Laboratorium 2

Algorytm genetyczny

Problem podziału (przypomnienie z wykładu)

W problemie podziału (Parition problem) pytamy, czy da się podzielić zbiór liczb S na dwa zbiory S_1 i S_2 w taki sposób, że liczby w jednym i drugim podzbiorze sumują się do tej samej liczby.

Rozpatrzmy poniższy zbiór o 15 liczbach:
S = [1, 2, 3, 6, 10, 17, 25, 29, 30, 41, 51, 60, 70, 79, 80]

Zadanie 1

Przedstawiony powyżej problem podziału został rozwiązany za pomocą paczki pygad:
https://pygad.readthedocs.io/en/latest/

Uruchom i przeanalizuj poniższe rozwiązanie (załączone jest też jako osobny plik partition_ga.py)

```
import pygad
import numpy

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

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

#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

#ile chromsomów w populacji
#ile genow ma chromosom
sol_per_pop = 10
num_genes = len(S)

#ile wylaniamy rodzicow do "rozmanazania" (okolo 50% populacji)
#ile pokolen
#ilu rodzicow zachowac (kilka procent)
num_parents_mating = 5
num_generations = 30
keep_parents = 2

#jaki typ selekcji rodzicow?
```

```
#sss = steady, rws=roulette, rank = rankingowa, tournament = turniejowa
parent_selection_type = "sss"

#w il =u punktach robic krzyzowanie?
crossover_type = "single_point"

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

#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()

#podsumowanie: najlepsze znalezione rozwiazanie (chromosom+ocena)
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_fitness=solution_fitness))

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

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

Problem plecakowy (przypomnienie z wykładu)

W problemie plecakowym dana jest lista przedmiotów o wartościach i wagach. Chcemy do plecaka zabrać najcenniejsze rzeczy. Pytanie brzmi: jaki zestaw przedmiotów (o łącznej maksymalnej wadze *n* kg) ma największą wartość?

Rozpatrzmy instancję problemu:

n = **25** kg (limit wagi), lista przedmiotów:

	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

Zadanie 2

Rozwiąż powyższy problem plecakowy w Pythonie z użyciem paczki pygad. Możesz skorzystać z kodu z zadania 2, który trzeba rozsądnie zmodyfikować. Najważniejsze jest poprawne napisanie funkcji fitness.

Dopasuj parametry algorytmu do powyższego problemu (wielkość populacji, mutacja, itp.)

Jakie jest najlepsze rozwiązanie? Które przedmioty powinniśmy zabrać? Jaką mają wartość?

Zadanie 3

Zapoznaj się z możliwością dodania warunków zatrzymania dla algorytmu genetycznego w `pygad`:
https://pygad.readthedocs.io/en/latest/README_pygad_ReadTheDocs.html#stop-criteria

Dla zadania 1, zmodyfikuj kod programu tak, aby:

a) Program tworzył nowe pokolenia dopóki nie znajdzie rozwiązania z fitness równym 0. Gdy fitness 0 zostanie osiągnięte, to algorytm przerwie działanie.
b) Po zakończeniu program wypisze, ile pokoleń minęło, aż do znalezienia najlepszego rozwiązania.
c) Zmierz, ile czasu działał algorytm genetyczny. Przed i po poleceniu `ga_instance.run()` trzeba zmierzyć czas systemowy i podać różnicę.

   Przykład z Internetu:

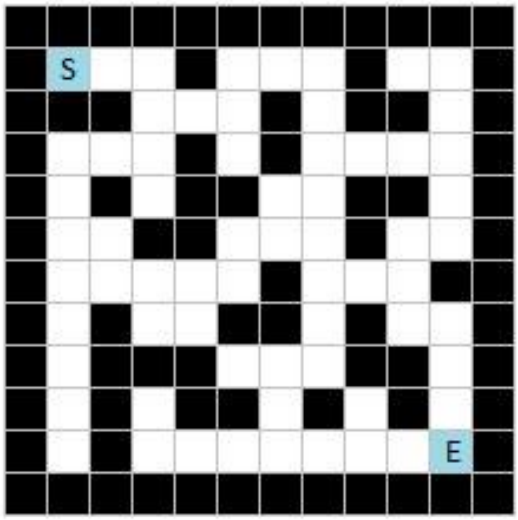
```
import time

start = time.time()
print("hello")
end = time.time()
print(end - start)
```

d) Zmierz czas 10 razy, zapisz wszystkie wyniki i podaj średnią z wyników. Ile średnio czasu zajmuje algorytmowi genetycznemu znalezienie rozwiązania?
e) Wykonaj powyższe polecenia a-d dla problemu plecakowego (złodziej) – zadanie 2. Możesz ustawić, że warunkiem przerwania będzie znalezienie przedmiotów o minimalnej wartości 1600.

Zadanie 4

Podobnie jak w zadaniu 1 i 2, stosując paczkę pygad, rozwiąż problem szukania drogi w labiryncie za pomocą algorytmu genetycznego. Możesz stosować pomysły przedstawione na wykładzie.

Weźmy następującą instancję problemu:
Czy istnieje droga o maksymalnie 30 krokach, ze startu do exitu, w labiryncie 12x12 przedstawionym na obrazku?



W rozwiązaniu uwzględnij następujące aspekty:

a) Jak zakodować labirynt? Macierz (lista list)?

b) Jak zakodować chromosom? Jakie ma znaki?

c) Na ile zwiększyć populację i odsetek rodziców do krzyżowania, by problem rozwiązywał się w dobrym czasie?

d) Jak wysoko ustawić szansę mutacji?

e) Którą funkcję fitness wybrać? A może przetestować obie z wykładu?
Uwaga, napisanie funkcji fitness przetwarzającej chromosomy jako sekwencje ruchów wymaga trochę umiejętności programistycznych. Należy w pętli przechodzić przez wszystkie ruchy w chromosomie i przesuwać naszą lokację w labiryncie na sąsiednie miejsce wskazywane przez ruch. Na końcu liczymy odległość.

f) Jeśli algorytm znajduje rozwiązanie, to wprowadź warunek stopu i zmierz średni czas z 10 uruchomień.

g) Dla chętnych: zrób własne ulepszenia funkcji fitness (np. punkty karne za uderzenia w ścianę lub powtarzane ruchy) i sprawdź czy algorytm działa lepiej.

Po zakończeniu rozwiązywania zadań (praca domowa)

Proszę założyć repozytorium **prywatne** o nazwie:

`[IO-2021/22-niestac] Imię Nazwisko`

Na jednym z portalów:

- https://github.com/
- https://bitbucket.org/

Następnie proszę mnie dodać do repozytorium. Namiary na mnie:

- https://github.com/gmadejsk
- https://bitbucket.org/grzesiekm/

W tym repozytorium proszę kolekcjonować zadania z zajęć. Będą one przeze mnie sprawdzane,

W terminie do kolejnych zajęć (7 listopada 2021) należy:

- Założyć w tym repozytorium **folder Lab02**
- W tym folderze umieścić **pliki z rozwiązaniami** zadań np. zad1.py, zad2.py, zad3ad.py, zad3e.py, zad4ver1.py, zad4ver2.py.
- W folderze umieścić też plik doc/pdf z krótkim **raportem** z wykonania zadań. W raporcie powinny się znaleźć:
    - Informacja o tym, które zadania się udało zrobić w całości, które częściowo. Czy były problemy?
    - Wyniki wszystkich zadań (skopiowany tekst z konsoli po uruchomieniu pliku)
    - Ewentualne wykresy, czy inne pliki graficzne, powstające w wyniku uruchomienia programu.