Podstawy Sztucznej Inteligencji

Laboratorium

Wstęp do Algorytmów Genetycznych.

Optymalizacja

Przygotował:

dr inż. Piotr Urbanek

Zadanie 1.

Puszka na napoje (funkcja jednej zmiennej)

Wykorzystując napisany na wcześniejszych zajęciach program wyznaczający maksimum funkcji jednej zmiennej za pomocą algorytmu genetycznego (rozwiązanie z wartościami rzeczywistymi w dziedzinie poszukiwania) znaleźć maksymalną objętość puszki do napojów wiedząc, że puszka jest walcem, którego powierzchnia A_0 jest dana i wynosi 4π.

Rozwiązanie analityczne:
Przy założeniu, że model geometryczny puszki jest walcem o promieniu r i wysokości h, poszukiwana jest maksymalna objętość $V(r,h)$ przy zadanej powierzchni A_0
Maksymalizacja: $V(r,h)=\pi r^2 h$ przy ograniczeniach:
$A(r,h)=2\pi r(r+h)=A_0$,
Dla $r>0$, $h>0$ $A(r,h) = 2\pi r(r+h) = A_0$
Czyli poszukiwane jest maksimum funkcji

$$V(h) = \pi h \left(-h + \sqrt{h^2 + \frac{A_0}{\pi}}\right)^2 \tag{1}$$

Zakładając, że $A_0=4\pi$

$$V(h) = \pi h \left(-h + \sqrt{h^2 + 4}\right)^2 \tag{2}$$

Założyć, że $0 \leq h \leq 9$

Aby sprawdzić poprawność otrzymanego wyniku proszę narysować zależność daną wzorem (2)

Zadanie 2.

FUNKCJA RASTRINGA (FUNKCJA DWU ZMIENNYCH)

Zadanie. Wyznaczyć za pomocą Algorytmu Genetycznego minimum i maksimum funkcji postaci:

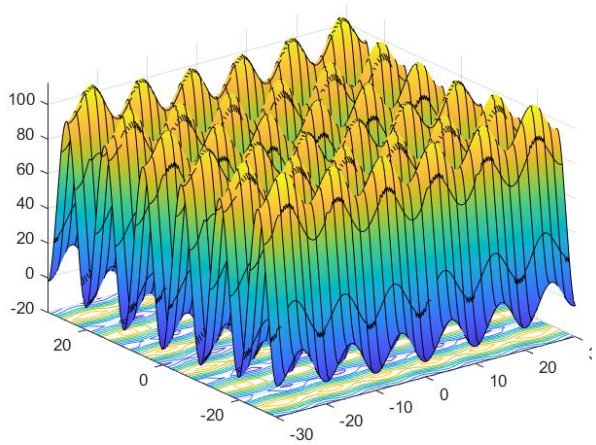
$$R = 20 + x^2 + y^2 - 10\big(cos(2\pi x) + cos(2\pi y)\big)$$

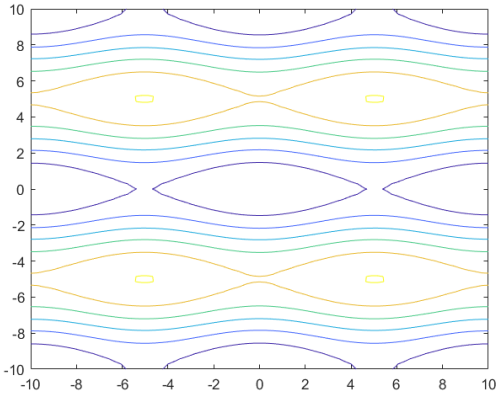
w przedziale zmienności $-30 \leq x \leq 30$ oraz $-30 \leq y \leq 30$.

Zbadać wpływ rodzaju prawdopodobieństwa i mutacji (minimum po dwa rodzaje) na zbieżność wyników.

Rozwiązanie:

Funkcja ta ma w analizowanym przedziale wiele minimów i maksimów, co można zobaczyć na wykresie przestrzennym oraz poziomicowym analizowanej funkcji:





Podpowiedź.

Proszę wykorzystać bibliotekę PyGad (instalacja: *pip3 install pygad*), dzięki której można rozwiązywać nawet skomplikowane zagadnienia wieloparametryczne.

Jako przykład niech posłuży rozwiązanie zagadnienia postaci:

y = f(w1:w6) = w1x1 + w2x2 + w3x3 + w4x4 + w5x5 + 6wx6

gdzie: (x1,x2,x3,x4,x5,x6)=(4, -2, 3.5, 5, -11, -4.7) a y=44

Więcej informacji można znaleźć na stronie: *https://pygad.readthedocs.io/en/latest/pygad.html*

Przykładowe rodzaje mutacji możliwe do zastosowania w bibliotece:

- random_mutation()
- swap_mutation()
- inversion_mutation()
- scramble_mutation()
- adaptive_mutation()

Przykładowe rodzaje krzyżowania możliwe do zastosowania w bibliotece:

- single_point_crossover()
- two_points_crossover()
- uniform_crossover()
- scattered_crossover()¶

Przykładowy program zamieszczony jest niżej:

```
# -*- coding: utf-8 -*-
"""
y = f(w1:w6) = w1x1 + w2x2 + w3x3 + w4x4 + w5x5 + w6x6
where (x1,x2,x3,x4,x5,x6)=(4,-2,3.5,5,-11,-4.7) and y=44
"""

import pygad
import numpy


function_inputs = [4,-2,3.5,5,-11,-4.7]
desired_output = 44

def fitness_func(ga_instance, solution, solution_idx):

    output = numpy.sum(solution*function_inputs)
    fitness = 1.0 / numpy.abs(output - desired_output)

    return fitness

fitness_function = fitness_func

num_generations = 500
num_parents_mating = 4

sol_per_pop = 8
num_genes = len(function_inputs)

init_range_low = -2
init_range_high = 5

parent_selection_type = "sss"
keep_parents = 1

#crossover_type = "single_point"
crossover_type = "uniform"

mutation_type = "random"
mutation_percent_genes = 0.01
ga_instance = pygad.GA(num_generations=num_generations,
                       num_parents_mating=num_parents_mating,
```

```
                fitness_func=fitness_function,
                sol_per_pop=sol_per_pop,
                num_genes=num_genes,
                init_range_low=init_range_low,
                init_range_high=init_range_high,
                parent_selection_type=parent_selection_type,
                keep_parents=keep_parents,
                crossover_type=crossover_type,
                mutation_type=mutation_type,
                mutation_percent_genes=mutation_percent_genes)


ga_instance.run()

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))

prediction = numpy.sum(numpy.array(function_inputs)*solution)
print("Predicted output based on the best solution : {prediction}".format(prediction=prediction))



ga_instance.plot_fitness()

# Returning the details of the best solution.
solution, solution_fitness, solution_idx =
ga_instance.best_solution(ga_instance.last_generation_fitness)
print(f"Parameters of the best solution : {solution}")
print(f"Fitness value of the best solution = {solution_fitness}")
print(f"Index of the best solution : {solution_idx}")

prediction = numpy.sum(numpy.array(function_inputs)*solution)
print(f"Predicted output based on the best solution : {prediction}")

if ga_instance.best_solution_generation != -1:
    print(f"Best fitness value reached after {ga_instance.best_solution_generation} generations.")

# Saving the GA instance.
filename = 'genetic' # The filename to which the instance is saved. The name is without extension.
ga_instance.save(filename=filename)

# Loading the saved GA instance.
loaded_ga_instance = pygad.load(filename=filename)
loaded_ga_instance.plot_fitness()
```