CS 3353 Algorithms: Lab 4 Report

By Xuan (James) Zhai

**1: Introduction**

In this lab, we are going to solve the traveling sales man problem (TSP) and find a Hamiltonian path in a graph. However, unlike Lab3, we will implement the Heuristic techniques which including the Tabu Search Algorithm and Genetic Algorithm. Those algorithms may not return the shortest path in the TSP problem, but they will approximate the optimal path in a short time (much shorter than Naïve and Dynamic Method). Like Lab3, we will start from 4 nodes and run all four algorithms (including the Naïve and Dynamic Method that implemented in Lab3). Every time we increment the number of nodes and to explore the change of execution time for those methods. Moreover, for the large-number nodes that we cannot use Naïve and Dynamic Method, we will only run the Tabu Method and Genetic Algorithm. In the end, we are going to build learning curves to find how those algorithms find the “minimum” path, and we will discuss how to optimize those methods to find a better solution.

**2: Code**

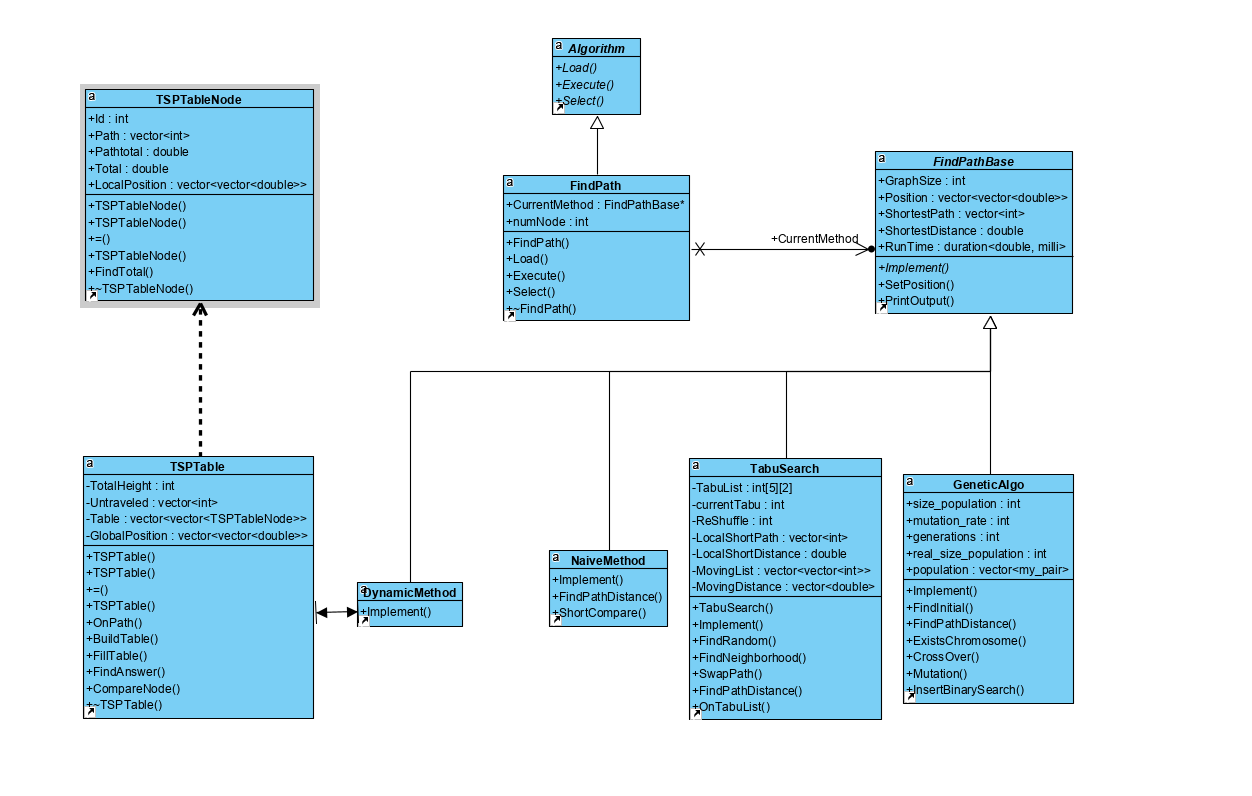


Figure 1: UML Diagram for Lab4

1. **Pattern interface:** In the “main.cpp”, I use a “Algorithm” pattern which goes through the pattern’s implementations from Load, to Execute, and to Select. I use a “FindPath” class to inherit the interface and achieve the functionality. In main, I use a for loop that loop the implementations twice so that I can run all four algorithm automatically. In the FindPath, I use a “FindPathBase” calls “CurrentMethod” to point to each algorithm; the default algorithm for CurrentMethod is the Tabu Search Method.
2. **Pattern-Load:** In the Load function in FindPath which is inherited from Algorithm, I pass a string file name and load the positions.txt which is the same as we did in Lab2.

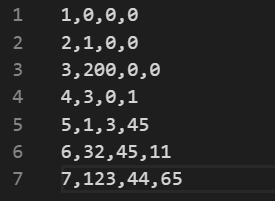


Figure 2: Sample Load file - “positions.txt”

Each line contains a node ID and its axis. I firstly use a temporary 2D double vector to store all axis. Since the first column is always in the form (1, 2, 3...), so the row x in the temporary vector represents the axis for number x+1. Next, we create a N by N matrix (2D double vector) to store the position from one number to another. For instance, (xi, yi) in the matrix contains the distance from xi + 1 to yi + 1. To find the distance, we apply the Distance Formula and use it to fill out our Position matrix. Eventually we set the Position matrix into each “FindPathBase” algorithm pointer.

1. **Pattern-Execute:** In the Execute function, the program will run each algorithm to find the shortest path with the shortest distance, and it will print out the result to the screen. At last, the program will also print the runtime (the time for finding the shortest path) to the screen. The time includes the time not only for find path in a space but may also the time for setting the space.
