Programowanie dynamiczne

Programowanie dynamiczne

- Równania Bellmana można rozwiązać analitycznie.
- Złożoność obliczeniowa sięga $\mathrm{O}(\mathrm{n}^3)$ gdzie n oznacza liczbę stanów.
- W przypadku rozwiązywania dużych problemów istnieje wiele metod iteracyjnych:
  - Exhaustive search
  - Programowanie dynamiczne
  - Metody Monte Carlo
  - Temporal Difference Learning

Programowanie dynamiczne

- Dużo efektywniejszym sposobem rozwiązywania procesów decyzyjnych Markowa jest wykorzystanie metod programowania dynamicznego.

- Jest to możliwe dzięki strukturze procesów decyzyjnych Markowa:
  - Rozwiązanie można rozbić na zagadnienie optymalizacji podproblemów **(zasada optymalności Bellmana).**
  - Podproblemy powtarzają się – to samo rozwiązanie można wykorzystać wiele razy.

Programowanie dynamiczne

- Dużo efektywniejszym sposobem rozwiązywania procesów decyzyjnych Markowa jest wykorzystanie metod programowania dynamicznego.

- Jest to możliwe dzięki strukturze procesów decyzyjnych Markowa:
  - Rozwiązanie można rozbić na zagadnienie optymalizacji podproblemów **(zasada optymalności Bellmana).**
  - Podproblemy powtarzają się – to samo rozwiązanie można wykorzystać wiele razy.
- **Zasada optymalności Bellmana:** optymalna strategia ma tą własność, że niezależnie od warunków początkowych i początkowej decyzji, akcje podjęte w każdej iteracji muszą stanowić optymalną strategię dla stanów wynikłych z poprzednich decyzji.

Programowanie dynamiczne

- **Ewaluacja** – dana jest strategia π, należy dokonać jej oceny:
  - Zainicjuj algorytm wybierając dowolne początkowe wartości funkcji wartości, np. $V_0(s) = 0\ \forall s$.
  - W każdym kroku k:
    - dla każdego ze stanów $s \in S$ uaktualnij V_k na podstawie $V_{k-1}(s')$:

$$V_k(s) = \sum_a \pi(s,a) \sum_{s',r} P(s,a,s')(r + \beta V_{k-1}(s'))$$

    - Zatrzymaj proces gdy:

$$\max_s |V_k(s) - V_{k-1}(s)| < \theta$$

Programowanie dynamiczne

- **Ewaluacja** – dana jest strategia π, należy dokonać jej oceny:
  - Zainicjuj algorytm wybierając dowolne początkowe wartości funkcji wartości, np. $V_0(s) = 0\ \forall s$.
  - W każdym kroku k:
    - dla każdego ze stanów $s \in S$ uaktualnij V_k na podstawie $V_{k-1}(s')$:

$$V_k(s) = \sum_a \pi(s,a) \sum_{s',r} P(s,a,s')(r + \beta V_{k-1}(s'))$$

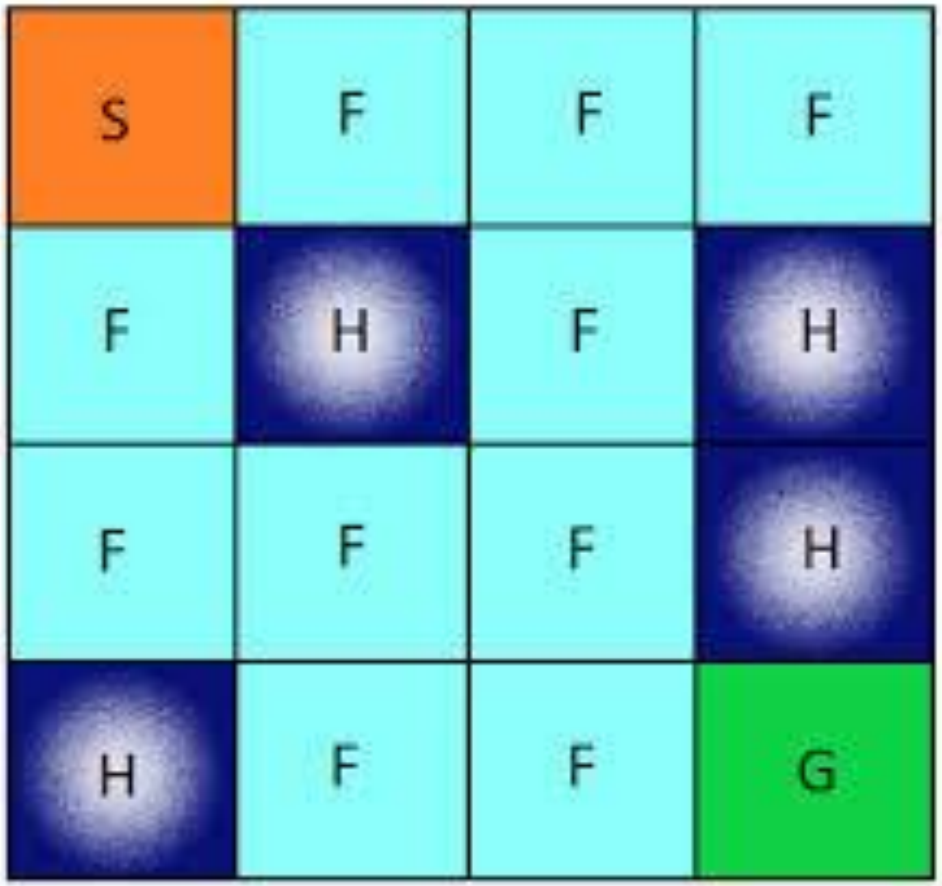
    - Zatrzymaj proces gdy:

$$\max_s |V_k(s) - V_{k-1}(s)| < \theta$$

- Na mocy **Twierdzenia Banacha o punkcie stałym** algorytm zbiega do V_π.

Przykład

Frozen Lake



- Celem agenta jest przejść z punktu S do Punktu G.
- Agent może iść po lodzie (pola oznaczone literą F), musi unikać wpadnięcia do przerębli (pola oznaczone jako H).
- Lód jest śliski; idąc przed siebie z pewnym prawdopodobieństwem p może się poślizgnąć i przesunąć w lewo lub w prawo w stosunku do swojej wyjściowej pozycji.

Programowanie dynamiczne

- **Optymalizacja strategii** – poszukiwanie optymalnej strategii π_* i optymalnej funkcji wartości stanu V_*.
- Metody rozwiązania:
  - Iterowanie po funkcji wartości **(value iteration)**.
  - Iterowanie po strategii **(policy iteration)**.

