دانشکده ریاضی و آمار

استاد: دکتر منصوری

نیمسال اول ۱۴۰۲-۱۴۰۱

مسائل MDP

هوش مصنوعی

نگارنده: عباسعلی رضائی

۱۳ دی ۱۴۰۱

فهرست مطالب

[1]Non-Deterministic Search

[2]Value Iteration

۱ جستجوی غیر قطعی [۳]

یک دونده را تصور کنید که به پایان اولین ماراتن خود رسیده است. اگرچه به نظر می رسد که او مسابقه را به پایان خواهد رساند اما به هیچ وجه تضمین شده نیست. ممکن است از خستگی بیهوش شود یا لیز بخورد و دچار آسیب جسمی شود. حتی غیر محتمل تر از آن، ممکن است یک زمین لرزهای رخ بدهد و دونده را تنها چند سانتی متر قبل از خط پایان دچار آسیب کند. این گونه احتمال ها، درجه ای از عدم قطعیت را به اعمال دونده اضافه می کند و این عدم قطعیت موضوع بحث بعدی ما هست.
در بخش های قبل، در مورد مشکلات مرسوم جستجو و چگونگی حل آنها صحبت کردیم، سپس الگوی خودمان رو تغییر دادیم تا رقیب ها و سایر عامل ها موجود در جهان که در مسیر به سمت حالت های هدف تأثیر می‌گذارند را در نظر بگیریم.
اکنون، دوباره الگو خود را تغییر می دهیم تا عامل تأثیرگذار دیگری را در نظر بگیریم یعنی پویایی خود جهان. محیطی که یک عامل در آن قرار می گیرد ممکن است اعمال عامل را در معرض **غیر قطعی** بودن قرار دهد. به این معنی که چندین حالت احتمالی وجود دارد که می تواند ناشی از یک عمل انجام شده در یک حالت باشد. این عدم قطعیت ذاتی در بسیاری از بازی های کارتی مانند پوکر یا blackjack ناشی از تصادفی بودن معامله کارت وجود دارد.
چنین مسائلی که در آن جهان دارای درجه ای از عدم قطعیت است، به عنوان مسائل جستجوی غیر قطعی شناخته می شوند و می توانند با الگوهای به نام Markov decision processes یا MDP حل شوند.

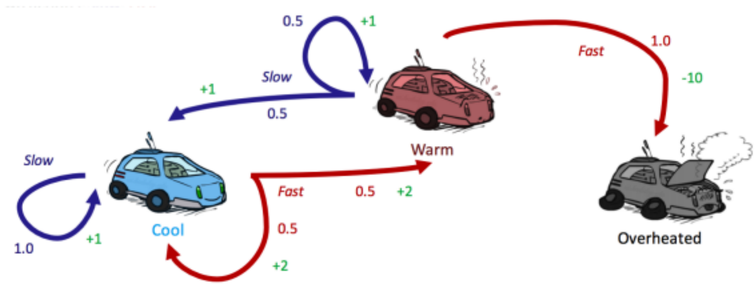
۱.۱ Markov decision processes (فرایند های تصمیم مارکوف)

یک MDP به صورت زیر تعریف می شود:

- یک مجموعه از حالت ها که آن را با S نمایش می دهیم. حالت ها در MDP به همان شکلی است که در مسائل جستجوی مرسوم نشان می دادیم.
- یک مجموعه از اعمال که آن را با A نمایش می دهیم. اعمال در MDP به همان شکلی است که در مسائل جستجوی مرسوم نشان می دادیم.
- یک حالت شروع و شاید یک حالت پایان.
- یک تابع تغییر حالت $T(s, a, s^{'})$. از آنجا که ما امکان اعمال غیر قطعی را معرفی کرده ایم، به راهی نیاز داریم تا احتمال انجام هر عمل معین از هر حالت را مشخص کنیم. تابع تغییر حالت برای یک MDP دقیقاً این کار را انجام می دهد – این تابع احتمال رفتن عامل از حالت $s \in S$ به حالت $s^{'} \in S$ با انجام $a \in A$ را نشان می هد.
- یک تابع پاداش $R(s, a, s^{'})$. به طور معمول، MDP ها با پاداش های کوچک در هر مرحله برای بقای یک عامل، همراه با پاداش های بزرگ برای رسیدن به یک وضعیت پایانی مدل می شوند. پاداش ها ممکن است بسته به اینکه آیا به سود عامل مورد نظر هستند یا خیر، مثبت یا منفی باشند، و هدف عامل به طور طبیعی به دست آوردن حداکثر پاداش ممکن قبل از رسیدن به حالت نهایی است.

