Inteligencja obliczeniowa - Algorytm genetyczny

Grzegorz Madejski

Inteligencja obliczeniowa

Definicja

Sztuczna inteligencja: zdolność maszyny do poprawnego interpretowania danych zewnętrznych, uczenia się z nich, i używania ich do osiągnięcia określonych celów dzięki adaptacji (Kaplan, Haenlein); ogólniej: dziedzina informatyki zajmująca się technikami wykorzystującymi powyższą zdolność

Inteligencja obliczeniowa

Definicja

Sztuczna inteligencja: zdolność maszyny do poprawnego interpretowania danych zewnętrznych, uczenia się z nich, i używania ich do osiągnięcia określonych celów dzięki adaptacji (Kaplan, Haenlein); ogólniej: dziedzina informatyki zajmująca się technikami wykorzystującymi powyższą zdolność

Definicja

Inteligencja obliczeniowa: poddziedzina sztucznej inteligencji skupiająca się na adapcyjnych algorytmach, które umożliwiają lub usprawniają inteligentne zachowanie w złożonym lub zmieniającym się środowisku (Engelbrecht)

Inteligencja obliczeniowa

Techniki z zakresu inteligencji obliczeniowej i sztucznej inteligencji:

- algorytmy ewolucyjne (w tym: genetyczne)
- sztuczne sieci neuronowe
- logika rozmyta
- inteligencja roju

Inteligencja obliczeniowa

Techniki z zakresu inteligencji obliczeniowej i sztucznej inteligencji:

- algorytmy ewolucyjne (w tym: genetyczne)
- sztuczne sieci neuronowe
- logika rozmyta
- inteligencja roju

Inne techniki, którymi zajmuje się już tylko sztuczna inteligencja:

- rozwiązywanie problemów przez przeszukiwanie przestrzeni stanów
- logika, wnioskowanie
- rozumowanie dedukcyjne
- systemy ekspertowe
- symboliczne i subsymboliczne paradygmaty

Inteligencja obliczeniowa

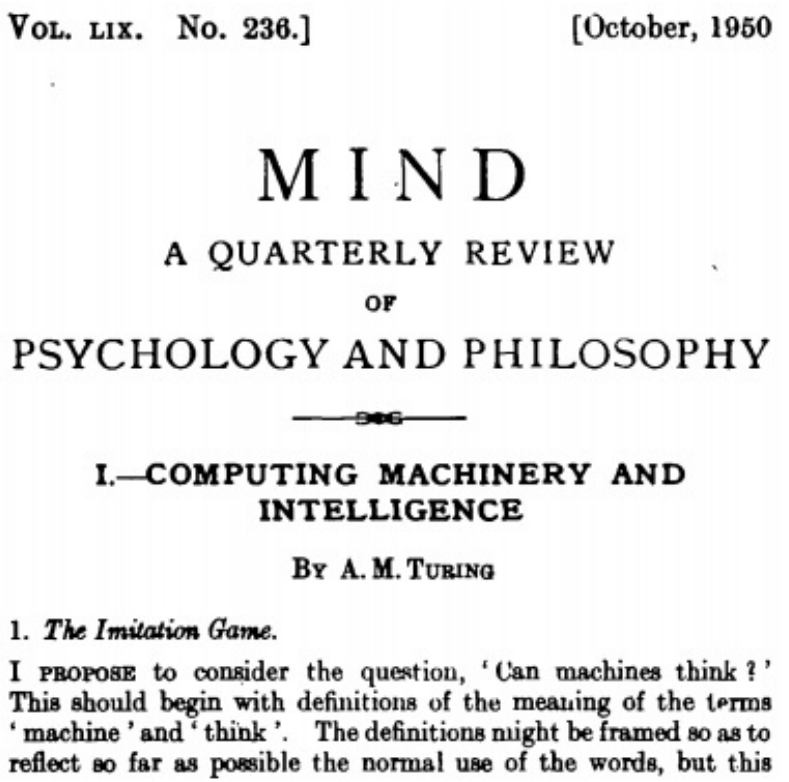
- Widać, że algorytmy inteligencji obliczeniowej są bardziej "miękkie", mocniej inspirowane biologią/naturą

Inteligencja obliczeniowa

- Widać, że algorytmy inteligencji obliczeniowej są bardziej "miękkie", mocniej inspirowane biologią/naturą
- Inteligencja obliczeniowa jest synonimiczna z terminem *soft computing*, czyli zbiorem algorytmów, które tolerują pewną niedokładność i aproksymują rozwiązania
- To przeciwieństwo tzw. *hard computing*, w którym algorytm musi znaleźć optymalne rozwiązanie

Historia algorytmu genetycznego

Rok 1950, Alan Turing wspomina o maszynie, która naśladowałaby ewolucję.



VOL. LIX. No. 236.] [October, 1950

MIND

A QUARTERLY REVIEW

OF

PSYCHOLOGY AND PHILOSOPHY

I.—COMPUTING MACHINERY AND INTELLIGENCE

BY A. M. TURING

1. *The Imitation Game.*

I PROPOSE to consider the question, 'Can machines think?' This should begin with definitions of the meaning of the terms 'machine' and 'think'. The definitions might be framed so as to reflect so far as possible the normal use of the words, but this

Historia algorytmu genetycznego

- Lata 1950-60, pierwsze próby symulacji komputerowych ze strategiami ewolucyjnym (Nils Aall Barricelli), rozwinięcie koncepcji algorytmów ewolucyjnych (Alx Fraser, Hans-Joachim Bremermann i inni).

Historia algorytmu genetycznego

- Lata 1950-60, pierwsze próby symulacji komputerowych ze strategiami ewolucyjnym (Nils Aall Barricelli), rozwinięcie koncepcji algorytmów ewolucyjnych (Alx Fraser, Hans-Joachim Bremermann i inni).
- Lata 1970, algorytmy ewolucyjne zyskują na popularności (Ingo Rechenberg, Hans-Paul Schwefel, Lawrence J. Fogel, John Holland i inni)

Historia algorytmu genetycznego

- Lata 1950-60, pierwsze próby symulacji komputerowych ze strategiami ewolucyjnym (Nils Aall Barricelli), rozwinięcie koncepcji algorytmów ewolucyjnych (Alx Fraser, Hans-Joachim Bremermann i inni).
- Lata 1970, algorytmy ewolucyjne zyskują na popularności (Ingo Rechenberg, Hans-Paul Schwefel, Lawrence J. Fogel, John Holland i inni)
- Lata 1980, pierwsza konferencja o algorytmach genetycznych, algorytmy genetyczne wykorzystywane w przemyśle, algorytm komercyjny Evolver (1989) na PC

Strategie inspirowane naturą

- *Algorytmy ewolucyjne, genetyczne*: wykorzystują symulację ewolucji, zmiany genów do rozwiązywania problemów

Strategie inspirowane naturą

- *Algorytmy ewolucyjne, genetyczne*: wykorzystują symulację ewolucji, zmiany genów do rozwiązywania problemów
- *Inteligencja roju* (Swarm Intelligence): wykorzystuje symulację roju owadów do określania najlepszego rozwiązania problemu

Strategie inspirowane naturą

- *Algorytmy ewolucyjne, genetyczne*: wykorzystują symulację ewolucji, zmiany genów do rozwiązywania problemów
- *Inteligencja roju* (Swarm Intelligence): wykorzystuje symulację roju owadów do określania najlepszego rozwiązania problemu
- *Sieci neuronowe*: naśladują mózg jako sieć neuronów

