Zaawansowane Metody Inteligencji Obliczeniowej
Wykład 8: Monte Carlo Tree Search

Michał Kempka Marek Wydmuch

25 kwietnia 2022

Fundusze Europejskie
Polska Cyfrowa

Rzeczpospolita Polska

Unia Europejska
Europejski Fundusz Rozwoju Regionalnego

"Akademia Innowacyjnych Zastosowań Technologii Cyfrowych (AI Tech)",
projekt finansowany ze środków Programu Operacyjnego Polska Cyfrowa POPC.03.02.00-00-0001/20

Plan wykładu

Planowanie

- Dotychczas poznane metody: *model-based* (bazujące na planowaniu/przeszukiwaniu) i *model-free* (bazujące na uczeniu się).
- W obu przypadkach estymowaliśmy $V(s)$ lub $Q(s, a)$.
- Dotychczas mówiąc, że znamy model, mieliśmy na myśli znajomość pełnego rozkładu $P(S', R|S, A)$ — **model z rozkładem (ang. distribution model albo transition model)**.
- Możliwe są także inne rodzaje modeli, np. takie, które pozwalają na uzyskanie próbki (przejście S, A, R, S') pochodzącej z prawdziwego rozkładu $P(S', R|S, A)$ — **model z próbkowaniem (ang. sample models)**.
- **Modele z próbkowaniem** są dużo łatwiejsze do otrzymania niż model z rozkładem.
- Mówiąc o **planowaniu** będziemy mieć na myśli proces, który poprawia politykę na podstawie podanego modelu.

Planowanie

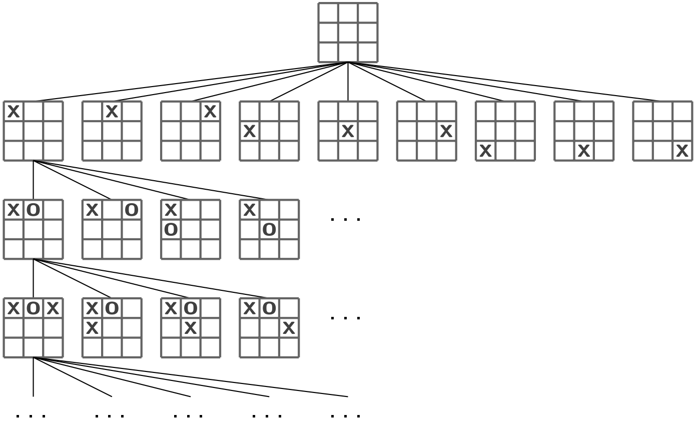
Rysunek: Garry Kasparov vs Deep Blue, 1997

Rysunek: Lee Sedol vs AlphaGo, 2016

Plan wykładu

Drzewo stanów

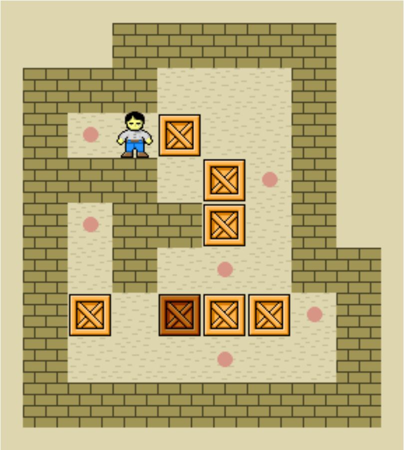
Reprezentacja środowiska jako drzewa, gdzie każdy stan odpowiada wierzchołkowi w drzewie a stany po nim następujące są jego dziećmi. Liście to stany terminalne.



Rysunek: Schemat drzewa stanów dla gry w kółko i krzyżyk.