[3] Non-Deterministic Search

ساخت MDP برای یک موقعیت کاملا شبیه به ساخت یک گراف فضای حالت برای یک مساله جستجو با چند شرط اضافی است. مثال ماشین مسابقه را در نظر بگیرید:



- حالت ها: Cool،Warm،Overheated
- اعمال: Slow،Fast

درست مانند یک گراف فضای حالت، هر یک از سه حالت با یک گره نشان داده می‌شود و یال‌های آن نشان‌دهنده عمل ها هستند. overheated یک حالت پایانی است، به این خاطر که هنگامی یک عامل "ماشین مسابقه ای" [4] به این حالت می رسد دیگر نمی تواند هیچ عملی را برای بدست آوردن پاداش بیشتر انجام دهد (در MDP به این حالت غرق شدن می گویم و هیچ یال خروجی وجود ندارد). برای عمل های غیر قطعی، یال های متفاوتی از یک حالت خارج می شود به عبارتی وقتی در یک حالت $s \in S$ قرار داریم با انجام عمل $a \in A$ ممکن است به چندین حالت متفاوت برویم.

- **Transition Function**: $T(s,a,s')$
  - $T(cool, slow, cool) = 1$
  - $T(warm, slow, cool) = 0.5$
  - $T(warm, slow, warm) = 0.5$
  - $T(cool, fast, cool) = 0.5$
  - $T(cool, fast, warm) = 0.5$
  - $T(warm, fast, overheated) = 1$
- **Reward Function**: $R(s,a,s')$
  - $R(cool, slow, cool) = 1$
  - $R(warm, slow, cool) = 1$
  - $R(warm, slow, warm) = 1$
  - $R(cool, fast, cool) = 2$
  - $R(cool, fast, warm) = 2$
  - $R(warm, fast, overheated) = -10$

حرکت یک عامل را از طریق حالت‌های مختلف MDP در طول زمان با بازه های زمانی گسسته نشان می‌دهیم، $s_t \in S$ و $a_t \in A$ را به صورت زیر تعریف می کنیم:

- s_t ، حالتی که عامل در زمان t در آن قرار دارد.
- a_t ، عملی که عامل در زمان t انجام می دهد.

[4] racecar

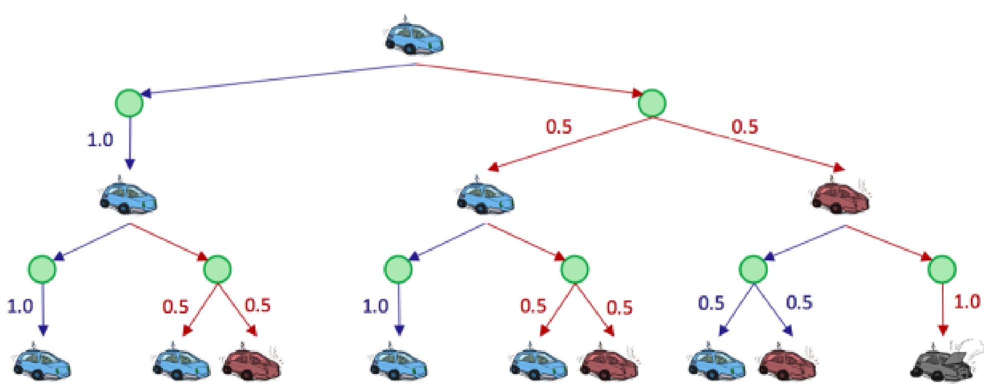
یک عامل در حالت s_0 در زمانی 0 شروع می کند و در هر زمان عملی انجام می دهد. بنابراین حرکت یک عامل از طریق یک MDP می تواند به صورت زیر مدل شود:

$$s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} s_3 \xrightarrow{a_3} \ldots$$

