمندی دنباله‌های حالت

تخفیف

DISCOUNTING

❑ چگونگی اعمال تخفیف؟

هر زمان که یک سطح پایین می‌آییم،
یک بار در تخفیف ضرب می‌کنیم.

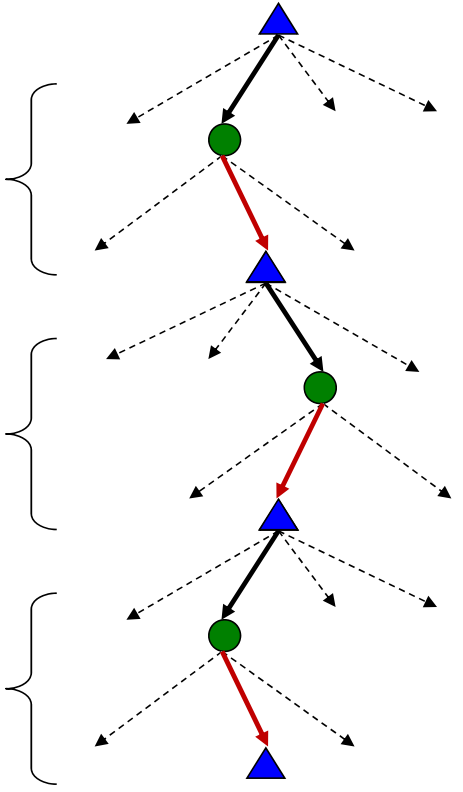
❑ چرا تخفیف؟

- پاداش‌های زودتر، احتمالاً سودمندی بیشتری دارند تا پاداش‌های دیرتر.
- همچنین به همگرایی الگوریتم کمک می‌کند.

1

γ

γ^2



مثال:

$$\gamma = 0.5$$

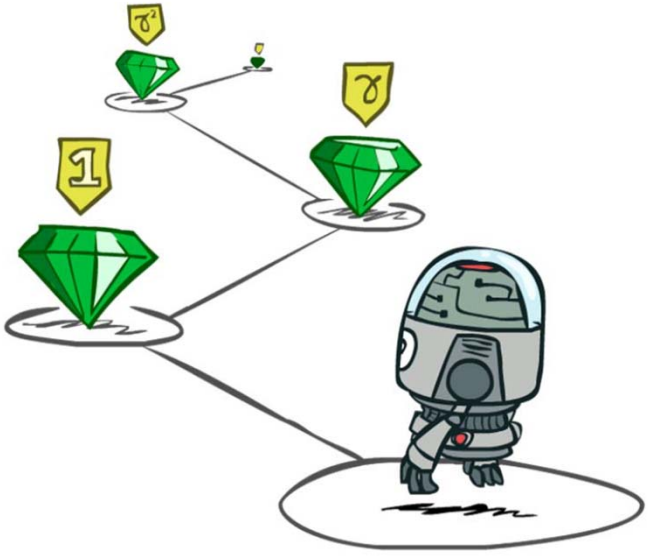
$$U([1,2,3]) = 1 \times 1 + 0.5 \times 2 + 0.25 \times 3$$

$$U([1,2,3]) < U([3,2,1])$$

سودمندی دنباله‌های حالت

ترجیح‌های ایستان

STATIONARY PREFERENCES



$$[r, r_0, r_1, r_2, \ldots] \succ [r, r'_0, r'_1, r'_2, \ldots] \Leftrightarrow [r_0, r_1, r_2, \ldots] \succ [r'_0, r'_1, r'_2, \ldots]$$

سودمندی حالت‌ها

UTILITY OF STATES

سودمندی یک حالت *Utility of a state*	ارزش یک حالت *Value of a state*	امید مجموع (تخفیف‌یافته) پاداش‌ها (تا رسیدن به پایان) با فرض کنش‌های بهینه	$U(s)$

با داشتن سودمندی حالت‌ها، انتخاب بهترین کنش صرفاً یک MEU است:
امید سودمندی مابعدهای بی‌واسطه را ماکزیمم کنید.

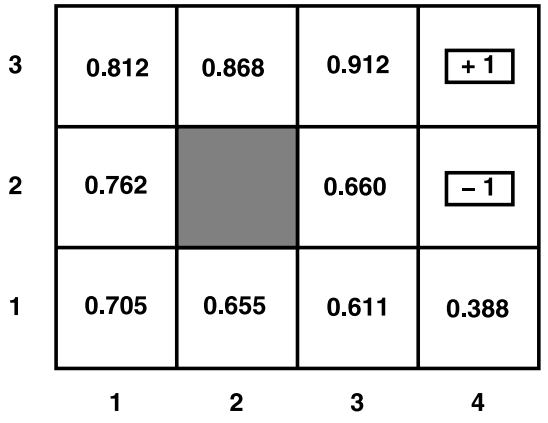
$$\pi^*(s) = \underset{a \in A(s)}{\operatorname{argmax}} \sum_{s'} P(s' \mid s, a) U(s')$$

سودمندی حالت‌ها

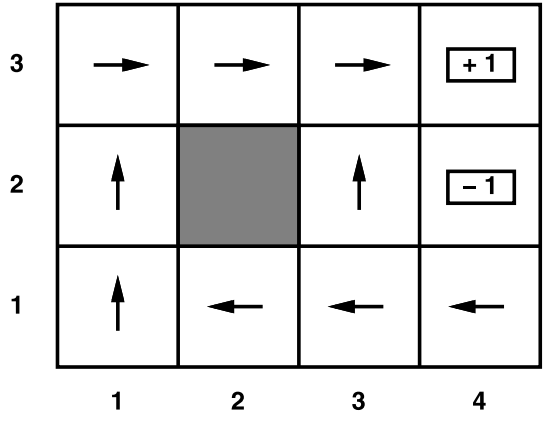
مثال

UTILITY OF STATES

Utility of States:



Optimal policy when state penalty $R(s)$ is –0.04:



نتایج پس از همگرایی الگوریتم

سودمندی حالت‌ها

برخورد با مشکل طول عمر نامتناهی

UTILITY OF STATES

مشکل: طول عمرهای نامتناهی ⇐ سودمندی‌های جمعی بی‌نهایت می‌شوند!

راه‌حل‌ها:

افق متناهی *Finite Horizon*	پایان در زمان ثابت T ⇐ سیاست **ناایستان:** $\pi(s)$ به زمان باقیمانده وابسته می‌شود.
حالت(های) جاذب *Absorbing State(s)*	با احتمال یک، عامل سرانجام می‌میرد، به‌ازای هر سیاست π ⇐ امید سودمندی هر حالت متناهی می‌شود.
تخفیف *Discounting*	$\gamma < 1, \quad R(s) \le R_{\max},$ $U[s_0, \ldots, s_\infty] = \sum_{t=0}^{\infty} \gamma^t R(s_t) \le R_{\max}/(1-\gamma)$ γ کوچکتر ⇐ افق کوتاه‌تر
ماکزیمم‌سازی بهره‌ی سیستم *Maximize System Gain*	بهره‌ی سیستم = پاداش متوسط برای هر گام زمانی

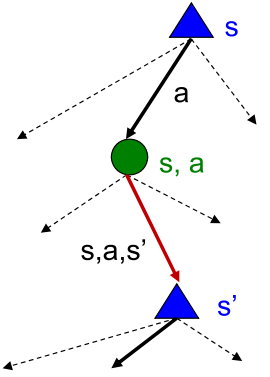
قضیه: سیاست بهینه پس از گذار اولیه، بهره‌ی ثابتی دارد.

