Solving Variants of TSP Using Genetic Algorithms

RAO Fenggui, 24037513R

I. INTRODUCTION

Traveling Salesman Problem (TSP) is an NP-hard combinational optimization problem that seeks the shortest possible route visiting each city exactly once and returning to the origin city. Genetic Algorithms (GA) provide an effective heuristic optimization method inspired by natural selection mechanisms, widely applied to complex optimization problems such as TSP.

This project explores GA-based approaches to various TSP variants, including classical single-objective TSP, large-scale TSP, multi-objective TSP, multi-tasking TSP, and bi-level depot selection TSP.

II. METHODOLOGY

A. Classical TSP (Single-objective GA)

In this task, we solved the classical 100-city Traveling Salesman Problem using a standard Genetic Algorithm based on the Geatpy toolkit. The framework involves the following key components:

- **Representation**: Each solution is encoded using **permutation encoding**, where a chromosome is an ordered sequence of integers from 0 to 99, representing the visiting order of cities.
- **Objective Function**: The fitness of an individual is defined as the total Euclidean distance of the round trip, including the return from the last city to the first.
- **Crossover Operator**: We adopted **Ordered Crossover (OX)** to preserve valid permutations and maintain subsequences.
- **Mutation Operator**: **Inversion Mutation** was used to reverse a randomly selected subsegment, improving local path optimization.
- **Selection Mechanism**: **Roulette Wheel Selection** is applied, favoring individuals with shorter tour lengths (i.e., higher fitness).
- **Evolutionary Framework**: The algorithm is built on the Simple Evolutionary Genetic Algorithm (SEGA) template from Geatpy, using a population size of 100 and 500 generations.

B. Large-scale TSP (Clustering + GA)

This task addresses scalability by increasing the number of cities from 100 to 200. The solution framework consists of a decomposition-based divide-and-conquer approach:

- **Data Augmentation**: We doubled the number of customers by horizontally shifting the original coordinates, creating 200 nodes.
- **Clustering**: **K-means clustering** is applied to partition the cities into k regional clusters (default $k = 4$).
- **Local Optimization**: Each cluster is independently solved as a smaller TSP using the same classical GA.
- **Path Integration**: After solving all clusters, local tours are concatenated in the order of cluster centroids (sorted by x-coordinate).
- **Operators**: Crossover, mutation, and selection mechanisms are reused from Task 1.

C. Multi-objective TSP (Distance and Profit)

In this task, each customer is assigned an intrinsic profit. The objective becomes twofold: minimize total distance and maximize net profit. Two algorithmic strategies were implemented:

- **Weighted-sum Method**: A single aggregated objective is constructed: $f = \text{distance} - \lambda \cdot \text{total\_profit}$, with tunable weight λ. This allows direct optimization using classical GA.
- **Pareto-based Method**: We adopt **NSGA-II** from Geatpy to simultaneously optimize two conflicting objectives—distance and negative profit—via non-dominated sorting and crowding distance.
- **Profit Calculation**: At each step of the tour, the profit from visiting a city is reduced by the cumulative distance up to that point.
- **Encoding and Operators**: Permutation encoding and OX/inversion mutation remain unchanged.
- **Selection**: Weighted method uses roulette selection; Pareto-based uses non-dominated front ranking.

D. Multi-tasking TSP (MFEA)

We design a **Multifactorial Evolutionary Algorithm (MFEA)** to solve two TSP instances simultaneously: one from the original 100-city coordinates, the other perturbed by random shifts.

- **Unified Population**: All individuals are encoded identically but assigned to one of the two tasks via **skill factors**.
- **Crossover**: **Ordered Crossover (OX)** is conditionally performed. If parents share skill factors or pass **Random Mating Probability (RMP)**, cross-task exchange occurs.
- **Mutation**: **Swap Mutation** randomly exchanges two positions, introducing diversity.
- **Reproduction**: Individuals inherit task labels from parents and are evaluated only on their assigned task.
- **Selection**: Within-task elitism is used; top individuals are preserved based on task-specific fitness.
- **Knowledge Transfer**: The effectiveness of cross-task learning is controlled by RMP; sensitivity studies are conducted with RMP=0.3, 0.5, 1.0.