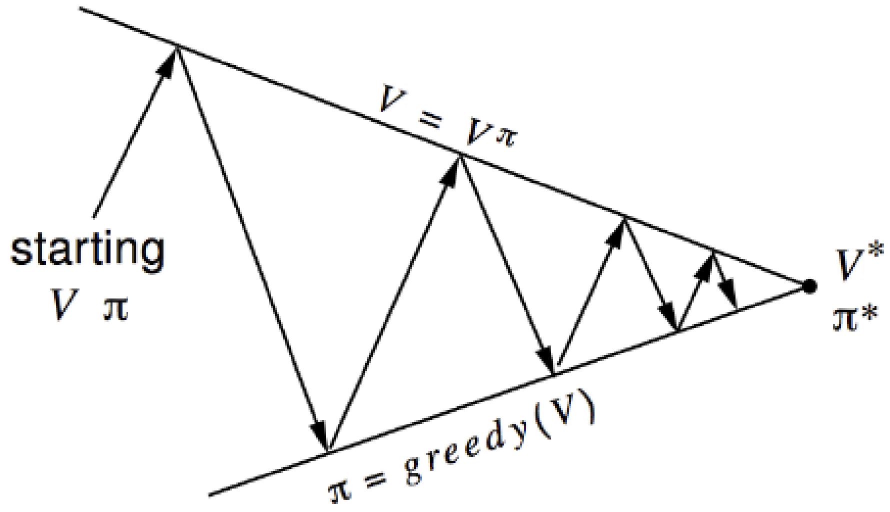
Value iteration

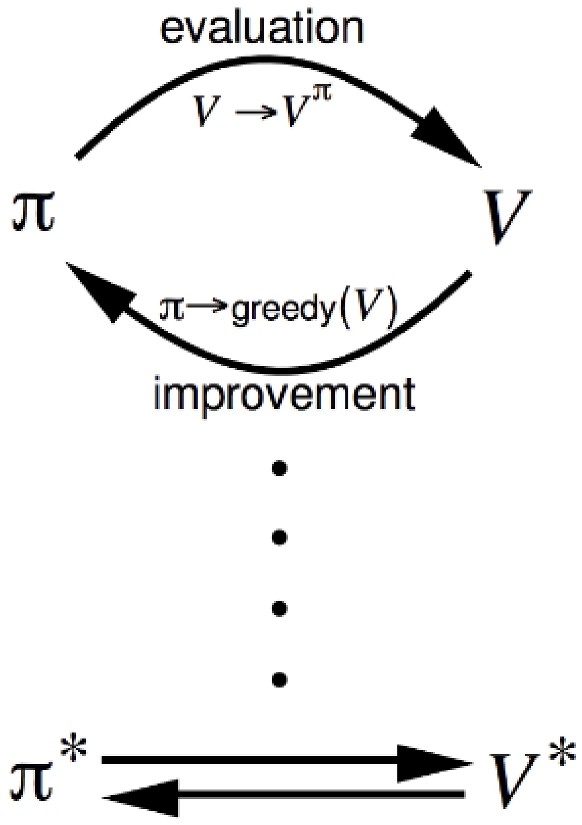
1. Zainicjuj algorytm wybierając dowolne początkowe wartości funkcji wartości, np. $V_0(s) = 0\ \forall s$.
2. W każdej iteracji k:
   - Uaktualnij wartość $V_k(s)$ wykorzystując do tego estymację funkcji $V_{k-1}(s')$:
   $$V_k(s) = \max_a \sum_{s',r} P(s,a,s')(r + \beta V_{k-1}(s'))$$
3. Zatrzymaj algorytm gdy wynik zbiegnie do $\max_s |V_k(s) - V_{k-1}(s)| < \theta$.
4. W ostatnim kroku wygeneruj optymalną strategię π_* wykorzystując do tego równanie:
   $$\pi_* = \operatorname{argmax}_a \sum_{s,r} P(s,a,s')(r + V_{\pi_*}(s'))$$

Policy iteration

1. Zainicjuj algorytm wybierając dowolne początkowe wartości funkcji wartości, np. $V_0(s) = 0\ \forall s$ i dowolną strategię π.
2. W każdej iteracji k:
   - Dokonaj oceny $V(s)$ wykorzystując do tego strategię π:
   $$V_k(s) = \sum_a \pi(s,a) \sum_{s',r} P(s,a,s')(r + \beta V_\pi(s'))$$
   - Popraw zaproponowaną strategię zachowując się zachłannie w stosunku do V_π:
   $$\pi' = greedy(V_\pi)$$
   $$\pi' = \operatorname*{argmax}_a \sum_{s,r} P(s,a,s')(r + V_\pi(s'))$$
3. Zatrzymaj algorytm gdy wynik zbiegnie do $\max_s |V_k(s) - V_{k-1}(s)| < \theta$.
4. Otrzymana w ten sposób strategia π' jest optymalna: $\pi' = \pi_*$.

Policy iteration





Policy evaluation Estimate v_π

Iterative policy evaluation

Policy improvement Generate $\pi' \geq \pi$

Greedy policy improvement

Źródło: http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching_files/DP.pdf

Programowanie dynamiczne

- Metody programowania dynamicznego gwarantują zbieżność do rozwiązania optymalnego.
- Jednak nadal charakteryzują się dużą złożonością obliczeniową, przez którą rozwiązywanie dużych problemów może być trudne.

Programowanie dynamiczne

- Przy czym nadal do poprawnego działania tych algorytmów konieczna jest pełna wiedza na temat rozpatrywanego procesu decyzyjnego Markowa (nazywamy to uczeniem opartym na modelu – *model based learning*).
- Konieczna jest znajomość macierzy przejścia i nagród za znalezienie się w każdym z osiągalnych stanów.
- Nie zawsze taka wiedza jest dostępna – wtedy warto skorzystać z metod uczenia wolnego od modeli (*model free learning*).

Metody Monte Carlo

Metody Monte Carlo

- Kluczowym problemem w przypadku korzystania z metod programowania dynamicznego jest fakt, że do ich rozwiązania konieczna jest znajomość pełnego modelu.

Metody Monte Carlo

- Kluczowym problemem w przypadku korzystania z metod programowania dynamicznego jest fakt, że do ich rozwiązania konieczna jest znajomość pełnego modelu.
- Oznacza to, że przed rozpoczęciem *planowania* musimy znać zarówno model przejścia $\boldsymbol{P}(\mathbf{s}, \mathbf{a}, \mathbf{s}')$– jak i wszystkie osiągalne nagrody : $\mathbf{R}(\mathbf{s}, \mathbf{a}, \mathbf{s}')$.

Metody Monte Carlo

- Kluczowym problemem w przypadku korzystania z metod programowania dynamicznego jest fakt, że do ich rozwiązania konieczna jest znajomość pełnego modelu.
- Oznacza to, że przed rozpoczęciem *planowania* musimy znać zarówno model przejścia $\boldsymbol{P}(\mathbf{s}, \mathbf{a}, \mathbf{s}')$– jak i wszystkie osiągalne nagrody : $\mathbf{R}(\mathbf{s}, \mathbf{a}, \mathbf{s}')$.
- Wykorzystując bezmodelowe metody uczenia jesteśmy w stanie obejść ten problem.

Metody Monte Carlo

- Kluczowym problemem w przypadku korzystania z metod programowania dynamicznego jest fakt, że do ich rozwiązania konieczna jest znajomość pełnego modelu.
- Oznacza to, że przed rozpoczęciem *planowania* musimy znać zarówno model przejścia $\boldsymbol{P}(\mathbf{s}, \mathbf{a}, \mathbf{s}')$– jak i wszystkie osiągalne nagrody : $\mathbf{R}(\mathbf{s}, \mathbf{a}, \mathbf{s}')$.
- Wykorzystując bezmodelowe metody uczenia jesteśmy w stanie obejść ten problem.
- W takim wypadku agent uczy się na podstawie swojego **doświadczenia,** poprzez powtarzające się **interakcje** z otoczeniem.

