Mohammed Alom
Student No – R00144214
Metaheuristic Optimisation – Second Assignment

Part 1:

The TSP Problem is one of the widely solved problems in the existing literature, and it is computationally very intensive (NP-hard) to obtain the optimal solution. The program used the heuristic method, and it is relatively fast to obtain a near-optimal tour. As it is called heuristics, but there is no guaranteed solution in terms of closeness to the optimal solution. Nonetheless, the local search algorithm techniques are widely used to optimize their easy implementation and decent performance.

The TSP program used an input file consisting of n number of cities, and the distance d is the distance between two cities. The program's goal is to find a minimum tour distance of the cities, where the algorithm will visit the city only once during the tour, or distance is minimized. The algorithm has to make sure the tour begins and ends in the same city. It used 3-OPT for move and 2-OPT for swap phase. It is a combinatorial optimization NP-Hard problem. The program uses heuristics and random search rather than an exact search, so it is not right that the program's calculated tour will be the best tour solution. The advantage of using the heuristic search is a shorter running time, which makes it suitable for problems with large instances.

Algorithm

The program local search comprises two-stage. First, the algorithm improves the initial solution in a local search stage until reaching a local minimum. In the Second Stage, in the perturbation phase or swap stage, the algorithm perturbs the incumbent solution to escape from difficult regions of the search. Finally, the program decides whether to update or not. Below is the program structure-

1. Initial Solution
2. Local Search
3. while some condition
4. Perturbation
5. Local Search
6 Acceptance criteria
7. Until no stopping criteria

Local search Algorithm

For this program Local Search Algorithm, the 3-OPT stage is applied but in a modified way. At first, a random edge is selected. A 3-OPT move is then performed with the above edge and with every possible combination of the other two edges, which speeds up the process. For each iteration during the search, only the best one is retained.

Two-OPT Phase

The perturbation stage is simply a 2-OPT swap to escape from the local minima. The program runs five times to have some randomness. To speed up the assignment, here tested it only once. The 2-Optimization, or just the 2-opt, is a heuristic method that takes an existing TSP route and optimizes it by removing two edges. However, if the resulting tour has a longer or equal length to the length of the tour before the swap, then the algorithm undo the swap. The 2-opt procedure has an upper bound of O (n 3), but this is after a preliminary route has been created once this stage is over a program called local search again.

Acceptance Criterion

In this stage program will decides which solution to keep. A random 0.05% probability is added to select the solution generated from the above steps, and then only the best is retained. The running time to improve the algorithms depends on the number of neighbourhoods.

Program Usage

The main class of ***tsp.py*** uses Multi-Processing Process to spawn five different processes to run the program five times at a time. Users can make it their own choice to 1 if the user wants to run it only once. Plotly Python library is used to plot the initial and final graph for the problem. The program can generate two **ways of the initial solution.**

1. Random Solution 2. Nearest neighbour solution

Euclidean distance method is used to calculate the tour cost. Here this can be optimized to improve the algorithms as the cost is calculated after every 2opt / 3opt move. To improve the algorithm, need more research and investigation.

The program is set to stop at 300 seconds (However, it takes around 2 to 3 hours to run the program to get the result. However, it is not enforced to stop the program. However, if the program execution is already started and surpassed the given seconds, it will not be terminated automatically. Three hundred seconds program execution time is relatively less, and it hugely depends on the program's complexity of the program data/file.

Program Results

Both the initial and final solution image graphs are stored in the program results directory and tested for inst-0.tsp and inst-13.tsp file.

