

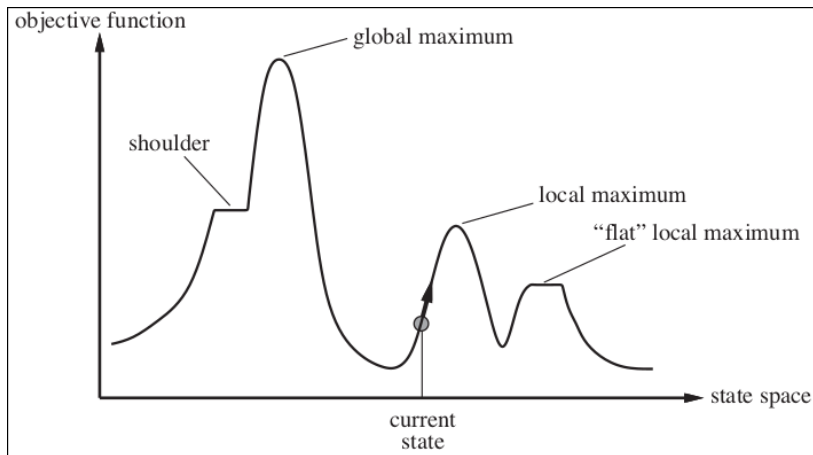
Przeszukiwanie lokalne i przeszukiwanie w grach

Paweł Rychlikowski

Instytut Informatyki UWr

9 kwietnia 2019

Krajobraz przeszukiwania lokalnego



Czym może być funkcja, którą minimalizujemy?

1. Liczbą **niespełnionych** więzów.

Czym może być funkcja, którą minimalizujemy?

1. Liczbą **niespełnionych** więzów.
2. **Wagą** niespełnionych więzów.

Czym może być funkcja, którą minimalizujemy?

1. Liczbą **niespełnionych** więzów.
2. **Wagą** niespełnionych więzów. Porównaj więzy:
 1. Nauczyciel ma tylko z jedną klasą lekcje na raz
 2. nikt nie ma dwóch biologii jednego dnia.

Czym może być funkcja, którą minimalizujemy?

1. Liczbą **niespełnionych** więzów.
2. **Wagą** niespełnionych więzów. Porównaj więzy:
 1. Nauczyciel ma tylko z jedną klasą lekcje na raz
 2. nikt nie ma dwóch biologii jednego dnia.
3. Czymś niezwiązanym bezpośrednio z więzami

Czym może być funkcja, którą minimalizujemy?

1. Liczbą **niespełnionych** więzów.
2. **Wagą** niespełnionych więzów. Porównaj więzy:
 1. Nauczyciel ma tylko z jedną klasą lekcje na raz
 2. nikt nie ma dwóch biologii jednego dnia.
3. Czymś niezwiązanym bezpośrednio z więzami
 - produktywnością zespołu robotników
 - zadowoleniem gości weselnych z towarzystwa przy stolikach,
 - potencjalnym zyskiem sklepu,
 - dopasowaniem do danych uczących

Hill climbing jest chyba najbardziej naturalnym algorytmem inspirowanym poprzednim rysunkiem.

- Dla stanu znajdujemy wszystkie następniki i wybieramy ten, który ma największą wartość.
- Powtarzamy aż do momentu, w którym nie możemy nic poprawić

Hill climbing jest chyba najbardziej naturalnym algorytmem inspirowanym poprzednim rysunkiem.

- Dla stanu znajdujemy wszystkie następniiki i wybieramy ten, który ma największą wartość.
- Powtarzamy aż do momentu, w którym nie możemy nic poprawić

Problem

Utknięcie w lokalnym maksimum.

Hill climbing z losowymi restartami

Uwaga

Możemy podjąć dwa działania, oba testowaliśmy w obrazkach logicznych:

1. Dorzucać ruchy niekoniecznie poprawiające (losowe, ruchy **w bok**)
2. Gdy nie osiągamy rozwiązania przez dłuższy czas rozpoczynamy od początku.

Hill climbing + random restarts (w trywialny sposób) jest algorytmem zupełnym z p-stwem 1 (bo kiedyś wylosujemy układ startowy)

Inne warianty Hill climbing

- a) **Stochastic hill climbing** – wybieramy losowo ruchy w górę (p-stwo stałe, albo zależne od wielkości skoku).
- b) **First choice hill climbing** – losujemy następnika tak długo, aż będzie on ruchem w górę
 - dobre, jeżeli następników jest bardzo dużo

Uwaga

Idee z tego i kolejnych algorytmów można dowolnie mieszać – na pewno coś wyjdzie!