Strategie inspirowane naturą

- *Algorytmy ewolucyjne, genetyczne*: wykorzystują symulację ewolucji, zmiany genów do rozwiązywania problemów
- *Inteligencja roju* (Swarm Intelligence): wykorzystuje symulację roju owadów do określania najlepszego rozwiązania problemu
- *Sieci neuronowe*: naśladują mózg jako sieć neuronów
- *Sztuczne życie, automaty komórkowe*: symulują rozwój mikroorganizmów i ich walkę o pożywienie, wykorzystywane do modelowania zjawisk, kryptografii

Algorytm genetyczny

Algorytm genetyczny (AG) to strategia rozwiązywania problemów za pomocą funkcji (meta)heurystycznej inspirowanej naturą, a konkretniej ewolucją. AG wykorzystuje pewne zjawiska biologiczne do znajdowania najlepszego rozwiązania danego problemu:

- rozwiązanie jako organizm z zestawem genów (chromosom)

Algorytm genetyczny

Algorytm genetyczny (AG) to strategia rozwiązywania problemów za pomocą funkcji (meta)heurystycznej inspirowanej naturą, a konkretniej ewolucją. AG wykorzystuje pewne zjawiska biologiczne do znajdowania najlepszego rozwiązania danego problemu:

- rozwiązanie jako organizm z zestawem genów (chromosom)
- Przystosowanie do warunków środowiska

Algorytm genetyczny

Algorytm genetyczny (AG) to strategia rozwiązywania problemów za pomocą funkcji (meta)heurystycznej inspirowanej naturą, a konkretniej ewolucją. AG wykorzystuje pewne zjawiska biologiczne do znajdowania najlepszego rozwiązania danego problemu:

- rozwiązanie jako organizm z zestawem genów (chromosom)
- Przystosowanie do warunków środowiska
- Selekcję najzdrowszych organizmów do dalszego przekazywania genów

Algorytm genetyczny

Algorytm genetyczny (AG) to strategia rozwiązywania problemów za pomocą funkcji (meta)heurystycznej inspirowanej naturą, a konkretniej ewolucją. AG wykorzystuje pewne zjawiska biologiczne do znajdowania najlepszego rozwiązania danego problemu:

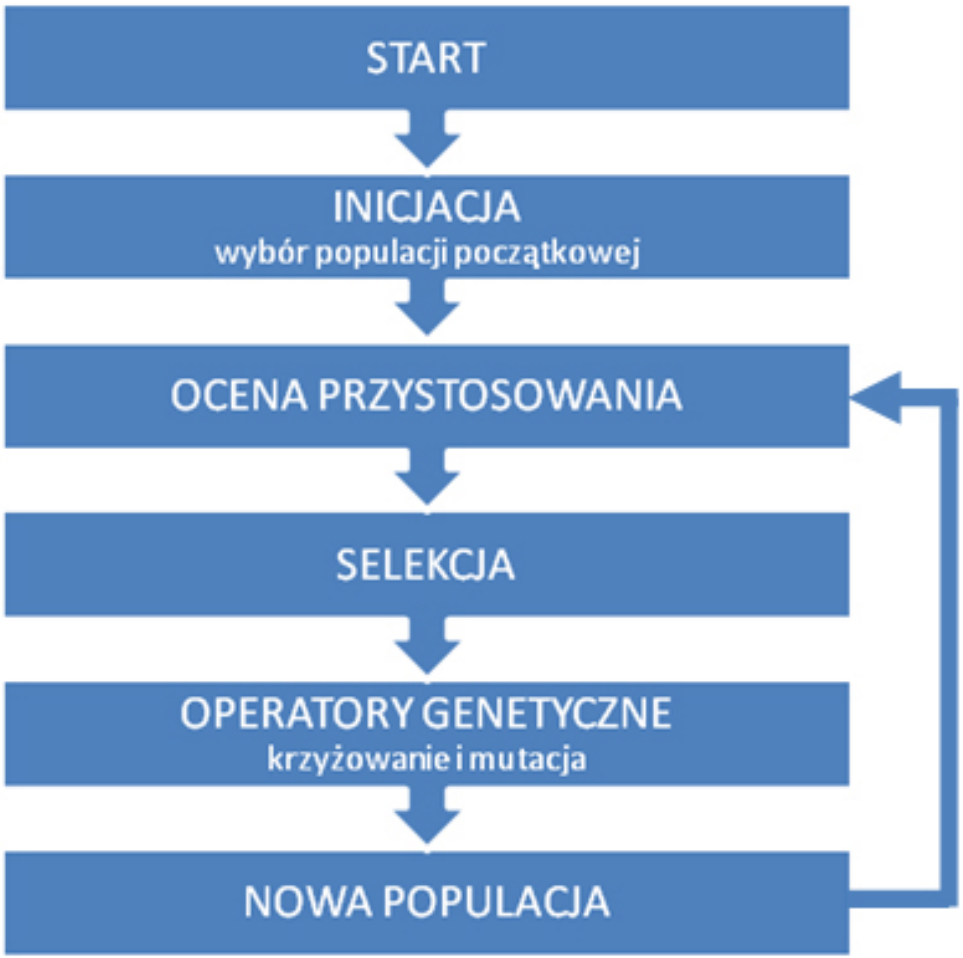
- rozwiązanie jako organizm z zestawem genów (chromosom)
- Przystosowanie do warunków środowiska
- Selekcję najzdrowszych organizmów do dalszego przekazywania genów
- Krzyżowanie się, mutację

Algorytm genetyczny

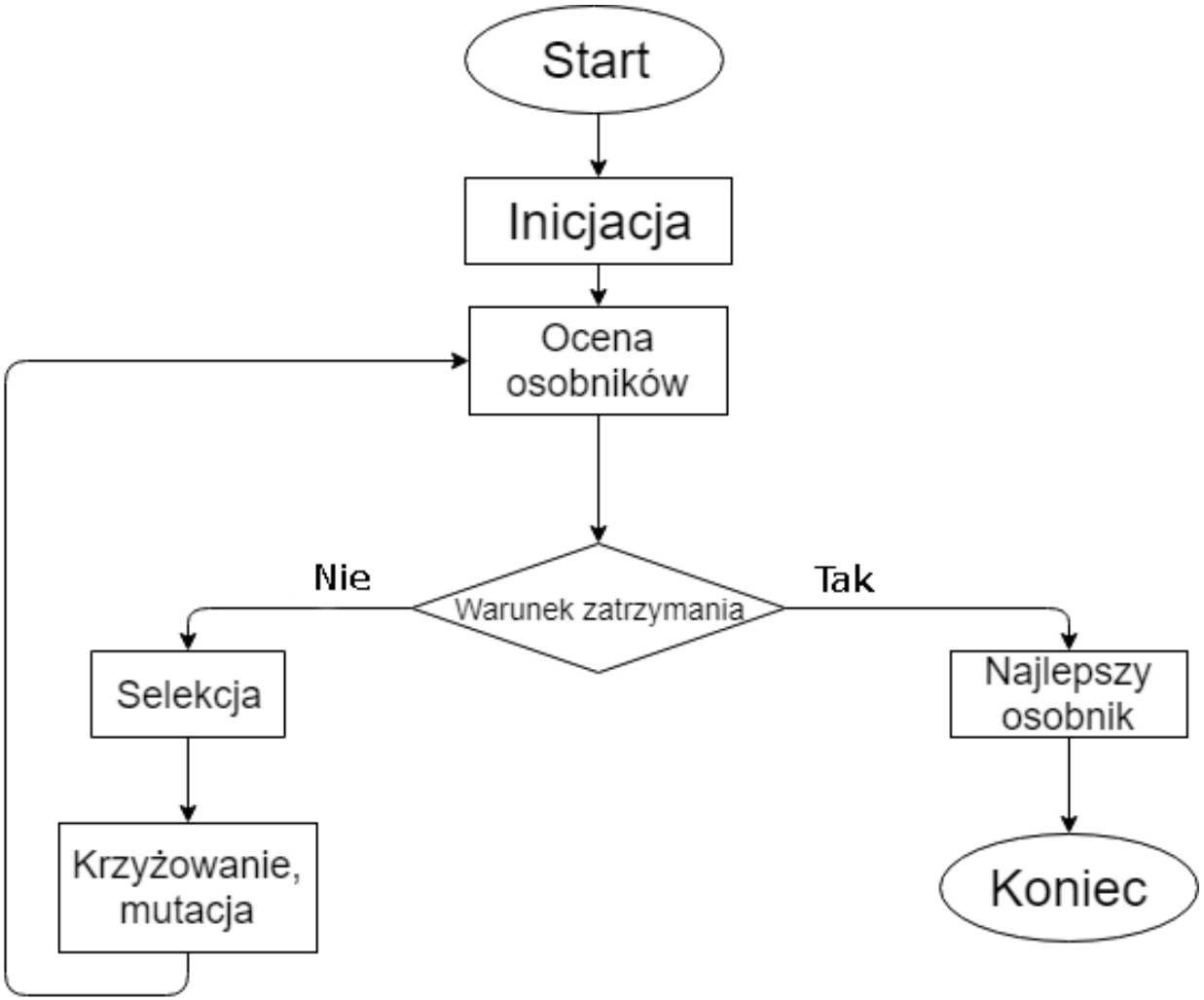
Algorytm genetyczny (AG) to strategia rozwiązywania problemów za pomocą funkcji (meta)heurystycznej inspirowanej naturą, a konkretniej ewolucją. AG wykorzystuje pewne zjawiska biologiczne do znajdowania najlepszego rozwiązania danego problemu:

- rozwiązanie jako organizm z zestawem genów (chromosom)
- Przystosowanie do warunków środowiska
- Selekcję najzdrowszych organizmów do dalszego przekazywania genów
- Krzyżowanie się, mutację
- Zmiany pokoleń, zachodząca ewolucja

Algorytm genetyczny



Algorytm genetyczny

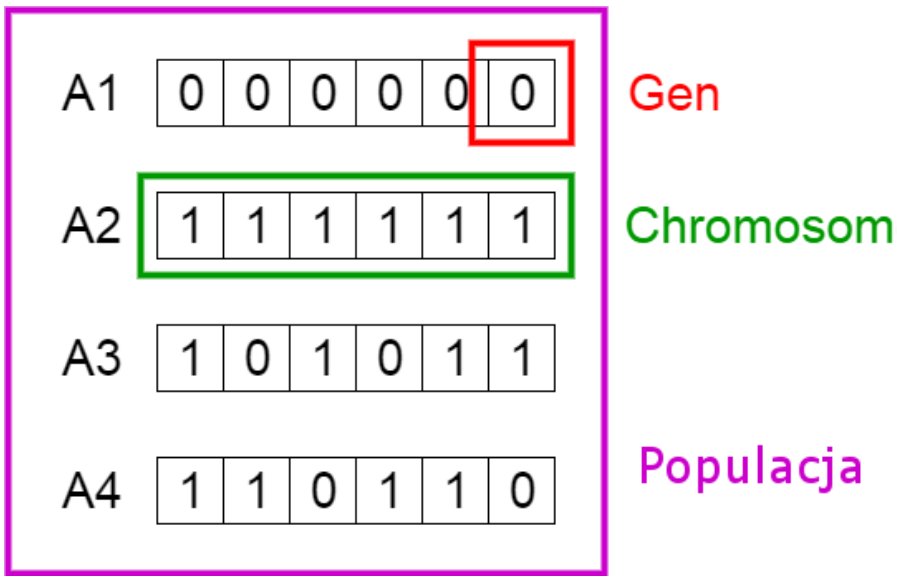


Algorytm genetyczny: populacja początkowa

Populację początkową wybieramy losowo.
Wybór wielkości populacji jest ważny. Z reguły zawiera setki lub tysiące osobników. Wielkość populacji zależy od złożoności problemu.

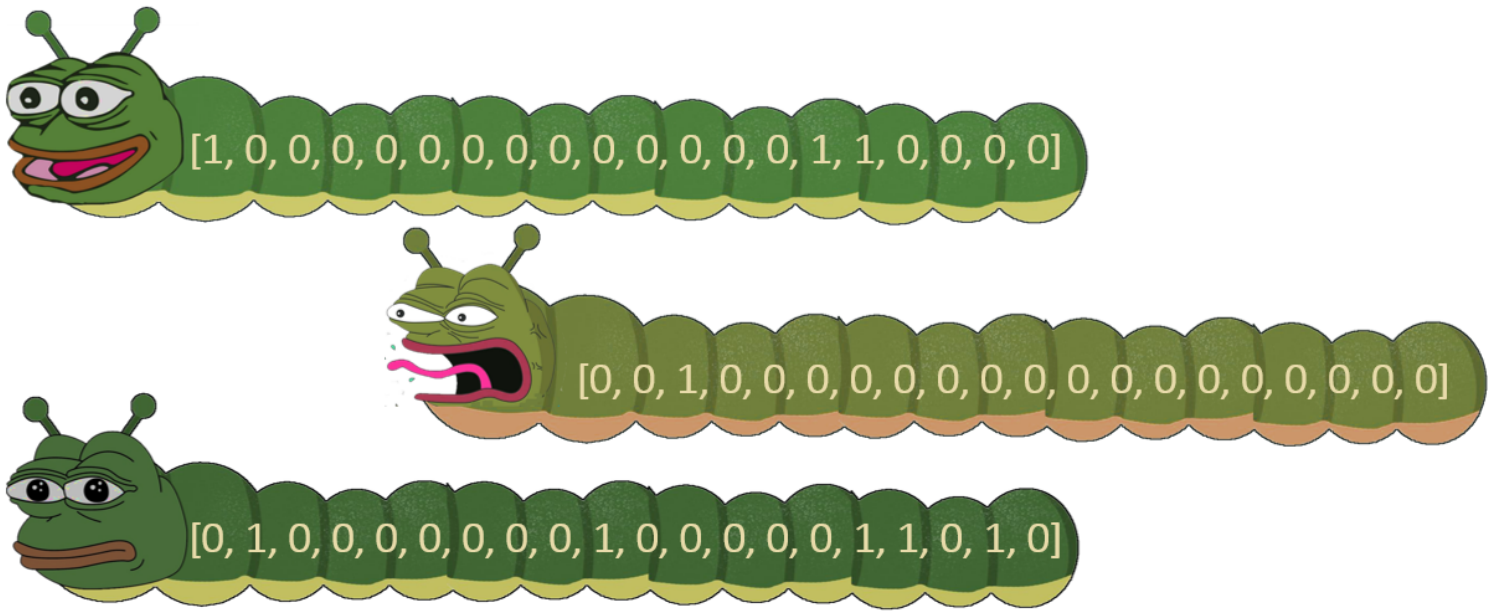
Algorytm genetyczny: osobnik, chromosom

Symulowane istoty żywe będziemy nazywać osobnikami lub (nieco umniejszając ich inteligencję, a koncentrując się na "wnętrzu") chromosomami. Przez chromosom rozumiemy ciąg znaków (string), bardzo często będzie to ciąg bitów. Każdy bit można nazwać genem. Chromosom koduje kandydata na rozwiązanie problemu.