2. **FindPathBase:** In the FindPathBase class, it has 1: GraphSize to store the number of nodes; 2: Position to store the position map we created before; 3: a vector of integer called “ShortestPath” to store the shortest Hamiltonian path; 4: ShortestDistance initialized with INT\_MAX to store the distance for the Hamiltonian path; 5: RunTime for the implementation of High Resolution Clock. Furthermore, FindPathBase has a visual function that was inherited by TabuSearch, GeneticAlgo, DynamicMethod, and NaiveMethod.
3. **Tabu-Search-Method:** In the implementation of Tabu Search Method, I firstly create a Tabu list a moving list, and a moving distance list, which moving list is the combination of possible swaps that can create neighborhood and moving distance list the distance change if we swap those two nodes. Furthermore, I create a reshuffle trigger. Since it is easy for Tabu Search Method to fall into the local minimum, the reshuffle trigger will pull the searching out and randomly create a new path if it becomes active.

In the source file, the implementation will firstly create a random path, and it will update local/global path and distance. After that, the algorithm will find all the possible neighbor and sort them based on their distance change. Then, it will swap the nodes if there are not on the Tabu list. The algorithm will add that combination on the Tabu list to prevent infinite loop. Finally, if the new distance is shorter than the global distance, it will update the global path and global distance. In the Tabu Search Method, there is an Aspiration criterion which is if the new distance is smaller than the global distance, the algorithm will do the swap even it is on the Tabu List.

In the Tabu Search, we set the Tabu list size as 5, so if the Tabu list is full, it will replace the first coming item on the list and follow the First-in-First-out rule like a queue.

We set 200 generations for the Tabu Search method (same as the generations in the Genetic Method). If the global distance is not updated in one generation, the reshuffle will increment by 1. If the reshuffle trigger achieves 50, it will redefine the Tabu List and randomly create a new local path.

1. **Genetic Algorithm**: The basic implementation of Genetic Algorithm was changed from [https://github.com/marcoscastro/tsp\_genetic](https://github.com/marcoscastro/tsp_genetic/blob/master/src/tsp.cpp). Firstly, like the Tabu Search Method, the algorithm will randomly create a path, but in the Genetic Algorithm the path does not include 1. So 2→4→5→3 reflects 1→2→4→5→3→1, and the distance we measured includes the start/end point 1. After we create the first path, it will create parents to build child.