Metody Monte Carlo

- Podstawową sposobem uczenia bez modelu są **Metody Monte Carlo.**
- Uczenie opiera się na generowaniu **pełnych epizodów** – agent nie bootstrapuje swojego doświadczenia na podstawie wcześniejszych wyników.

Metody Monte Carlo

- Podstawową sposobem uczenia bez modelu są **Metody Monte Carlo.**
- Uczenie opiera się na generowaniu **pełnych epizodów** – agent nie bootstrapuje swojego doświadczenia na podstawie wcześniejszych wyników.
  - Oznacza to, że metody Monte Carlo można stosować jedynie w przypadku skończonych procesów decyzyjnych Markowa.

Metody Monte Carlo

- Podstawową sposobem uczenia bez modelu są **Metody Monte Carlo.**
- Uczenie opiera się na generowaniu **pełnych epizodów** – agent nie bootstrapuje swojego doświadczenia na podstawie wcześniejszych wyników.
  - Oznacza to, że metody Monte Carlo można stosować jedynie w przypadku skończonych procesów decyzyjnych Markowa.
  - Ale też oznacza to, że możliwe jest uczenie się problemów, które nie spełniają własności Markowa.

Metody Monte Carlo

- Idea uczenia MC jest prosta:
  - Wartość funkcji wartości w k-tej iteracji jest równa przeciętnej skumulowanej przyszłej nagrodzie otrzymanej w poprzednich iteracjach.
  - Na mocy prawa wielkich liczb, gdy ilość symulowanych epizodów będzie dążyła do nieskończoności to oszacowanie będzie dążyło do prawdziwej funkcji wartości.

Monte Carlo Prediction

- **Ewaluacja** – dana jest strategia π, należy dokonać jej oceny:
  - Zainicjuj algorytm wybierając dowolne początkowe wartości funkcji wartości, np. $V_0(s) = 0\ \forall s$.

<table>
<tr>
<td>

W każdym kroku k:

- Wygeneruj epizod na podstawie strategii π: $\mathrm{s}_0, \mathrm{a}_0, \mathrm{r}_1, \dots, s_{T-1}, a_{T-1}, r_T$
- Dla każdego ze stanów $s \in S$ należących do wygenerowanego epizodu, **gdy odwiedzasz go za pierwszym razem**:
  - Uaktualnij licznik odwiedzin stanu s:
  $$n_s = n_s + 1$$
  - Uaktualnij przeciętną wartość funkcji wartości:
  $$V_k(s) = V_{k-1}(s) + \frac{1}{n_s}(R_k(s) - V_{k-1}(s))$$

</td>
<td>

W każdym kroku k:

- Wygeneruj epizod na podstawie strategii π: $\mathrm{s}_0, \mathrm{a}_0, \mathrm{r}_1, \dots, s_{T-1}, a_{T-1}, r_T$
- Dla każdego ze stanów $s \in S$ należących do wygenerowanego epizodu, **za każdym razem gdy go odwiedzasz**:
  - Uaktualnij licznik odwiedzin stanu s:
  $$n_s = n_s + 1$$
  - Uaktualnij przeciętną wartość funkcji wartości:
  $$V_k(s) = V_{k-1}(s) + \frac{1}{n_s}(R_k(s) - V_{k-1}(s))$$

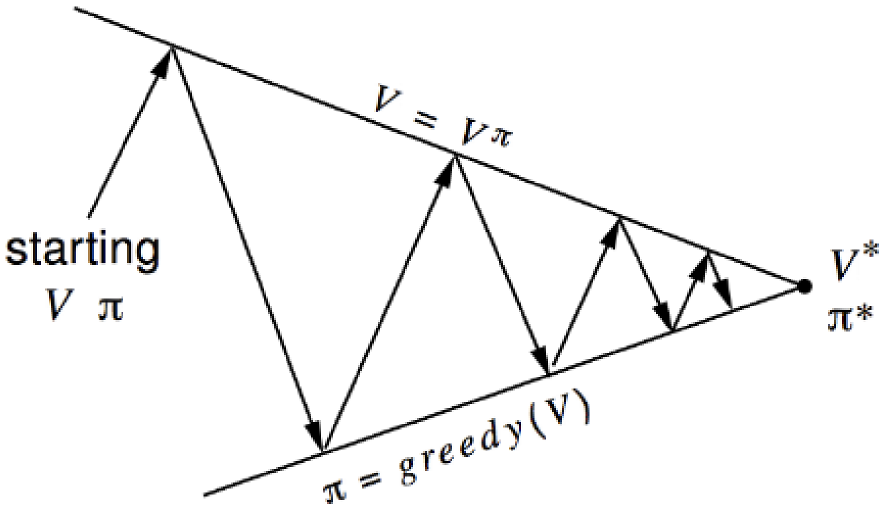
</td>
</tr>
</table>

- Obie metody zbiegną $V(s) \to V_\pi(s)$ gdy $n_s \to \infty$.

Monte Carlo Control

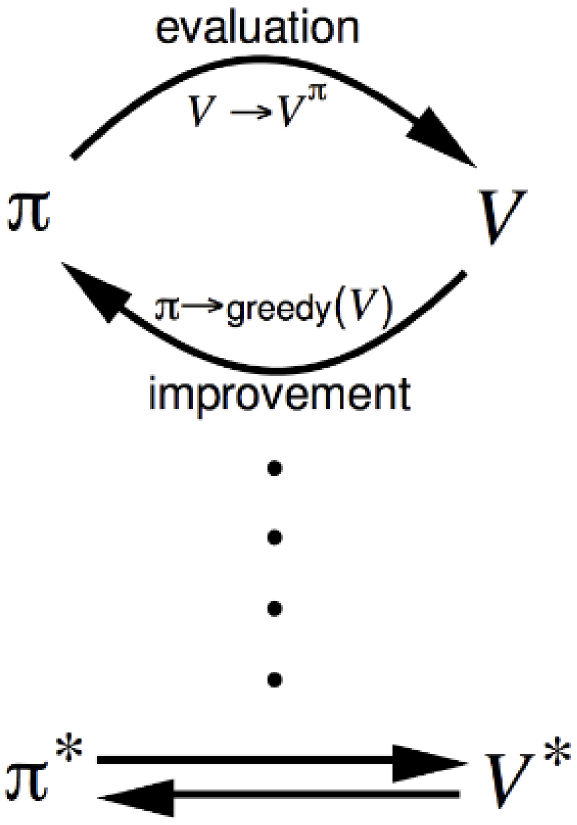
- W przypadku poszukiwania optymalnej strategii π_* za pomocą metod Monte Carlo konieczna jest modyfikacja podejścia do rozwiązywanego problemu:

Monte Carlo Control



Policy evaluation Estimate v_π
Iterative policy evaluation

Policy improvement Generate $\pi' \geq \pi$
Greedy policy improvement



Źródło: http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching_files/DP.pdf

Monte Carlo Control

- Zachłanna iteracja po funkcji $V(s)$ jest niemożliwa; wymaga znajomości modelu:

$$\pi' = \underset{a}{\operatorname{argmax}} \sum_{s,r} P(s, a, s')(r + V_\pi(s'))$$

Monte Carlo Control

- Zachłanna iteracja po funkcji $V(s)$ jest niemożliwa; wymaga znajomości modelu:

$$\pi' = \underset{a \in A}{\operatorname{argmax}} \sum_{s,r} P(s, a, s')(r + V_\pi(s'))$$