مسائل تصمیم‌گیری مارکوف

کمیت‌های بهینه

MDP'S OPTIMAL QUANTITIES

- The value (utility) of a state s:
  $V^*(s)$ = expected utility starting in s and acting optimally
- The value (utility) of a q-state (s,a):
  $Q^*(s,a)$ = expected utility starting out having taken action a from state s and (thereafter) acting optimally
- The optimal policy:
  $\pi^*(s)$ = optimal action from state s



s is a state

(s, a) is a q-state

(s,a,s') is a transition

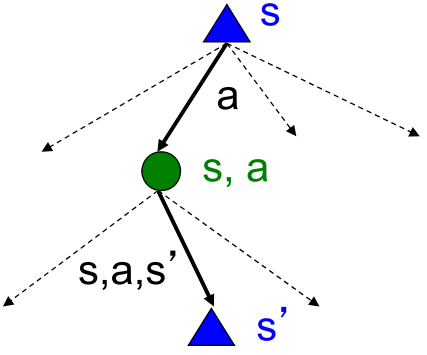
امید سودمندی با شروع از s و کنش به‌صورت بهینه	**ارزش حالت** *Value of a State*	$V^*(s)$
امید سودمندی با شروع از s و اجرای کنش a در آن و کنش به‌صورت بهینه پس از آن	**ارزش حالت q** *Value of a q-State*	$Q^*(s,a)$
کنش بهینه از حالت s	**ارزش حالت** *Value of a State*	$\pi^*(s)$

مسائل تصمیم‌گیری مارکوف

ارزش حالت‌ها

MDP'S VALUES OF STATES

Fundamental operation: compute the (EXPECTIMAX) value of a state
- Expected utility under optimal action
- Average sum of (discounted) rewards
- This is just what EXPECTIMAX computed!



تعریف‌های بازگشتی ارزش:

$$V^*(s) = \max_a Q^*(s,a)$$

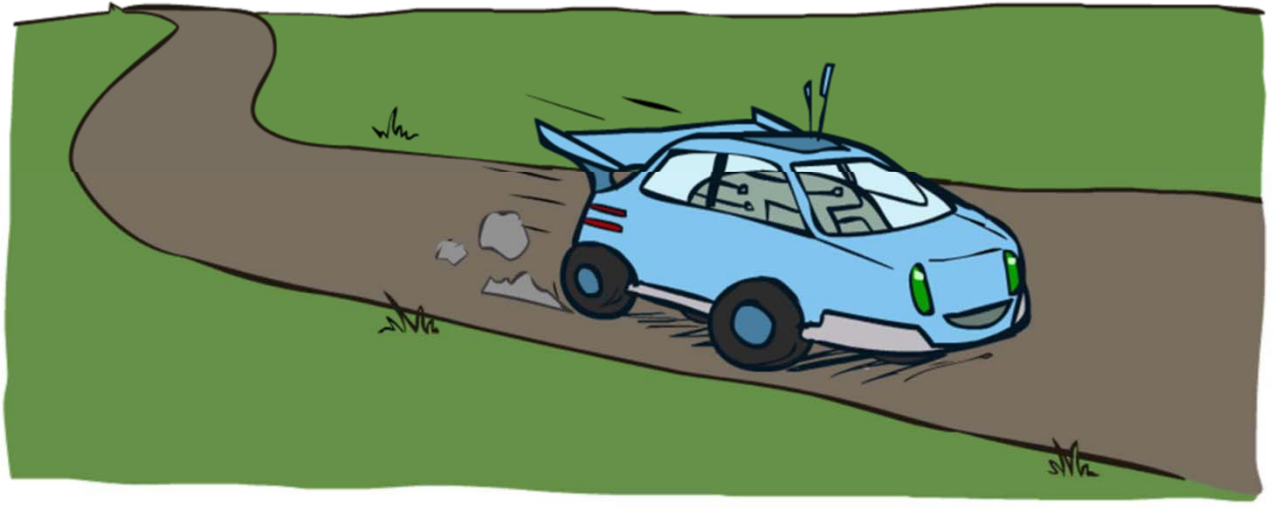
$$Q^*(s,a) = \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^*(s')]$$