E. Bi-level TSP with Depot Selection

This problem mimics real-world logistics by introducing depot selection as an upper-level decision. The process is hierarchically structured:

- **Upper Level**: Choose two distinct depots (from 10 candidates). One must be the starting and ending point; the other is a mandatory stop after visiting the first 50 customers.
- **Lower Level**: For each pair of depots, solve the modified TSP that enforces a depot stop in the middle of the route, using a custom fitness function cost = $d_1 \rightarrow$ 50cust $\rightarrow d_2 \rightarrow$ 50cust $\rightarrow d_1$.
- **Nested Approach**: The upper GA optimizes the depot indices; the lower GA solves the TSP with fixed depots. The fitness of the lower level propagates to the upper level.
- **Single-Level Reformulation**: A Random Key method encodes depot selection and TSP routing jointly. Sorting the last 100 genes determines customer visiting order, and the first 2 genes round to depot indices.
- **Validation**: Each solution is validated for feasibility (route length = 100, no duplicates, valid pair of depots), and the fitness reported for the recomputed distance matches.

III. EXPERIMENTAL RESULTS

The code is in the folder 'src', the experimental process ipynb is in the folder 'experiment', and this chapter focuses on describing the experimental results obtained.

A. Classical TSP

In order to solve the classic TSP with 100 customers, we employ a genetic algorithm (GA) via Geatpy. In a baseline experiment, we set the population size (`pop_size`) to 100 and the maximum number of generations (`max_gen`) to 500, obtaining a best route distance of 572.4207. Below are the first 20 customers in the best route:

```
[45 36 73 83 60 98 85 17 28 48 39 10 74 51 95 43 62 27 97 34].
```

Figure 1 shows a visualization of the resulting best route:

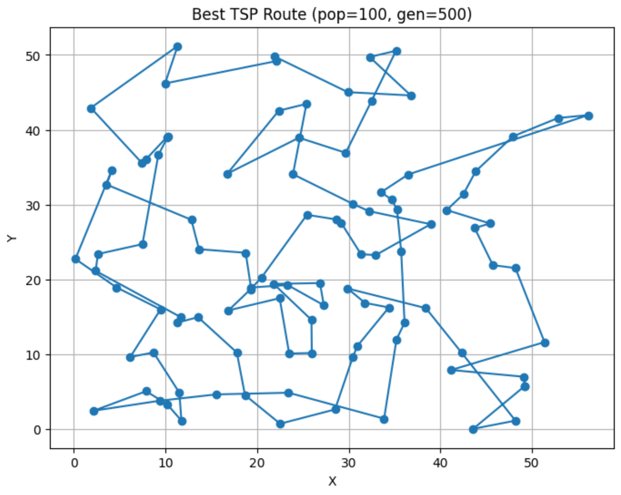

Figure 1: Best TSP Route (`pop=100`, `gen=500`)

Next, we investigate how changes in `pop_size` and `max_gen` affect solution quality by conducting a small parameter study, where:

$$\texttt{pop\_size} \in \{50, 100, 150\}, \quad \texttt{max\_gen} \in \{200, 500\}.$$

Table 1 summarizes the results:

Table 1: Effect of population size and number of generations on best route distance

pop_size	max_gen	best_distance
50	200	967.582658
50	500	637.034276
100	200	921.345686
100	500	625.262528
150	200	855.563563
150	500	537.773996

Figure 2 provides a visual comparison, indicating that larger population sizes and more generations generally yield lower distances.

