Tyler Vincent

Genetic Metaheuristic Algorithm

AUGUST 2019 – MAY 2020

The Objective:

Develop a multi-objective optimization algorithm in Python to schedule 50+ military, earth-to-space antennae positions that verifies & records 100+ satellites and deep space objects positions in Geosynchronous Earth orbit with radar every 24 hours for the Aerospace Corporation.

The Solution:

Using my operations research and programming knowledge, I designed a Genetic Algorithm, which is a metaheuristic, inspired by the process of natural selection that is used to generate useful solutions to optimization and search problems. Genetic algorithms are frequently used to find optimal or near-optimal solutions to difficult problems which otherwise would take a lifetime to solve. Here is the code itself, in a simplified, releasable version .py file. Run it in a compiler of your choice. It’s also below.

Here's a general outline of my approach:

We start with a population of random solutions. Each solution is an assignment of satellites to antennas and has a certain "fitness" - the total movement needed to cover all the satellites.

* We implement genetic operators to create new solutions: "crossover" (combining parts of two solutions) and "mutation" (randomly changing part of a solution).

* We select the best solutions from the current and the new generation.

* We repeat the process until we reach a satisfactory solution or a stop condition.

The output will be the fitness of the best assignment found, and the schedule for each antenna.

Limitations:

* The result is random due to the nature of the genetic algorithm, so if you run the code multiple times, you'll likely get different outputs. However, the fitness of the best assignment should be roughly similar between runs.
* It is a heuristic algorithm, which means it doesn't guarantee to find the optimal solution. It may get stuck in a local optimum, especially if the mutation rate is too low.
* The performance of the algorithm heavily depends on the parameters such as the population size and the number of generations. Finding good values for these parameters can be challenging.
* This algorithm assumes that the movement of an antenna is the same regardless of the direction or the distance. In practice, this may not be true. For example, moving a large distance in a short time may not be feasible.
* The algorithm also assumes that it's possible to instantly switch from one satellite to another, which might not be the case in reality due to physical constraints.

▼▽▼ See the code below ▼▽▼

```python
import numpy as np
from operator import attrgetter

class Assignment:
    def __init__(self, antennas, schedule):
        self.antennas = antennas
        self.schedule = schedule
        self.fitness = self.calculate_fitness()

    def calculate_fitness(self):
        # Compute total distance travelled by all antennas
        return sum(np.sum(np.abs(np.diff(self.schedule[i]))) for i in
range(len(self.schedule)))

def generate_random_assignment(antennas, num_satellites):
    # Create a random schedule for each antenna
    schedule = [np.random.choice(num_satellites, size=6, replace=False) for _ in
range(antennas)]
    return Assignment(antennas, schedule)

def crossover(a1, a2):
    # Choose a random crossover point
    point = np.random.randint(len(a1.schedule))
    # Create a new schedule by combining parts of two parents' schedules
    new_schedule = a1.schedule[:point] + a2.schedule[point:]
    return Assignment(a1.antennas, new_schedule)

def mutate(a, num_satellites):
    # Choose a random antenna and a random new satellite
    antenna = np.random.randint(a.antennas)
    new_satellite = np.random.randint(num_satellites)
    # Replace one of the satellites in the antenna's schedule
    a.schedule[antenna][np.random.randint(6)] = new_satellite
    a.fitness = a.calculate_fitness()

def genetic_algorithm(antennas, num_satellites, population_size=100,
generations=200):
    # Generate initial population
    population = [generate_random_assignment(antennas, num_satellites) for _ in
range(population_size)]

    for _ in range(generations):
        # Generate new population
        new_population = [crossover(np.random.choice(population),
np.random.choice(population)) for _ in range(population_size)]
        # Apply mutation
        for a in new_population:
            if np.random.rand() < 0.1:
                mutate(a, num_satellites)
        # Select the best individuals from the current and the new population
        population = sorted(population + new_population,
key=attrgetter('fitness'))[:population_size]

    # Return the best assignment found
    return min(population, key=attrgetter('fitness'))

# Number of antennas and satellites
num_antennas = 5
num_satellites = 10

# Generate satellites' positions
satellites = [(i, np.random.rand(6, 2)) for i in range(num_satellites)]

best_assignment = genetic_algorithm(num_antennas, num_satellites)

print("Best assignment fitness: ", best_assignment.fitness)
for i in range(best_assignment.antennas):
    print(f"Antenna {i} schedule: {best_assignment.schedule[i]}")


print("\nThe best assignment found by the algorithm minimizes the total movement of
all antennas.")
print(f"The total movement of the antennas in the best assignment is:
{best_assignment.fitness}\n")

print("Here's how the antennas should be scheduled to achieve this minimum
movement:")

for i in range(best_assignment.antennas):
    schedule = best_assignment.schedule[i]
    print(f"\nAntenna {i}:")

    for j in range(6):
        satellite_id = schedule[j]
        azimuth, elevation = satellites[satellite_id][1][j]
        print(f"At the {j*4}-th hour, point to satellite {satellite_id} at azimuth
{azimuth:.2f} and elevation {elevation:.2f}.")
```