$$V^*(s) = \max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^*(s')]$$

مسائل تصمیم‌گیری مارکوف

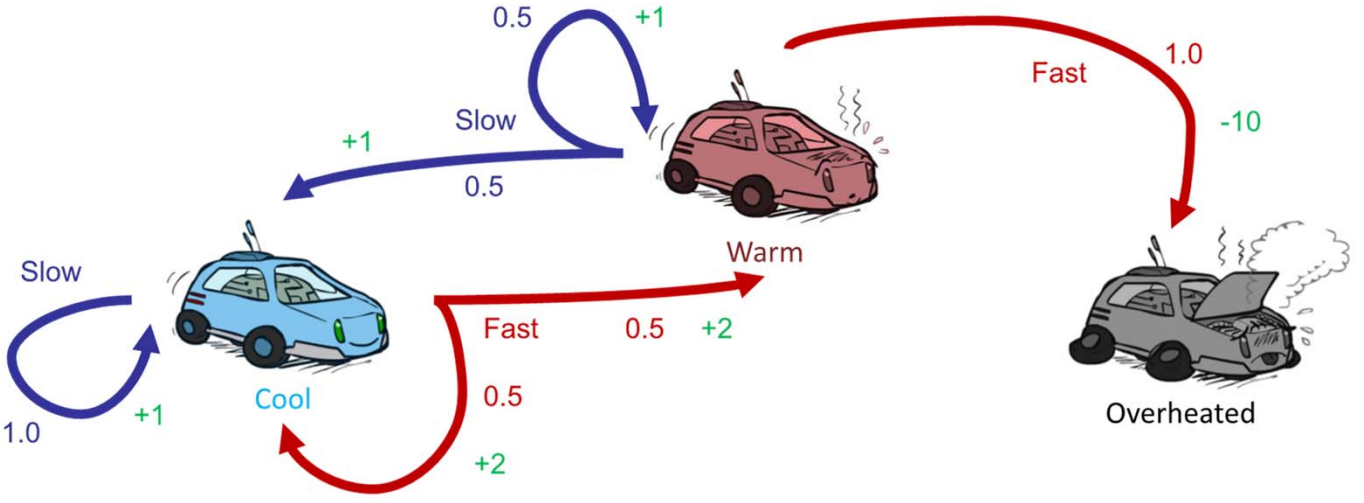
مثال: مسابقه‌ی اتوموبیل‌رانی

RACING



مسائل تصمیم‌گیری مارکوف

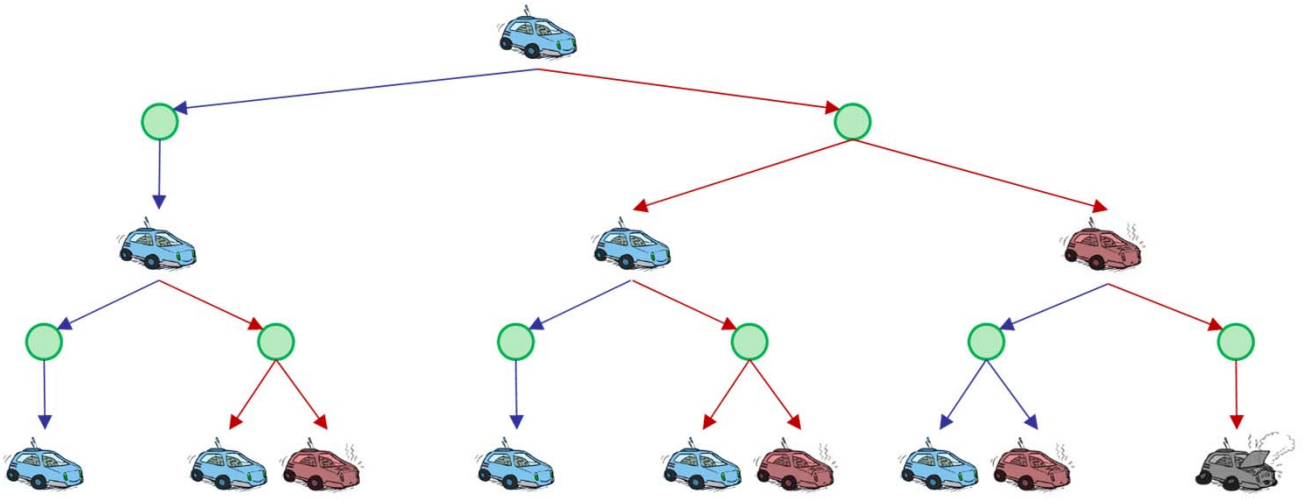
مثال: مسابقه‌ی اتوموبیل‌رانی



- A robot car wants to travel far, quickly
- Three **states**: Cool, Warm, Overheated
- Two **actions**: *Slow*, *Fast*
- Going faster gets double reward

مسائل تصمیم‌گیری مارکوف

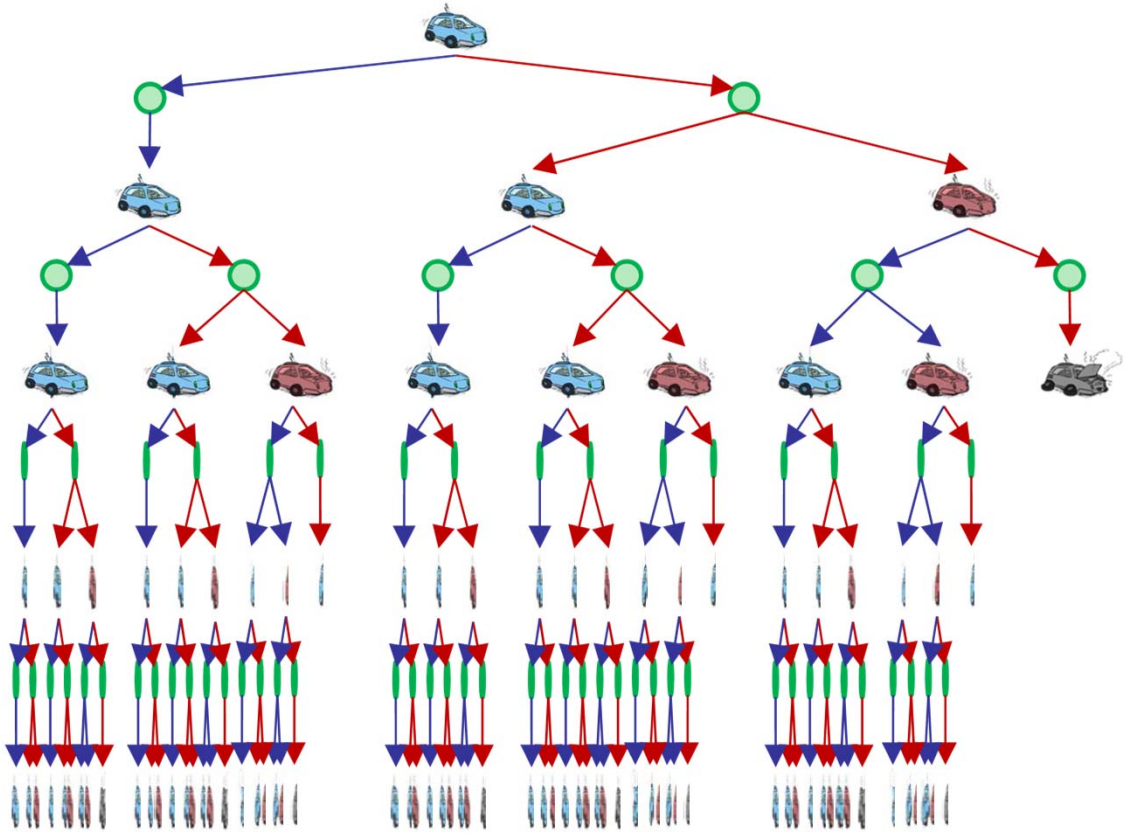
مثال: مسابقه‌ی اتوموبیل‌رانی: درخت جستجو



- Three **states**: Cool, Warm, Overheated
- Two **actions**: *Slow*, *Fast*

مسائل تصمیم‌گیری مارکوف

مثال: مسابقه‌ی اتوموبیل‌رانی: درخت جستجو



مسائل تصمیم‌گیری مارکوف

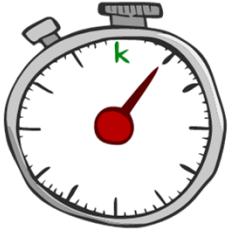
ارزش‌های دارای محدودیت زمانی

TIME-LIMITED VALUES

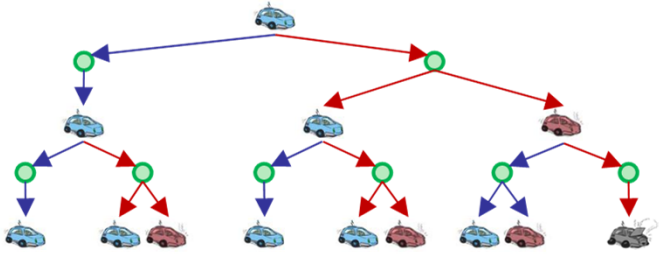
$V_k(s)$ = ارزش بهینه‌ی حالت s
اگر بازی در k گام دیگر خاتمه یابد.