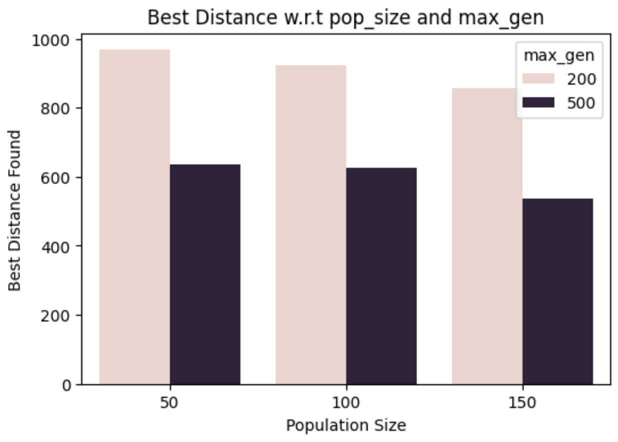

Figure 2: Best Distance with respect to `pop_size` and `max_gen`

As seen in Table 1, using `pop_size` = 150 and `max_gen` = 500 yields the best distance of 537.773996. In an additional run, we obtained an even lower best distance of 531.6974 with a corresponding route whose first 20 customers are

```
[8 45 94 81 48 6 87 39 10 74 68 59 65 26 11 40 77 15 52 43].
```

Figure 3 depicts the resultant route for these parameters.

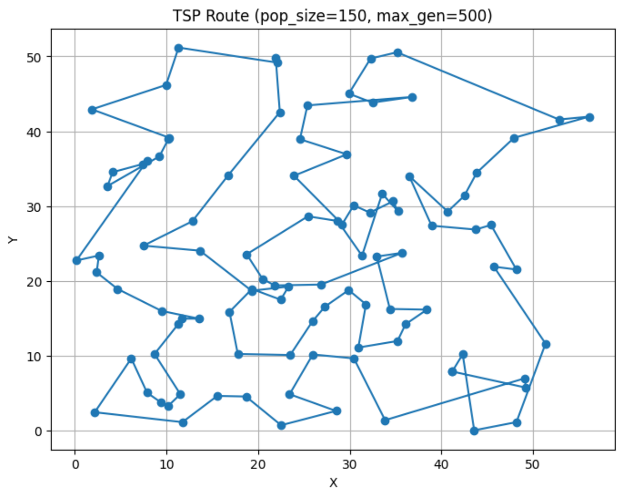

Figure 3: Improved TSP Route (`pop=150`, `gen=500`)

Overall, these experiments suggest that increasing both population size and generations can improve the quality of the classical TSP solution, although at the cost of increasing computational effort.

B. Large-scale TSP

In order to scale the classical TSP from 100 to 200 customers, we generate an additional 100 cities by shifting the original coordinates in the x-axis by a constant offset of 100. This transformation doubles the problem size while preserving the overall structure of the data. Next, we employ K-means clustering to divide the resulting 200 customers into several sub-regions, each of which is then solved locally by a genetic algorithm (GA) similar to that used in Task 1. Finally, the locally optimized routes are concatenated to form a global solution.

Baseline Experiment. With `pop_size_local` = 50, `max_gen_local` = 200, and `n_clusters` = 4, we obtain a global route distance of 1325.4233. The first 20 city indices of this route are:

`[2, 6, 19, 67, 82, 21, 95, 43, 62, 34, 97, 1, 27, 89, 96, 26, 52, 15, 87, 29].`

Figure 4 illustrates the final 200-customer route.

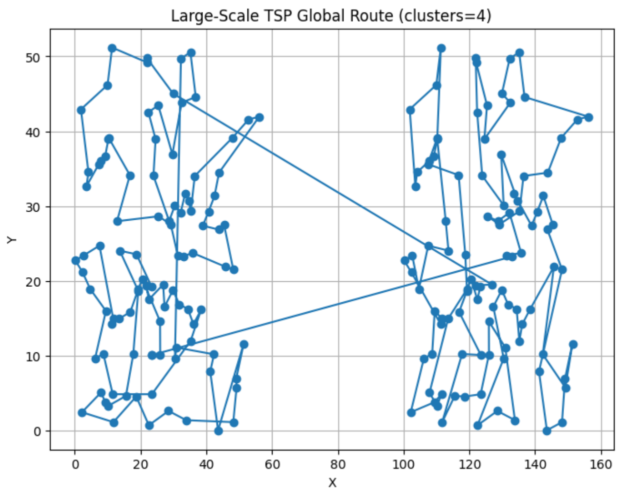

Figure 4: Large-Scale TSP Global Route (`n_clusters` = 4).