- Konieczne jest wykorzystanie funkcji $Q(s, a)$:

$$\pi' = \underset{a \in A}{\operatorname{argmax}}\, Q(s, a)$$

Monte Carlo Control z Eksploracją punktów startowych

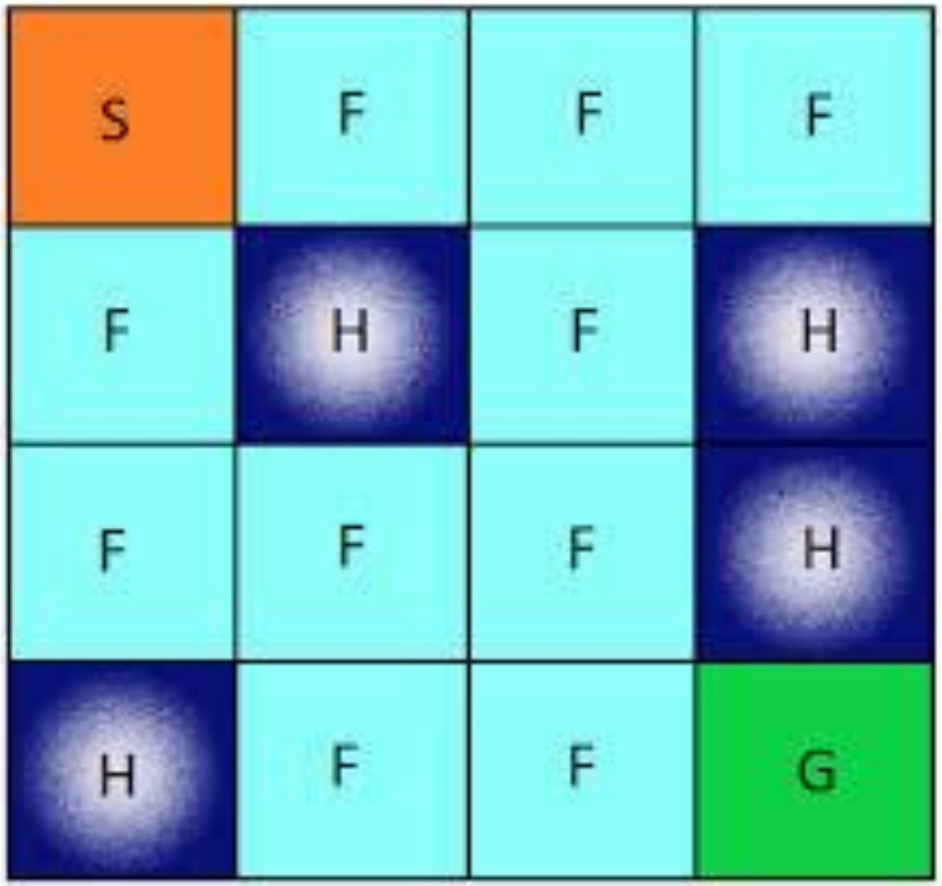
1. Zainicjuj algorytm wybierając dowolne początkowe wartości funkcji wartości, np. $Q_0(s,a) = 0 \; \forall s, a$, wektor odwiedzin part stan i akcja: $N(s,a) = 0 \; \forall s, a$ i dowolną strategię π.

2. W każdej iteracji k:
   - Wylosuj początkową parę $\mathrm{s}_0 \in S, \mathrm{a}_0 \in A$
   - Wygeneruj epizod na podstawie strategii π: $\mathrm{s}_0, \mathrm{a}_0, \mathrm{r}_1, \dots, s_{T-1}, a_{T-1}, r_T$.
   - Przyjmij wartość skumulowanej przyszłej nagrody $R = 0$
   - Dla każdego $\mathrm{t} = \mathrm{T} - 1, \mathrm{T} - 2, \dots, 1, 0$:
     - Przypisz nową wartość $R = \beta R + r_{t+1}$
     - Jeżeli para $\mathrm{s_t}, \mathrm{a_t}$ nie występuje w $s_0, a_0, r_1, \dots, s_{t-1}, a_{t-1}$ (**jest pierwszym wystąpieniem**):
       - Uaktualnij licznik odwiedzin pary $\mathrm{s_t}, \mathrm{a_t}$:
         $$N(s_t, a_t) = N(s_t, a_t) + 1$$
       - Uaktualnij funkcję wartości akcji:
         $$Q(s_t, a_t) = Q(s_t, a_t) + \frac{1}{N(s,a)}\left(R - Q(s_t, a_t)\right)$$
       - Popraw zaproponowaną strategię zachowując się zachłannie:
         $$\pi(s) = \underset{a \in A}{\operatorname{argmax}}\, Q(s,a)$$

3. Gdy $n \to \infty$ to $\pi \approx \pi_*$

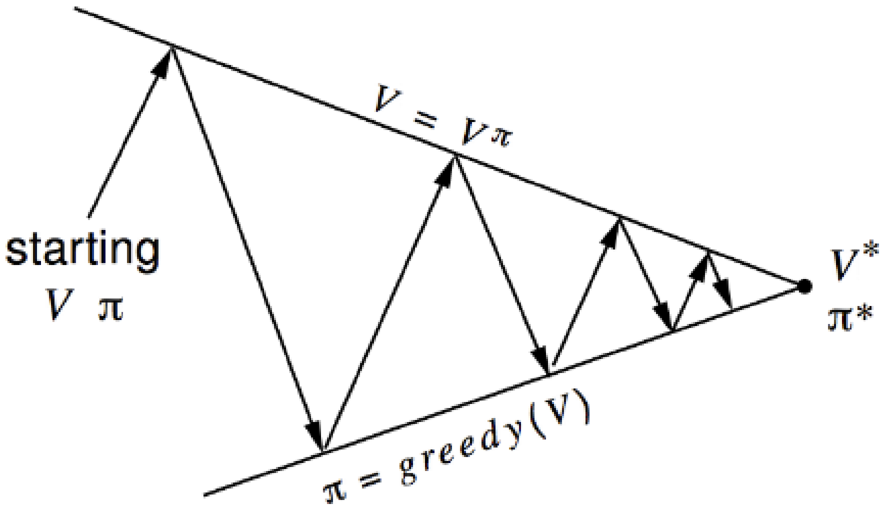
Przykład

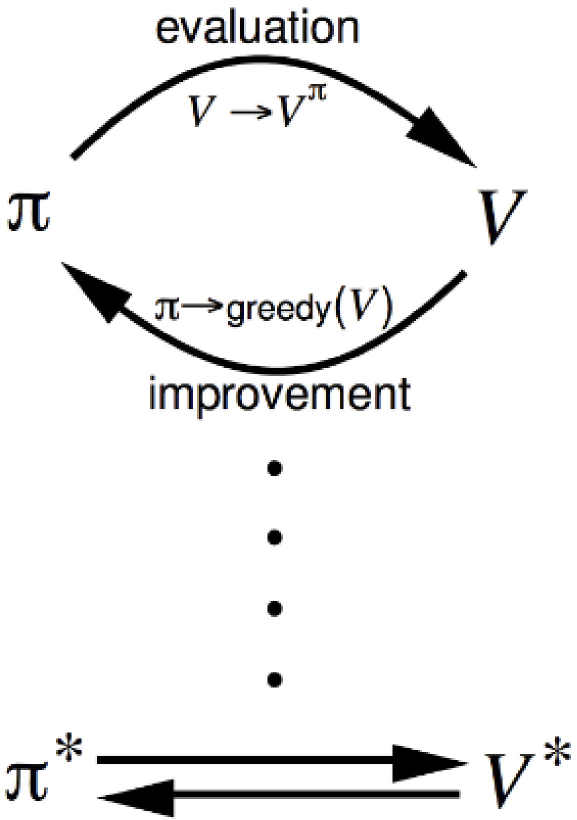
Frozen Lake cd.