Przeszukiwanie drzewa stanów

- Jeśli problem jest środowiskiem jedno-agentowym:
  - można go rozwiązać przeszukując jego drzewo stanów za pomocą jednego z powszechnych algorytmów przeszukiwania, np. BFS (ang. Breadth-first Search), DFS (ang. Depth-First Search)
  - jeśli drzewo zawiera dużo wierzchołków można zastosować algorytm A* z heurystyką typującą najlepsze wierzchołki do odwiedzenia w pierwszej kolejności **(dla odpowiedniej heurystyki, algorytm A* gwarantuje znalezienie optymalnego rozwiązania w minimalnej liczbie odwiedzonych wierzchołków)**
- W większość gier gramy przeciwko innym graczom – zwykłe przeszukiwanie nie działa



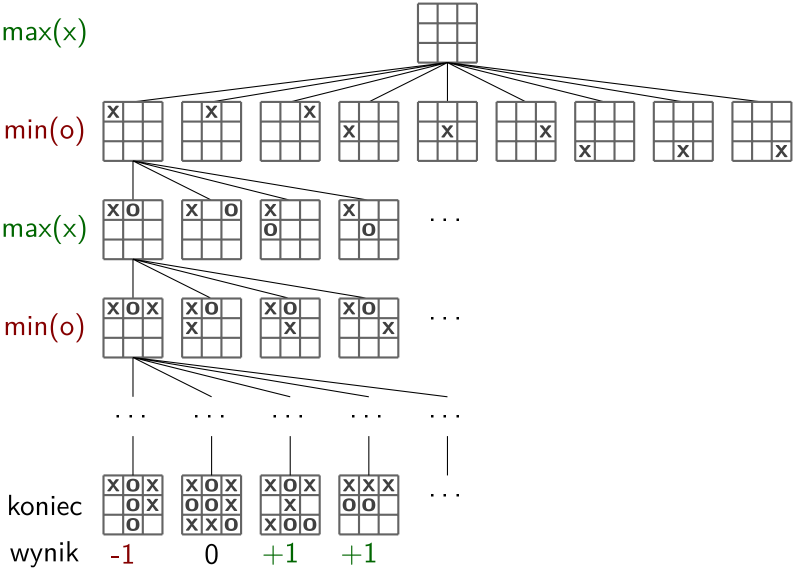
Rysunek: Przykładowa plansza z gry Sokoban

Minimax

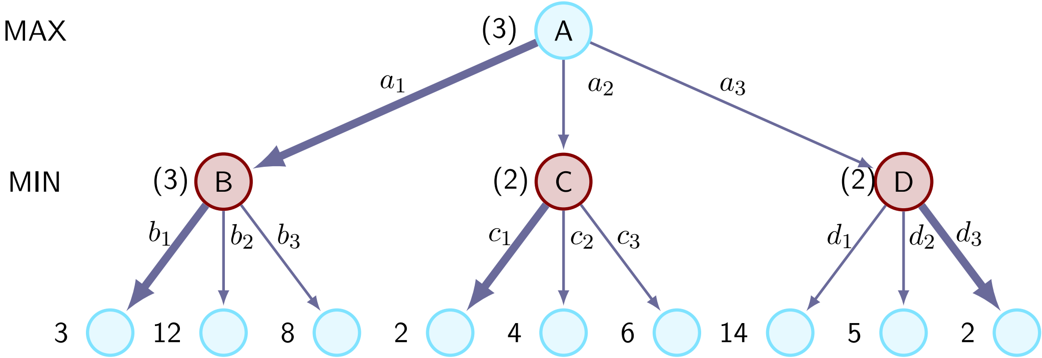
- Rozważmy grę dla dwóch graczy, deterministyczną z perfekcyjną informacją (w pełni obserwowalną), o **sumie zerowej (ang. zero-sum)**
- Musimy zmodyfikować algorytm przeszukiwania i uwzględnić przeciwnika.
- Cel przeciwnika: maksymalizacja swojej własnej nagrody $\rightarrow$ minimalizacją naszej nagrody (gra o sumie zerowej).
- Algorytm **Minimax** wykonuje przeszukiwanie wybierając akcje dla gracza, które maksymalizują nagrodę, i zakłada, że gra toczy się przeciwko nieomylnemu przeciwnikowi, który wybiera akcję minimalizujące nagrodę.
- Mając dane drzewo stanów, wartość każdego stanu zgodnie z algorytmem Minimax wyliczamy w następujący sposób:

$$\text{Minimax}(s) = \begin{cases} \text{CałkowitaNagroda}(s) & \text{if StanKońcowy}(s) \\ \max_{a \in \mathcal{A}_s} \text{Minimax}(\text{Nast.Stan}(s, a)) & \text{if Gracz}(s) = \text{MAX} \\ \min_{a \in \mathcal{A}_s} \text{Minimax}(\text{Nast.Stan}(s, a)) & \text{if Gracz}(s) = \text{MIN} \end{cases}$$