Parameter Sensitivity. To further explore how clustering and local GA parameters affect the global route, we vary:

$$\texttt{n\_clusters} \in \{2, 4, 6\}, \quad \texttt{pop\_size\_local} \in \{50, 100\}, \quad \texttt{max\_gen\_local} \in \{100, 200\}.$$

Table 2 summarizes the corresponding global route distances.
Figures 5 and 6 provide a visual comparison. From these results, we observe that increasing `n_clusters` can reduce local TSP complexity but may introduce additional distances between clusters. Similarly, raising `pop_size_local` or `max_gen_local` generally improves local solutions, thus lowering the final global route distance—albeit at a higher computational cost.

Table 2: Effects of clustering and local GA parameters on the global route distance.

n_clusters	pop_size_local	max_gen_local	global_distance
2	50	100	2312.936275
2	50	200	2171.280895
2	100	100	2281.868919
2	100	200	2152.295962
4	50	100	1308.068670
4	50	200	1294.696144
4	100	100	1324.540843
4	100	200	1307.463329
6	50	100	1250.918339
6	50	200	1246.608688
6	100	100	1266.948666
6	100	200	1258.935634

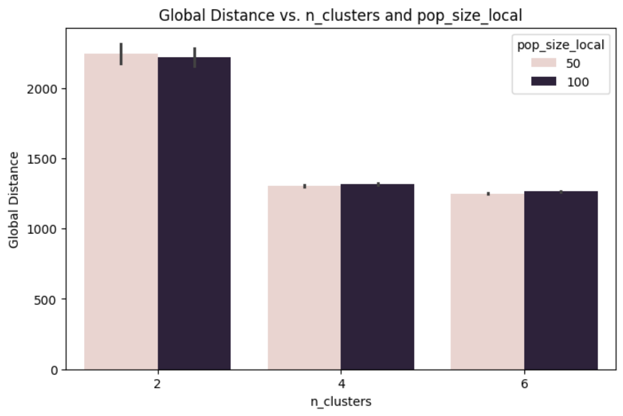

Figure 5: Global Distance vs. `n_clusters` and `pop_size_local`.

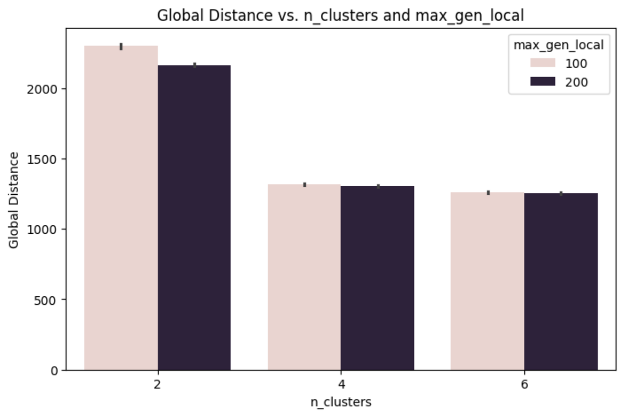

Figure 6: Global Distance vs. `n_clusters` and `max_gen_local`.

In short, clustering reduces problem scale within each region and enables faster local GA convergence, but the chosen cluster count and local GA parameters must be balanced to avoid excessive inter-cluster travel. A more adaptive clustering strategy or additional refinement steps could further enhance overall solution quality.

C. Multi-objective TSP

We address the problem of simultaneously optimizing total travel distance and profit using two different approaches:

1. **Pareto mode**: Minimizing f_1 = distance and $f_2 = -(\text{profit})$ simultaneously via an NSGA-II algorithm.
2. **Weighted mode**: Combining distance and profit into a single objective $f = \text{distance} - \lambda \times \text{profit}$.

The function `run_multiobjective_tsp(...)` in `task3_multiobjective.py` provides an option to switch between these two modes.

Weighted Method. To illustrate the weighted-sum approach, we set $\lambda = 0.1$, a population size of 100, and a maximum of 500 generations. Figure 7 shows a typical convergence trace under these conditions, and Figure 8 displays the corresponding best route in the coordinate plane.