- Celem agenta jest przejść z punktu S do Punktu G.
- Agent może iść po lodzie (pola oznaczone literą F), musi unikać wpadnięcia do przerębli (pola oznaczone jako H).
- Lód jest śliski; idąc przed siebie z pewnym prawdopodobieństwem p może się poślizgnąć i przesunąć w lewo lub w prawo w stosunku do swojej wyjściowej pozycji.

Monte Carlo Control





Policy evaluation Estimate v_π

Iterative policy evaluation

Policy improvement Generate $\pi' \geq \pi$

Greedy policy improvement

Źródło: http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching_files/DP.pdf

Monte Carlo Control

- Problematyczny jest też fakt zapewnienia odpowiedniej ilości epizodów na podstawie których możliwe będzie oszacowanie $Q(s, a)$.
  - **Eksploracja**
  - **Eksploatacja**

Monte Carlo Control

- Problematyczny jest też fakt zapewnienia odpowiedniej ilości epizodów na podstawie których możliwe będzie oszacowanie $Q(s, a)$.
    - **Eksploracja**
    - **Eksploatacja**
- Tradycyjny algorytm zachłanny bardzo szybko zacznie eksploatować rozwiązanie suboptymalne.

Monte Carlo Control

- **On-policy**
  - Zoptymalizuj strategię π bazując na doświadczeniu próbkowanym na podstawie tej właśnie strategii. (*„ucz się na swoich błędach”*)
- **Off-policy**
  - Zoptymalizuj strategię π bazując na doświadczeniu próbkowanym z innych strategii μ. (*„ucz się na czyichś błędach”*)

Monte Carlo Control

- **Strategia ϵ-zachłanna (ϵ-greedy)**
- Najprostszy algorytm pozwalający na zachowanie ciągłej eksploatacji w trakcie uczenia się.
- Zapewnia, że każda z dopuszczalnych m akcji będzie wybierana z niezerowym prawdopodobieństwem.
  - Z $\mathrm{p} = 1 - \epsilon$ wybierz akcję zachłannie.
  - W przeciwnym wypadku wybieraj losowo.

Monte Carlo Control

- **Strategia ϵ-zachłanna (ϵ-greedy)**

$$\pi(a|s) = \begin{cases} \dfrac{\epsilon}{m} + 1 - \epsilon, gdy\ a_* = \operatorname*{argmax}_{a \in A} Q(s,a) \\ \dfrac{\epsilon}{m}, w\ przeciwnym\ wypadku \end{cases}$$

Monte Carlo On Policy

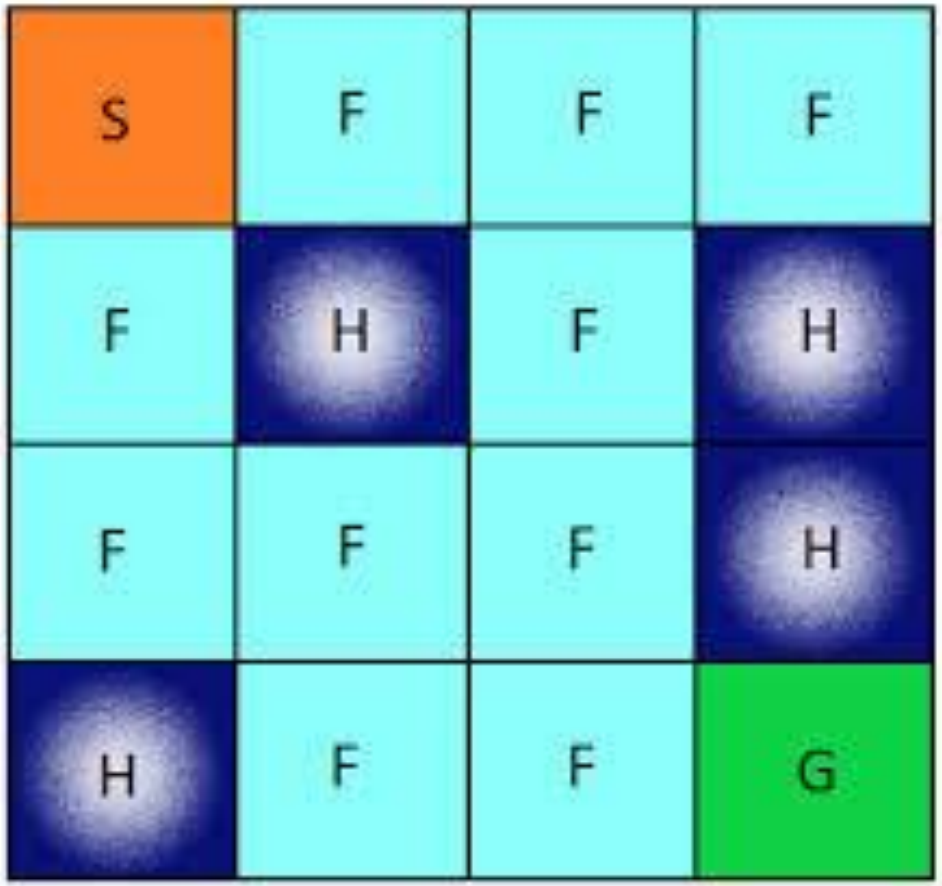
1. Zainicjuj algorytm wybierając dowolne początkowe wartości funkcji wartości, np. $Q_0(s,a) = 0 \ \forall s,a$, wektor odwiedzin part stan i akcja: $N(s,a) = 0 \ \forall s,a$ i dowolną ϵ-zachłanną strategię π.

2. W każdej iteracji k:
   - Wygeneruj epizod na podstawie strategii π: $\mathrm{s}_0, \mathrm{a}_0, \mathrm{r}_1, \dots, s_{T-1}, a_{T-1}, r_T$.
   - Przyjmij wartość skumulowanej przyszłej nagrody $R = 0$
   - Dla każdego $\mathrm{t} = \mathrm{T}-1, \mathrm{T}-2, \dots, 1, 0$:
     - Przypisz nową wartość $R = \beta R + r_{t+1}$
     - Jeżeli para $\mathrm{s_t}, \mathrm{a_t}$ nie występuje w $s_0, a_0, r_1, \dots, s_{t-1}, a_{t-1}$ (**jest pierwszym wystąpieniem**):
       - Uaktualnij licznik odwiedzin pary $\mathrm{s_t}, \mathrm{a_t}$:
       $$N(s_t,a_t) = N(s_t,a_t) + 1$$
       - Uaktualnij funkcję wartości akcji:
       $$Q(s_t,a_t) = Q(s_t,a_t) + \frac{1}{N(s,a)}\left(R - Q(s_t,a_t)\right)$$
       - Popraw zaproponowaną strategię zachowując się zachłannie:
       $$\pi(a|s) \leftarrow \begin{cases} 1 - \epsilon + \epsilon/|A(S_t)| & gdy\ a = \operatorname*{argmax}_{a\in A} Q(s,a) \\ \epsilon/|A(S_t)| & gdy\ a \neq \operatorname*{argmax}_{a\in A} Q(s,a) \end{cases} \quad \forall a \in A(S_t)$$

3. Gdy $n \to \infty$ to $\pi \approx \pi_*$

Przykład

Frozen Lake cd.