Algorytm genetyczny: osobnik, chromosom

Symulowane istoty żywe będziemy nazywać osobnikami lub (nieco umniejszając ich inteligencję, a koncentrując się na "wnętrzu") chromosomami. Przez chromosom rozumiemy ciąg znaków (string), bardzo często będzie to ciąg bitów. Każdy bit można nazwać genem. Chromosom koduje kandydata na rozwiązanie problemu.



Populacja

Dobór wielkości populacjima spore znaczenie. Im większa, tym większa szansa na znalezienie rozwiązania, ale też spowalnia to algorytm genetyczny. Dobór wielkosci populacji można uzależnić od długości chromosomu np.

- Chromosom 10-liczbowy $\Rightarrow$ populacja kilkadziesiąt osobników
- Chromosom 30-liczbowy $\Rightarrow$ populacja 100-200 osobników
- Chromosom 100-liczbowy $\Rightarrow$ populacja kilkaset osobników

Zmieniamy oczywiscie dodatkowo liczbę rodziców do krzyżowania ustalając ich na około 30-50% z wszystkich osobników

Algorytm genetyczny: funkcja przystosowania

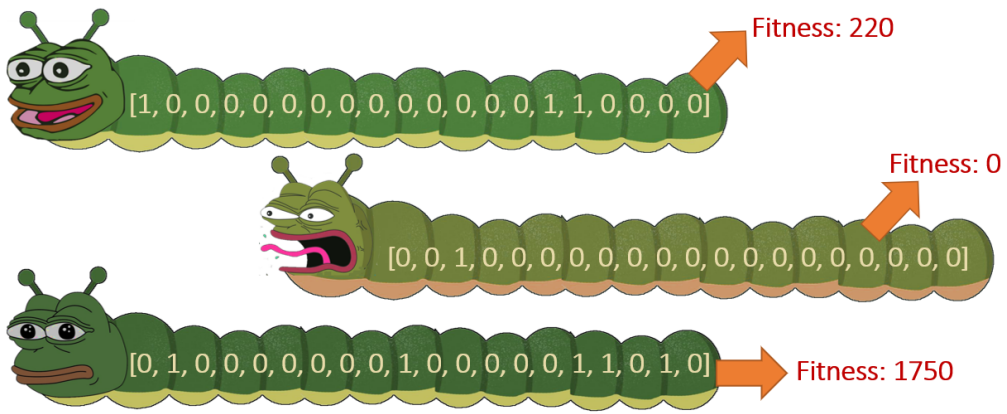
- W ewolucji osobniki słabsze (zwykle) giną i nie przekazują swoich genów (to brutalne!). Co znaczy "słaby", co "silny" w naszym algorytmie?

Algorytm genetyczny: funkcja przystosowania

- W ewolucji osobniki słabsze (zwykle) giną i nie przekazują swoich genów (to brutalne!). Co znaczy "słaby", co "silny" w naszym algorytmie?
- Decyduje o tym programista określając tzw. *funkcję przystosowania*, oceny (fitness function). Funkcja musi premiować osobniki, które przypominają rozwiązanie dla naszego problemu.

Algorytm genetyczny: funkcja przystosowania

- W ewolucji osobniki słabsze (zwykle) giną i nie przekazują swoich genów (to brutalne!). Co znaczy "słaby", co "silny" w naszym algorytmie?
- Decyduje o tym programista określając tzw. *funkcję przystosowania*, oceny (fitness function). Funkcja musi premiować osobniki, które przypominają rozwiązanie dla naszego problemu.
- Jest to najcięższa część tworzenia algorytmu. Czasem nie wiadomo jak oceniać!



Algorytm genetyczny: selekcja

Chromosomy oceniliśmy. Musimy dać szansę tym lepszym na przekazywanie swoich genów. Przeprowadzamy *selekcję*. Jest na to kilka strategii, m.in.:

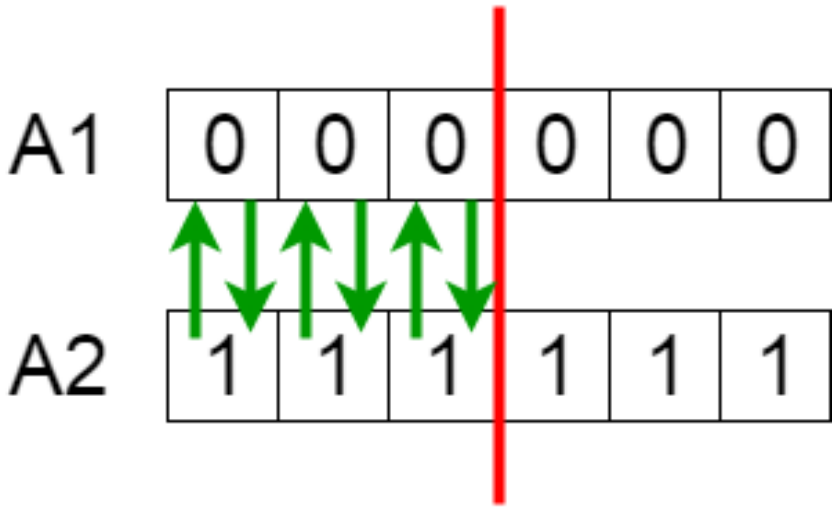
- *selekcja stabilnego stanu* (steady state selection, sss) - wybierz pewien procent najlepszych rodziców, wyprodukuj z nich potomków i zastąp najgorsze chromosomy potomkami. Reszta przeżywa do nastepnego pokolenia.
- *metoda ruletki*: losujemy osobniki z populacji, chromosomy o lepszym przystosowaniu mają zwiększoną szansę na wylosowanie
- *metoda turnieju*: losujemy grupę k osobników z populacji, osobniki walczą ze sobą (walka polega na wybraniu osobnika z najlepszym fitness), który zostaje rodzicem

Algorytm genetyczny: krzyżowanie

Wybrane w selekcji chromosomy krzyżują się dając życie nowym, być może lepszym, osobnikom. Zbliżamy się powoli do lepszego rozwiązania problemu!

Algorytm genetyczny: krzyżowanie

Wybrane w selekcji chromosomy krzyżują się dając życie nowym, być może lepszym, osobnikom. Zbliżamy się powoli do lepszego rozwiązania problemu!