- Motywacja fizyczna: ustalanie struktury krystalicznej metalu.
- Jeżeli będziemy ochładzać powoli, to metal będzie silniejszy (bliżej globalnego minimum energetycznego).
- **Symulowane wyżarzanie** – próba oddania tej idei w algorytmie.

Algorytm

Symulujemy opadającą temperaturę, prawdopodobieństwo ruchu chaotycznego zależy **malejąco** od temperatury.

Symulowane wyżarzanie (2)

- Przykładowa implementacja bazuje na **first choice hill climbing**.
- Jak wylosowany ruch (**r**) jest lepszy (czyli $\Delta F > 0$), to go wykonujemy (maksymalizacja F).
- W przeciwnym przypadku wykonujemy ruch **r** z p-stwem $p = e^{\frac{\Delta F}{T}}$
- Pilnujemy, żeby T zmniejszało się w trakcie działania (i było cały czas dodatnie)

Komentarze do wzoru

- $\Delta F \leq 0$, $T > 0$, czyli $0 \leq p \leq 1$.
- Im większe pogorszenie, tym mniejsze p-stwo
- Im większa temperatura, tym większe p-stwo.

Problem

Być może płaskie maksimum lokalne.

Rozwiązanie

Dodajemy pamięć algorytmowi, zabraniamy powtarzania ostatnio odwiedzanych stanów.

Local beam search

- Zamiast pamiętać pojedynczy stan, pamiętamy ich k (wiązkę).
- Generujemy następniki dla każdego z k stanów.
- Pozostawiamy k liderów.

Uwaga 1

To nie to samo co k równoległych wątków hill-climbing (bo uwaga algorytmu może przerzucać się do bardziej obiecujących kawałków przestrzeni)

Uwaga 2

Beam search jest bardzo popularnym algorytmem w różnych zadaniach wykorzystujących sieci neuronowe do modelowania sekwencji (np. tłumaczenie maszynowe).

- Zarządzamy **populacją** osobników (czyli np. pseudorozwiązań jakiegoś problemu więzowego).
- Mamy dwa rodzaje operatorów:
 - a) Mutacja, która z jednego osobnika robi innego, podobnego.
 - b) Krzyżowanie, która z dwóch osobników robi jednego, w jakiś sposób podobnego do „rodziców”.
- Nowe osobniki oceniane są ze względu na wartość **funkcji przystosowania**
- Przeżywa k najlepszych.

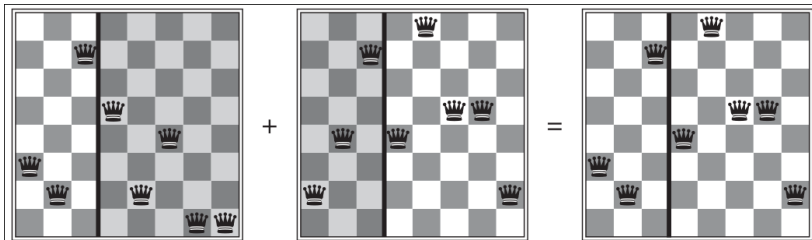
Uwaga

Zauważmy, że choć zmienił się język, jeżeli pominiemy krzyżowanie, to otrzymamy wariant Local beam search (mutacja jako krok w przestrzeni stanów).

Krzyżowanie. Przykład

Pytanie

Czym mogłoby być krzyżowanie dla zadania z N hetmanami?



Algorytmy ewolucyjne. Kilka uwag

1. Krzyżowanie i mutacje można zorganizować tak, że najpierw powstają **dzieci**, a następnie się mutują z pewnym prawdopodobieństwem.
2. Wybór osobników do rozmnażania może zależeć od funkcji dopasowania (większe szanse na reprodukcję mają lepsze osobniki)
3. Można mieć wiele operatorów krzyżowania i mutacji.

Przeszukiwanie w grach

- Gracz **A** wybiera jeden z trzech zbiorów:
 1. $\{-50, 50\}$
 2. $\{1, 3\}$
 3. $\{-5, 15\}$
- Następnie gracz **B** wybiera liczbę z tego zbioru.