- Celem agenta jest przejść z punktu S do Punktu G.
- Agent może iść po lodzie (pola oznaczone literą F), musi unikać wpadnięcia do przerębli (pola oznaczone jako H).
- Lód jest śliski; idąc przed siebie z pewnym prawdopodobieństwem p może się poślizgnąć i przesunąć w lewo lub w prawo w stosunku do swojej wyjściowej pozycji.

Temporal Difference Learning

Temporal Difference Learning

- Temporal difference learning (co możemy przetłumaczyć na uczenie oparte na różnicach czasowych) jest grupą algorytmów, które w pewnym sensie łączy ze sobą sposób działania metod Monte Carlo i podejścia związanego z dynamicznym programowaniem.

Temporal Difference Learning

- Temporal difference learning (co możemy przetłumaczyć na uczenie operte na różnicach czasowych) jest grupą algorytmów, które w pewnym sensie łączy ze sobą sposób działania metod Monte Carlo i podejścia związanego z dynamicznym programowaniem.
- Metody TD polegają na uczeniu się zarówno na podstawie **doświadczenia,** jak i **oczekiwań**.

Temporal Difference Learning

- Najprostszy algorytm ($TD(0)$):

$$V_\tau(s_t) = V_{\tau-1}(s_t) + \alpha\left(r_{t+1} + \beta V_{\tau-1}(s_{t+1}) - V_{\tau-1}(s_t)\right)$$

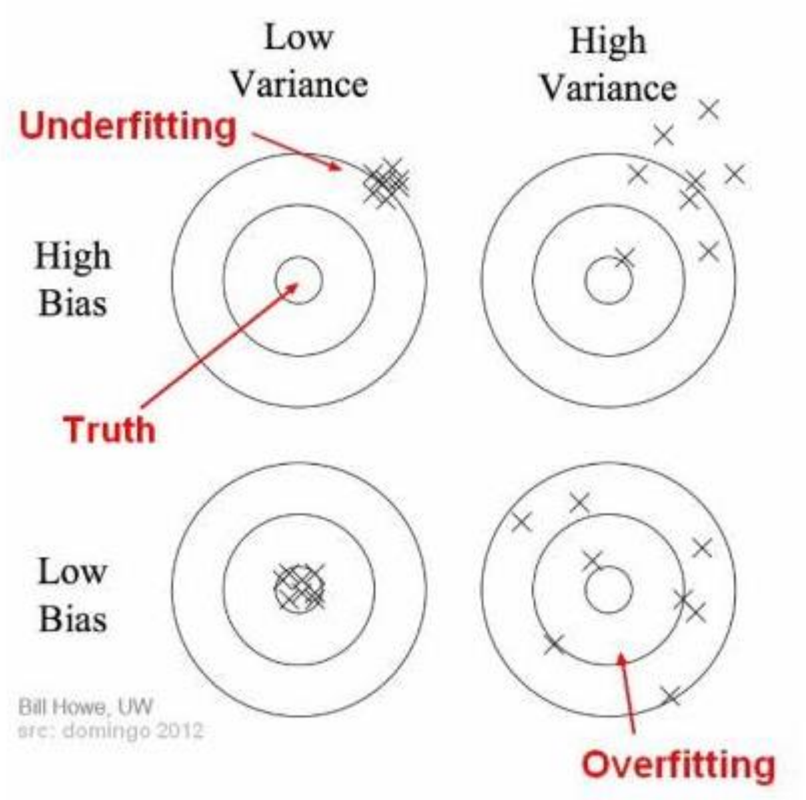
TD vs. MC

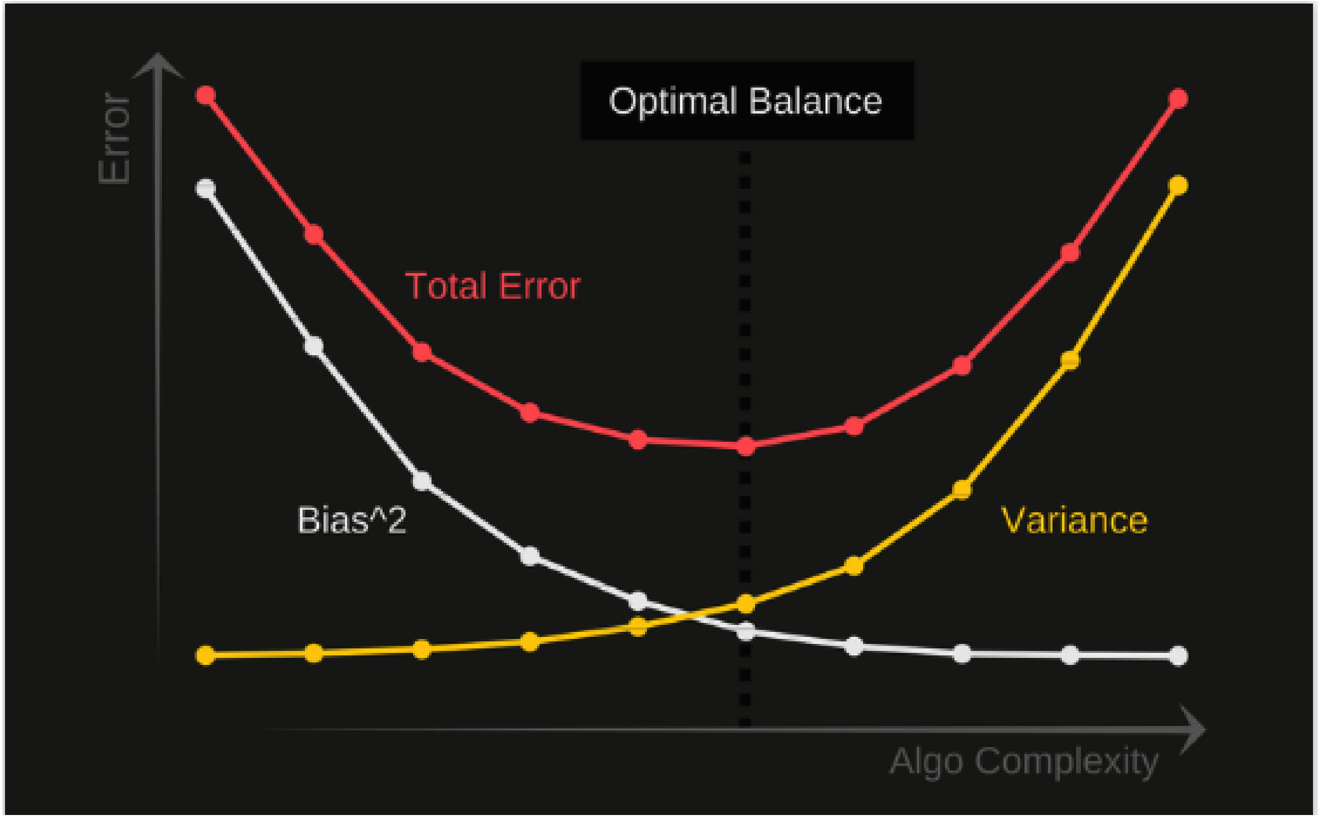
Temporal difference learning:	**Metody Monte Carlo**:
• Uczy się w trakcie trwania epizodu. • Oznacza to, że metoda może się uczyć nawet w przypadku problemów z nieskończonym horyzontem czasowym	• Uaktualnianie dzieje się po skończeniu epizodu.
• Jest obciążonym estymatorem, posiada jednak niską wariancję .	• Ma bardzo wysoką wariancję, zerowe obciążenie.
• Dużo wrażliwszy na stan początkowy.	• Niewrażliwy na stan początkowy,