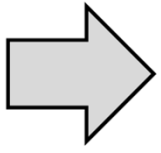
Key idea: time-limited values

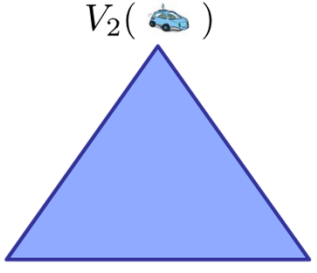
Define $V_k(s)$ to be the optimal value of s
if the game ends in k more time steps



Equivalently, it's what a depth-k EXPECTIMAX would give from s



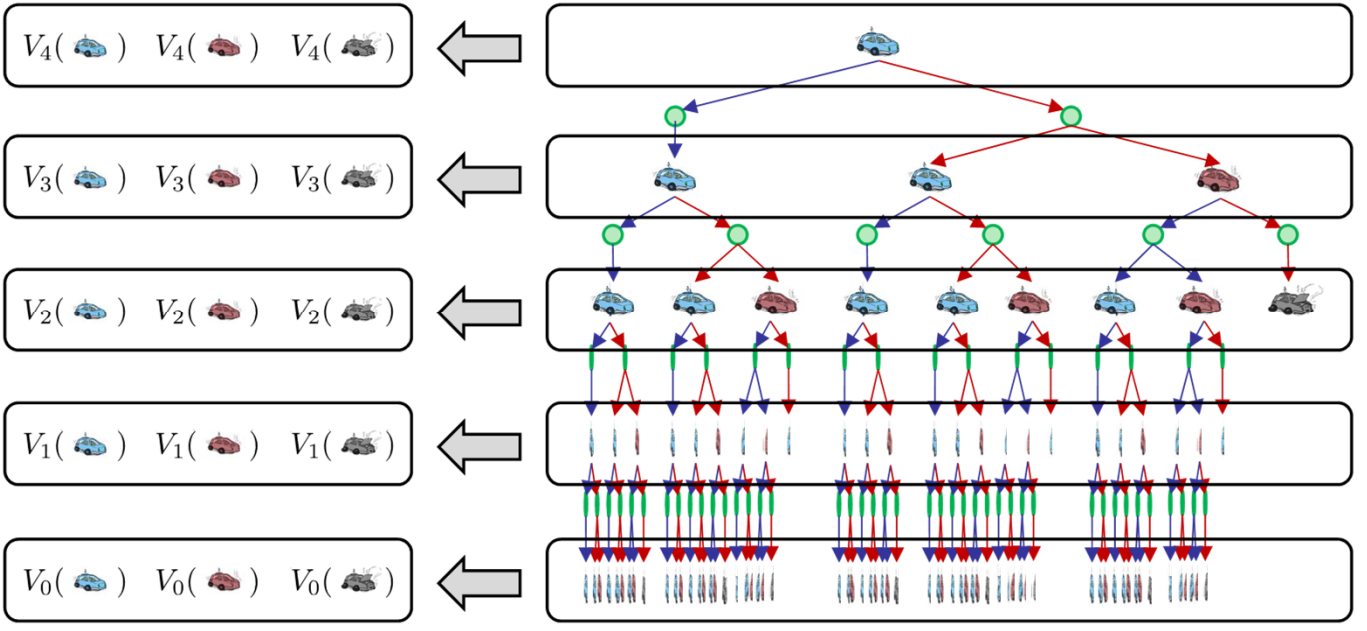




ارزش‌های دارای محدودیت زمانی

محاسبه

TIME-LIMITED VALUES



هوش مصنوعی

اتخاذ تصمیم‌های پیچیده

۲

تکرار ارزش

تکرار ارزش

VALUE ITERATION

تکرار ارزش، در اصل، نسخه‌ای از جستجوی گرافی EXPECTIMAX است، اما:

- **در هر گام پاداش‌هایی وجود دارد** (بر خلاف سودمندی که فقط در گره‌ی پایانی است).
- **از پایین به بالا اجرا می‌شود** (بر خلاف اجرای بازگشتی که بالا به پایین است).

مسئله: چگونه باید طرح‌ریزی کنیم، وقتی کنش‌ها ممکن است شکست بخورند؟
(جستجوی غیرقطعی)

معادله‌ی بلمن

DYNAMIC PROGRAMMING: THE BELLMAN EQUATION

Definition of utility of states leads to a simple relationship among utilities of neighboring states:

expected sum of rewards
= current reward
+ $\gamma \times$ expected sum of rewards after taking best action

Bellman equation (1957):

$$U(s) = R(s) + \gamma \max_a \Sigma_{s'} U(s') T(s, a, s')$$

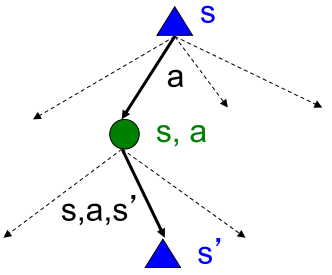
$$\begin{aligned} U(1,1) = -0.04 \\ + \gamma \max\{ & 0.8U(1,2) + 0.1U(2,1) + 0.1U(1,1), && \textit{up} \\ & 0.9U(1,1) + 0.1U(1,2) && \textit{left} \\ & 0.9U(1,1) + 0.1U(2,1) && \textit{down} \\ & 0.8U(2,1) + 0.1U(1,2) + 0.1U(1,1)\} && \textit{right} \end{aligned}$$

One equation per state = n **nonlinear** equations in n unknowns

تکرار ارزش

با افق متناهی بدون تخفیف

VALUE ITERATION FOR FINITE HORIZON AND NO DISCOUNTING



Initialization: $\forall s \in S \quad V_0^*(s) \leftarrow 0$

for $i \leftarrow 1, 2, \ldots, H$

  for all $s \in S$

    for all $a \in A$

      $Q_i^*(s,a) \leftarrow \sum_{s'} T(s,a,s')[R(s,a,s') + V_{i-1}^*(s')]$

    $V_i^*(s) \leftarrow \max_{a \in A} Q_i^*(s,a)$

    $\pi_i^*(s) \leftarrow \arg\max_{a \in A} Q_i^*(s,a)$

$V_i^*(s)$: امید سودمندی (مجموع پاداش‌های انباشته) با شروع از s
و کنش به‌صورت بهینه پس از آن، برای یک افق با i گام زمانی

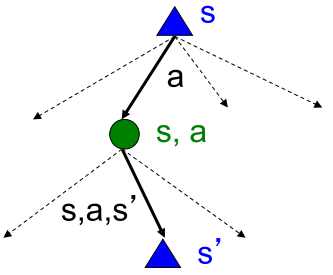
$Q_i^*(s,a)$: امید سودمندی (مجموع پاداش‌های انباشته) با شروع از s و اجرای کنش a در آن
و کنش به‌صورت بهینه پس از آن، برای یک افق با i گام زمانی

چگونه بهینه کنش کنیم؟ سیاست بهینه $\pi_i^*(s)$ را اجرا می‌کنیم، وقتی i گام باقی مانده است.