Algorytm genetyczny: krzyżowanie

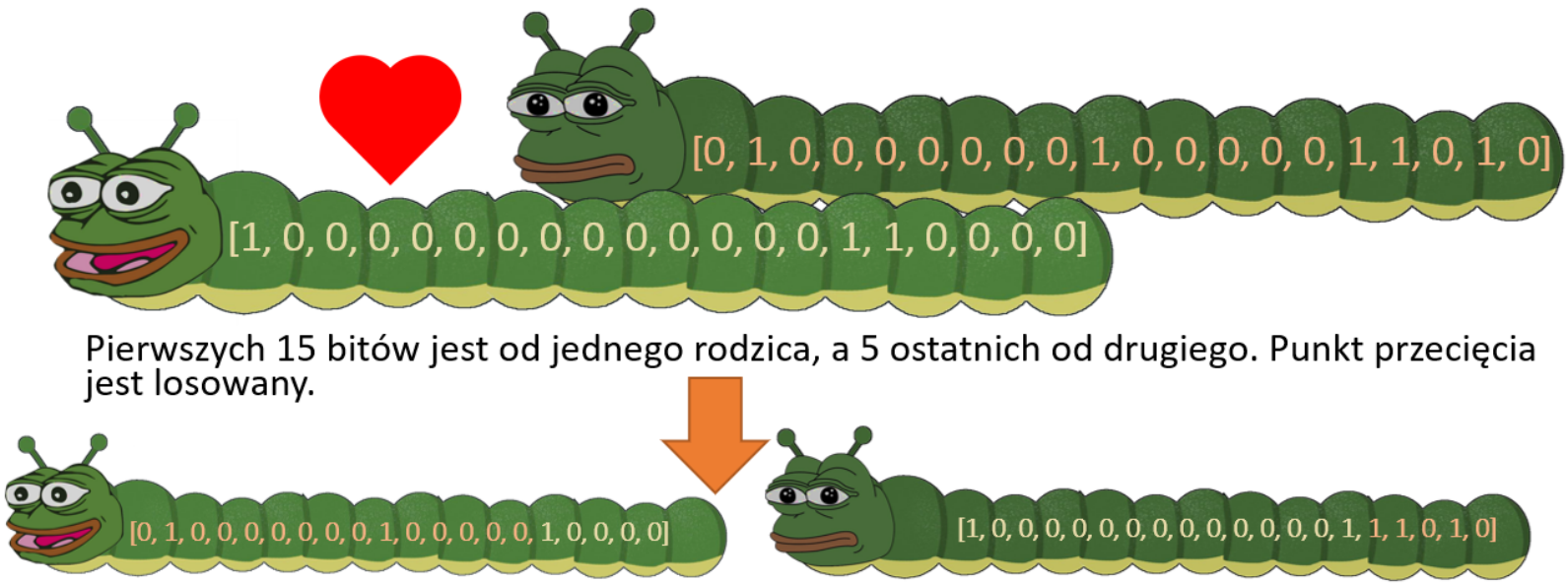
Wybrane w selekcji chromosomy krzyżują się dając życie nowym, być może lepszym, osobnikom. Zbliżamy się powoli do lepszego rozwiązania problemu!

A5	1	1	1	0	0	0

A6	0	0	0	1	1	1

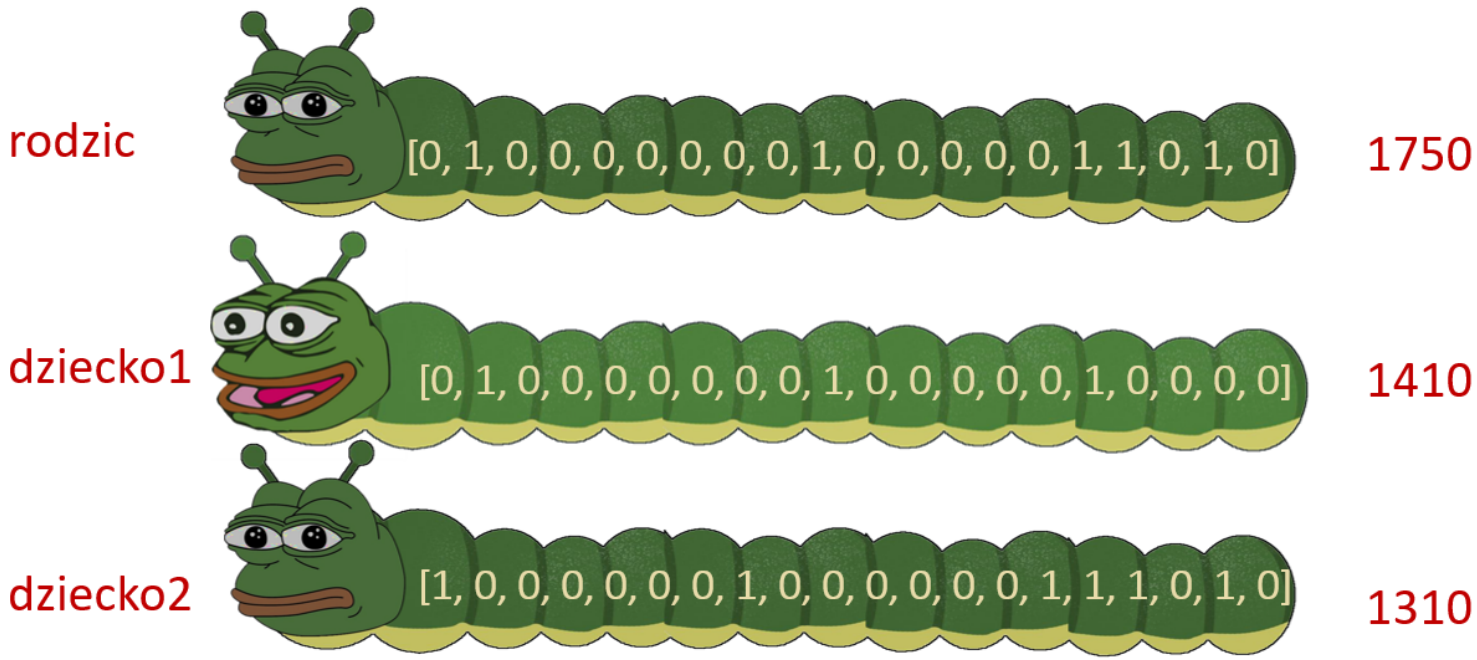
Algorytm genetyczny: krzyżowanie

Wybrane w selekcji chromosomy krzyżują się dając życie nowym, być może lepszym, osobnikom. Zbliżamy się powoli do lepszego rozwiązania problemu!



Algorytm genetyczny: krzyżowanie

Jest szansa, że z dwojga dobrych rodziców powstanie jeszcze lepsze dziecko. Ale nie zawsze tak jest...

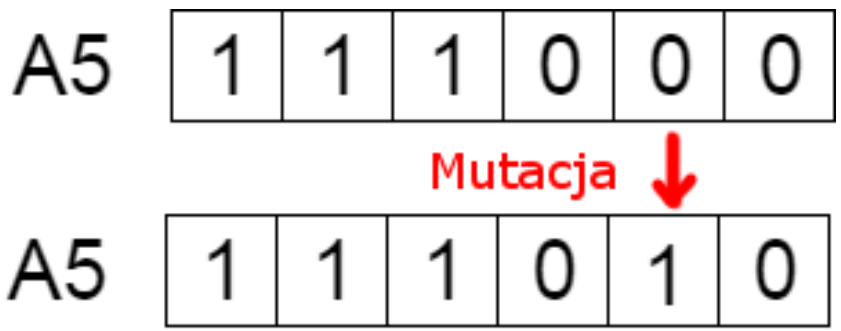


Algorytm genetyczny: krzyżowanie

Krzyżowanie też może podlegać modyfikacjom. Zamiana krzyżowania z przecięciem w jednym punkcie (single-point crossover) na taką w wielu punktach (multi-point crossover) może pomóc w generowaniu większej różnorodności rozwiązań. Jest to dobry pomysł, gdy rozwiązania mają tendencję do utykania w minimum (maksimum) lokalnym. Krzyżowanie można również zmodyfikować wg swoich upodobań, np. zwiększając szansę na przecięcie chrosomomów w wybranych pozycjach.

