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.**
  It is particularly effective for **complex search spaces** where exhaustive search is impractical.

- ## Why Use Genetic in the Drone Delivery Problem?
  - Efficiently finds **near-optimal routes** among a very large number of possible paths.
  - Handles **complex and nonlinear constraints**, such as distance limits or multiple delivery points.
  - Avoid local optima using **mutation and crossover.**
  - Scales better than classical search algorithms for large graphs.

- ## GA Design Details
  1. ### Solution Representation
     Each solution is represented as a permutation of delivery points, ensuring:
     - Each location is visited exactly once.
     - No duplicate nodes exist in a route.
  2. ### Fitness Function
     It evaluates the **total distance of a route**.
     The objective is to **minimize total travel distance,** meaning shorter routes have higher fitness.
  3. ### Population Initialization

The algorithm starts with a **random population of routes,** ensuring diversity in the initial search space.

4. **Selection Method**

   Tournament Selection is used to choose parent routes. **These balances:**
   - Exploitation of good solutions.
   - Exploration of new areas in the search space.

5. **Crossover Operator**

   Order Crossover is applied to combine two parent routes while preserving the relative order of nodes.

6. **Mutation Operator**

   Swap Mutation randomly exchanges two delivery points in a route to **maintain population diversity** and **prevent premature convergence** by enabling exploration of new regions of the search space.

7. **Elitism**

   The best solution from each generation is **carried forward unchanged** to ensure solution quality doesn't degrade.

8. **Termination Criteria**

   The algorithm stops when:
   - A fixed number of generations is reached.
   - No significant improvement is observed over multiple generations.

- **Python Implementation**

```python
import random
import time

# ------------------------
# 1. Delivery points & distances
# ------------------------
delivery_points = ['A', 'B', 'C', 'D', 'E', 'F']

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
for (u,v),d in list(distance.items()):
    distance[(v,u)] = d
```

```python
# ------------------------
# 2. Fitness: total route distance (lower is better)
# Includes returning to start
# ------------------------
def compute_distance(route):
    total = 0
    for i in range(len(route) - 1):
        total += distance[(route[i], route[i+1])]
    # Return to start
    total += distance[(route[-1], route[0])]
    return total

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

# ------------------------
# 4. Selection - Tournament
# ------------------------
def select(pop, fitnesses, k=3):
    selected = []
    for _ in range(len(pop)):
        candidates = random.sample(range(len(pop)), k)
        best = min(candidates, key=lambda i: fitnesses[i])
        selected.append(pop[best])
    return selected
```

```python
# 5. Crossover - Order Crossover
# ------------------------
def crossover(parent1, parent2):
    size = len(parent1)
    start, end = sorted(random.sample(range(size), 2))
    child = [-1]*size
    child[start:end] = parent1[start:end]

    pointer = end
    for gene in parent2:
        if gene not in child:
            if pointer >= size:
                pointer = 0
            child[pointer] = gene
            pointer += 1
    return child

# ------------------------
# 6. Mutation - Swap Mutation
# ------------------------
def mutate(route, mutation_rate=0.1):
    r = route.copy()
    for i in range(len(r)):
        if random.random() < mutation_rate:
            j = random.randrange(len(r))
            r[i], r[j] = r[j], r[i]
    return r
```

```python
# 7. Genetic Algorithm with Elitism
# ------------------------
def genetic_algorithm(pop_size=50, generations=100, mutation_rate=0.1):
    start_time = time.time()
    population = create_population(pop_size)
    best_route = min(population, key=lambda r: compute_distance(r))
    for gen in range(generations):
        fitnesses = [compute_distance(r) for r in population]
        parents = select(population, fitnesses)
        next_pop = []

        # Elitism: carry forward best solution
        next_pop.append(best_route)
        # Create children
        for i in range(0, len(parents), 2):
            p1 = parents[i]
            p2 = parents[(i+1)%len(parents)]
            child1 = mutate(crossover(p1, p2), mutation_rate)
            child2 = mutate(crossover(p2, p1), mutation_rate)
            next_pop += [child1, child2]

        # Keep population size fixed
        population = next_pop[:pop_size]
        # Update best route
        current_best = min(population, key=lambda r: compute_distance(r))
        if compute_distance(current_best) < compute_distance(best_route):
            best_route = current_best
    exec_time = time.time() - start_time
    return best_route, compute_distance(best_route), exec_time
```

```python
# ------------------------
# 8. Run GA
# ------------------------
best_route, best_distance, execution_time = genetic_algorithm()
print(f"Best Route: {' → '.join(best_route)} → {best_route[0]}")
# print("Best Route:", best_route)
print("Total Distance:", best_distance)
print("Execution Time:", round(execution_time,4), "seconds")
```

```
Best Route: F → D → E → A → B → C → F
Total Distance: 19
Execution Time: 0.052 seconds
```

- ## Output


```python
# ------------------------
# 8. Run GA
# ------------------------
best_route, best_distance, execution_time = genetic_algorithm()
print(f"Best Route: {' → '.join(best_route)} → {best_route[0]}")
# print("Best Route:", best_route)
print("Total Distance:", best_distance)
print("Execution Time:", round(execution_time,4), "seconds")
```

```
Best Route: F → D → E → A → B → C → F
Total Distance: 19
Execution Time: 0.052 seconds
```


- GA finds a **feasible, near-optimal route** visiting all delivery points.
- Minimizes total travel distance efficiently.
- Produces **practical paths** ready for implementation.

- ## 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**

   It depends on **population size** and **number of delivery points.**

3. **Success Rate**

   GA reliably finds feasible routes by visiting all delivery points.

4. **Solution Optimality**

   GA produces near optimal or optimal routes, especially for larger complex graphs.

5. **Scalability**

   By adjusting population size and number of generations, it can handle more delivery points without exploring unnecessary paths.