Minimax



Minimax

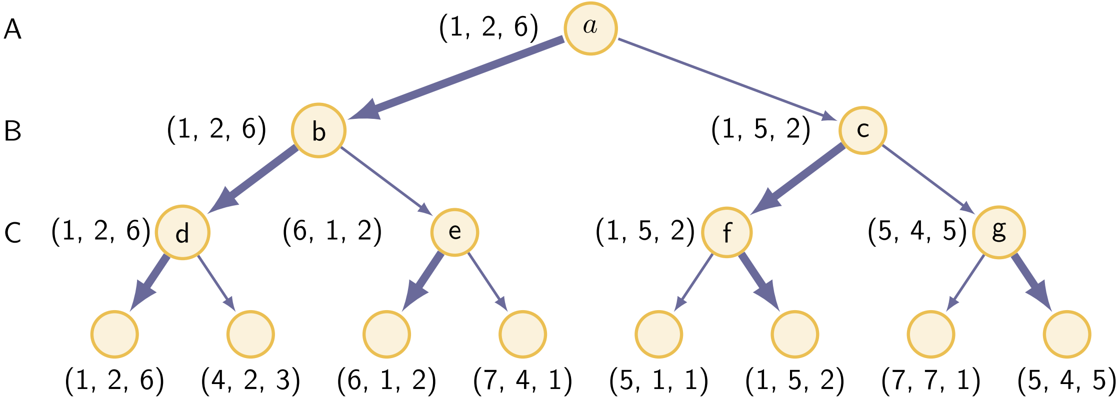


Rysunek: Przykład algorytmu Minimax.

Minimax

- Minimax wyznacza najlepszą strategię przeciwko optymalnemu przeciwnikowi.
- Jeśli przeciwnik nie gra optymalnie, Minimax zapewni nam lepszą lub równą nagrodą jak przy grze z optymalnym przeciwnikiem.
- Inna strategia może być lepsza przeciwko nieoptymalnemu przeciwnikowi.
- Minimax łatwo uogólnić dla większej liczby graczy.
- Wymaga wtedy przechowywania wektora wartości nagród (po jednej dla każdego gracza).

Minimax dla większej ilości graczy



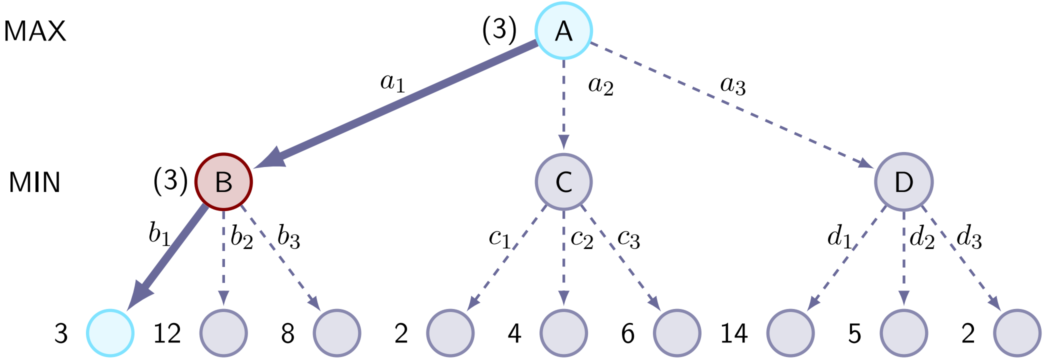
Rysunek: Przykład algorytmu Minimax dla trzech graczy A, B i C.