LENGTH OF THE ROUTE IS : 24691214.918058332

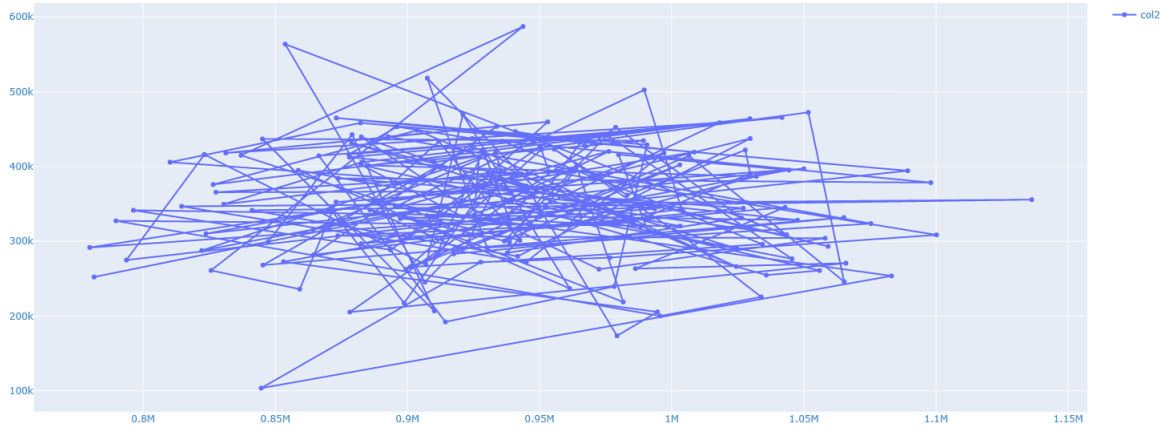

Fig -1: Initial solution inst-0.tsp

LENGTH OF THE ROUTE IS : 4109367.804768901

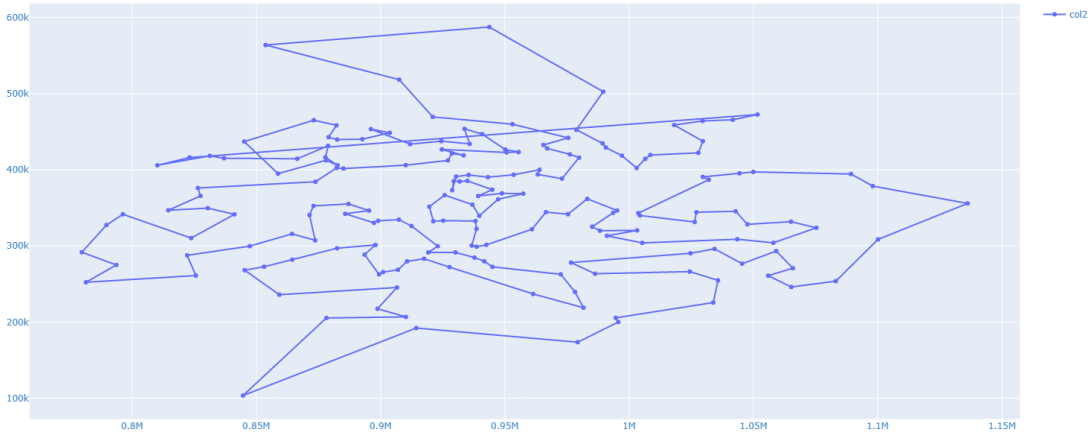

Fig-2: Final solution inst-0.tsp

LENGTH OF THE ROUTE IS : 120643819.40005808

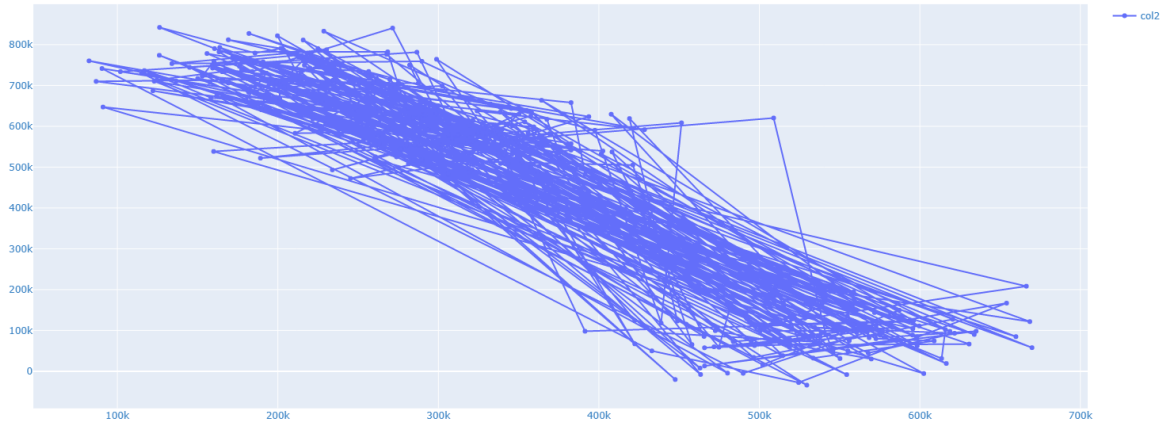

Fig -3: Initial solution of inst-13.tsp

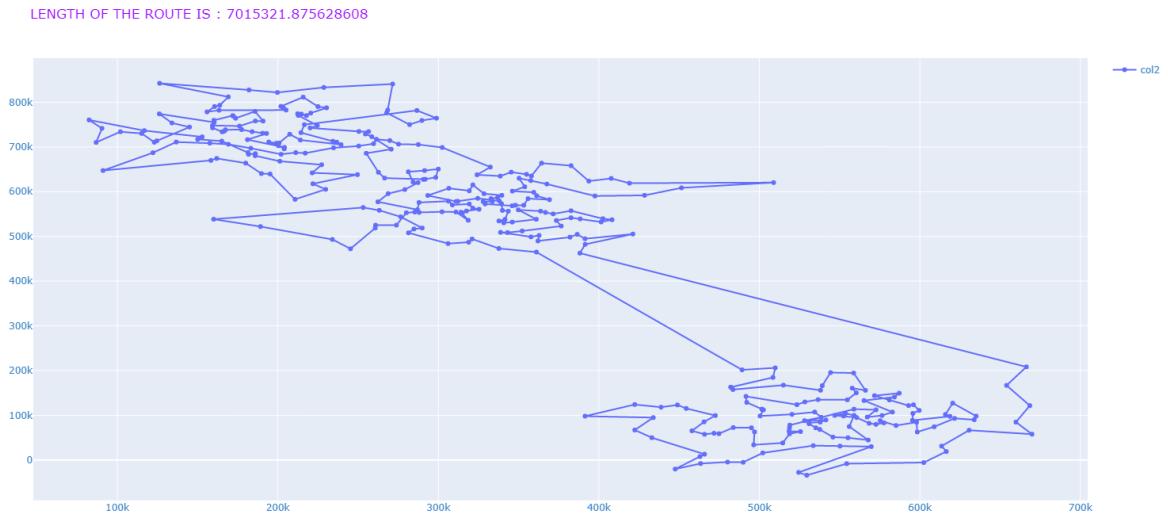

Fig 4: Final solution of inst-13.tsp

The rationale for the Algorithms

From the above four graphs, it is clear that the problem is optimized, but there is a lot to improve. Nearest Neighbor is fast for small input files, but it failed to find a solution after over 30 minutes of running on the larger input sets (input files over 10000 bytes). However, the runtime for 3-Opt was n^3, and was also too slow for larger values of n. It may not be very optimized,

Requirements to run the program

- Python 3 - Plotly

Running the program Just run the tsp.py file in the command prompt.

python tsp.py

Program Tested on below hardware specification

This program is tested on HP

- Windows 10
- AMD A9-9410 RADEON R5, 5 COMPUTE CORES 2C + 3G 2.90 GHz Processor
- 8 Gb RAM (Memory)
- 64-bit operating system, x64-based processor
- Anaconda – Spyder IDE used

Part 2:

In part two, the program's main objective is to implement the N Queens problem by using the MaxMin hill-climbing search and its variants with different runtime distribution. This program will take the number of queens as a variable N. The program will allow the user to input N's value.

This is a local search algorithm, an iterative algorithm that starts with an initial solution and then looks to find a better solution by incrementally changing a single solution element. If the change finds a better solution, then an incremental change is made to the new solution, thus steps repeating continue until no further improvements can be found.

Hill climbing: For this variant, the queens are set on board at random positions. Using the heuristic value, we are calculating the attacking pairs. Also, we select the random child from the set of lowest cost heuristics. Finally, we again calculate the heuristic until we reached the goal or a failed state is reported.