علاوه بر این، با دانستن اینکه هدف یک عامل حداکثر کردن پاداش در تمام زمان ها است، می توانیم به صورت ریاضی به عنوان حداکثر کردن **تابع سودمندی** [5] زیر بیان کنیم:

$$U([s_0, a_0, s_1, a_1, s_2, ...]) = R(s_0, a_0, s_1) + R(s_1, a_1, s_2) + R(s_2, a_2, s_3) + ...$$

فرایندهای تصمیم گیری مارکوف، مانند گراف های فضای حالت، می توانند با درخت های جستجو حل شوند. عدم قطعیت در این درخت‌ها با Q-states که به عنوان حالت‌های عمل نیز شناخته می‌شوند، مدل‌سازی می‌شود که اساساً با گره‌های شانس expectimax یکسان هستند. Q-states از احتمالات برای مدل‌سازی عدم قطعیت استفاده می‌کنند که محیط یک عامل را در یک حالت معین قرار می‌دهد، درست همانطور که گره‌های شانس expectimax از احتمالات برای مدل‌سازی عدم قطعیت استفاده می‌کنند، عامل های رقیب از طریق حرکتی که انتخاب می کنند، عامل ما را در یک وضعیت معین قرار می‌دهد. Q-state با انجام عمل a از حالت s به صورت تاپل (s،a) نشان داده می شود.
در شکل زیر درخت جستجوی باز شده برای ماشین مسابقه ای را مشاهده می کنید، که تا عمق -۲ کوتاه شده است:



گره های سبز نشان دهنده Q-states هستند که در آن ها یک عمل از یک حالت انجام شده است اما هنوز به حالت جانشین ختم نشده است.

عامل ها زمان صفر را در Q-states صرف می کنند، و Q-states صرفاً ساختاری هستند که برای سهولت نمایش و توسعه الگوریتم های MDP ایجاد شده اند.

[5] utility function

۲.۱ Finite horizons and Discount factors

یک مشکل ذاتی MDP ماشین مسابقه ای این هست که هیچ محدودیت زمانی برای تعداد بازه زمانی که یک ماشین مسابقه می تواند عملی انجام دهد و پاداش جمع آوری کند، قرار نداده ایم. با فرمول بندی فعلی ما، عامل همیشه می تواند عمل $a = slow$ را در هر زمانی انتخاب کند ،و به طور ایمن و کارا پاداش نامحدود بدون خطر جوش آوردن را به دست آورد.
به طور خلاصه مشکلی که وجود دارد این است که اگر بازی تا ابد ادامه داشته باشد، آیا پاداش بی نهایت دریافت خواهیم کرد؟
راه حل:

- Finite horizons (همانند جستجوی عمقی محدود)

MDP که یک Finite horizons اعمال می کند اساساً یک "طول عمر" را برای عامل ها تعریف می کند، که به آنها زمان n می دهد تا قبل از خاتمه خودکار، تا آنجا که می توانند پاداش دریافت کنند.

- Discount factors: (ضرایب کاهش)

ضرایب کاهش کمی پیچیده تر هستند و برای مدل سازی یک کاهش نمایی در مقدار پاداش ها در طول زمان معرفی می شوند. انجام عمل a_t از حالت s_t در زمان t به حالت s_{t+1} ختم می شود، و در نتیجه منجر به پاداش $\gamma R(s_t, a_t, s_{t+1})$ به جای $R(s_t, a_t, s_{t+1})$.
به جای به حداکثر رساندن سودمندی جمع شونده

$$U([s_0, a_0, s_1, a_1, s_2, ...]) = R(s_0, a_0, s_1) + R(s_1, a_1, s_2) + R(s_2, a_2, s_3) + ...$$

تلاش می کنیم سودمندی کاهش یابنده را به حداکثر برسانیم

$$U([s_0, a_0, s_1, a_1, s_2, ...]) = R(s_0, a_0, s_1) + \gamma R(s_1, a_1, s_2) + \gamma^2 R(s_2, a_2, s_3) + ...$$

با توجه به اینکه تعریف فوق از تابع سودمندی کاهش یابنده شبیه به یک سری هندسی با نسبت γ است. می توانیم ثابت کنیم که تا زمانی که محدودیت وجود دارد، ارزش متناهی بودن آن تضمین شده است $|\gamma| < 1$ (که در آن |n| عملگر قدر مطلق را نشان می دهد) از طریق منطق زیر برآورده می شود

$$\begin{aligned} U([s_0, s_1, s_2, ...]) &= R(s_0, a_0, s_1) + \gamma R(s_1, a_1, s_2) + \gamma^2 R(s_2, a_2, s_3) + ... \\ &= \sum_{t=0}^{\infty} \gamma^t R(s_t, a_t, s_{t+1}) \leq \sum_{t=0}^{\infty} \gamma^t R_{max} = \frac{R_{max}}{1-\gamma} \end{aligned}$$