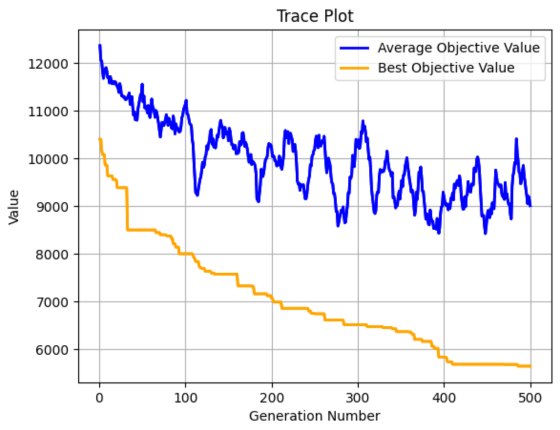

Figure 7: Convergence trace for the weighted single-objective method ($\lambda = 0.1$).

The best objective value is approximately 5650.4114, corresponding to

$$f = \text{Distance} - \lambda \times \text{profit}, \quad \text{where } \lambda = 0.1.$$

Increasing λ emphasizes profit more strongly, which can lead to longer routes if the profit gain outweighs the additional distance.

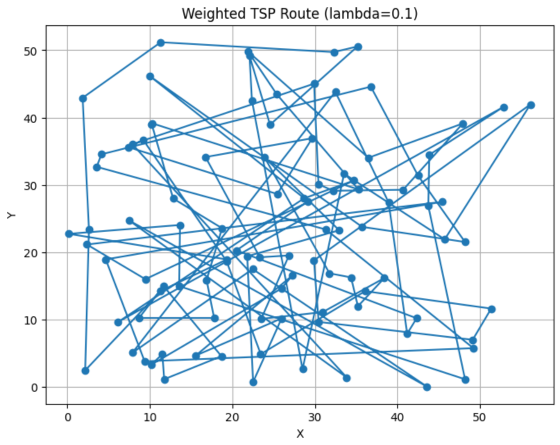

Figure 8: Weighted TSP Route ($\lambda = 0.1$).

To explore this trade-off, we tried $\lambda \in \{0.01, 0.1, 0.5, 1.0\}$ with the same GA settings and recorded the objective values shown in Figure 9. The resulting best objective values range from 2134.52 (for $\lambda = 0.01$) up to 44333.61 (for $\lambda = 1.0$), reflecting how shifting weight from distance to profit influences the final solution.

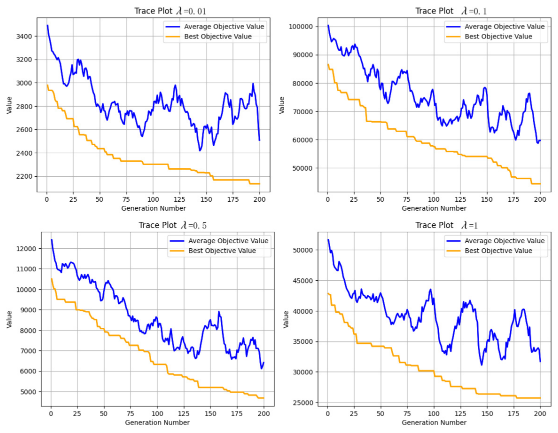

Figure 9: Comparison of different λ values in the weighted-sum approach.

Pareto-based Method. We then switch to `mode="pareto"`, enabling a multi-objective GA NSGA-II that treats distance and negative profit as two distinct objectives. The algorithm produces a set of non-dominated solutions, each balancing distance and profit differently. Figure 10 shows an example Pareto front, where points closer to the bottom left corner represent routes with shorter distance and higher profit.

In practice, this set of Pareto-optimal solutions enables decision makers to pick the route that best aligns with their distance and profit preferences.