• Zbiega do estymatora największej wiarygodności (MLE) procesu decyzyjnego Markowa opisującego problem:
    • Estymata $\left(S, A, \hat{P}, \hat{R}, \beta\right)$ najlepiej dopasowana do doświadczenia zbieranego w trakcie uczenia się
	• Zbiega do rozwiązania z najmniejszym błędem średniokwadratowym:
	    • Dopasowuje się do obserwowanych w trakcie epizodów nagród: $\mathrm{MSE} = \sum_{k=1}^{n} \sum_{t=1}^{\mathrm{T}_k} \left(R_t^k - V(S_t^k)\right)^2$

• Efektywnie wykorzystuje własność Markowa.	• Nie opiera się na własności Markowa, dzięki czemu możliwe jest rozwiązywanie problemów nie będących procesami decyzyjnymi Markowa.

Bias-Variance Tradeoff



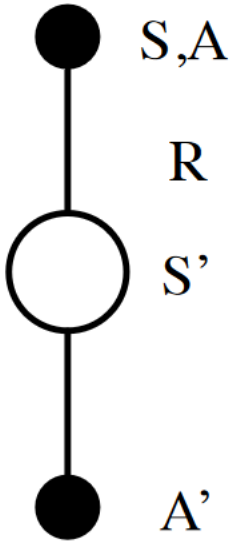


Źródło: https://towardsdatascience.com/understanding-the-bias-variance-tradeoff-165e6942b229

Temporal Difference Learning

- **On-policy**
  - SARSA
- **Off-policy**
  - Q- learning

SARSA



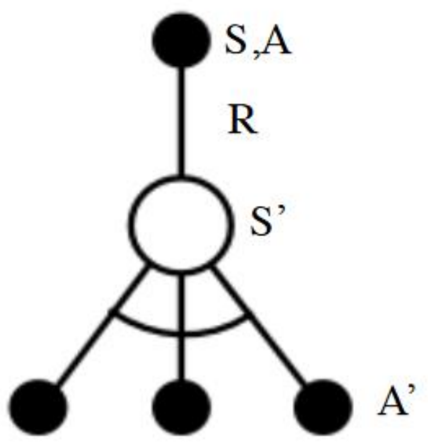
$$Q(S, A) \leftarrow Q(S, A) + \alpha \left(R + \gamma Q(S', A') - Q(S, A) \right)$$

Źródło: http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching_files/control.pdf

SARSA

1. Zainicjuj algorytm wybierając dowolne początkowe wartości funkcji wartości, np. $Q_0(s, a) = 0$ $\forall s$,a i $0 < \epsilon < 1$.
2. W każdej iteracji k:
   - Wyznacz stan początkowy S i akcję A za pomocą strategii wyprowadzonej z Q (np. ϵ-zachłannej).
   - Dla każdego S, A należącego do epizodu:
     - Podejmij akcję A i zaobserwuj nagrodę R i stan S'.
     - Wybierz A' za pomocą strategii wyprowadzonej z Q (np. ϵ-zachłannej).
     - Uaktualnij wartość funkcji $Q_k(S, A)$:
       $$Q_k(S, A) = Q_{k-1}(S, A) + \alpha(R + \beta Q_{k-1}(S', A') - Q_{k-1}(S, A))$$
     - Przyjmij, że: $\mathrm{S} = S'$ i $\mathrm{A} = A'$; kontynuuj działanie aż osiągniesz stan terminalny.

Q-learning



$$Q(S, A) \leftarrow Q(S, A) + \alpha \left(R + \gamma \max_{a'} Q(S', a') - Q(S, A) \right)$$

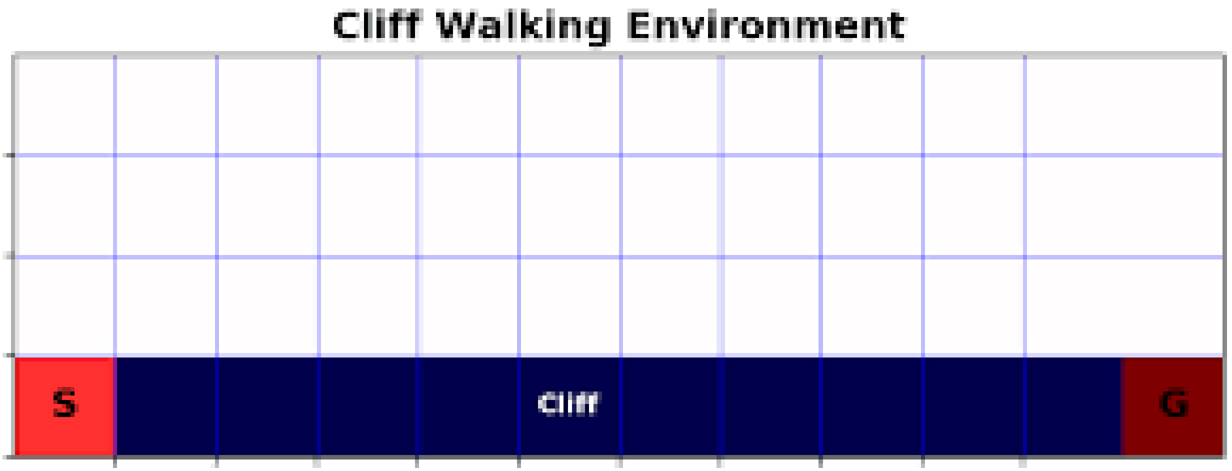
Źródło: http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching_files/control.pdf

Q-learning

1. Zainicjuj algorytm wybierając dowolne początkowe wartości funkcji wartości, np. $Q_0(s,a) = 0\ \forall s$,a i $0 < \epsilon < 1$.
2. W każdej iteracji k:
   - Wyznacz stan początkowy S
   - Dla każdego S należącego do epizodu:
     - Wyznacz akcję A za pomocą strategii wyprowadzonej z Q (np. ϵ-zachłannej).
     - Podejmij akcję A i zaobserwuj nagrodę R i stan S'.
     - Uaktualnij wartość funkcji $Q(S,A)$:
     $$Q_k(S,A) = Q_{k-1}(S,A) + \alpha(R + \beta \max_a Q_{k-1}(S',a) - Q_{k-1}(S,A))$$
     - Przyjmij, że: $S = S'$; kontynuuj działanie aż osiągniesz stan terminalny.

Przykład

Cliff Learning



Źródło: https://yun-long.github.io/2017/11/RL_CW_TD/

- Celem agenta jest przejść z punktu S do Punktu G.
- Agent może iść po białych polach musi uniknąć spadnięcia z klifu (pola granatowe).
- Tym razem świat jest w pełni deterministyczny.