که در آن R_{max} حداکثر پاداش ممکن در هر زمان معین در MDP است. معمولا، γ به طور دقیق از محدوده $0 < \gamma < 1$ انتخاب می شود زیرا مقادیر در محدوده $-1 < \gamma \leq 0$ در بیشتر موقعیت های دنیای واقعی معنی ندارند. یک مقدار منفی برای γ به این معنی است که پاداش برای یک حالت s بین مقادیر مثبت و منفی در بازه های زمانی متناوب جابه جا می شود.

۳.۱ Markovianess

فرآیندهای تصمیم مارکوف «مارکوفی» هستند به این معنا که ویژگی مارکوف یا بدون حافظه را برآورده می کنند، بیان می کند که آینده و گذشته با توجه به زمان حال به طور مشروط مستقل هستند. به طور شهودی، این بدان معنی است که اگر وضعیت حال را بشناسیم، دانستن گذشته اطلاعات بیشتری در مورد آینده به ما نمی دهد.
برای بیان این موضوع به صورت ریاضی، عاملی را در نظر بگیرید که پس از انجام اعمال $a_0, a_1, ..., a_{t-1}$ از $s_0, s_1, ..., s_t$ بازدید کرده است و به تازگی عمل a_t را انجام داده است. احتمال اینکه این عامل پس از آن به حالت s_{t+1} برسد با توجه به سابقه حالت های قبلی که بازدید کرده و عمل های انجام شده، می توان به صورت زیر نوشت:

$$P(S_{t+1} = s_{t+1}|S_t = s_t, A_t = a_t, S_{t-1} = s_t, A_{t-1} = a_{t-1}, ..., S_0 = s_0)$$

که در آن

- S_t متغیر تصادفی است که حالت عامل ما را در زمان t نشان می دهد.
- A_t متغیر تصادفی است که نشان دهنده عملی است که عامل ما در زمان t انجام می دهد.

ویژگی مارکوف بیان می کند که احتمال فوق را می توان به صورت زیر ساده کرد:

$$P(S_{t+1} = s_{t+1}|S_t = s_t, A_t = a_t, S_{t-1} = s_t, A_{t-1} = a_{t-1}, ..., S_0 = s_0) = P(S_{t+1} = s_{t+1}|S_t = s_t, A_t = a_t)$$

زمانه که می گوییم «**بدون حافظه**» [۶] است به این معناست که احتمال رسیدن به حالت $s^{'}$ در زمان $t+1$ فقط به وضعیت s و اقدام a در زمان t بستگی دارد، نه به حالت ها یا اعمال قبلی: $T(s, a, s^{'}) = P(s^{'}|s, a)$.

به طور خلاصه داریم:

- فرایند «مارکوف» معمولا به این معنا است که با داشتن حالت فعلی، حالت بعدی و حالت قبلی از یکدیگر مستقل هستند.
- در فرایندهای تصمیم مارکوف، منظور از «مارکوف» این است که نتیجه ی هر عمل تنها به حالت فعلی بستگی دارد.

[6] memoryless

۴.۱ حل مسائل MDP

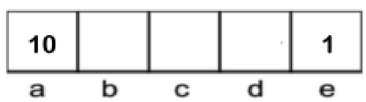
در مسائل جستجوی تک_عاملی قطعی، ما به دنبال یک مسیر بهینه بودیم. (یک دنباله از عملیات از حالت شروع به حالت هدف)
از طرف دیگر، حل یک فرآیند تصمیم مارکوف به معنای یافتن یک سیاست بهینه $\pi^* : S \to A$ است، تابعی که هر حالت $s \in S$ را به یک عمل $a \in A$ نگاشت می کند.

- یک سیاست صریح، بیانگر یک عامل واکنشی است.
- سیاست π تعیین کننده‌ی یک عمل در هر حالت است.
- سیاستی بهینه است که در صورت دنبال کردن، سودمندی مورد انتظار را بیشینه سازد.