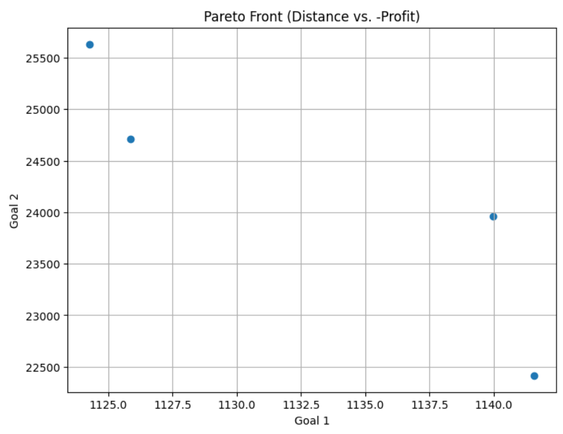

Figure 10: Pareto Front (f_1 = distance, $f_2 = -$profit).

D. Multi-tasking TSP (MFEA)

In this section, we explore the multi-tasking TSP problem using a Multifactorial Evolutionary Algorithm (MFEA). Two related TSP instances are addressed:

- **Task 0**: The original 100-city TSP.
- **Task 1**: A perturbation of the same TSP, where each city's x-coordinate is randomly offset within $[0, 50]$.

Our primary objectives are to:

1. Compare performance on both tasks under a single population framework.
2. Investigate how the random mating probability (`RMP`) controls cross-task genetic exchange and impacts final solution quality.

Baseline Experiment. With default parameters `RMP` = 0.3, `pop_size` = 100, and `max_gen` = 500, MFEA yields the following best distances:

- Task 0: 879.1500, with a sample route prefix `[23, 47, 49, 78, ...]`.
- Task 1: 1331.1426, with a sample route prefix `[37, 46, 39, 53, ...]`.

RMP Sensitivity Study. To see how different `RMP` values affect each task's best distance, we test `RMP` $\in \{0.0, 0.1, 0.3, 0.5, 0.8, 1.0\}$. Table 3 summarizes the results, and Figure 11 visualizes them.

Observations for Task 0 (Original). Task 0 achieves its best results around `RMP` = 0.0, 0.5, or 1.0. Notably, `RMP` = 0.8 yields a higher distance ($\approx$ 1064), suggesting that excessive cross-task mating may occasionally disrupt Task 0's elite structures.

Table 3: Effect of RMP on best distances in the multi-task TSP

RMP	Best Dist. (Task 0)	Best Dist. (Task 1)
0.0	963.588820	1437.927403
0.1	979.897317	1648.691916
0.3	1058.177958	1382.142548
0.5	966.469141	1507.147494
0.8	1064.081072	1468.868613
1.0	962.327637	1423.931875

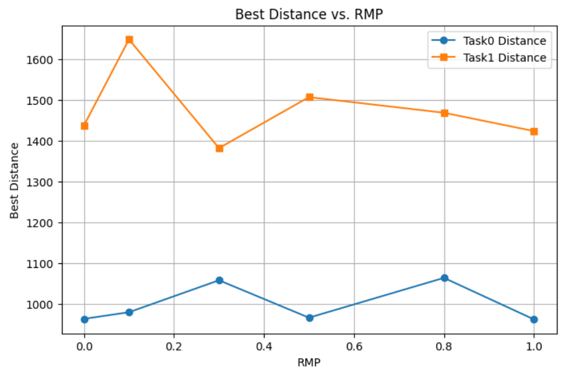

Figure 11: Best Distance vs. RMP for Task 0 and Task 1.

Observations for Task 1 (Perturbed). Task 1 exhibits more dramatic fluctuations. At `RMP` = 0.1, the distance exceeds 1648, indicating detrimental knowledge transfer. By contrast, `RMP` = 0.3 or higher generally helps, with performance improving again near `RMP` = 1.0. Since Task 1 is generated by perturbing Task 0, moderate-to-high cross-task exchange can be beneficial, though too-frequent mating may still degrade performance for the original task.

Overall Insights. No single RMP optimizes both tasks simultaneously, reflecting different sensitivity to cross-task information. Nevertheless, `RMP` = 1.0 consistently produces near-best results for both tasks, implying that Task 1 gains from broader genetic exchange, while Task 0 is not overly harmed. Figure 12 shows example routes for `RMP` = 1.0.

