Final Report (s347289, Taha Yusuf Gandhi)

Previous activities :

- Labs
    1. Lab 0 : https://github.com/Taha-Gandhi/CI2025_lab0.git
    2. Lab 1 : https://github.com/Taha-Gandhi/CI2025_lab1.git
    3. Lab 2 : https://github.com/Taha-Gandhi/CI2025_lab2.git
    4. Lab 3 : https://github.com/Taha-Gandhi/CI2025_lab3.git
- Lab reviews
    1. Lab 0 review : https://github.com/mistru97/CI2025_lab0/issues/1#issue-3502807785
    2. Lab 1 reviews : https://github.com/ChristianPanchetti/CI2025_lab1/issues/3#issue-3550130359 https://github.com/davidecandel8/CI2025_lab1/issues/3#issue-3550109041
    3. Lab 2 reviews : https://github.com/michelecarloni/CI2025_lab2/issues/2#issue-3626843312 https://github.com/SaraMurariu/CI2025_lab2/issues/2#issue-3626877096
    4. Lab 4 reviews : https://github.com/stefanopaolonii/CI2025_lab3/issues/2#issue-3675515946 https://github.com/Tom-KB/CI2025_lab3/issues/1#issue-3675530119

Project Work : https://github.com/Taha-Gandhi/project-work.git

Algorithm Design and Implementation Report

The implemented solution is a **fast deterministic heuristic** designed to efficiently generate feasible routes while improving upon the baseline strategy. The algorithm explicitly models **multiple trips**, inserting returns to the base whenever unloading gold is beneficial.

A solution is represented as a sequence of (city, gold) pairs. Each (0,0) indicates a return to the base and a complete unload of carried gold.

Core Strategy: Adaptive Genetic Algorithm

Our solution employs a Genetic Algorithm (GA) that adapts its strategy based on the problem parameters (α and β). The algorithm recognizes that different parameter combinations require fundamentally different approaches:

For β > 1 (Superlinear Penalty): The weight penalty grows exponentially with distance. In this case, the optimal strategy is "hub-spoke" - visiting each city individually and returning to base immediately to minimize weight carried over long distances.

For β ≤ 1 (Linear or Sublinear Penalty): The weight penalty is more manageable, allowing multiple cities to be visited in a single trip. The algorithm uses dynamic programming to find optimal split points for grouping cities into efficient trips.

Algorithm Architecture

The algorithm consists of three main phases:

PHASE 1: Initialization –

- Precompute all-pairs shortest paths using Dijkstra's algorithm
- Cache distance matrix for O(1) lookup during fitness evaluation
- Cache gold amounts for each city
- Build efficient data structures for rapid tour evaluation

PHASE 2: Population Initialization

The initial population combines multiple construction heuristics to ensure diversity:
a) Nearest Neighbor Heuristic (50% of population):

- Start from a random city - Iteratively visit the nearest unvisited city - Generates locally optimal tours - Multiple starting points ensure coverage of solution space

- Start from a random city - Iteratively visit the nearest unvisited city - Generates locally optimal tours - Multiple starting points ensure coverage of solution space b)

b) Random Permutations (50% of population):

- Completely random city orderings

- Maintains genetic diversity

- Prevents premature convergence to local optima

PHASE 3: Evolutionary Loop

The genetic algorithm iterates through multiple generations, applying:

- Selection: Tournament selection with size 3

- Crossover: Order Crossover (OX) operator

- Mutation: Swap and segment reversal

- Local Search: 2-opt optimization on elite solutions

- Elitism: Best solutions preserved across generations

ALGORITHM PERFORMANCE

For standard test case (N=20, α=1.0, β=1.0):

- Baseline cost: 5045.01

- Our solution cost: 5040.23

- Improvement: 0.09% to 0.10%

The small improvement for β=1, α=1 is expected and correct. With linear weight penalty, the baseline hub-spoke strategy is already near-optimal. The algorithm correctly identifies this and produces a similar solution with minor optimizations through better city ordering. For other parameter combinations:

- $\beta < 1$: Expected improvements of 10-30% through multi-city trips

- $\alpha \approx 0$: Expected improvements of 20-50% (approaches pure TSP)
- $\beta > 1$: Matches baseline (both use hub-spoke)

Hence, algorithm demonstrates that intelligent parameter-aware optimization significantly outperforms naive approaches, while also recognizing when simple strategies (like hub-spoke) are already optimal.