A MEMETIC ALGORITHM FOR ORIENTEERING PROBLEM WITH HOTEL AND RESTAURANT SELECTION AND TIME WINDOW

BY

MR. APISIT CHENG

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF MASTER OF ENGINEERING IN

LOGISTICS AND SUPPLY CHAIN SYSTEMS ENGINEERING

SIRINDHORN INTERNATIONAL INSTITUTE OF TECHNOLOGY

THAMMASAT UNIVERSITY

ACADEMIC YEAR 2020

A MEMETIC ALGORITHM FOR ORIENTEERING PROBLEM WITH HOTEL AND RESTAURANT SELECTION AND TIME WINDOW

BY

MR. APISIT CHENG

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF MASTER OF ENGINEERING IN

LOGISTICS AND SUPPLY CHAIN SYSTEMS ENGINEERING

SIRINDHORN INTERNATIONAL INSTITUTE OF TECHNOLOGY

THAMMASAT UNIVERSITY

ACADEMIC YEAR 2020

THAMMASAT UNIVERSITY
SIRINDHORN INTERNATIONAL INSTITUTE OF TECHNOLOGY

THESIS

BY

MR. APISIT CHENG

ENTITLED

A MEMETIC ALGORITHM FOR ORIENTEERING PROBLEM WITH HOTEL AND RESTAURANT SELECTION AND TIME WINDOW

was approved as partial fulfillment of the requirements for
the degree of Master of Engineering in Logistics and Supply Chain Systems Engineering

on July 29, 2020

Chairperson _____
(Associate Professor Pisit Chanvarasuth, Ph.D.)

Member and Advisor _____
(Associate Professor Aussadavut Dumrongsiri, Ph.D.)

Member _____
(Assistant Professor Nattharika Rittippant, Ph.D.)

Member _____
(Associate Professor Charoenchai Khompatraporn, Ph.D.)

Director _____
(Professor Pruettha Nanakorn, D.Eng.)

Thesis Title	A MEMETIC ALGORITHM FOR ORIENTEERING PROBLEM WITH HOTEL AND RESTAURANT SELECTION AND TIME WINDOW
Author	Mr. Apisit Cheng
Degree	Master of Engineering in Logistics and Supply Chain Systems Engineering
Faculty/University	Sirindhorn International Institute of Technology/ Thammasat University
Thesis Advisor	Associate Professor Aussadavut Dumrongsiri, Ph.D.
Academic Years	2020

ABSTRACT

To design a tour plan which provide a maximum statisfaction, before have any experiences with the destination can ba hard and time consuming process. The goal of this study is to create an algorithm that efficiently generate a tour plan with high or maximum satisfaction within a reasonable processing time. The memetic algorithm which is a combination of genetics algorithm and local search algorithm would be created to solve this problem. This study used real data gathered from trusted tourist community in Thailand such as TripAdvisor.com, Wongnai.com, etc. Information of 181 sites were gathered including 39 hotels, 48 lunch restaurants, 49 dinner restaurants, and 45 tour sites. These information were used as an input for optimization and memetic algorithm approach. The result of this study shown that MA could provide avg. saving in computation time of 158.92 seconds, with the maximum of 736.69 seconds. Avg. %gap in total tour score of 1.59 with standard deviation of 2.24. As a summary, the MA approach could solve tour trip design problem efficiently since both saving in computation time and %gap are in a good shape and well-balanced.

Keywords: Tour trip design problem, Memetic algorithm, Mathematical model, Travel routing, Orienteering problem

ACKNOWLEDGEMENTS

First, I would like to take this opportunity to say thank you to my advisor, Associate Professor Dr. Aussadavut Dumrongsiri, for everything since the beginning of my thesis through many years of wonderful research. I have learned many thing throughout this period.

I also extremely grateful to the member of examination committee, Associate Professor Dr. Pisit Chanvarasuth, Assistant Professor Dr. Natthrika Rittippant and the external examiner, Associate Professor Dr Charoenchai Khompatraporn for their useful comments and advices.

Special thanks to the present secretary of the School of Management Technology for there support during my study.

Lastly, I would like to thank you to my family for there support. And also thanks for Excellent Thai Students Schorlarship by Sirindhorn International Institute of Technology, Thammasat University for supporting my graduate study.

Mr. Apisit Cheng

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF FIGURES

LIST OF SYMBOLS/ABBREVIATIONS

Symbols/Abbreviations	Terms
SIIT	Sirindhorn International Institute of Technology
TU	Thammasat University
OP	Orienteering Problem
OPTW	Orienteering Problem with Time Window
OPHS	Orienteering Problem with Hotel Selection
OPHRSTW	Orienteering Problem with Hotel and Restaurant Selection and Time Window
MA	Memetic Algorithm
TTDP	Tour Trip Design Problem

CHAPTER 1
INTRODUCTION

1.1 Overview of tourism in Thailand and orienteering problem with hotel and restaurant selection and time window (OPHRSTW)