Plan wykładu

Alpha-beta pruning

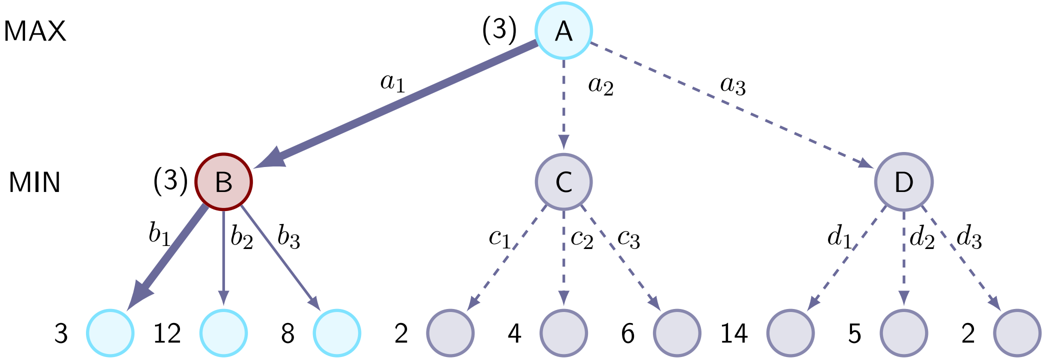
- Problem: Minimax wymaga przeszukania całej przestrzeni stanów.
- Możliwe jest wyliczenie wyniku algorytmu Minimax bez przeszukiwania wszystkich wierzchołków drzewa — algorytm **Alpha-beta pruning**.
- Idea polega na pominięciu wierzchołków, których wartości jest *niezależna* od podejmowanej decyzji.
- Alpha-beta pruning nie eliminuje wykładniczej złożoności ale znacząco ją redukuje.

Alpha-beta pruning



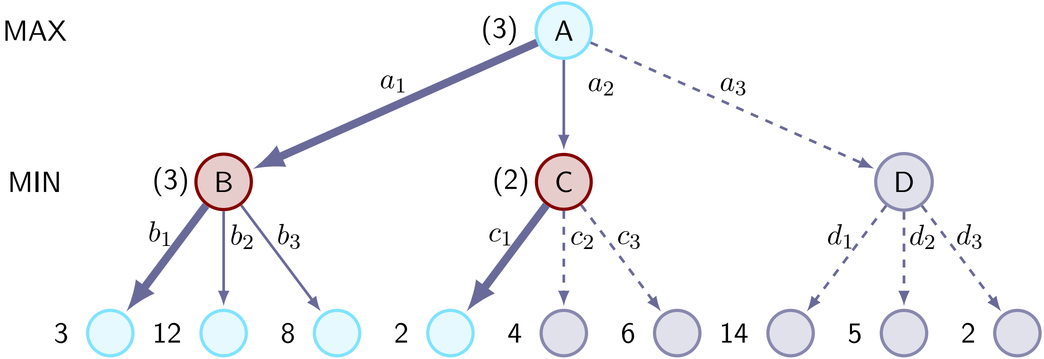
Rysunek: Przykład algorytmu Alpha-beta pruning.

Alpha-beta pruning



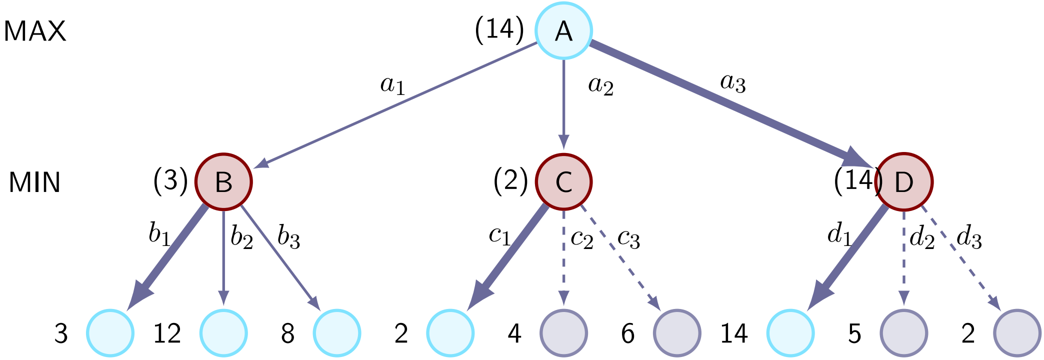
Rysunek: Przykład algorytmu Alpha-beta pruning.