In theGenetic Algorithm, the most important parts are Selection, CrossOver, and Mutation. The Selection is composed in the implement method, and it selects two parents in the population list to generate children (normally the best fit parents). Furthermore, after each generation, the algorithm will eliminate those parents who have worst fitness (Extinction). For the CrossOver method, I changed it to the Order-Based Crossover (OX). Basically, a swath of consecutive alleles from parent 1 drops down, and remaining values are placed in the child in the order which they appear in parent 2 (like Figure 3 below). In the method, I randomly choose the chose substring, and by using two random start point and end point, the CrossOver may create children that have better fitness. Finally, in the Mutation, I set a mutation rate which is 4. If the mutate trigger is lower than the mutation rate, a mutation fill happened. In the mutation function, we will randomly swap two nodes in the children path and record it as new chromosomes.

Notice, in the new generation, the algorithm will rearrange the population, and the first on the population list has the best fitness. In this TSP problem, the fitness is the total distance of the path.

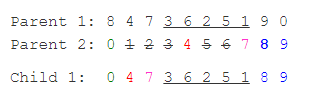


Figure 3: Basic idea about Order-Based Crossover (OX)

1. **Pattern-Select:** When the program finished one algorithm, it will call a Select method that will delete current algorithm “CurrentMethod” and create a new one that has different algorithm.

**3: Data**

1) When we use 4-13 nodes for this TSP problem, we find out that the time costs for Tabu Search Method and Genetic Method, from the Figure 4, are far smaller than Naïve Method and Dynamic Programming Method, and their shortest paths are close to the true answer. For the Tabu Search Method. It forms a time change that is close to a linear time complexity because every time we find a swap it is a linear search, and the Tabu List is too small to make a change. On the other hand, the Genetic Algorithm costs even smaller and forms a time complexity around constant. We need to do further test to understand what caused the change in timing. In the Figure 5, I combine the time we have in Figure 4 with the actual and asymptotic time of Naïve and Dynamic method. We also plot graphs (Figure 6-10) to show those results.

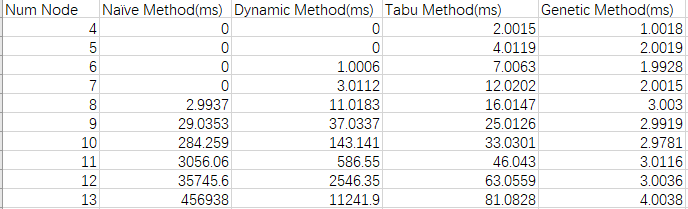


Figure 4: Time to solve TSP

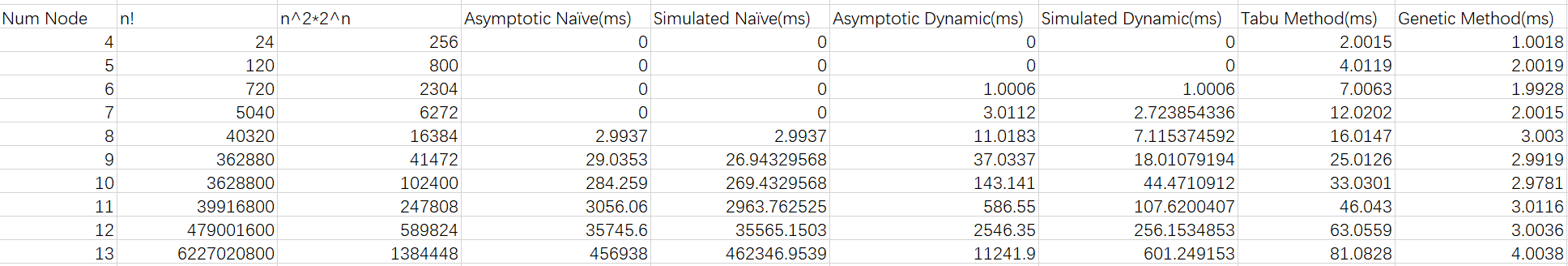


Figure 5: Time to solve TSP (with actual and asymptotic data)