According to report from World Travel & Tourism Council (WTTC), a total contribution of travel & tourism to Thailand GDP was 19.7% of GDP in 2019 (Those number calculated from total spending on Travel & Tourism within Thailand for business and leisure purposes by residents and non-residents, and spending by government on Travel & Tourism service directly linked to visitors). This show that Travel & Tourism is one of the largest Thailand economic sectors, therefore the goal for Thailand is to increase or at least maintain the same level of tourism. One possible solution for Thailand to achieve this goal is to create a tour plan that could leads to a high overall consumer tour satisfaction by using an information from past consumption experiences. Many studies, shows that overall consumer's tour satisfaction is an important factor directly affect intension of tourist to recommend (positive-review) a tour to their peer and relatives, and also affect intension to re-buy the tour or re-visit a region (Marin & Taberner, 2008; Valle, Silva, Mendes, & Guerreiro, 2006; Yoon & Uysal, 2005). According to Vlachos (2012), past consumption experiences on the internet is the main source of information for tourist to be used as a guideline for creating a tour plan. Because assessing quality of experiential goods (hotels, attractions, and restaurants) before consumption occurred is difficult (Liu & Park, 2015). Thus, consumer tent to used online comments or reviews scores, which is easy to access, as a source of information for them to reduce their level of perceived uncertainty (Ye, Law, Gu & Chen, 2011). So consumer's reviews on an online communities could be used as a source of information to create a tour plan that satisfied consumer's needs and conditions. However, tour planning is a time consuming process because consumer have to repeatedly making a decision to include or exclude each of available destination from their tour plan before obtaining a good tour plan. A large amount of tourism data leads to more time used for tour planning process. Consumer's tour

planning process could take several days in which they would like to cover several attraction that would resulting in a high satisfaction. In this study, an approach to efficiently plan a tour for consumer would be studied. The objective of this study is to obtain a tour planning approach that efficiently create a tour plan with high overall consumer's tour satisfaction and also satisfy general tour planning condition within a short period of time. A complete tour in this study is defined as a tour which consists of a D trip which connect initial and final location. Each trip must have start location, tourist attraction, restaurant for lunch and dinner, and end location. And the tour have a time and budget constraint as a limits. According to the above condition, in this study, tour planning process/ tour trip design problem (TTDP) could be seen as Orienteering Problem with Hotel and Restaurant Selection and Time Window (OPHRSTW). Since its shared the following similarities: 1. Objective of an orienteering is to maximize the score which tour planning process also aim to visit a place that have a high consumer's satisfaction which could be measured by review score from past consumer experiences on the internet, 2. Objective value could only be increase by visiting as much place as possible, 3. There is a time limit in each trip. Based on the similarity described above, the real world tour planning problem could be converted to a mathematical model for computer to solve this kind of problem, which called "Optimization approach", to the optimal point (the best solution).

The Orienteering Problem with Hotel and Restaurant Selection and Time Window (OPHRSTW) is a new extension of the Orienteering Problem (OP). In this study, OPHRSTW is applied to tour planning related problem, where a tourists want to select a hotel, restaurant for dinner and lunch, and attraction for their tour to maximize their enjoyment. Length of the tour can be several days. Tourists cannot visit all the available location since they have a limited amount of time and budget and have to return to the initial location that they start their travel. The term "trip" refer to a sequence of location that have hotel/depot as a start and end location, while the term "tour" refer to a sequence of trip that connect initial location with final location.

The OP was proved to be NP-hard by Golden, Levy, & Vohra (1987). Since the OP is NP-hard, it extensions such as OPHRSTW will also be NP-hard. The term

NP-hard mean that to solve the problem to optimality is very time consuming. Thus, heuristic approach would also be studied along with optimization approach to compare performances of both approach.

1.2 Statement of the problem

Due to the NP-hard by nature of OP and it relatives problem, so when the data getting bigger and bigger the optimization approach would solve the problem slower and slower. Because of slow in performance of optimization approach, it is not practical for this approach to be used as a tour planner in real world. Thus, a heuristic approach, which tent to have better performance in terms of runtime, should be studied to explore a heuristic that could efficiently solve OPHRSTW. In this study, a heuristic approach that would be studied is called “memetic algorithm”

1.3 Objectives

The purpose of this study is to develop a Memetic Algorithm (MA) that can solve OPHRSTW for an appropriate solution within a reasonable time. In the MA development process, the following algorithm will be used:

1) Genetics algorithm: Using one point crossover and randomized mutation.

2) Local search algorithm: Variable Neighborhood Descent (VND) with seven local search heuristics.

CHAPTER 2
REVIEW OF LITERATURE

2.1 Orienteering problem and it extensions

Sport game named Orienteering is the origin of the Orienteering Problem (OP) (Chao, Golden, & Wasil, 1996). The objective of this sport game is to maximize a total collected score that obtained from visiting each check points and come back to starting point within a time limit. Divsalar, Vansteenwegen, & Cattrysse (2013) considered it as a combination between the Knapsack Problem and the Travelling Salesperson Problem (TSP), and also known as the Selective Traveling Salesperson Problem (Gendreau, Laporte, & Semet, 1998; Laporte & Martello, 1990) and Maximum Collection Problem (Butt & Cavalier, 1994; Kataoka & Morito, 1988).