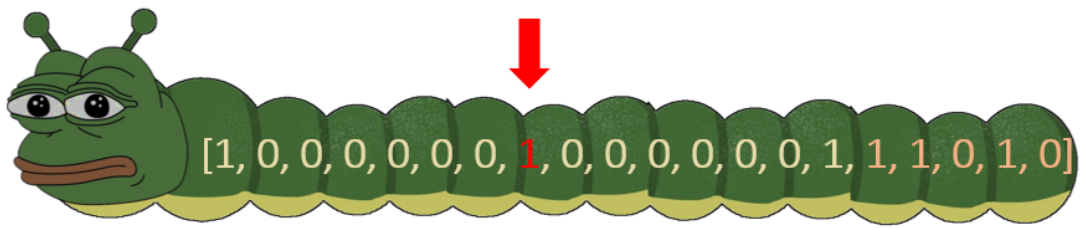
Algorytm genetyczny: mutacja

Bezpośrednio po krzyżowaniu można zastosować operator mutacji na dziecku. Z małym prawdopodobieństwem (około $\frac{1}{dlugosc(chrom)}$) zmieniamy bity na przeciwne. Pomaga to utrzymać genetyczną różnorodność.



Algorytm genetyczny: mutacja

Bezpośrednio po krzyżowaniu można zastosować operator mutacji na dziecku. Z małym prawdopodobieństwem (około $\frac{1}{dlugosc(chrom)}$) zmieniamy bity na przeciwne. Pomaga to utrzymać genetyczną różnorodność.



Algorytm genetyczny: mutacja

W algorytmie genetycznym warto uwzględnić mutację, ale na niewielkim poziomie. Warto, by w krótszych chromosomach (np. do 100 liczb), mutowana była 1 liczba. Z tego względu, należy dobrze ustawić prawodobodobieństwo mutacji:

- Chromosom 10-liczbowy $\Rightarrow$ 10% szansy na mutację $\Rightarrow$ 1 liczba się zmieni
- Chromosom 15-liczbowy $\Rightarrow$ 7% szansy na mutację $\Rightarrow$ 1 liczba się zmieni
- Chromosom 100-liczbowy $\Rightarrow$ 1% szansy na mutację $\Rightarrow$ 1 liczba się zmieni
- Chromosom 200-liczbowy $\Rightarrow$ 1% szansy na mutację $\Rightarrow$ 2 liczby się zmienią

Problem podziału i algorytm genetyczny

Problem podziału

Mamy zbiór liczb naturalnych $S = \{1, 4, 6, 11, 13, 20, 35\}$. Czy da się podzielić ten zbiór na dwie części S_1, S_2 tak, by obie części miały taką sumę liczb?

Problem podziału i algorytm genetyczny

Problem podziału

Mamy zbiór liczb naturalnych $S = \{1, 4, 6, 11, 13, 20, 35\}$. Czy da się podzielić ten zbiór na dwie części S_1, S_2 tak, by obie części miały taką sumę liczb?

Ustalanie struktury chromosomu:

Problem podziału i algorytm genetyczny

Problem podziału

Mamy zbiór liczb naturalnych $S = \{1, 4, 6, 11, 13, 20, 35\}$. Czy da się podzielić ten zbiór na dwie części S_1, S_2 tak, by obie części miały taką sumę liczb?

Ustalanie struktury chromosomu:

- Chromosomy mają 7 bitów.

Problem podziału i algorytm genetyczny

Problem podziału

Mamy zbiór liczb naturalnych $S = \{1, 4, 6, 11, 13, 20, 35\}$. Czy da się podzielić ten zbiór na dwie części S_1, S_2 tak, by obie części miały taką sumę liczb?

Ustalanie struktury chromosomu:

- Chromosomy mają 7 bitów.
- Każdy bit wskazuje na jedną liczbę w zbiorze (wg kolejności rosnącej liczb). Gdy jest 0, to liczbę bierzemy do S_1, a gdy 1 to do S_2.

Problem podziału i algorytm genetyczny

Problem podziału

Mamy zbiór liczb naturalnych $S = \{1, 4, 6, 11, 13, 20, 35\}$. Czy da się podzielić ten zbiór na dwie części S_1, S_2 tak, by obie części miały taką sumę liczb?

Ustalanie struktury chromosomu:

- Chromosomy mają 7 bitów.
- Każdy bit wskazuje na jedną liczbę w zbiorze (wg kolejności rosnącej liczb). Gdy jest 0, to liczbę bierzemy do S_1, a gdy 1 to do S_2.
- Jaki podział zbioru wskazuje chromosom $x = 0001110$?

Problem podziału i algorytm genetyczny

Problem podziału

Mamy zbiór liczb naturalnych $S = \{1, 4, 6, 11, 13, 20, 35\}$. Czy da się podzielić ten zbiór na dwie części S_1, S_2 tak, by obie części miały taką sumę liczb?

Ustalanie struktury chromosomu:

- Chromosomy mają 7 bitów.
- Każdy bit wskazuje na jedną liczbę w zbiorze (wg kolejności rosnącej liczb). Gdy jest 0, to liczbę bierzemy do S_1, a gdy 1 to do S_2.
- Jaki podział zbioru wskazuje chromosom $x = 0001110$?
- x dzieli zbiór na $S_1 = \{1, 4, 6, 35\}$, $S_2 = \{11, 13, 20\}$. Czy x jest "dobrym" chromosomem?

Problem podziału i algorytm genetyczny

Problem podziału

Mamy zbiór liczb naturalnych $S = \{1, 4, 6, 11, 13, 20, 35\}$. Czy da się podzielić ten zbiór na dwie części S_1, S_2 tak, by obie części miały taką sumę liczb?

Ustalanie funkcji przystosowania:

Problem podziału i algorytm genetyczny

Problem podziału

Mamy zbiór liczb naturalnych $S = \{1, 4, 6, 11, 13, 20, 35\}$. Czy da się podzielić ten zbiór na dwie części S_1, S_2 tak, by obie części miały taką sumę liczb?

Ustalanie funkcji przystosowania:

- Chromosomy są tym lepsze, im lepiej dzielą zbiór,

Problem podziału i algorytm genetyczny

Problem podziału

Mamy zbiór liczb naturalnych $S = \{1, 4, 6, 11, 13, 20, 35\}$. Czy da się podzielić ten zbiór na dwie części S_1, S_2 tak, by obie części miały taką sumę liczb?

Ustalanie funkcji przystosowania:

- Chromosomy są tym lepsze, im lepiej dzielą zbiór, tzn. sumy w S_1 i S_2 muszą być podobne.
- Funkcja przystosowania: weź chromosom (S_1 i S_2 są przez niego wskazywane) i policz sumy dla obu zbiorów Sum_1,Sum_2, a następnie zwróć

Problem podziału i algorytm genetyczny

Problem podziału

Mamy zbiór liczb naturalnych $S = \{1, 4, 6, 11, 13, 20, 35\}$. Czy da się podzielić ten zbiór na dwie części S_1, S_2 tak, by obie części miały taką sumę liczb?

Ustalanie funkcji przystosowania:

- Chromosomy są tym lepsze, im lepiej dzielą zbiór, tzn. sumy w S_1 i S_2 muszą być podobne.
- Funkcja przystosowania: weź chromosom (S_1 i S_2 są przez niego wskazywane) i policz sumy dla obu zbiorów Sum_1,Sum_2, a następnie zwróć $|Sum_1 - Sum_2|$.
- Jaki oceniony będzie chromosom z poprzednich slajdów: $x = 0001110$?

Problem podziału i algorytm genetyczny

Problem podziału

Mamy zbiór liczb naturalnych $S = \{1, 4, 6, 11, 13, 20, 35\}$. Czy da się podzielić ten zbiór na dwie części S_1, S_2 tak, by obie części miały taką sumę liczb?