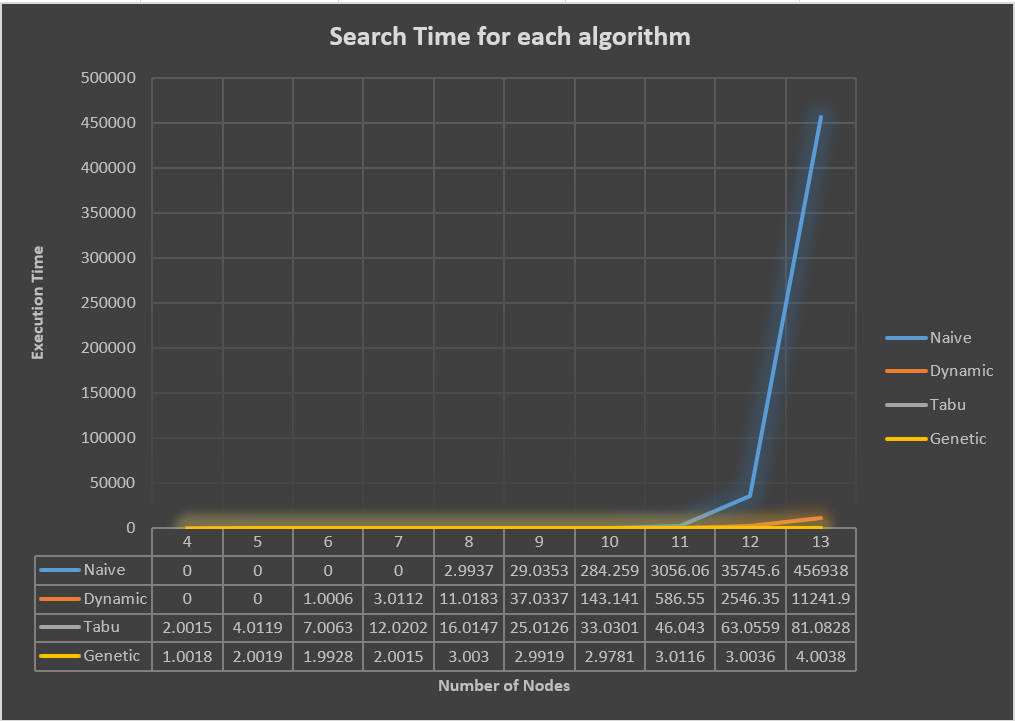


Figure 6: Graph for solving TSP timing (Combined)

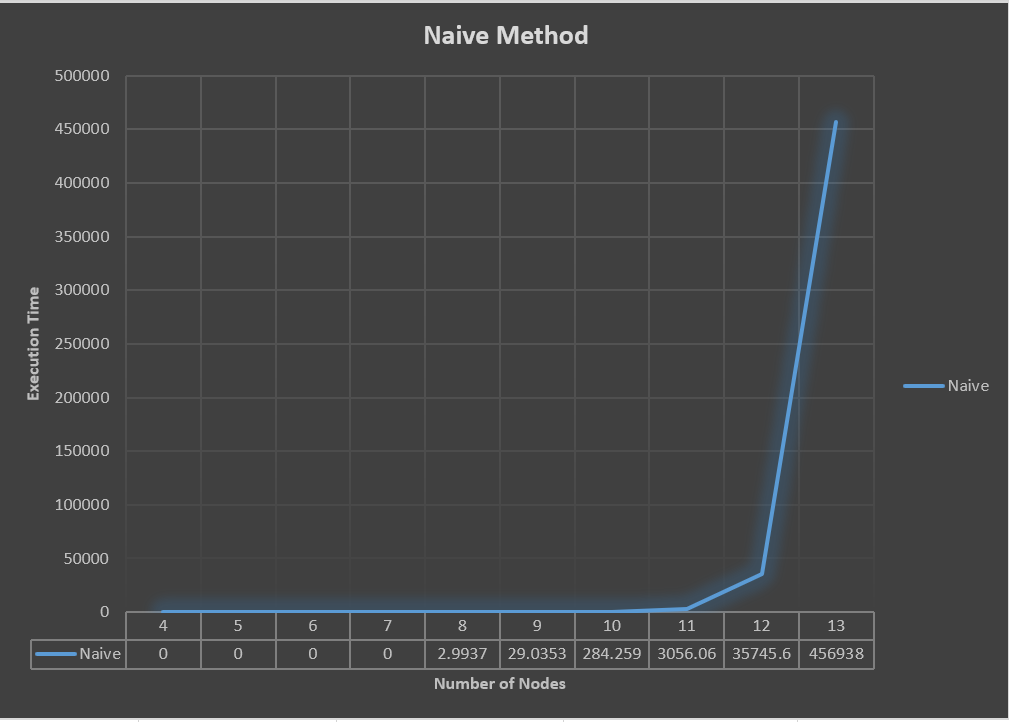


Figure 7: Graph for solving TSP timing (Naïve Method)