تکرار ارزش

با افق متناهی با تخفیف

VALUE ITERATION FOR FINITE HORIZON AND WITH DISCOUNTING



Initialization: $\forall s \in S \quad V_0^*(s) \leftarrow 0$

for $i \leftarrow 1, 2, \ldots, H$

  for all $s \in S$

    for all $a \in A$

      $Q_i^*(s,a) \leftarrow \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V_{i-1}^*(s')]$

    $V_i^*(s) \leftarrow \max_{a \in A} Q_i^*(s,a)$

    $\pi_i^*(s) \leftarrow \arg\max_{a \in A} Q_i^*(s,a)$

$V_i^*(s)$: امید سودمندی تخفیف‌یافته (مجموع پاداش‌های انباشته‌ی تخفیف‌یافته) با شروع از s و کنش به‌صورت بهینه پس از آن، برای یک افق با i گام زمانی

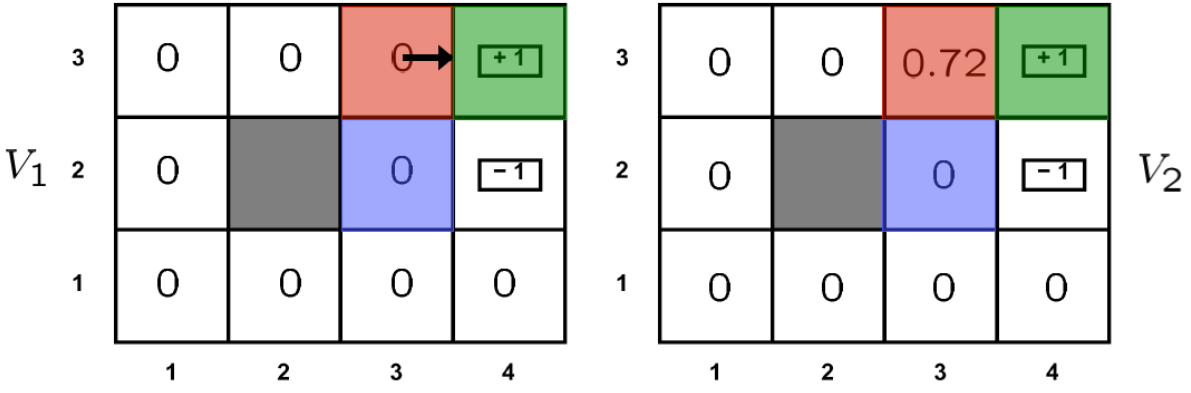
$Q_i^*(s,a)$: امید سودمندی تخفیف‌یافته (مجموع پاداش‌های انباشته‌ی تخفیف‌یافته) با شروع از s و اجرای کنش a در آن و کنش به‌صورت بهینه پس از آن، برای یک افق با i گام زمانی

چگونه بهینه کنش کنیم؟ سیاست بهینه $\pi_i^*(s)$ را اجرا می‌کنیم، وقتی i گام باقی مانده است.

تکرار ارزش

مثال

$$\gamma = 0.9$$



$$V_{i+1}(s) = \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V_i(s') \right]$$

$$V_2(\langle 3,3 \rangle) = \sum_{s'} T(\langle 3,3 \rangle, \text{right}, s') \left[R(\langle 3,3 \rangle) + 0.9 V_1(s') \right]$$

max happens for a=right, other actions not shown

$$= 0.9 \left[0.8 \cdot 1 + 0.1 \cdot 0 + 0.1 \cdot 0 \right]$$

تکرار ارزش

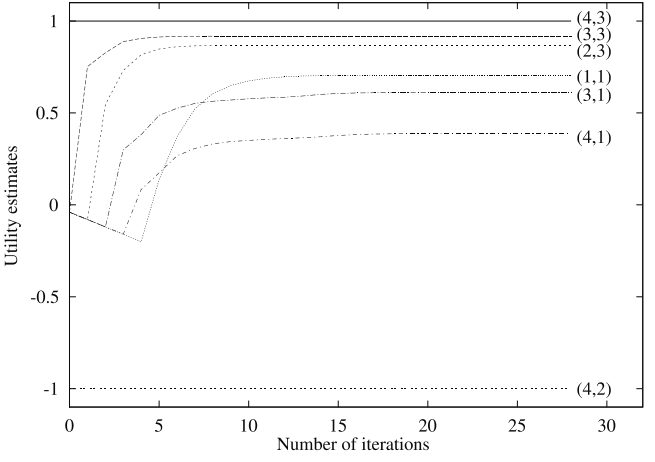
الگوریتم

VALUE ITERATION ALGORITHM

Idea: Start with arbitrary utility values
Update to make them locally consistent with Bellman eqn.
Everywhere locally consistent $\Rightarrow$ global optimality

Repeat for every s simultaneously until "no change"

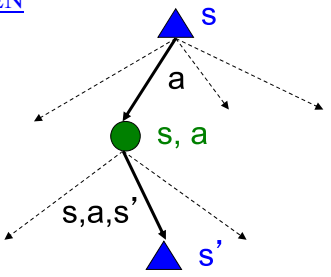
$$U(s) \leftarrow R(s) + \gamma \max_a \sum_{s'} U(s') T(s, a, s') \qquad \text{for all } s$$



تکرار ارزش

با افق متناهی با تخفیف: بازنویسی

VALUE ITERATION FOR FINITE HORIZON AND WITH DISCOUNTING: REWRITTEN



مناسب برای کدنویسی

Initialization: $\forall s \in S \;\; V_0^*(s) \leftarrow 0$
for $i \leftarrow 1, 2, \ldots, H$
  for all $s \in S$
    for all $a \in A$
      $Q_i^*(s,a) \leftarrow \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V_{i-1}^*(s')]$
    $V_i^*(s) \leftarrow \max_{a \in A} Q_i^*(s,a)$

با جایگذاری عبارت Q_i^*

مناسب برای تحلیل همگرایی

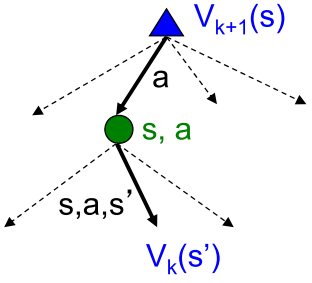
Initialization: $\forall s \in S \;\; V_0^*(s) \leftarrow 0$
for $i \leftarrow 1, 2, \ldots, H$
  for all $s \in S$
    $V_i^*(s) \leftarrow \max_{a \in A} \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V_{i-1}^*(s')]$

صراحتاً دیده می‌شود که الگوریتم تکرار ارزش، مقادیر دنباله‌ی ... $V_0, V_1, V_2,$ را محاسبه می‌کند.

تکرار ارزش

الگوریتم

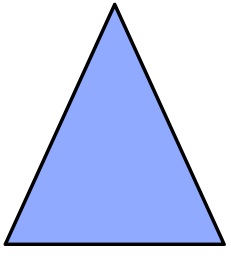
VALUE ITERATION ALGORITHM



- با $V_0(s) \leftarrow 0$ شروع می‌کنیم
  (هیچ گام زمانی باقی نمانده است، یعنی: امید مجموع پاداش‌ها از این به بعد صفر است)
- با داشتن بردار ارزش‌های $V_k(s)$ یک مرحله الگوریتم EXPECTIMAX را برای هر حالت اجرا می‌کنیم.