Alpha-beta pruning



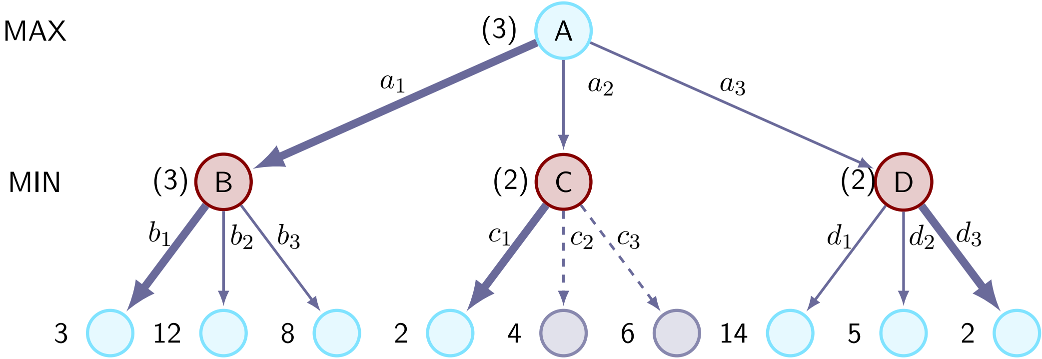
Rysunek: Przykład algorytmu Alpha-beta pruning.

Alpha-beta pruning



Rysunek: Przykład algorytmu Alpha-beta pruning.

Alpha-beta pruning



Rysunek: Przykład algorytmu Alpha-beta pruning.

Plan wykładu

Heuristic Minimax/Alpha-beta pruning

- Przeszukiwanie do danej głębokości/odcięcie przeszukiwania (Heurisitic-Minimax):

$$\text{H-Minimax}(s) = \begin{cases} \text{Ewaluacja}(s) & \text{if TestOdcięcia}(s, d) \\ \max_{a \in \mathcal{A}_s} \text{H-Minimax}(\text{Nast.Stan}(s, a), d+1) & \text{if Gracz}(s) = \text{MAX} \\ \min_{a \in \mathcal{A}_s} \text{H-Minimax}(\text{Nast.Stan}(s, a), d+1) & \text{if Gracz}(s) = \text{MIN} \end{cases}$$

- Beam-search — przeszukiwanie tylko z góry narzuconej liczby najbardziej obiecujących wierzchołków
- Look-up table — zapisana tablica optymalnych ruchów dla podzbiorów stanów, stworzona przez eksperta

Stochastic Minimax/Alpha-beta pruning

- Algortym Minimax można także zastosować do stochastycznych środowisk poprzez wprowadzenie dodatkowych wierzchołków tam gdzie następują zdarzenia losowe i wyliczaniu wartości oczekiwanej wyniku (Expected-Minimax) (wymaga znajomości modelu z rozkładem!):

$$\text{E-Minimax}(s) = \begin{cases} \text{CałkowitaNagroda}(s) & \text{if StanTerminalny}(s) \\ \max_{a \in \mathcal{A}_s} \text{E-Minimax}(\text{Nast.Stan}(s,a)) & \text{if Gracz}(s) = \text{MAX} \\ \min_{a \in \mathcal{A}_s} \text{E-Minimax}(\text{Nast.Stan}(s,a)) & \text{if Gracz}(s) = \text{MIN} \\ \sum p(s', r | a, s) \times \text{E-Minimax}(s') & \text{if Gracz}(s) = \text{CHANCE} \end{cases}$$

