Laboratorium

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 wstępne

Przedstawiony powyżej problem podziału został rozwiązany za pomocą paczki pygad (co omówiono na wykładzie) 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 1

Rozwiąż powyższy problem plecakowy w Pythonie z użyciem paczki pygad. Możesz skorzystać z kodu z partition_ga.py, który trzeba rozsądnie zmodyfikować. Najważniejsze jest poprawne napisanie funkcji fitness – wskazówki były na wykładzie.
Dopasuj parametry algorytmu do powyższego problemu (wielkość populacji, mutacja, itp.)

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

Zadanie 2

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 1600. Gdy fitness 1600 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 poleceniach:

```
ga_instance = pygad.GA(…)
ga_instance.run()
```

trzeba zmierzyć czas systemowy i podać różnicę czasu.
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?

Rozwiązania zadań 1 i 2 najlepiej przedstaw w postaci notatnika (wstawki kodu + wyniki + komentarze).

*Zadanie dodatkowe, nieobowiązkowe

Zrób eksperyment z problemem podziału z większą liczbą elementów w S niż w zadaniu wstępnym. Kilka aspektów do przemyślenia:

- Jak wygenerować większe zbiory S? Czy zbiory te będą miały idealne rozwiązanie (fitness = 0)? Czy da się je wygenerować tak, aby miały idealny podział?
- Jak skalibrować parametry algorytmu genetycznego, by rozwiązywał on większe problemy?
- Czy udało się rozwiązać problem algorytmowi genetycznemu?