Hill-climbing with no sideways move – here we proceed with no sideways moves also. However, if a local minimum is attained, we again select one of the lowest cost children and find a shoulder from where the global minimum can be attained.

Random restart hill climbing without sideways move- for this variant, we are starting a search from a randomly chosen start point, going all the way uphill. When stuck at the local minimum, we select a random start point again to search. If the first hill-climbing attempt does not work, then try again and again and again. That is, generate random initial states and perform hill-climbing again and again. This is a random-restart. The number of attempts needs to be limited, and this number depends on the problem.

It often fails to find a goal when one already exists because they get stuck on local maxima. To overcome this problem, Random-restart can be used to solve local maxima, as it conducts a series of hill-climbing searches from randomly generated initial states until a goal is found.

Below is the result screenshot

```
ENTER THE NUMBER OF QUEENS :8

HILL CLIMBING NO SIDEWAYS MOVES ALLOWED PROGRAM STARTED
FINDING THE SOLUTIOIN ... PLEASE WAIT ...

                                  SUCCESSES      FAILURE

    SUCCESS AND FAILURE RATE :  16.0%          84.0%
    AVERAGE NUMBER OF STEPS :   4              3

RANDOM RESTART HILL CLIMBING NO SIDE WAYS MOVES ALLOWED PROGRAM STARTED:
FINDING THE SOLUTIOIN ... PLEASE WAIT ...

                                SUCCESSES      FAILURE

SUCCESS AND FAILURE RATE:       100.0%         0.0%
AVERAGE NUMBER OF RE-START :        7.152777777777778
   AVERAGE NUMBER OF STEPS :        29             0
```

Fig -5: Hill-climbing run time distribution result

Source Consulted

1. https://stackoverflow.com/questions/53275314/2-opt-algorithm-to-solve-the-travelling-salesman-problem-in-python
2. Greekforgreek.com
3. Lecture Notes