Pytanie

Co powinien zrobić **A**, żeby uzyskać jak największą liczbę?

Nasza gra

1. $\{-50, 50\}$
2. $\{1, 3\}$
3. $\{-5, 15\}$

Racjonalny wybór dla **A** zależy od (modelu) gracza **B**

- Współpracujący: Oczywiście 1.
- Losowy (z $p = \frac{1}{2}$)) Wybór 3 (średnio 5)
- „Złośliwy”: wybór 2 (gwarantujemy wartość 1)

- Nieco inna rodzina zadań wyszukiwania, w których mamy dwóch (lub więcej) agentów.
- Interesy agentów są (przynajmniej częściowo) rozbieżne.
- Rozgrywka przebiega w turach, w których gracze na zmianę wybierają swoje ruchy.

Definicja

Gra jest problemem przeszukiwania, zadany przez następujące składowe:

1. Zbiór stanów, a w nim S_0 , czyli stan początkowy
2. $\text{player}(s)$, funkcja określająca gracza, który gra w danym stanie.
3. $\text{actions}(s)$ – zbiór ruchów możliwych w stanie s
4. $\text{result}(s,a)$ – funkcja zwracająca stan powstały w wyniku zastosowania akcji a w stanie s .
5. $\text{terminal}(s)$ – funkcja sprawdzająca, czy dany stan kończy grę.
6. $\text{utility}(s, \text{player})$ – funkcja o wartościach rzeczywistych, opisująca wynik gry z punktu widzenia danego gracza.

Definicja

W **grze o sumie zerowej** suma wartości stanów terminalnych dla wszystkich graczy jest stała (niekonieczne zera, ale...)

Konsekwencje:

- Zysk jednego gracza, jest stratą drugiego.
- Kooperacja nic nie daje.

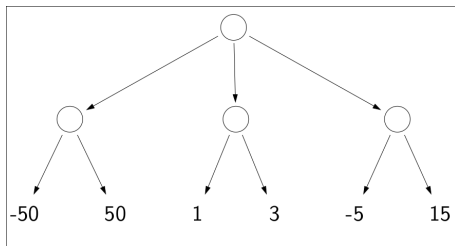
Uwaga

Zaczniemy od gier o sumie zerowej i gracza, wcześniej nazwanego **złośliwym** (lepiej go nazwać **racjonalnym**)

Różnice między grami a zwykłym przeszukiwaniem

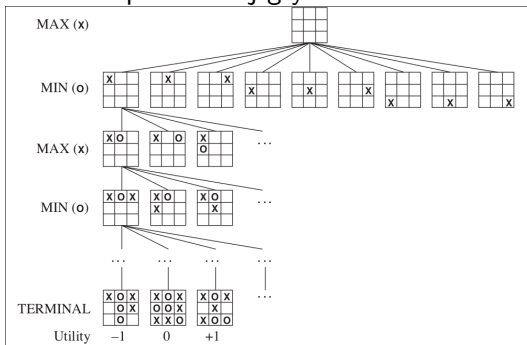
- ❶ Mamy graczy: stan gry wskazuje na gracza, który ma się ruszać.
- ❷ Stany końcowe mają wartości, różne dla różnych graczy.
- ❸ Koszt jest zwykle jednostkowy (inny można uwzględnić w końcowej wypłacie, dodając do stanu „finanse” gracza)

Drzewo gry



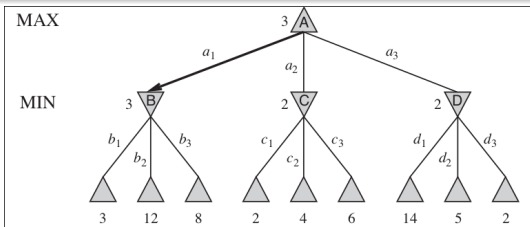
Kółko i krzyżyk. Drzewo gry

Fragment drzewa dla prawdziwej gry



Drzewo gry (2)

- Mamy dwóch graczy Max i Min (jeden chce maksymalizacji, drugi minimalizacji).
- Wartość dla Max-a to liczba przeciwna wartości dla Min-a.
- Mamy dwa ruchy, zaczyna gracz maksymalizujący.