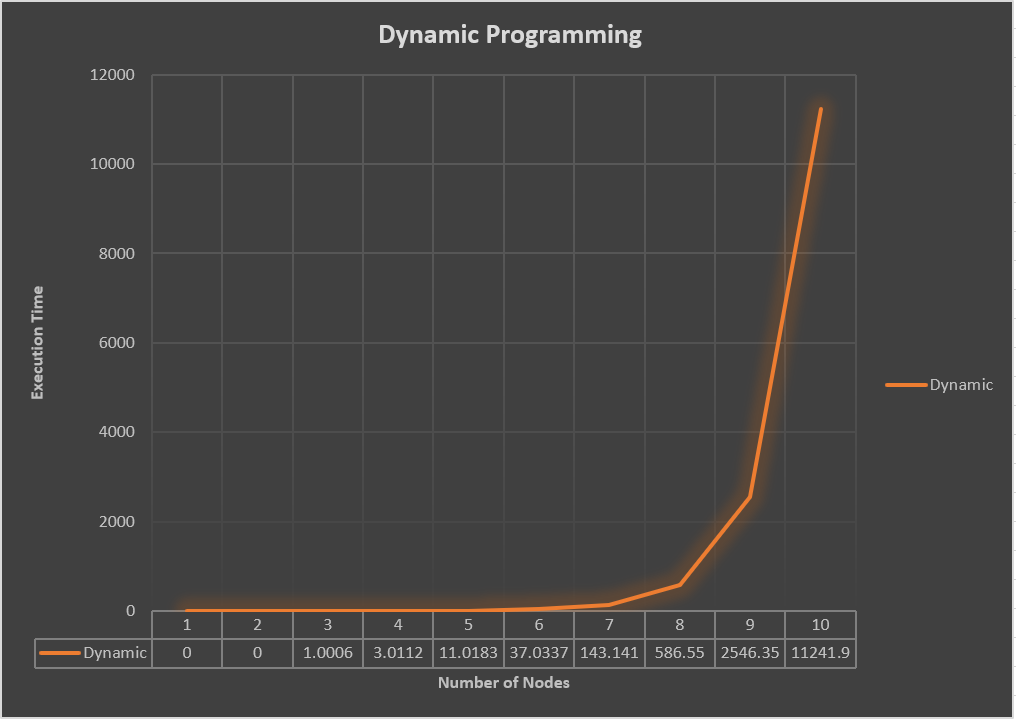


Figure 8: Graph for solving TSP timing (Dynamic Programming)