$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V_k^*(s')]$$

- این کار را تا **همگرایی** تکرار می‌کنیم.



* پیچیدگی الگوریتم برای هر تکرار $O(|S|^2|A|)$ است.

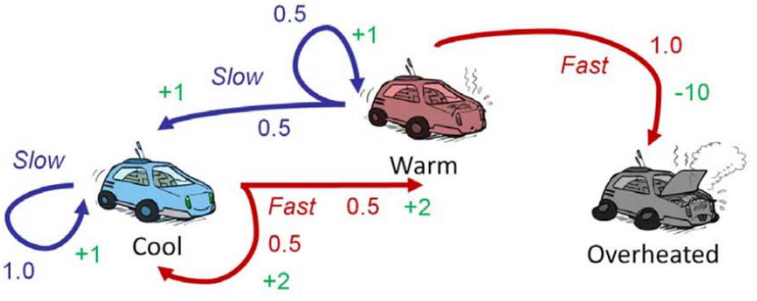
قضیه‌ی همگرایی ثابت می‌کند که ارزش‌ها به مقادیر بهینه‌ی یکتای آنها همگرا می‌شود.
- ایده‌ی اثبات: تقریب‌ها به‌سوی مقادیر بهینه اصلاح می‌شوند.

نکته: سیاست می‌تواند بسیار زودتر از ارزش‌ها همگرا شود.

تکرار ارزش

الگوریتم: مثال

VALUE ITERATION ALGORITHM



Assume no discount!

$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V_k(s')\right]$$

تکرار ارزش

همگرایی الگوریتم

VALUE ITERATION ALGORITHM: CONVERGENCE

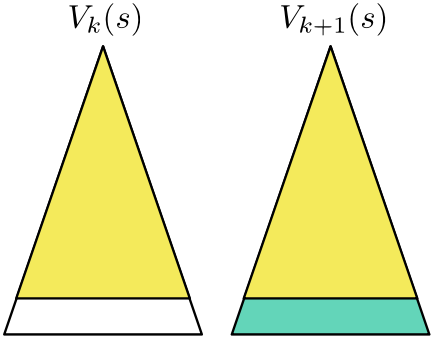
از کجا می‌دانیم بردارهای V_k به سمت همگرایی می‌روند؟

حالت ۱: درخت دارای حداکثر عمق M است.

- در این صورت V_M مقادیر واقعی بدون تقریب را دربردارد.

حالت ۲: اگر نرخ تخفیف کمتر از یک باشد.

- برای هر حالت، V_k و V_{k+1} می‌توانند به‌عنوان نتایج الگوریتم EXPECTIMAX در عمق $k+1$ درخت‌های تقریباً یکسان دیده شود.
- تفاوت در لایه‌ی پایین است:
  V_{k+1} پاداش‌های واقعی را دربردارد، در حالی که V_k صفر است.
- لایه‌ی آخر در بهترین حالت همگی $R_{\max}$ است.
- لایه‌ی آخر در بدترین حالت همگی $R_{\min}$ است.
- اما همه چیز با ضریب γ^k دچار تخفیف می‌شود.
- بنابراین تفاوت V_k و V_{k+1} حداکثر $\gamma^k \max|R|$ است.
- پس با افزایش k مقادیر به سمت همگرایی می‌روند.



تکرار ارزش

قضیه‌ی همگرایی

VALUE ITERATION ALGORITHM: CONVERGENCE THEOREM

Define the max-norm $||U|| = \max_s |U(s)|$,
so $||U - V||$ = maximum difference between U and V

Let U^t and U^{t+1} be successive approximations to the true utility U

Theorem: For any two approximations U^t and V^t

$$||U^{t+1} - V^{t+1}|| \leq \gamma\, ||U^t - V^t||$$

I.e., any distinct approximations must get closer to each other
so, in particular, any approximation must get closer to the true U
and value iteration converges to a unique, stable, optimal solution

Theorem: if $||U^{t+1} - U^t|| < \epsilon$, then $||U^{t+1} - U|| < 2\epsilon\gamma/(1-\gamma)$
I.e., once the change in U^t becomes small, we are almost done.

MEU policy using U^t may be optimal long before convergence of values

تکرار ارزش

قضیه‌ی همگرایی: نتیجه

VALUE ITERATION ALGORITHM: CONVERGENCE THEOREM

الگوریتم تکرار ارزش همگرا می‌شود:
در همگرایی، مقدار بهینه‌ی تابع V^* را برای مسئله‌ی افق نامتناهی تخفیف‌یافته، می‌یابیم
که معادلات بلمن را ارضا می‌کند:

$$\forall s \in S \quad V^*(s) = \max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^*(s')]$$

اکنون مشخص است که برای افق نامتناهی و پاداش‌های تخفیف‌یافته چگونه باید کنش را انتخاب کنیم:

الگوریتم **تکرار ارزش** را فراخوانی می‌کنیم تا به همگرایی برسد.
کنش بهینه بر اساس تابع V^* مشخص می‌شود:

$$\pi^*(s) = \arg\max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^*(s')]$$

نکته: سیاست بهینه با افق نامتناهی، **ایستان** است
(یعنی کنش بهینه در حالت s در تمام زمان‌ها یکسان است)

تکرار ارزش

انتخاب کنش بهینه

VALUE ITERATION ALGORITHM: OPTIMAL ACTION SELECTION

برای انتخاب کنش بهینه در هر حالت:

اگر مقادیر بهینه‌ی V^* را داشته باشیم:

$$\pi^*(s) = \arg\max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^*(s')]$$

اگر مقادیر بهینه‌ی Q^* را داشته باشیم:

$$\pi^*(s) \leftarrow \arg\max_a Q^*(s,a)$$

نتیجه: انتخاب کنش با استفاده از Q ساده‌تر است.

هوش مصنوعی

اتخاذ تصمیم‌های پیچیده

۳

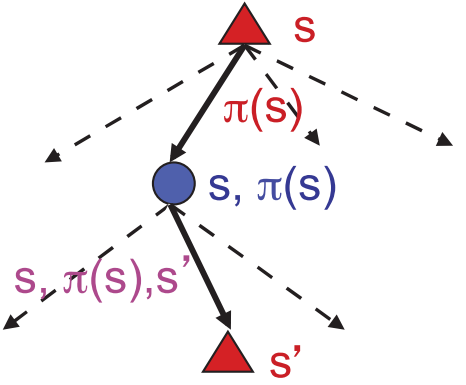
تکرار سیاست

ارزیابی سیاست

POLICY EVALUATION ALGORITHM

می‌توان سودمندی یک حالت را تحت یک سیاست ثابت (نه لزوماً بهینه) محاسبه کرد.

$$V^{\pi}(s) = \sum_{s'} T(s, \pi(s), s')[R(s, \pi(s), s') + \gamma V^{\pi}(s')]$$



سودمندی یک حالت s تحت یک سیاست ثابت π:

$V^{\pi}(s)$: امید سودمندی تخفیف‌یافته (مجموع پاداش‌های انباشته‌ی تخفیف‌یافته) با شروع از s و دنبال کردن سیاست π

ارزیابی سیاست

الگوریتم

POLICY EVALUATION ALGORITHM

روش محاسبه‌ی مقادیر V برای یک سیاست ثابت:

راه‌حل ۱: تغییر معادلات به‌هنگام‌سازی بلمن

$$V_0^\pi(s) = 0$$

$$V_{i+1}^\pi(s) = \sum_{s'} T(s, \pi(s), s')[R(s, \pi(s), s') + \gamma V_i^\pi(s')]$$

راه‌حل ۲: حل دستگاه معادلات خطی حاصل از معادلات زیر!