Plan wykładu

Monte Carlo Tree Search

- Wymyślony w 2007 roku **Monte Carlo Tree Search (MCTS)** jest bazą dla aktualnych programów do gry w szachy, go, pokera itp.
- MCTS nie wylicza wartości akcji dla każdego z możliwych stanów, a dokonuje tzw. planowania w momencie decyzji (otrzymania nowego stanu s).
- Algorytm MCTS jest oparty o inny algorytm, tzw. **rollout**

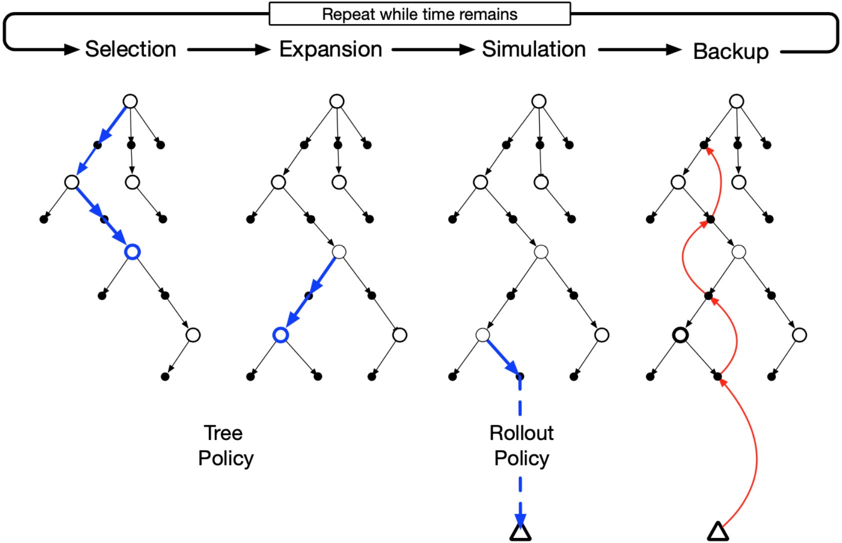
Rollout

- Rollout estymuje wartość akcji w stanie s poprzez uśrednianie wyników symulacji epizodów, które zaczynają się w stanie s oraz wybraniem każdej z możliwych akcji $a \in \mathcal{A}_s$ aż do stanu terminalnego zgodnie z daną **polityką rolloutu π (ang. rollout policy, albo ang. sample policy)**.
- Celem rolloutu nie jest estymacja optymalnej funkcji wartości albo estymacja funkcji wartości dla polityki π. Celem jest jedynie wyestymowanie wartości dla par $(s, a), a \in \mathcal{A}_s$ dla polityki π.
- Jeśli mamy dwie polityki π oraz π', które są identyczne, z wyjątkiem, że $\pi'(s) = a \neq \pi(s)$ dla jakiegoś stanu s, a $q_\pi(s, a) > v(s)$, wtedy polityka π' jest lepsza od π.
- Jeśli π to polityka rolloutu, to dochodzimy do tego, że celem rolloutu jest znalezienie lepszej polityki niż π a nie znalezienie optymalnej polityki.
- W praktyce, nawet jeśli π jest losową polityką, rollout jest skuteczny dla wielu problemów.

Monte Carlo Tree Search

- MCTS łączy ideę rollout π_r z polityką drzewa π_t, która bazując na wartościach otrzymanych przez symulacje epizodów wybiera akcje i stany, dla których warto wykonać dalsze symulacje.
- Algorytm MCTS składa się z 5 etapów, pierwsze 4 powtarzane są w pętli zadaną ilość razy (np. do upływu czasu na obliczenia):
    1. Selekcja (ang. Selection)
    2. Ekspansja (ang. Expansion)
    3. Symulacja (ang. Simulation)
    4. Powrót (ang. Backup)
    5. Wyboru akcji

Monte Carlo Tree Search



Monte Carlo Tree Search

Algorytm 1: Pseudokod dla algorytmu Monte Carlo Tree Search

Inicjalizacja:
s stan dla którego mamy wybrać akcję, π_r polityka rolloutu, π_t polityka drzewa
Ilość symulacji I, drzewo $T = \{s\}$, tablica wartości $V(S)$, początkowo $V(S) = 0$

for $i \leftarrow 1, \ldots, I$ **do**
    1. Selekcja (...)
    2. Ekspansja (...)
    3. Symulacja (...)
    4. Powrót (...)
5. Wybór akcji