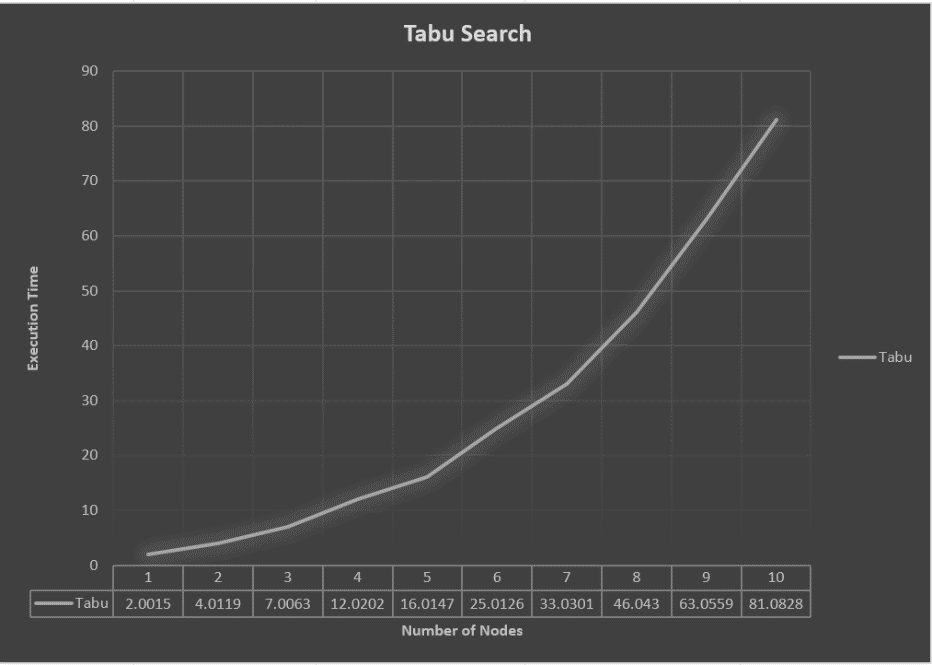


Figure 9: Graph for solving TSP timing (Tabu Search)

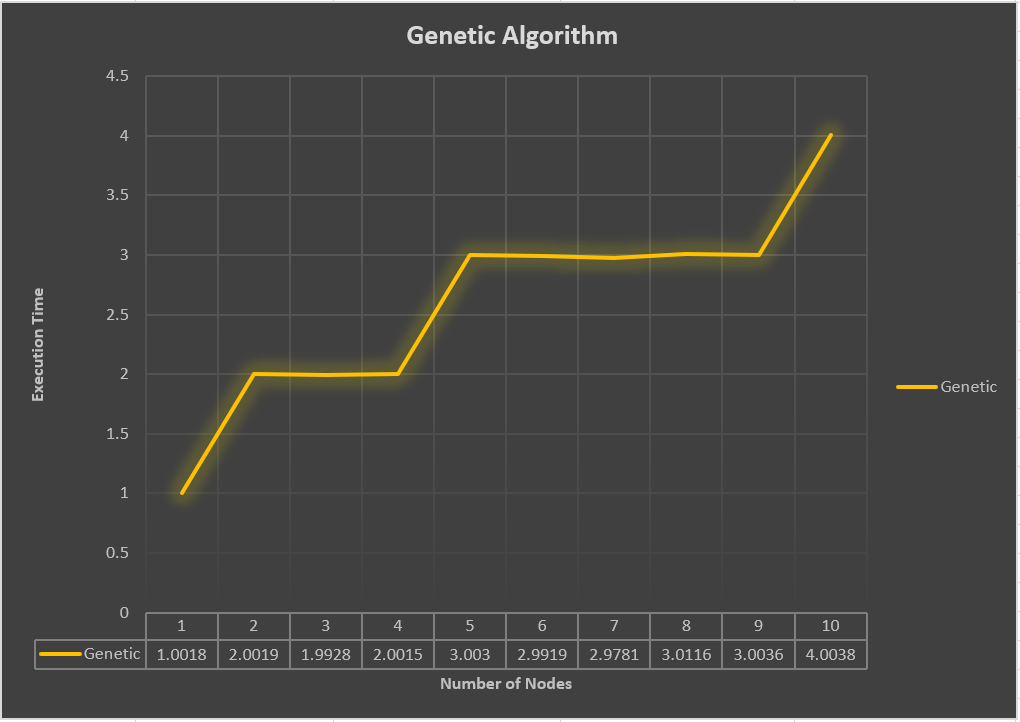


Figure 10: Graph for solving TSP timing (Genetic Algorthm)

In the next part, we will plot the learning curves (Figure 11-13). We choose 20 nodes in the tsp problem, and we run Tabu Search Algorithm and Genetic Algorithm both with 200 generation. Finally, we build the graph where the x axis is the epoch(generations), while y axis is the fitness. Since in this problem I uses the distance as the fitness, so **smaller y refers to better fitness**. Moreover, for the Tabu Search, I plot both the local minimum and the global minimum for each generation, and that’s because in this algorithm, I designed that if the reshuffle is greater or equal to 50, the path will be redefine. Therefore, in Figure 11 and Figure 12, the Tabu Local minimum will sometimes have dramatic increase, and at that time the path is rerandomized to find not just local but global minimum.