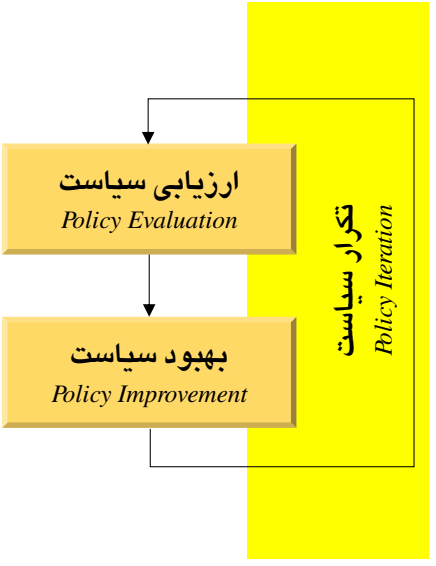
$$V^\pi(s) = \sum_{s'} T(s, \pi(s), s')[R(s, \pi(s), s') + \gamma V^\pi(s')]$$

برای هر حالت s یک معادله نوشته می‌شود.

تکرار سیاست

الگوریتم

POLICY ITERATION ALGORITHM



مقادیر سودمندی را برای یک سیاست ثابت (نه لزوماً بهینه) محاسبه می‌کنیم تا همگرایی حاصل شود.

$$V_{i+1}^{\pi_k}(s) = \sum_{s'} T(s, \pi_k(s), s')[R(s, \pi_k(s), s') + \gamma V_i^{\pi_k}(s')]$$

سیاست را با استفاده از معادله‌ی بلمن و مقادیر همگراشده محاسبه می‌کنیم (مقادیر آینده)

$$V^{\pi}(s) = \sum_{s'} T(s, \pi(s), s')[R(s, \pi(s), s') + \gamma V^{\pi}(s')]$$

one-step lookahead/ Bellman equation

((تکرار تا همگرایی سیاست))

کنش بهینه در هر حالت:

$$\pi_{k+1}(s) = \arg\max_a \sum_{s'} T(s, a, s')[R(s, a, s') + \gamma V^{\pi_k}(s')]$$

تکرار سیاست

الگوریتم

POLICY ITERATION ALGORITHM

Howard, 1960: search for optimal policy and utility values simultaneously

Algorithm:

$\pi \leftarrow$ an arbitrary initial policy
repeat until no change in π
  compute utilities given π
  update π as if utilities were correct (i.e., local MEU)

To compute utilities given a fixed π (value determination):

$$U(s) = R(s) + \gamma \sum_{s'} U(s') T(s, \pi(s), s') \qquad \text{for all } s$$

i.e., n simultaneous **linear** equations in n unknowns, solve in $O(n^3)$

تکرار ارزش ناهمگام

ASYNCHRONOUS VALUE ITERATION

در تکرار ارزش، هر حالت در هر تکرار به‌روزرسانی می‌شد.

در واقع، هر دنباله از به‌روزرسانی‌های بلمن، همگرا خواهد شد
اگر هر حالت بی‌نهایت مرتبه دیده شود.

در واقعیت، می‌توانیم سیاست را به‌ندرت و گاهی در صورت تمایل به‌روزرسانی کنیم،
و هنوز به همگرایی برسیم.

ایده: حالت‌هایی را به‌روزرسانی کنیم که انتظار داریم ارزش آنها تغییر کند:
اگر $|V_{i+1}(s) - V_i(s)|$ بزرگ بود، آنگاه ماقبل‌های s را به‌روزرسانی می‌کنیم.

تکرار سیاست

شکل اصلاح شده

MODIFIED POLICY ITERATION ALGORITHM

Policy iteration often converges in few iterations, but each is expensive

Idea: use a few steps of value iteration (but with π fixed)
starting from the value function produced the last time
to produce an approximate value determination step.

Often converges much faster than pure VI or PI

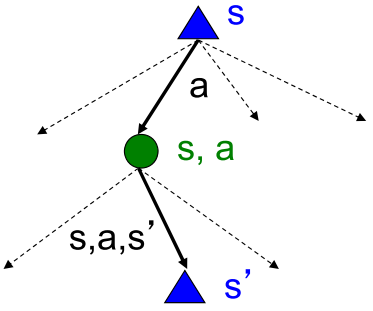
Leads to much more general algorithms where Bellman value updates and Howard policy updates can be performed locally in any order

Reinforcement learning algorithms operate by performing such updates based on the observed transitions made in an initially unknown environment

مسائل تصمیم‌گیری مارکوف

MDPs

مؤلفه‌های تعریف یک MDP		
تابع پاداش *Reward Function*	**مدل گذر** *Transition Model*	**حالت آغازین** *Initial State*
$R(s)$ $R(s,a)$ $R(s,a,s')$	$T(s,a,s')$	s_0
States $s \in S$, actions $a \in A$		



روش‌های راه‌حل		
تکرار ارزش *Value Iteration (VI)*	**تکرار سیاست** *Policy Iteration (PI)*	...

محدودیت‌ها:

* فضای حالت نباید زیاد بزرگ باشد.
* فرض شده است T و R (مدل محیط) معلوم است.

راه‌حل: روش‌های یادگیری تقویتی (Reinforcement Learning)

هوش مصنوعی

اتخاذ تصمیم‌های پیچیده

۴

MDPهای مشاهده‌پذیر جزئی (POMDPs)

MDPهای مشاهده‌پذیر جزئی (POMDPs)

POMDP has an observation model $O(s, e)$ defining the probability that the agent obtains evidence e when in state s

Agent does not know which state it is in

$\Rightarrow$ makes no sense to talk about policy $\pi(s)$!!