Ustalanie funkcji przystosowania:

- Chromosomy są tym lepsze, im lepiej dzielą zbiór, tzn. sumy w S_1 i S_2 muszą być podobne.
- Funkcja przystosowania: weź chromosom (S_1 i S_2 są przez niego wskazywane) i policz sumy dla obu zbiorów Sum_1,Sum_2, a następnie zwróć $|Sum_1 - Sum_2|$.
- Jaki oceniony będzie chromosom z poprzednich slajdów: $x = 0001110$?
- Dobre są oceny niskie czy wysokie?

Problem podziału i algorytm genetyczny

Funkcja FITNESS ma dostęp do zmiennej globalnej S przechowującej tablicę z liczbami zbioru.

```
funkcja FITNESS(chromosom)
    Sum1 ← 0
    Sum2 ← 0
    wykonuj w pętli od i = 1 do  DŁUGOŚĆ(chromosom)
        jeśli chromsom[i] = 0
            Sum1 = Sum1 + S[i]
        jeśli chromsom[i] = 1
            Sum2 = Sum2 + S[i]
    zwróć WARTOŚĆ-BEZWZGLĘDNA(Sum1-Sum1)
```

Złodziej

Złodziej

Złodziej wchodzi do czyjegoś domu z chęcią zrabowania go. Ma do wyboru przedmioty o różnej wartości [zł] i wadze [kg]:

	przedmiot	wartosc	waga
1	zegar	100	7
2	obraz-pejzaż	300	7
3	obraz-portret	200	6
4	radio	40	2
5	laptop	500	5
6	lampka nocna	70	6
7	srebrne sztućce	100	1
8	porcelana	250	3
9	figura z brązu	300	10
10	skórzana torebka	280	3
11	odkurzacz	300	15

Niestety złodziej ma udźwig jedynie 25 kg. Jakie przedmioty powinien wziąć złodziej, by najwięcej na tym zarobił?

Złodziej

Dla danego problemu:

- Ustal strukturę chromosomu. Jaką długość mają chromosomy? Czy są binarne?
- Podaj przykład chromosomu i powiedz co oznaczają jego bity.
- Podaj funkcję przystosowania dla tego problemu.
- Oblicz dla wybranego chromosomu jego przystosowanie.
- Dobre są oceny niskie czy wysokie? Jaka jest maksymalna i minimalna ocena?

Idea rozwiązania

Problem plecakowy (złodziej) - wybieramy zestaw przedmiotów o najwyższej cenie, nie przekraczających limitu wagi. *Chromosom* = ciąg zer i jedynek, jedynki wskazują przedmioty do wzięcia. *Fitness* = wartość wziętych przedmiotów (im wyższa tym lepsza), nie może przekraczać wagi.

jak za duzo przedmiotów bierzemy - to wyceruje nam fitness -- nie mozemy unieść plecaka

Paczka pygad

Do rozwiązywania problemów będziemy wykorzystywać paczkę PyGAD.

Paczka pygad

- Paczka pygad oferuje rozwiązywanie problemów za pomocą algorytmu genetycznego.
- Algorytm genetyczny jest zaimplementowany. Wystarczy go odpowiednio skonfigurować (to jest temat tego praktycznego wykładu).
- Link `https://pygad.readthedocs.io/en/latest/`
- Paczka jest stale ulepszana. Sprawdź datę ostatniej aktualizacji ;)

Problem Podziału - przypomnienie

Problem podziału, instancja z 15 liczbami:

Problem podziału

Mamy zbiór liczb naturalnych $S = \{1, 2, 3, 6, 10, 17, 25, 29, 30, 41, 51, 60, 70, 79, 80\}$. Czy da się podzielić ten zbiór na dwie części S_1, S_2 tak, by obie części miały taką sumę liczb?

Zakodowanie problemu w Pythonie

- Problem należy tak zakodować, aby pobieranie wszystkich danych było dla nas wygodne.
- Możemy użyć list pythonowych, słowników, arrayów z `numpy`.
- W naszym przypadku wystarczy lista. Ma ona 15 liczb.

```
import pygad
import numpy

# nasz problem
S = [1, 2, 3, 6, 10, 17, 25, 29, 30, 41, 51, 60, 70, 79, 80]
```

Kodowanie chromosomów

- Przechodzimy do kodowania chromosomów.
- Przypomnienie: chromosom oznacza 0 i 1 do jakiego z dwóch zbiorów (S_1 czy S_2) ma trafić liczba.
- Zatem przestrzeń wszystkich możliwych genów to $\{0, 1\}$.
- Natomiast chromosom to tablica 15 bitów.

```
# definiujemy parametry chromosomu
# geny to liczby: 0 lub 1
gene_space = [0, 1]

#ile genow ma chromosom, długość S czyli 15
num_genes = len(S)
```

Kodowanie chromosomów

- Jeśli chcemy zwiększyć liczbę genów np. do czterech, można zmodyfikowaćprzestrzeń genów:

$$gene\_space = [0, 1, 2, 3]$$

- Jeśli chcemy liczby zmiennoprzecinkowe, musimy ustawić limit dolny i górny na przedział. Jesli chcemy wszystkie liczby z przedziału od 2 do 4 to realizujemy to tak:

$$gene\_space = \{'low' : 2, 'high' : 4\}$$

Tworzenie populacji

- Przechodzimy do tworzenia populacji.
- Do ustawienia jest właściwie tylko jeden parametr: wielkość populacji, czyli liczba chromosomów/rozwiązań.
- Wielkość ta, w przypadku naszych zajęć, raczej nie powinna przekraczać 100-200 osobników. Ustawimy małą, tylko 10.
- Dodatkowo, ustawiamy jaki jest limit pokoleń (w zależności od skomplikowania problemu). U nas: 30.

```
# ile chromosomów w populacji
# solutions per population
sol_per_pop = 10

# ile pokolen (generations)
num_generations = 30
```

Selekcja

- Przechodzimy do wyboru typu selekcji.
- Najprostsza to sss (steady state selection) - patrz poprzedni wykład. - bierze część ze wszystkich osobników (elita)
- Dodatkowo ustawiamy parametry: ilu rodziców ma brać udział w selekcji. Należy wybrać jakąś część populacji, np. 50%, u nas to 5 osobników.
- Oraz mówimy ilu najlepszych rodziców (tzw. elita) ma przeżyć do następnego pokolenia. Zwyczajowo daje się kilka procent. U nas, to aż 20% (2 osobniki), bo mamy małą populację.

Selekcja

```
# ile wylaniamy rodzicow do "rozmanazania" (okolo 50% populacji)
# ilu rodzicow zachowac (kilka procent)
num_parents_mating = 5
keep_parents = 2

#jaki typ selekcji rodzicow?
#sss = steady, rws=roulette, rank = rankingowa, tournament = turniejowa
parent_selection_type = "sss"
```

5 osobników z 10

Krzyżowanie

- Standardowym wyborem dla krzyżowania jest jednopunktowe: "single_point"
- Inne możliwości:
  - Dwupunktowe: "double_point" - rodzice wymieniają się środkiem.
  - Jednorodne: "uniform" - każdy gen dziecka wybierany jest losowo od jednego z rodziców.

```
#w ilu punktach robic krzyzowanie?
crossover_type = "single_point"
```

Mutacja

- Mutacja powoduje losowe zmiany w chromosomie. Jest kilka jej typów.
  - random: wylosuj geny i zmień je na inne. - jeśli jest za duża to będzie zbyt randomowa
  - swap: wylosuj parę genów i zamień je miejscami.
  - scramble: wylosuj przedział i wylosuj nowe miejsca dla genów z przedziału.
  - inversion: wylosuj predział i odwróć w nim kolejność genów.
- Procent genów do mutacji nalezy ustawić na conajmniej 1/długość chrosmosomu. U nas: $1/15 = 6{,}67\%$. Czyli 7% czy 8% będzie ok. - najoptymalniejsza wartość, nie mniej niż 1%

```
# mutacja ma dzialac na ilu procent genow?
# trzeba pamietac ile genow ma chromosom
mutation_type = "random"
mutation_percent_genes = 8
```