به عبارتی با توجه به حالت $s \in S$ سیاست π عمل $a = \pi(s)$ را انجام می دهد به طوری که $a \in A$.
مسئله MDP زیر را در نظر بگیرید:

- $S = \{a, b, c, d, e\}$
- $A = \{East, West, Exit\}$
- عمل $Exit$ فقط در حالت های a, e یک عمل معتبر محسوب می شود و به ترتیب پاداش $10, 1$ را به همراه دارد.
- ضریب کاهش $\gamma = 0.1$

و انتقال های قطعی :



دو سیاست برای این MDP به شرح زیر است:

(a) Policy 1 (b) Policy 2

با کمی بررسی، تعیین بهینه بودن سیاست ۲ دشوار نیست. با دنبال کردن این سیاست تا زمان انجام عمل $a = Exit$ پاداش های زیر را برای هر حالت شروع به دست می دهد:

Start State	Reward
a	10
b	1
c	0.1
d	0.1
e	1

اکنون یاد خواهیم گرفت که چگونه چنین MDP ها (و موارد بسیار پیچیده تر!) را به صورت الگوریتمی با استفاده از **معادله بلمن** [7] برای فرآیندهای تصمیم مارکوف حل کنیم.

[7] Bellman equation

۵.۱ معادله بلمن

به منظور صحبت در مورد معادله بلمن برای MDP ها، ابتدا باید دو کمیت ریاضی جدید را معرفی کنیم:

- $U^*(s)$ = ارزش یا سودمندی حالت s :
  سودمندی مورد انتظار با شروع از s و بهینه عمل کردن.
  توجه داشته باشید که اغلب در متون همین مقدار را با $V^*(s)$ نشان داده می شود.
- $Q^*(s,a)$ سودمندی مورد انتظار با شروع از s و انتخاب عمل a و بهینه عمل کردن [از این به بعد].

با استفاده از این دو کمیت جدید و سایر کمیت های MDP که قبلاً مورد بحث قرار گرفت، معادله بلمن به صورت زیر تعریف می شود:

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

قبل از اینکه به تفسیر معادله بالا بپردازیم، اجازه دهید معادله مقدار بهینه یک Q-state را نیز تعریف کنیم (که معمولاً به عنوان Q-value بهینه شناخته می شود):

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

توجه داشته باشید که این تعریف دوم به ما اجازه می دهد تا معادله بلمن را به صورت زیر تعریف کنیم

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

معادله بلمن مثالی از یک معادله برنامه ریزی پویا است، معادله ای که یک مساله را از طریق یک ساختار بازگشتی ذاتی به زیرمسئله های کوچک تر تجزیه می کند. می‌توانیم این بازگشت ذاتی را در معادله Q–value یک حالت، در عبارت $[R(s,a,s') + \gamma U^*(s')]$ مشاهده کنیم. این عبارت نشان‌دهنده کل سودمندی است که یک عامل با انجام عمل a از s و رسیدن به s' دریافت می‌کند و از این پس به‌طور بهینه عمل می‌کند. پاداش حاصل از عمل انجام شده a یعنی $R(s,a,s')$، به مجموع ضریب کاهشی بهینه پاداش های قابل دستیابی از s یعنی $U^*(s')$ افزوده می شود، که توسط γ کاهش می یابد تا یک مرحله زمانی در انجام عمل a در نظر گرفته شود.
اکنون می توانیم یک گام دیگر به سمت بیرون برداریم و معادله کامل Q–value را در نظر بگیریم. دانستن اینکه $[R(s,a,s') + \gamma U^*(s')]$ نشان دهنده سودمندی است که با عمل بهینه پس از رسیدن به حالت s' از Q-State (s,a) حاصل می شود، بدیهی است که کمیت