Theorem (Astrom, 1965): the optimal policy in a POMDP is a function $\pi(b)$ where b is the belief state (probability distribution over states)

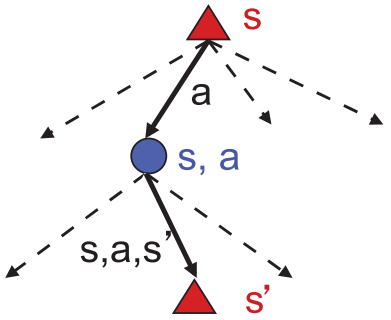
Can convert a POMDP into an MDP in belief-state space, where $T(b, a, b')$ is the probability that the new belief state is b' given that the current belief state is b and the agent does a. I.e., essentially a filtering update step

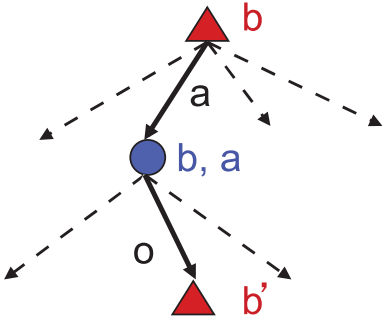
MDPهای مشاهده‌پذیر جزئی (POMDPs)

در POMDPها موارد زیر به MDP اضافه می‌شود:

مؤلفه‌ها:	
مشاهده‌ها *Observations*	**مدل مشاهده** *Observation Model*
O	$P(o \mid s)$ $O(s,o)$

POMDPها به‌نوعی MDPها روی **حالت‌های باور** هستند.





MDPهای مشاهده‌پذیر جزئی (POMDPs)

ویژگی‌ها

Solutions automatically include information-gathering behavior

If there are n states, b is an n-dimensional real-valued vector
$\Rightarrow$ solving POMDPs is very (actually, PSPACE-) hard!

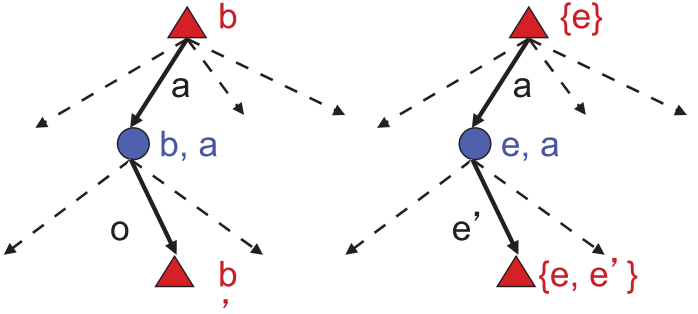
The real world is a POMDP (with initially unknown T and O)

MDPهای مشاهده‌پذیر جزئی (POMDPs)

مثال

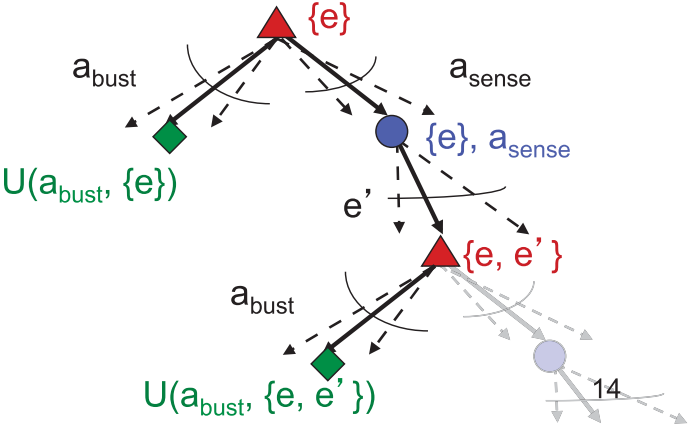
In (static) Ghostbusters:

- Belief state determined by evidence to date $\{e\}$
- Tree really over evidence sets
- Probabilistic reasoning needed to predict new evidence given past evidence



Solving POMDPs

- One way: use truncated ExpectiMax to compute approximate value of actions
- What if you only considered busting or one sense followed by a bust?
- You get a VPI-based agent!



هوش مصنوعی

اتخاذ تصمیم‌های پیچیده

۵

تصمیم‌هایی با عامل‌های چندگانه: نظریه‌ی بازی

تصمیم‌هایی با عامل‌های چندگانه

نظریه‌ی بازی

DECISIONS WITH MULTIPLE AGENTS: GAME THEORY

هوش مصنوعی

اتخاذ تصمیم‌های پیچیده

۶

طراحی مکانیسم

طراحی مکانیزم

MECHANISM DESIGN

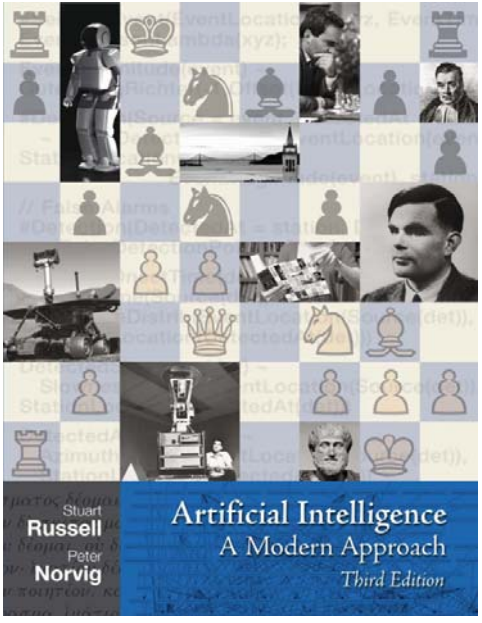
هوش مصنوعی

اتخاذ تصمیم‌های پیچیده

۷

منابع، مطالعه، تکلیف

منبع اصلی



Stuart Russell and Peter Norvig,
Artificial Intelligence: A Modern Approach,
3rd Edition, Prentice Hall, 2010.

Chapter 17

17 MAKING COMPLEX DECISIONS

In which we examine methods for deciding what to do today, given that we may decide again tomorrow.

SEQUENTIAL DECISION PROBLEM

In this chapter, we address the computational issues involved in making decisions in a stochastic environment. Whereas Chapter 16 was concerned with one-shot or episodic decision problems, in which the utility of each action's outcome was well known, we are concerned here with **sequential decision problems**, in which the agent's utility depends on a sequence of decisions. Sequential decision problems incorporate utilities, uncertainty, and sensing, and include search and planning problems as special cases. Section 17.1 explains how sequential decision problems are defined, and Sections 17.2 and 17.3 explain how they can be solved to produce optimal behavior that balances the risks and rewards of acting in an uncertain environment. Section 17.4 extends these ideas to the case of partially observable environments, and Section 17.4.3 develops a complete design for decision-theoretic agents in partially observable environments, combining dynamic Bayesian networks from Chapter 15 with decision networks from Chapter 16.

The second part of the chapter covers environments with multiple agents. In such environments, the notion of optimal behavior is complicated by the interactions among the agents. Section 17.5 introduces the main ideas of **game theory**, including the idea that rational agents might need to behave randomly. Section 17.6 looks at how multiagent systems can be designed so that multiple agents can achieve a common goal.

17.1 SEQUENTIAL DECISION PROBLEMS

Suppose that an agent is situated in the 4×3 environment shown in Figure 17.1(a). Beginning in the start state, it must choose an action at each time step. The interaction with the environment terminates when the agent reaches one of the goal states, marked +1 or –1. Just as for search problems, the actions available to the agent in each state are given by ACTIONS(s), sometimes abbreviated to $A(s)$; in the 4×3 environment, the actions in every state are *Up*, *Down*, *Left*, and *Right*. We assume for now that the environment is **fully observable**, so that the agent always knows where it is.

645