Funkcja fitness

Przyjrzyjmy się funkcji fitness: - ocena chromosomu

```
#definiujemy funkcję fitness
def fitness_func(solution, solution_idx):
    sum1 = numpy.sum(solution * S)
    solution_invert = 1 - solution
    sum2 = numpy.sum(solution_invert * S)
    fitness = -numpy.abs(sum1-sum2)
    #lub: fitness = 1.0 / (1.0 + numpy.abs(sum1-sum2))
    return fitness

fitness_function = fitness_func
```

chromosom, klucz chromosomu / id

- sumujemy to co pod jedynkami jest

- odwracamy 0 na 1 i 1 na 0

- sumujemy to co pod jedynkami jest

różnica sum
im lepsza ocena tym wyżej - dlatego minus dodajemy

- wyciągamy 0 i 1, sumujemy liczby pod S wskazane przez 1'ki

Funkcja fitness

Funkcja fitness sprytnie liczy sumy wykorzystując mnożenie po współrzędnych, sumowanie i inwersję bitów:

```
]#           S = [1, 2, 3, 6, 10, 17]
 #   solution = [0, 1, 0, 1, 0,  0]
 # S*solution = [0, 2, 0, 6, 0,  0]
 # S1 = sum(S*solution) = 8
 #   solution_invert = [1, 0, 1, 0, 1, 1]
 # S*solution_invert = [1, 0, 3, 0, 10, 17]
 # S2 = sum(S*solution_invert) = 31
```

inwersja 1 na 0 i 0 na 1
zamieniamy-

Funkcja fitness

Zauważ, że można wprowadzać różne skale ocen, ale premiowane są zawsze chromosomy z **wyższą** oceną. Pamiętając o tym, trzeba czasem zmodyfikować wzór na ocenianie. Przykłady dobrych funkcji:

- $-|Sum_1 - Sum_2|$ (skala od $-Sum_{max}$ do 0)
- $Sum1 + Sum2 - |Sum_1 - Sum_2|$ (skala od 0 do Sum_{max})
- $\frac{1}{1+|Sum_1-Sum_2|}$ (skala od ulamka bliskiego zeru do 1)

Uruchomienie algorytmu

Wszystkie zdefiniowane atrybuty wkładamy do inicjowanego obiektu GA. Następnie uruchamiamy ten algorytm komendą `run()`.

GA - genetic algorithm

```
#inicjacja algorytmu z powyzszymi parametrami wpisanymi w atrybuty
ga_instance = pygad.GA(gene_space=gene_space,
                       num_generations=num_generations,
                       num_parents_mating=num_parents_mating,
                       fitness_func=fitness_function,
                       sol_per_pop=sol_per_pop,
                       num_genes=num_genes,
                       parent_selection_type=parent_selection_type,
                       keep_parents=keep_parents,
                       crossover_type=crossover_type,
                       mutation_type=mutation_type,
                       mutation_percent_genes=mutation_percent_genes)

#uruchomienie algorytmu
ga_instance.run()
```

Wyniki

Po uruchomieniu w obiekcie `ga_instance` będą przechowywane informacje o najlepszym rozwiązaniu, które można pobrać i wyświetlić tak:

podsumowanie, zachowa best chromosom

```
solution, solution_fitness, solution_idx = ga_instance.best_solution()
print("Parameters of the best solution : {solution}".format(solution=solution))
print("Fitness value of the best solution = {solution_fitness}".format(solution_fi

#tutaj dodatkowo wyswietlamy sume wskazana przez jedynki
prediction = numpy.sum(S*solution)
print("Predicted output based on the best solution : {prediction}".format(predicti
```

- połowa sumy, ile siedzi pod jedynkami

```
Parameters of the best solution : [0. 0. 0. 0. 0. 1. 1. 0. 1. 1. 0. 1. 0. 1. 0.]
Fitness value of the best solution = -0.0
Predicted output based on the best solution : 252.0
```

- sumy są idealnie podzielone -- ich suma wyszła 0

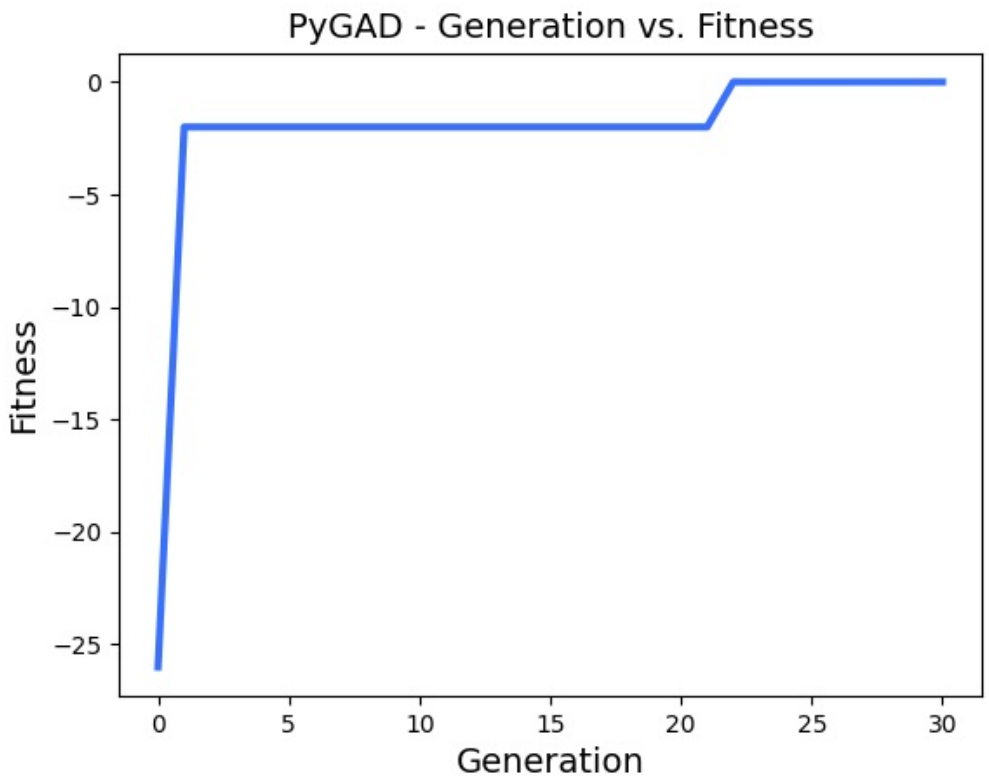
Wykres

Bardzo ciekawą opcją jest też wyświetlenie wykresu obrazującego przebieg ewolucji rozwiązań.

```
#wyswietlenie wykresu: jak zmieniala sie ocena na przestrzeni pokolen
ga_instance.plot_fitness()
```

Wykres

Na wykresie na osi X są kolejne pokolenia, a na osi Y fitness najlepszego rozwiązania w danym pokoleniu. Spójrz na wykres i opowiedz: w którym pokoleniu osiągnięto maksymalną ocenę?



tak przebiegała ewolucja - 30 generacji

Wykres

Co ciekawe (ale i spodziewane), za każdym uruchomieniem algorytmu dostaniemy inny wynik. W algorytmie genetycznym jest dużo losowości (populacja początkowa, selekcja, mutacja).