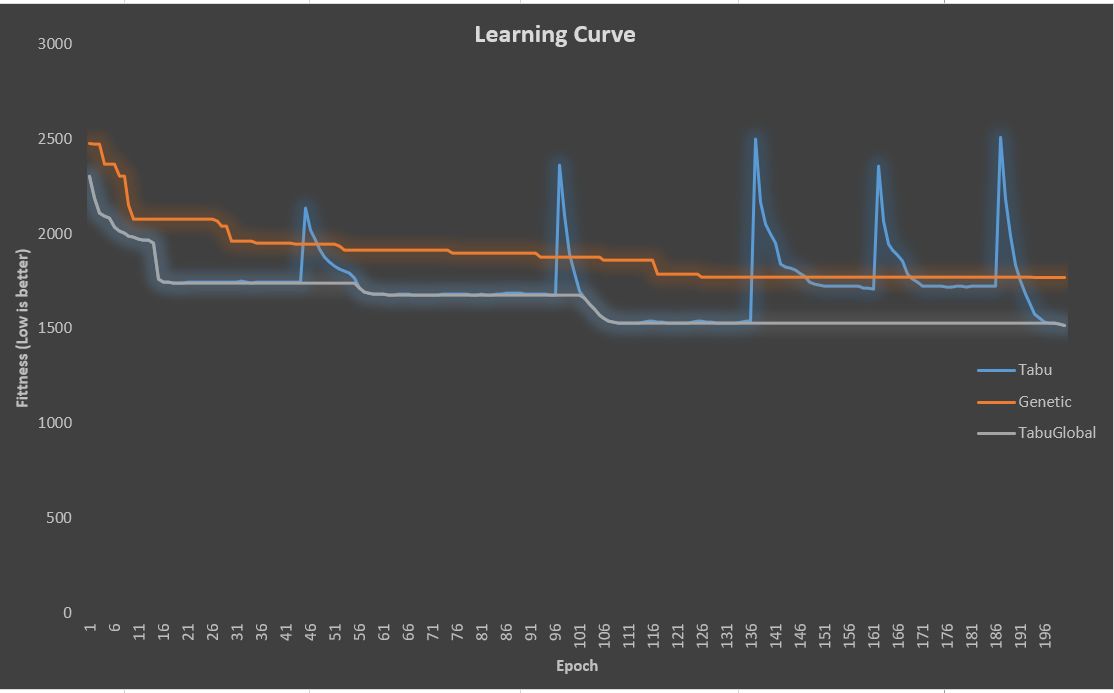


Figure 11: Learning Curve (Combined)