Monte Carlo Tree Search

Algorytm 2: Pseudokod dla algorytmu Monte Carlo Tree Search

Inicjalizacja:
s stan dla którego mamy wybrać akcję, π_r polityka rolloutu, π_t polityka drzewa
Ilość symulacji I, drzewo $T = \{s\}$, tablica wartości $V(S)$, początkowo $V(S) = 0$

for $i \leftarrow 1, \ldots, I$ **do**
    1. Selekcja:
    $S \leftarrow s$
    while $Dzieci(S) \notin T$ **do**
        Przejdź do następnego stanu S' zgodnie z $\pi_t(S)$ (uwzględniając V)
        $S \leftarrow S'$
    2. Ekspansja:
    $T \leftarrow T \cup Dzieci(S)$
    3. Symulacja (...)
    4. Powrót (...)
5. Wybór akcji

Monte Carlo Tree Search

Algorytm 3: Pseudokod dla algorytmu Monte Carlo Tree Search

Inicjalizacja:

s stan dla którego mamy wybrać akcję, π_r polityka rolloutu, π_t polityka drzewa

Ilość symulacji I, drzewo $T = \{s\}$, tablica wartości $V(S)$, początkowo $V(S) = 0$

for $i \leftarrow 1, \ldots, I$ **do**

  1. Selekcja (...)

  2. Ekspansja (...)

  3. Symulacja:

  $S' \leftarrow S$

  while $\neg Terminalny(S')$ **do**

    Przejdź do nast. stanu S'' zgodnie z $\pi_r(S')$

    $S' \leftarrow S''$

  Wylicz sum. nagrodę R od S do S'

  4. Powrót:

  Uaktualnij estymację $V(S)$ zgodnie z R

  while $S \neq s$ **do**

    $S' \leftarrow Rodzic(S)$

    Uaktualnij estymację $V(S')$ zgodnie z R, oraz nagrodą r z przejścia S' do S.

    $S \leftarrow S'$

5. Wybór akcji

Polityka drzewa

- Polityka drzewa może być prostą polityką, ale musi dbać o eksplorację drzewa (np. ϵ-greedy).
- Dla gier dla dwóch graczy, gdzie nagroda jest przyznawana wyłącznie za wygraną grę popularnym wyborem polityka **UCB**, gdzie wybierana jest akcja maksymalizująca poniższe wyrażenie:

  $$\frac{w}{n} + c\sqrt{\frac{\ln t}{n}}$$

  gdzie w to liczba wygranych symulacji przechodzących przez dany wierzchołek, n to liczba symulacji, które przeszły przez dany wierzchołek/stan, c to stała kontrolująca eksplorację, t liczba symulacji, które przeszły przez wierzchołek rodzica.

Wybór akcji

- Ostatnim krokiem algorytmu jest wybór akcji dla stanu s (który ciągle reprezentuję aktualny stan środowiska).
- Wybór ostatecznej akcji powinien zależeć od zgromadzonych statystyk.
- Może być to akcja z największą wartością akcji.
- Może być to akcja, która została odwiedzona największą liczbę razy by uniknąć wybrania odstającej obserwacji (outlier).

Bibliografia

[1] Russell, S. and Norvig, P. (2010). *Artificial Intelligence: A Modern Approach*. Prentice Hall, third edition.

[2] Sutton, R. S. and Barto, A. G. (2018). *Reinforcement Learning: An Introduction*. The MIT Press, second edition.

Fundusze Europejskie Polska Cyfrowa

Rzeczpospolita Polska

Unia Europejska Europejski Fundusz Rozwoju Regionalnego

"Akademia Innowacyjnych Zastosowań Technologii Cyfrowych (AI Tech)",
projekt finansowany ze środków Programu Operacyjnego Polska Cyfrowa POPC.03.02.00-00-0001/20