Laboratorium 2
Wykorzystanie sztucznej inteligencji w IT

Kacper Studenny
Robert Jaworski

Cel: Należy rozpisać 2 iteracje algorytmu genetycznego rozwiązujący problem plecakowy dla przedstawionych danych.
20 osobników, prawdopodobieństwo krzyżowania 0,8, prawdopodobieństwo mutacji 0,1.Należy wyraźnie opisać w jaki sposób algorytm poradzi sobie z osobnikiem stanowiącym przeładowany plecak (gdy dla danego osobnika waga plecaka przekracza wartość waga max).

Zestaw 1										
nr przedmiotu	1	2	3	4	5	6	7	8	9	10
waga	12	8	15	2	9	17	36	8	14	9
wartość	25	32	5	8	16	12	19	2	14	3

Opis najważniejszych części programu:

- Ruletka
  Program losuje liczbę w przedziale od **[0 , 1]**. Następnie sprawdzane jest który osobnik z populacji zawiera tę liczbę w swoim przedziale, gdzie rozmiar jego przedziału to: wartość **dopasowania** podzielone przez **sumę wszystkich dopasowań**, początek to 0 lub koniec przedziału poprzedniego osobnika, a koniec to powyższa liczba powiększona o rozmiar przedziału osobnika. Nasz **los** musi się zawierać w tym przedziale, aby osobnik był wybrany do krzyżowania.

```
def Ruletka(population, wynikDopasowania):
    start = 0.0
    suma = sum(wynikDopasowania)
    los = random.random()
    nowyStart = 0.0
    for x in range(len(wynikDopasowania)):
        nowyStart += wynikDopasowania[x] / suma
        if (start <= los < nowyStart):
            return population[x]
        start = nowyStart
    print("Awaria poza zakresem")
    return population[1]
```

- W przypadku wybrania **tej samej osoby** dwukrotnie powtarzamy losowanie drugiej osoby, aż będzie się różnić od pierwszej.
- Pomimo wylosowania dwójki potencjalnych rodziców wciąż, sprawdzane jest prawdopodobieństwo ich krzyżowania **80%.** W wyniku niepowodzenia, ze względu na brak informacji co zrobić teraz zdecydowaliśmy, że rodzice przechodzą do przyszłej populacji osobiście.

```
while rodzic2 == rodzic1:
    rodzic2 = Ruletka(populacja, wynikDopasowania)
if random.uniform(0, 1) <= wspolczynnikKrzyzowania:
    dziecko1, dziecko2 = krzyzowanie(rodzic1, rodzic2)
else:
    dziecko1 = rodzic1
    dziecko2 = rodzic2
```

- Mutacja
  Mutacja została ustawiona na **10%** dla każdego genu w genotypie. 10% w trakcie testów okazało się wysoką wartością przy założeniu, że w najgorszym wypadku ulec mutacji może każdy genom w genotypie.

```
def mutate(jednostka, szansa=0.1):
    for i in range(len(jednostka)):
        if random.random() < szansa:
            jednostka[i] = 1 - jednostka[i]
```

- Krzyżowanie (jednopunktowe)
  Krzyżowanie w projekcie przebiega w losowym jednym punkcie. W wyniku tworzone są dwa nowe genotypy, które zwracane są do przyszłej puli populacji

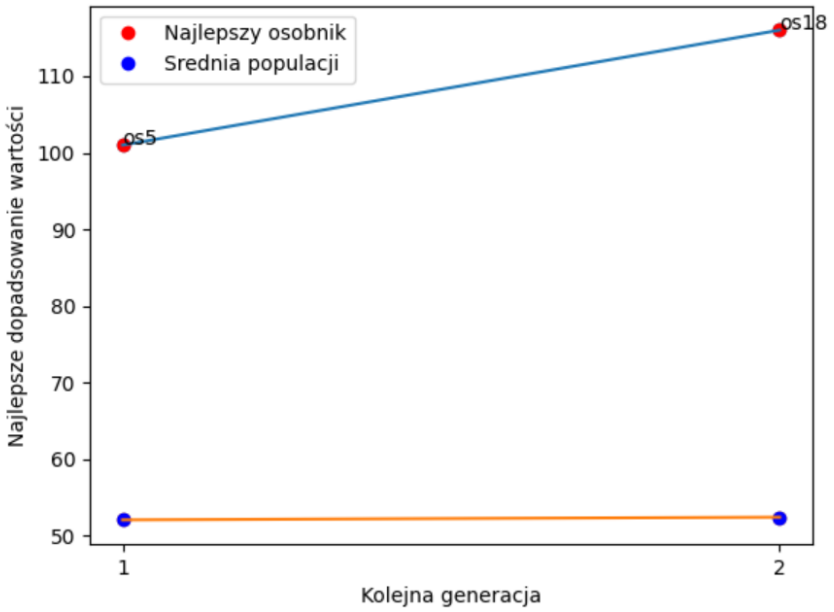
```
def krzyzowanie(rodzic1, rodzic2):
    punktKrzyzowania = random.randint(1, len(rodzic1) - 1)
    dziecko1 = rodzic1[:punktKrzyzowania] + rodzic2[punktKrzyzowania:]
    dziecko2 = rodzic2[:punktKrzyzowania] + rodzic1[punktKrzyzowania:]
    return dziecko1, dziecko2
```