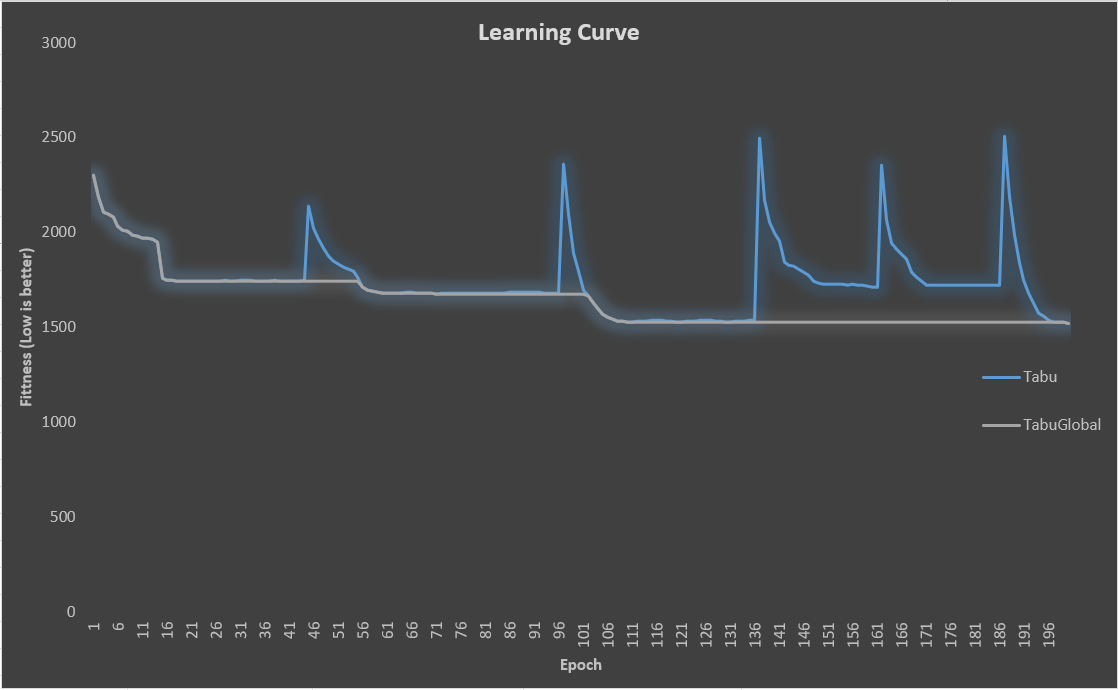


Figure 12: Learning Curve (Tabu Search)

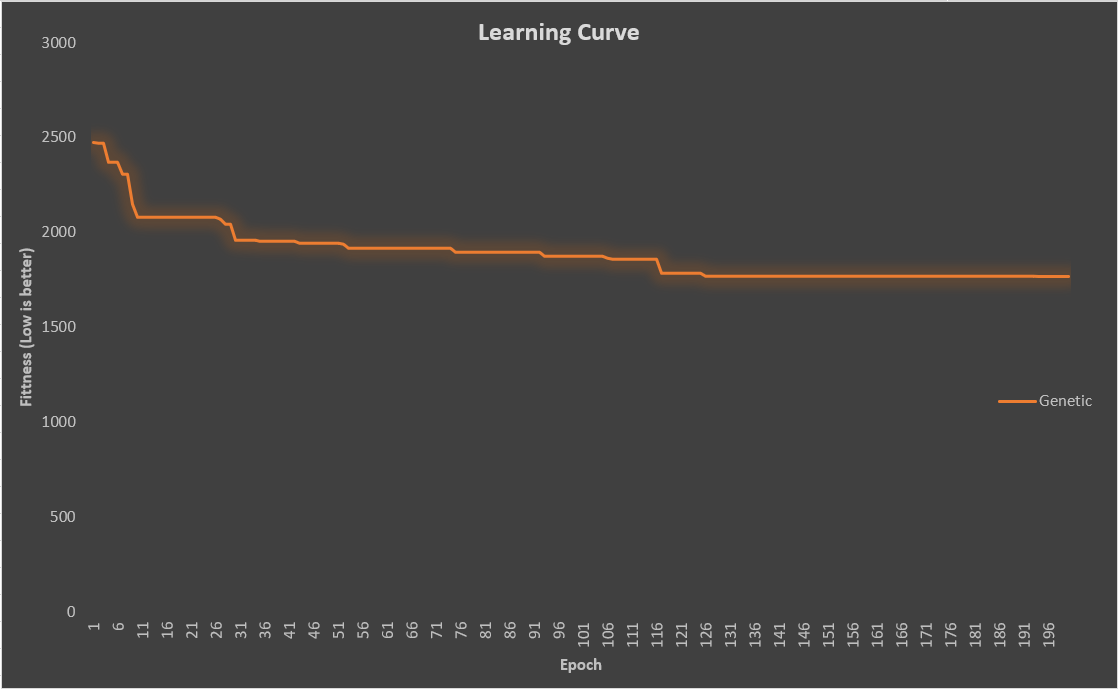


Figure 13: Learning Curve (Genetic Algorithm)

**4: Analysis**

In those graphs above, we find out that for the same problems with same generations, the Genetic Algorithm would not return the optimal path if the size is big. Since we use Order-Based Crossover (OX) with the random interval to do Crossover, and we use random location to mutate, the algorithm is not intelligent. To get a closer answer, we can crossover the shortest part within two parents so that the children may be more fit, or we can have more “clever” mutation strategy so that the children can jump close to the global minimum. Besides, we can modify the generation and the generation we used, because a higher generation may return a better approximation. On the other hand, for the Tabu Search method, we can change the size of Tabu List and remodify the neighborhood to improve this algorithm. Overall, for both algorithms, there learning curve shows that they are improving their results, while Tabu Search method can return a closer result, the Genetic Algorithm takes little time.