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

صرفاً یک مجموع وزنی از سودمندی ها است که هر سودمندی بر اساس احتمال وقوعش وزن دهی می شود. طبق تعریف سودمندی مورد انتظار عمل بهینه از Q-State (s,a) به بعد است! مقدار بهینه یک حالت یعنی $U(s')$ حداکثر سودمندی مورد انتظار برای همه عمل های ممکن از حالت s است.
محاسبه حداکثر سودمندی مورد انتظار برای یک حالت s اساساً با اجرای expectimax یکسان است – ابتدا سودمندی مورد انتظار را از هر Q-State (s,a) محاسبه می کنیم (معادل محاسبه مقدار گره های شانس). سپس برای اینکه حداکثر سودمندی مورد انتظار (معادل محاسبه مقدار یک گره حداکثر ساز) را محاسبه کنید، maximum را روی این گره ها پیدا کنید .
اگر بتوانیم مقدار $U(s)$ را برای هر حالت $s \in S$ به نحوی تعیین کنیم که معادله بلمن برای هر یک از این حالت ها صادق باشد، می توانیم نتیجه بگیریم که این مقادیر، مقادیر بهینه برای حالات مربوطه خود هستند. درواقع ارضای این شرط به این معناست که $\forall s \in S, U(s) = U^*(s)$.

۶.۱ Value Iteration [8]

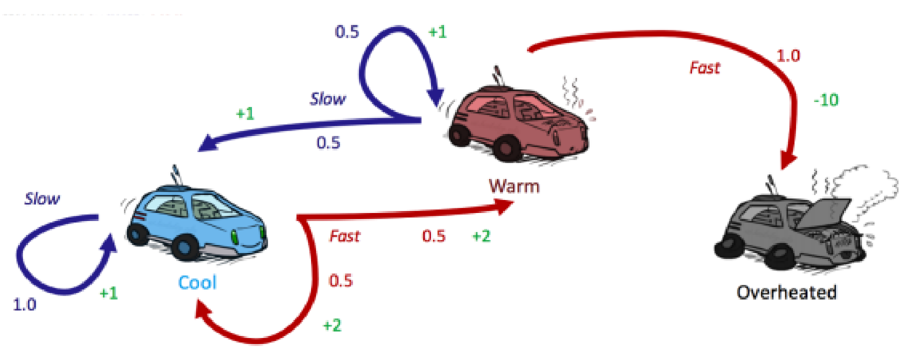
حال که یک چارچوب برای آزمایش بهینگی مقادیر حالت ها در یک MDP داریم، سوال دیگری که پیش می آید این است که چگونه این مقادیر بهینه را محاسبه کنیم. برای پاسخ به این سوال، به ارزش های با زمان محدود [9] (نتیجه طبیعی اجرای Finite horizons) نیاز داریم. ارزش با زمان محدود برای حالت s با محدودیت k بازه زمانی به صورت $U_k(s)$ نشان داده می شود، و نشان دهنده حداکثر سودمندی مورد انتظار قابل دستیابی از حالت s با توجه به اینکه فرآیند تصمیم گیری مارکوف تحت بررسی در k مرحله خاتمه یابد. به طور معادل، این همان چیزی است که یک expectimax با عمق محدود k در درخت جستجو برای یک MDP برمی گرداند.

Value Iteration یک الگوریتم برنامه ریزی پویا است که از یک محدودیت زمانی تکراری برای محاسبه ارزش های با زمان محدود تا همگرایی استفاده می کند (یعنی تا زمانی که مقادیر U برای هر حالت مانند تکرار گذشته یکسان باشد: $\forall s, U_{k+1}(s) = U_k(s)$) به صورت زیر عمل می کند:

- $\forall s \in S$ مقدار اولیه برای U داریم که $U_k(s) = 0$.
- قانون به روزرسانی زیر را تا زمان هم گرایی مقادیر تکرار کنید:

$$\forall s \in S, U_{k+1}(s) \leftarrow max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma U^*(s')]$$