- Przekroczenie Limitu plecaka
  W przypadku przekroczenia limitu wagi osobnik karany jest poprzez przypisanie wyniku dopasowania równemu **zero.** Co w praktyce wiąże się z eliminacją z procesu krzyżowania - śmiercią jego genotypu.

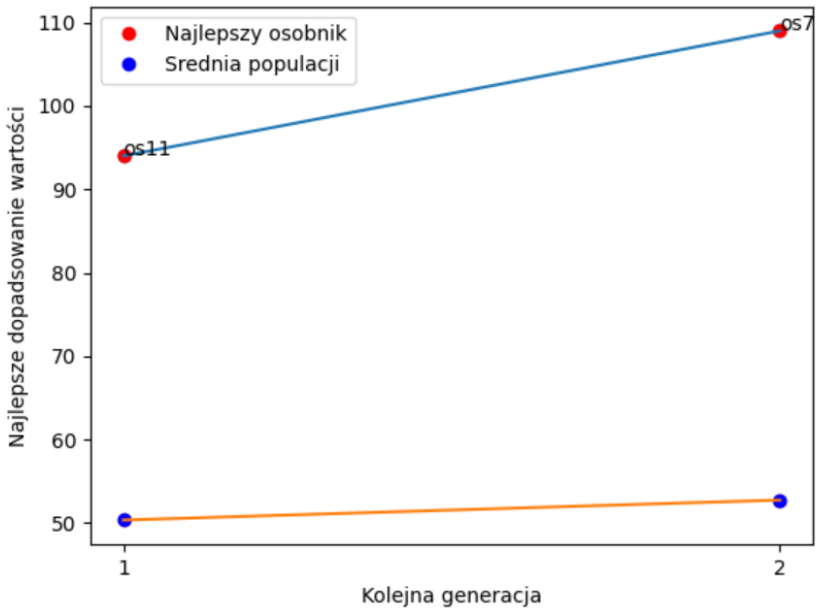
```
def OcenaDopasowan(jednostka):
    liczbakrzyzowek = 0
    wagaCalkowita = sum([jednostka[i] * wagaPrzedmiotu[i] for i in range(len(wagaPrzedmiotu))])
    wartoscCalkowita = sum([jednostka[i] * wartoscPrzedmiotu[i] for i in range(len(wartoscPrzedmiotu))])
    # print("+")
    return wartoscCalkowita if wagaCalkowita <= pojemnoscPlecaka else 0
```

Ilustracja wyników:

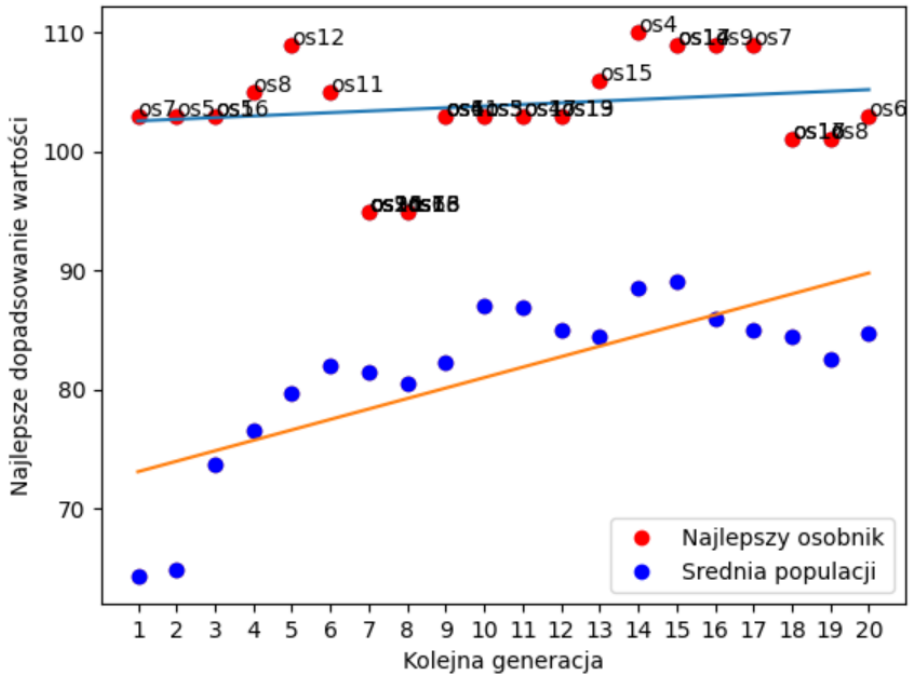
- Ilustracja wyników zgodnych z założeniami zadania:
  liczba generacji = 2
  osobników na generację = 20
  prawdopodobieństwo krzyżowania = 0,8
  prawdopodobieństwo mutacji = 0,1
  próba 1:



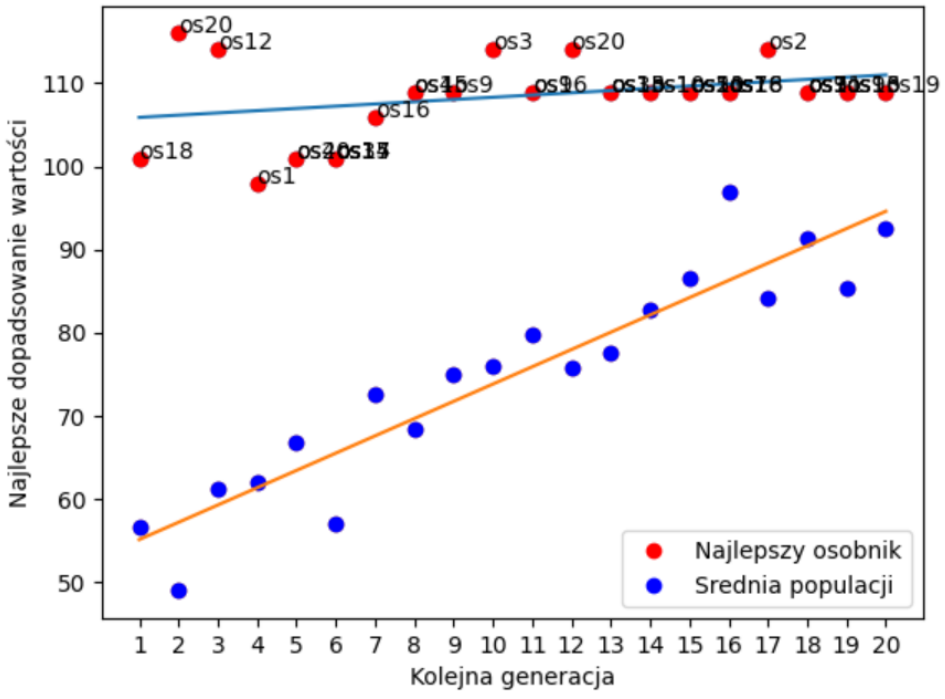
próba 2:



- Ze względu lepsze zobrazowanie wyników zmieniliśmy niektóre parametry; **zwiększono** ilość **generacji** do **20** oraz zmniejszono **możliwość mutacji** z 10% do **1%**. Oto wyniki :
próba 1:



próba 2:



Przykładowy wypis danych wyjściowych:

```
Wyniki dopasowania całej polpulacji  [0, 100, 17, 74, 0, 79, 45, 72, 87, 108, 0, 65, 97, 65, 56, 22, 42, 52, 0, 61]
osobnik # 1    [0, 0, 1, 1, 0, 1, 1, 1, 1, 1] Wynik dopasowania tego osobnika :  0 Osobnik  1 jest przeładowany i ponosi karę
osobnik # 2    [1, 1, 1, 1, 1, 1, 0, 1, 0, 0] Wynik dopasowania tego osobnika :  100
osobnik # 3    [0, 0, 1, 0, 0, 1, 0, 0, 0, 0] Wynik dopasowania tego osobnika :  17
osobnik # 4    [1, 1, 1, 0, 0, 1, 0, 0, 0, 0] Wynik dopasowania tego osobnika :  74
osobnik # 5    [1, 0, 1, 0, 0, 0, 1, 1, 1, 1] Wynik dopasowania tego osobnika :  0 Osobnik  5 jest przeładowany i ponosi karę
osobnik # 6    [1, 0, 0, 0, 1, 0, 1, 1, 1, 1] Wynik dopasowania tego osobnika :  79
osobnik # 7    [1, 0, 1, 0, 0, 1, 0, 0, 0, 1] Wynik dopasowania tego osobnika :  45
osobnik # 8    [1, 1, 0, 0, 0, 1, 0, 0, 0, 1] Wynik dopasowania tego osobnika :  72
osobnik # 9    [1, 1, 0, 1, 0, 0, 1, 0, 0, 1] Wynik dopasowania tego osobnika :  87
osobnik # 10   [1, 1, 0, 0, 1, 0, 1, 1, 1, 0] Wynik dopasowania tego osobnika :  108
osobnik # 11   [0, 1, 1, 1, 0, 1, 1, 0, 1, 0] Wynik dopasowania tego osobnika :  0 Osobnik  11 jest przeładowany i ponosi karę
osobnik # 12   [1, 1, 0, 1, 0, 0, 0, 0, 0, 0] Wynik dopasowania tego osobnika :  65
osobnik # 13   [1, 1, 0, 1, 1, 0, 0, 1, 1, 0] Wynik dopasowania tego osobnika :  97
osobnik # 14   [1, 0, 0, 0, 1, 0, 1, 1, 0, 1] Wynik dopasowania tego osobnika :  65
osobnik # 15   [0, 0, 1, 0, 1, 0, 1, 1, 1, 0] Wynik dopasowania tego osobnika :  56
osobnik # 16   [0, 0, 1, 0, 0, 1, 0, 1, 0, 1] Wynik dopasowania tego osobnika :  22
osobnik # 17   [0, 0, 1, 0, 1, 0, 1, 1, 0, 0] Wynik dopasowania tego osobnika :  42
osobnik # 18   [1, 0, 0, 1, 1, 0, 0, 0, 0, 1] Wynik dopasowania tego osobnika :  52
osobnik # 19   [1, 1, 1, 1, 0, 1, 1, 1, 1, 0] Wynik dopasowania tego osobnika :  0 Osobnik  19 jest przeładowany i ponosi karę
osobnik # 20   [0, 1, 0, 1, 0, 0, 1, 1, 0, 0] Wynik dopasowania tego osobnika :  61


Średnia wartości dopasowań populacji  52.1
Najlepszy Osobnik: [1, 1, 0, 0, 1, 1, 0, 1, 1, 0]
Najlepsze Dopasowanie Wartości: 101
Przy Wadze: 68
Ilość osobników w populacji ukaranych za przeładowanie: ) 4

**********************************************************************
```

Wnioski:

W trakcie tworzenia algorytmu w języku python zauważyliśmy kilka różnych zależności. Jedną z nich było zmniejszenie prawdopodobieństwa wystąpienia mutacji, które znacząco poprawiło rozwój populacji na przestrzeni generacji umożliwiając szybsze polepszanie populacji.

Inną sprawdzą jest oczywista poprawa wyników na przestrzeni zwiększonej ilości generacji np. 20, ale zauważyliśmy tutaj, że przy wysokim współczynniku mutacji (10%), generacje (i ich średnie populacyjne) zaczynały się pogarszać. Prawdopodobnie ze względu na to, że osobniki bliskie optymalnej wartości przy występowaniu mutacji zaczynały wykraczać poza limit wagi i były wykluczane z krzyżowania, przez co gorsze osobniki zajmowały ich miejsce i pogarszały stan populacji.

Większa mutacja utrudniała szybkość polepszania się populacji na przestrzeni generacji.

Na przestrzeni kilku testów testów zauważyliśmy też, że zwiększona ilość populacji (50 osób) wpłynęła na szybsze polepszanie się populacji.

Ostatecznie można powiedzieć, że mutacja dodaje pewien czynnik losowości, ale przy większych wartościach znacząco utrudnia rozwój populacji. Zwiększona ilość generacji i wielkość populacji wpłynęła pozytywnie w naszym odczuciu na polepszenie się wyników.