Delivery Drone Route Optimization Using Genetic Algorithm

- ## What is the Genetic Algorithm?

  A GA is an optimization technique inspired by the principles of natural selection, and genetics.

  ### Main Idea:

  - **Population Initialization:** start with many random routes.
  - **Fitness Evaluation:** measure how good each route.
  - **Selection:** pick better routes to reproduce.
  - **Crossover:** combine two routes to produce new ones.
  - **Mutation:** randomly alter parts of a route.
  - **Generation Loop:** repeat to improve solutions over time.

- ## Why Use Genetic in the Drone Delivery Problem?

  - Efficiently finds near-optimal routes among many possible paths.
  - Handles complex and nonlinear constraints, such as distance limits or multiple delivery points.
  - Mutation and crossover allow the algorithm to explore diverse solutions and avoid getting stuck in poor routes.

- ## Python Implementation

```python
import random
# Example list of delivery points
delivery_points = ['A', 'B', 'C', 'D', 'E', 'F']
# Example distance matrix (dictionary of distances)
distance = {
    ('A','B'): 2, ('A','C'): 4, ('A','D'): 7, ('A','E'): 3, ('A','F'): 5,
    ('B','C'): 1, ('B','D'): 5, ('B','E'): 6, ('B','F'): 4,
    ('C','D'): 8, ('C','E'): 2, ('C','F'): 3,
    ('D','E'): 4, ('D','F'): 6,
    ('E','F'): 2,
}
# Make symmetric, means distance of A,b = distance of b,a
for (u,v),d in list(distance.items()):
    distance[(v,u)] = d

# Fitness: total route distance (lower is better)
def compute_distance(route):
    total = 0
    for i in range(len(route) - 1):
        total += distance[(route[i], route[i+1])]   ## add distance between current and next point
    return total

# Generate initial random population
def create_population(size):
    return [random.sample(delivery_points, len(delivery_points)) for _ in range(size)]
```

```python
# Selection (tournament)
def select(pop, fitnesses):
    selected = []
    for _ in range(len(pop)):
        i, j = random.sample(range(len(pop)), 2)
        selected.append(pop[i] if fitnesses[i] < fitnesses[j] else pop[j])
    return selected

# Crossover (ordered)
def crossover(parent1, parent2):
    size = len(parent1)
    start, end = sorted(random.sample(range(size), 2))
    child = parent1[start:end]
    for gene in parent2:
        if gene not in child:
            child.append(gene)
    return child

# Mutation: swap two points
def mutate(route, rate=0.1):
    r = route.copy()
    for i in range(len(r)):
        if random.random() < rate:
            j = random.randrange(len(r))
            r[i], r[j] = r[j], r[i]
    return r
```

```python
# GA main loop
def genetic_algorithm(generations=100, pop_size=50):
    pop = create_population(pop_size)
    for gen in range(generations):
        fitnesses = [compute_distance(r) for r in pop]
        parents = select(pop, fitnesses)
        next_pop = []
        for i in range(0, len(parents), 2):
            p1, p2 = parents[i], parents[(i+1)%len(parents)]
            child1 = crossover(p1, p2)
            child2 = crossover(p2, p1)
            next_pop += [mutate(child1), mutate(child2)]
        pop = next_pop

    best = min(pop, key=compute_distance)
    return best, compute_distance(best)

best_route, best_dist = genetic_algorithm()
print("Best Route:", best_route)
print("Distance:", best_dist)
```

- **Output**

```
Best Route: ['A', 'B', 'C', 'F', 'E', 'D']
Distance: 12
```

  - GA efficiently reached all delivery points efficiently.
  - The path is optimal.
  - GA minimizes total route distance.

- **Advantages of Using a GA for Drone Delivery**
  - Produces shorter routes than simple search methods.
  - Explores multiple solutions simultaneously.
  - Handles large and complex search spaces.
  - Flexible for multi-objective problems.

- **Limitations**
    - GA doesn't guarantee a mathematically optimal solution.
    - GA uses more resources (time and memory) than simpler algorithms like DFS.
    - Parameter selection (population size, mutation rate) is critical, which means poor choices may reduce performance or prevent convergence.

- **Genetic Algorithm (GA) Evaluation Metrics**
    1. **Execution Time**
       GA may take longer than simple search algorithms because it evaluates many solutions over multiple generations.
    2. **Memory Usage**
       Stores the population and fitness values; memory grows with population size and number of delivery points.
    3. **Success Rate**
       GA reliable finds a feasible route that visits all delivery points. Mutation and crossover help explore diverse solutions and avoid getting stuck.
    4. **Solution Optimality**
       GA produces near optimal or optimal routes, minimizing total distance or cost.
    5. **Scalability**
       By adjusting population size and number of generations, it can handle more delivery points without exploring unnecessary paths.