Warunki zbieżności

- Algorytm zbiega do rozwiązania optymalnego $Q_*(s, a)$ gdy:
- *Greedy in the Limit with Infinite Exploration* (GLIE):
  - Wszystkie pary s, a są odwiedzone nieskończoną liczbę razy:
$$\lim_{k \to \infty} N_k(s, a) = \infty$$
  - Strategia zbiega do strategii zachłannej:
$$\lim_{k \to \infty} \pi_k\,(a|s) = \mathrm{I}(a_* = \operatorname*{argmax}_{a \in A} Q_k(s, a))$$
    - np. gdy ustalimy, że $\epsilon = \frac{1}{k}$
  - Stopa uczenia się α spełnia warunek Robbinsa-Monro:
$$\sum_{k=1}^{\infty} \alpha_k = \infty$$
$$\sum_{k=1}^{\infty} \alpha_k^2 < \infty$$

Współczynnik eksploracji ϵ

- W przypadku uczenia bezmodelowego (metody Monte Carlo, uczenie różnicowe) chcemy zachować balans pomiędzy **eksploracją** a **eksploatacją**.
- Przyjęcie stałej wartości współczynnika eksploracji ϵ jest dużym uproszczeniem z dwóch podstawowych powodów:
  - W początkowych fazach procesu uczenia chcemy żeby agent eksplorował możliwie jak najwięcej w celu zdobycia możliwie jak najpełniejszej informacji na temat otaczającego go świata.
  - Ale później, gdy jego strategia zaczyna zbiegać do strategii optymalnej, chcemy żeby jego zachowania były możliwie jak najbliższe optymalnej strategii – dzięki temu będzie on maksymalizował swoją użyteczność w każdym kolejnym epizodzie.
- Dlatego zazwyczaj przyjmujemy, że ϵ maleje z czasem.

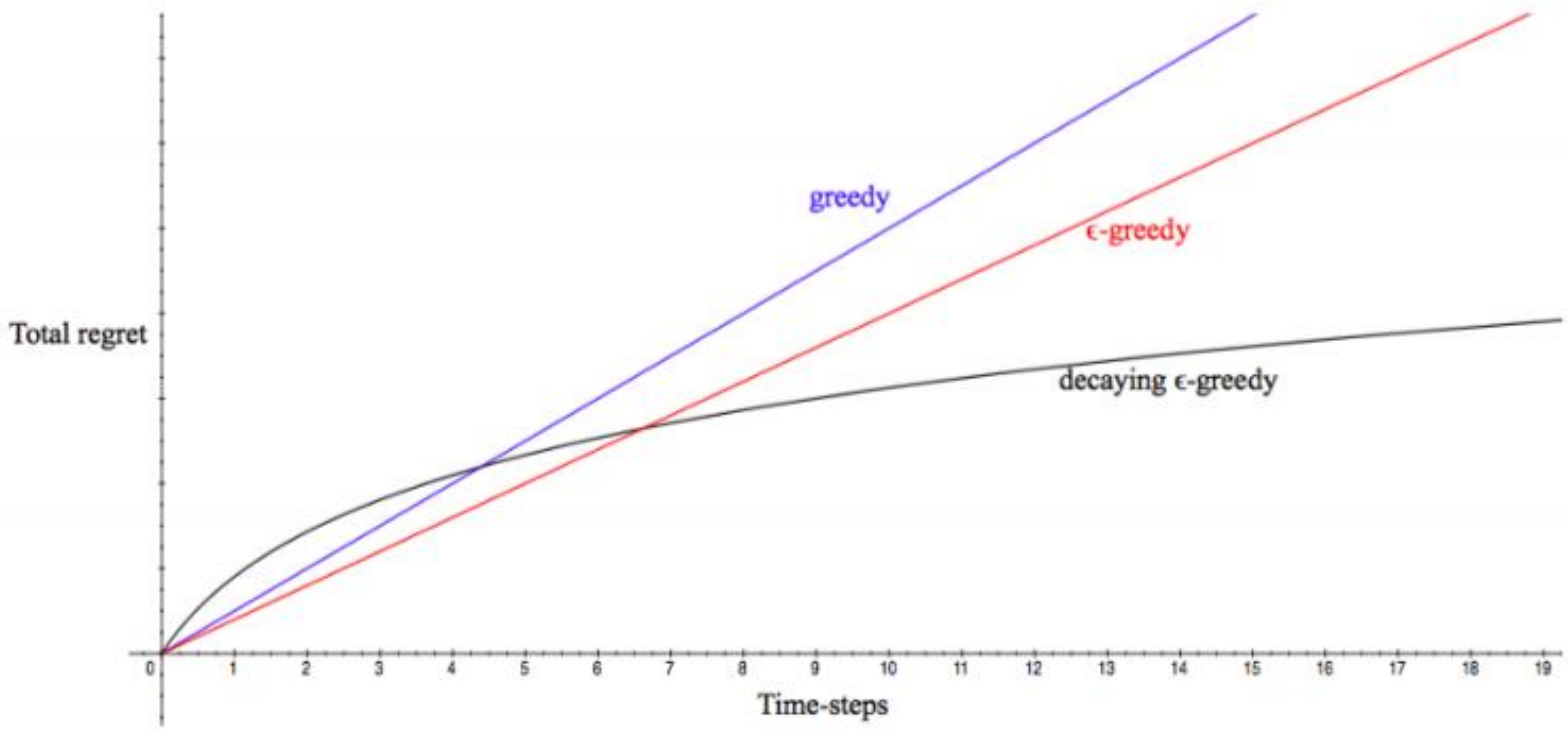
Współczynnik eksploracji ϵ

- Aby lepiej to zrozumieć zdefiniujmy pojęcie **żalu:**

$$Reg_N = NE[R(A^*)] - \sum_{k=1}^{N} R_k(A_k)$$

gdzie A^* oznacza zbiór optymalnych akcji, A_k zbiór akcji podjętych w k-tym epizodzie a $R(\cdot)$ skumulowaną nagrodę z danego epizodu.

Współczynnik eksploracji ϵ



Źródło: https://towardsdatascience.com/exploration-in-reinforcement-learning-e59ec7eeaa75

Współczynnik eksploracji ϵ

- Jak w takim razie kontrolować proces eksploracji?
  - $\epsilon = \frac{1}{k}$ - poprawny teoretycznie, w praktyce problematyczny
  - ϵ malejący liniowo – spada wolniej niż $\epsilon = \frac{1}{k}$, łatwiej go kontrolować
  - Dodać proces wypalania – agent przez pierwsze n iteracji zachowuje się całkowicie losowo, dopiero później zaczyna podejmować decyzję zgodnie ze strategią ϵ-zachłanną:

$$\epsilon = \begin{cases} 1 \; gdy \; k < n \\ \frac{1}{k} \; gdy \; k \geq n \end{cases}$$

  - Zamiast definiować ϵ zastosować **górną granicę ufności** *(upper confidence bound)*.

Upper Confidence Bound

- **Górną granicę ufności** *(upper confidence bound, UBC)*, dla epizodu t, stanu s i akcji a definiujemy jako:

$$U_t(s,a) = \sqrt{\frac{2\ln t}{N_t(s,a)}}$$

  gdzie $N_t(s,a)$ jest licznikiem odwiedzin pary s,a.

- Przy tak zdefiniowanym $U_t(s,a)$ agent wybiera akcję zachłannie maksymalizując:

$$a^* = \underset{a \in A}{\operatorname{argmax}}(Q_t(s,a) + U_t\ (s,a))$$