در تکرار k از الگوریتم Value Iteration، از ارزش های با زمان محدود با عمق k برای هر حالت استفاده می کنیم تا ارزش های با زمان محدود شده را با عمق $(k+1)$ تولید کنیم.
در اصل، ما از راه حل های محاسبه شده برای زیرمسئله ها استفاده می کنیم (همه $U_k(s)$) تا به طور تکراری راه حل هایی برای زیرمسئله های بزرگ تر بسازیم (همه $U_{k+1}(s)$)؛ این همان چیزی است که الگوریتم Value Iteration را به یک الگوریتم برنامه ریزی پویا تبدیل می کند.
توجه داشته باشید که اگرچه معادله بلمن اساساً در ساختار با قانون به روز رسانی بالا یکسان به نظر می رسد، اما آنها یکسان نیستند. معادله بلمن یک شرط برای بهینه بودن می دهد، در حالی که قانون به روز رسانی روشی را برای به روز رسانی مکرر مقادیر تا زمان همگرایی ارائه می دهد. هنگامی که به همگرایی رسید، معادله بلمن برای هر حالت برقرار می شود: $\forall s \in S, U_k(s) = U_{k+1}(s) = U^*(s)$.
به اختصار، اغلب $U_{k+1}(s) \leftarrow max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma U^*(s')]$ را به صورت $U_{k+1} \leftarrow BU_k$ نشان می‌دهیم، که در آن B عملگر بلمن نامیده می‌شود.
بیایید چند به‌روزرسانی از الگوریتم Value Iteration را با بازبینی مجدد MDP مسئله ماشین مسابقه‌ای، با معرفی ضریب کاهش $\gamma = 0.5$ مشاهده کنیم:



Value Iteration را با مقدار دهی اولیه $U_0(s) = 0$ شروع می کنیم:

	cool	warm	overheated
U_0	0	0	0

[8] Value Iteration
[9] time-limited values

در دور اول به‌روزرسانی‌ها، می‌توانیم $\forall s \in S, U_1(s)$ را به صورت زیر محاسبه کنیم:

$$\begin{aligned}
U_1(cool) &= \max\{1\cdot[1+0.5\cdot 0],\ 0.5\cdot[2+0.5\cdot 0]+0.5\cdot[2+0.5\cdot 0]\} \\
&= \max\{1,2\} \\
&= \boxed{2} \\
U_1(warm) &= \max\{0.5\cdot[1+0.5\cdot 0]+0.5\cdot[1+0.5\cdot 0],\ 1\cdot[-10+0.5\cdot 0]\} \\
&= \max\{1,-10\} \\
&= \boxed{1} \\
U_1(overheated) &= \max\{\} \\
&= \boxed{0}
\end{aligned}$$

	cool	**warm**	**overheated**
U_0	0	0	0
U_1	2	1	0

به طور مشابه، ما می توانیم این روند را برای محاسبه دور دوم به روز رسانی ها با مقادیر جدیدمان برای $U_1(s)$ برای محاسبه $U_2(s)$ تکرار کنیم.

$$\begin{aligned}
U_2(cool) &= \max\{1\cdot[1+0.5\cdot 2],\ 0.5\cdot[2+0.5\cdot 2]+0.5\cdot[2+0.5\cdot 1]\} \\
&= \max\{2,2.75\} \\
&= \boxed{2.75} \\
U_2(warm) &= \max\{0.5\cdot[1+0.5\cdot 2]+0.5\cdot[1+0.5\cdot 1],\ 1\cdot[-10+0.5\cdot 0]\} \\
&= \max\{1.75,-10\} \\
&= \boxed{1.75} \\
U_2(overheated) &= \max\{\} \\
&= \boxed{0}
\end{aligned}$$

	cool	**warm**	**overheated**
U_0	0	0	0
U_1	2	1	0
U_2	2.75	1.75	0

لازم به ذکر است که $U^*(s)$ برای هر حالت پایانی باید ۰ باشد، زیرا از هر حالت پایانی هیچ عملی را برای بدست آوردن پاداش نمی توانیم انجام دهیم .

۷.۱ policy extraction(استخراج سیاست)

هدف نهایی ما در حل MDP تعیین سیاست بهینه است.این کار بسیار ساده است.در هر حالت باید عملی انجام شود که حداکثر سودمندی مورد انتظار رابه همراه دارد.این عمل، عملی است که ما را به Q-state ای می برد که حداکثر مقدار Q را دارد. که می توانیم آن را با استفاده از معادله زیر بیان کنیم:

$$Q_{k+1}(s,a) \leftarrow \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma \max_{a'} Q(s',a')]$$

برای استخراج سیاست بهینه، در صورتیکه مقادیر Q-state ها را ذخیره کرده باشیم.باانجام یک عمل argmax می توانیم به راحتی عملی که ما را از حالت فعلی به Q-state ای با مقدار بهینه می برد، پیدا کنیم. در غیر این صورت باید با استفاده ازمعادله بلمن ، مقادیر Q-state ها را محاسبه کنیم و سپس argmax را روی مقادیر آنها اعمال کنیم.