E. Bi-level TSP

In this section, we compare two GA-based strategies for the Bi-Level TSP problem with depot selection:

1. **Nested Evolutionary Algorithm** (`method="nested"`):
    - *Upper Level*: Select a pair of depots (d_1, d_2) from 10 candidates.

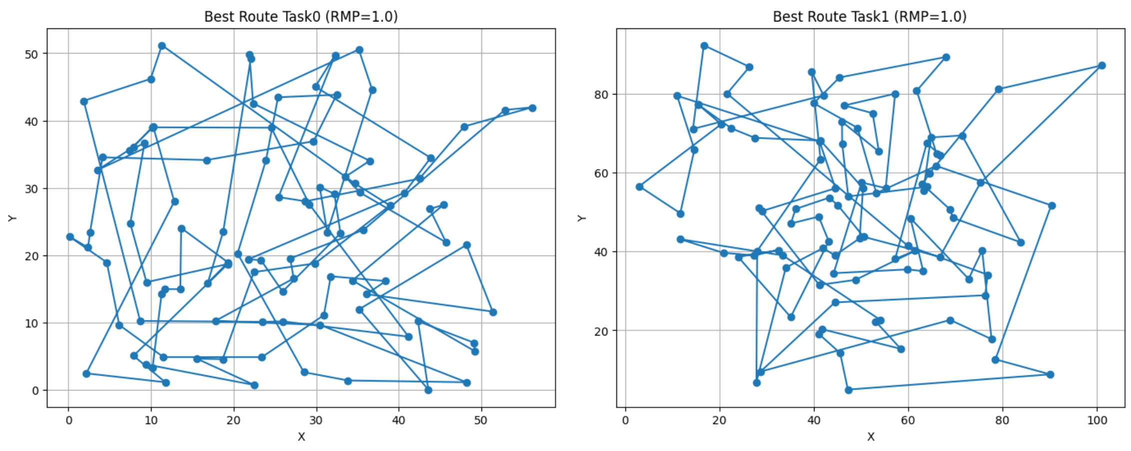

Figure 12: Best routes under MFEA at `RMP` = 1.0 for Task 0 (left) and Task 1 (right).

- *Lower Level*: Given (d_1, d_2), solve the TSP route, forcing a stop at d_2 immediately after visiting 50 customers.

2. **Single-Level (Random Key) Approach** (`method=“single”`):
   - Encodes both the depot pair and the 100-customer permutation in a single chromosome.
   - A random-key scheme ensures a valid permutation for the TSP segment while embedding depot choices as real-valued genes in the same individual.

The function `run_bilevel_tsp(...)` from `task5_bilevel.py` implements both methods.

Nested Evolutionary Algorithm. To illustrate the nested approach, we use a smaller upper-level population and generation count for demonstration, plus standard parameters for the lower-level GA. As shown below, the upper-level GA discovers a depot pair $(d_1, d_2) = (4, 8)$, and the lower-level TSP then achieves a route distance of approximately 1562.6479. The validated route is length 100, visits all customers exactly once, and successfully stops at depot 8 after 50 customers:

```
[3, 46, 96, 99, 22, ..., 71, 91, 27, 89],
```

with no discrepancies found (distance matched within 10^{-6}).

Single-Level (Random Key) Approach. Next, we switch `method="single"` to unify depot selection and TSP encoding in a single chromosome. This scheme yields a best depot pair $(d_1, d_2) = (7, 8)$ and a total route distance of around 1100.9127. The resulting route is also verified to be length 100 and free of duplicates, with a midpoint stop at the chosen second depot. In this particular run, the single-level approach found a noticeably smaller distance than the nested method:

```
[80, 23, 47, 95, 43, ..., 7, 78, 49],
```

again matching the computed distance to high precision.

Comparison and Observations. Although both methods satisfy the bi-level depot constraints, they can produce different final solutions:

- **Nested Approach**:
  - Separates depot selection (upper-level) from TSP route optimization (lower-level).
  - Each depot pair candidate triggers a full TSP solution, leading to potentially high computational cost.
- **Single-Level Approach**:
  - Encodes all decisions in one chromosome, reducing the nested calls.
  - May converge faster to a high-quality solution, but is more complex to design (especially if the problem constraints grow more intricate).