Algorytm MinMax

```
MAX = 1
MIN = 0

def decision(state):
    """decision for MAX"""
    return max(a for actions(state),
               key = lambda a : minmax(result(a,state), MIN))

def minmax(state, player):
    if terminal(state): return utility(state)

    values = [minmax(result(a,state), 1-player) for a in actions(state)]
    if player == MIN:
        return min(values)
    else:
        return max(values)
```

- $O(d)$ – pamięć
- $O(b^{2d})$ czas, gdzie d jest liczbą ply's (półruchów)
- Dla szachów $b \approx 35$, $d \approx 50$
- Dla go: 250, 150

Algorytm MinMax (wersja realistyczna)

- Algorytm MinMax działa jedynie dla bardzo małych, sztucznych gier (ewentualnie dla końcówek prawdziwych gier).
- Żeby go uczynić realistycznym, musimy:
 - a) Przerwać poszukiwania na jakiejś głębokości.
 - b) Umieć szacować wartość nieterminalnych sytuacji na planszy.

Algorytm MinMax z głębokością

```
def decision(state):  
    return max(a for actions(state),  
               key = lambda a : minmax(result(a,state), MIN), 0))  
  
def minmax(state, player, depth):  
    if terminal(state): return utility(state)  
    if cut_off_test(state, depth):  
        return heuristic_value(state)  
  
    values = [minmax(result(a,state), 1-player, depth+1) for a in actions(state)]  
    if player == 0:  
        return min(values)  
    else:  
        return max(values)
```


Dwa parametry algorytmu wyszukiwania

1. **cut_off_test**: kiedy kończymy przeszukiwanie
 - najłatwiej: jak osiągniemy maksymalny poziom, biorąc pod uwagę możliwości
 - Nie jest to jedyne wyjście (ani najlepsze)
2. Co to znaczy funkcja **heuristic_value**

Jak szacować wartość sytuacji?

Wariant 1

Korzystamy z wiedzy eksperta, próbując ją sformalizować.

Wariant 2

Próbujemy zaprząć jakiś mechanizm uczenia (lub przeszukiwania), żeby tę funkcję wybrać.

Jak szacować wartość sytuacji? (2)

Generalne wskazówki:

1. Przewaga materialna (więcej, lepszych figur)
2. Ustawienie figur (ruchliwość – liczba możliwych ruchów)
3. Szacowana liczba ruchów do zwycięstwa (zagrożony król, itp).
4. Ochrona naszych figur (jak mnie zbijesz, to ja cię zaraz zbiję)

Przewaga materialna

- Wartość materialną liczą powszechnie szachiści:
 - a) pion: 1
 - b) skoczek, goniec: 3
 - c) wieża: 5
 - d) hetman: 9
- Sprawdzono doświadczalnie, że te wartości są dobrze dobrane (jak sobie wyobrazić taki eksperyment?)

Uwaga

Nawet nie wiedząc nic o uczeniu, możemy sobie wyobrazić łatwo jakąś procedurę wyznaczania tych wartości. Na przykład:

1. Losujemy 100 zestawów:
(1, wartość-gońca, wartość-skoczka, wartość-wieży, wartość-hetmana).
2. Przeprowadzamy pojedynki każdy z każdym.
3. Wybieramy zwycięzcę.

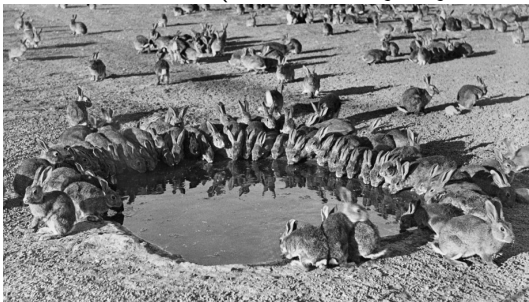
- Istnieje pokusa, żeby zastosować algorytmy ewolucyjne (bo zadanie przypomina ewolucję, w której osobniki toczą ze sobą walkę).
- **Problem:** Jak wyznaczyć funkcję celu?
 - a) Rozgrywać turnieje, przystosowaniem jest średni wynik.
 - b) Wybrać grupę przeciwników (stałą), przystosowaniem X -a będzie średni wynik z tymi przeciwnikami.

Drobna uwaga o ewolucji

Uwaga

Opcja pełnej ewolucji trochę niebezpieczna, często łączy się oba warianty.

Króliczki w Australii (obok australijskiej ewolucji):



AlphaGo zachowywało swoje poprzednie wersje i pilnowało, by kolejna wersja ciągle umiała pokonać starsze modele