In our example, the single-level approach yielded a significantly smaller route distance. However, performance depends on parameter tuning (e.g. population size, number of generations) and the specifics of the depot distribution and TSP structure. For real-world bi-level logistics scenarios, the choice between nested vs. single-level methods involves a trade-off between code complexity, computational overhead, and the depth of insight gained into separate depot selection and route optimization stages.

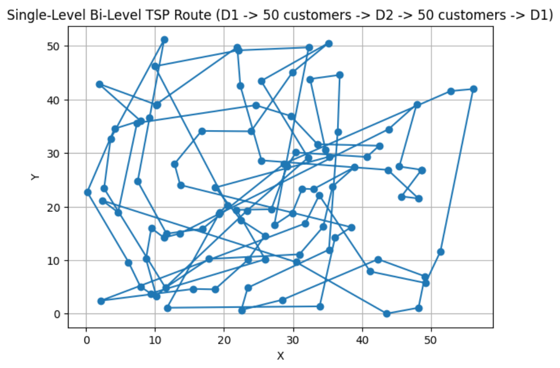

Figure 13: Single-Level Bi-Level TSP Route (D1 - 50 customers - D2 - 50 customers - D1)

IV. CONCLUSION

This project involved examining five distinct variations of the Traveling Salesman Problem (TSP) and developing tailored genetic algorithm (GA) solutions for each. We provided an in-depth discussion of essential operators like crossover and mutation, along with selection strategies and frameworks. Additionally, we performed sensitivity analyses on vital parameters such as population size, crossover and mutation rates, number of generations, and random mating probability (RMP), presenting our findings through numerous tables and figures to demonstrate their impacts.

- **Classical TSP (Task 1)**: By examining how the population size and the number of generations affect the final length of the route, we confirmed that increasing these parameters typically results in shorter trips, albeit at higher computational cost. Multiple runs demonstrated improved stability in identifying near-optimal routes, underscoring the importance of parameter tuning.
- **Large-scale TSP (Task 2)**: We scaled TSP from 100 to 200 cities by duplicating and shifting coordinates, then applied a K-means clustering strategy to divide and conquer. Sensitivity experiments revealed that cluster count, local GA population, and generations influence how effectively the method balances intra-cluster optimization vs. inter-cluster transitions.
- **Multi-objective TSP (Task 3)**: We explored both weighted-sum (with a tunable parameter λ) and the Pareto-based NSGA-II methods, analyzing trade-offs between distance and profit. By repeatedly running the algorithm with different λ values or objectives, we demonstrated how solutions shift in terms of distance–profit compromises, and how the Pareto approach can yield a diverse set of non-dominated solutions.
- **Multi-tasking TSP (Task 4)**: Two related TSPs were solved concurrently using a Multifactorial Evolutionary Algorithm (MFEA), with cross-task knowledge transfer regulated by random mating probability (RMP). Our experiments confirmed that RMP has a significant impact on the quality of the solution for each task, particularly when the tasks share structural similarities. Systematic testing with multiple RMP values provided clear insight into when cross-task mating is beneficial or disruptive.
- **Bi-level TSP (Task 5)**: We investigated a hierarchical problem where the selection of the upper level depot interacts with a lower level TSP. We compared a two-level nested GA vs. a single-level reformulation that encodes both depot choices and route permutations in one chromosome. Repeated runs showed how different parameter settings can strongly favor one method over the other, reflecting trade-offs in complexity, computational overhead, and depot–route integration.

Generally, comprehensive algorithmic designs(such as operators, selection mechanisms, and encodings) along with broad parameter explorations (including population size, crossover/mutation rates, and generations) were crucial to reveal the capabilities of each approach. Having the outcomes in tables and figures allowed us to visualize and quantify the improvements in various settings, and multiple independent trials confirmed the consistency of the observed patterns. Future research might include adaptive parameter tuning, hybrid local search strategies, or sophisticated heuristics for clustering and depot selection, aiming to advance solution quality and scalability in real-world routing applications.