There are many extension of the OP such as the OP with Time Window (OPTW), The Team OP (TOP), the generalized OP (GOP), the OP with Hotel Selection (OPHS), and the OP with Mandatory visits and Exclusionary Constraints (OPMVEC). For a recent survey, see (Gunawan, Lau, & Vansteenwegen, 2016; Lu, Benlic, & Wu, 2018; Vansteenwegen, Souffriau, & Oudheusden, 2011). In this study, only OPHS and OPTW will be discussed further in **Section 2.2**. Due NP-hard nature of the OP and it extensions, many study have develop a heuristic algorithm to solve them. For example, Tabu Search (TS), Greedy heuristics, Variable Neighborhood Search (NVS), Skewed Variable Neighborhood Search (SVNS), Simulated Annealing (SA), Artificial Neural Network (ANN), Ant Colony Optimization (ACO), and Memetic Algorithm (MA). See (Divsalar et al., 2013; Divsalar, Vansteenwegen, Sörensen, & Cattrysse, 2014; Kobeaga, Merino, & Lozano, 2018; Lu et al., 2018) for a recent survey.

2.2 Orienteering problem with hotel and restaurant selection and time window

As OPHRSTW is one of a variant of OP, so it shares some similarities and have some differences with several other extension of OP. The OPHRSTW can be seen as a combination of OPHS, OPTW, and Restaurant Selection condition. In the OPHS (Divsalar et al., 2013) a set of hotel and attraction are given with assigned score but in this study, the score would also assigned to each hotel. The tour consist

of D trip which start and end at one of the hotel. Each hotel can be visited more than once. Each attraction can be visited only once. The time t_{ij} required to travel from location i to j is given for all pairs of location. The objective is to maximize a total collected score from all D trip in the tour, while OPTW have the same objective and almost the same constraint as OPHS. Except that OPTW have just one trip in the tour and each trip must be start and end at the same location. To visit each location in OPTW, the time window must be check whether arrive time fall within a given time window (Kantor et al., 1992).

Based on the above section, the main obvious difference between OPHRSTW and OPHS is that Time Window is not considered in OPHS and there is no consideration about node of type restaurant. For OPHRSTW and OPOTW, there are three obvious difference between OPHRSTW and OPTW where the first difference is that multiple trip is not considered in OPTW. The second difference is that each trip have to start and end at the same location and the last one is there is no consideration about restaurant node in the problem. Thus, OPHRSTW is a research that would be a combination of two existing research fields and also have more node type to be considered than those existing problems.

2.3 Other related research Main heading

Many studies propose method and algorithm to solve OP and its variant in field of tour planning to create a tour planning that could support tourism decisions which could leads to success tourism. Yan (2014), create a mathematical model for tour planning process which result produced by the model show priority of the tourist attraction site. Information of available tourism route played an important role in the tour planning process when choosing which tourist attraction to be included to the tour. Liao & Zheng (2018), create a model to solve tour planning problem with time dependent stochastic environment where historical data played an important roles to the distribution of TDSVs to make the algorithm create a realistic solutions. They used a combination of genetic algorithm (GA) and difference evolution algorithm (DEA) to solve the problem. The result generated by this algorithm could provide an effective tour plan that could satisfied the stochastic condition of the problem. Similarly, Nedjati (2017) used one variant of GA algorithm, which is non-

dominated sorting genetic algorithm II (NSGAII), to deal with bi-objective covering tour routing problem. Result of the study shows that NSGAII could solve the bi objective problem within a short-period of time while also provide excellence result in terms of tour plan quality. Xiao (2017), used a neural net buffer algorithm to deal with tour planning problem in Zhengzhou, China. Popular attractions classified into number of tourism attractions, sight-seeing locations, and the sightseeing time limit to generate a multi-perceptron (MP) nerve cell route planning model. The result from this model was feasible and valuable and can provide effective decision support for tourists. Zhu (2012), create an algorithm that could solve a tour trip design problem that included an uncertainty of time and environment into account, such as traveling times or waiting times. The algorithm of this study was able to find a good approximate solutions in a very short time. An overview of related research of TTDP are shown in below **Table 2.1**.

Table 2.1 List of related studies

Author	Topic	Hotel	Restaurant	Multi-Day	Time Window
Yan et al. (2014)	Tour Route Multiobjective Optimization Design Based on the Tourist Satisfaction	-	-	-	-
Zhu et al. (2012)	On the tour planning problem	X	-	X	X
Xiao et al. (2017)	Tourism Route Decision Support Based on Neural Net Buffer Analysis	X	X	-	-
Wu (2017)	A tour route planning model for tourism experience utility maximization	X	-	-	X
Liao & Zheng et al. (2018)	Using a heuristic algorithm to design a personalized day tour route in a time-dependent stochastic environment	-	-	-	X
Nedjati et al. (2017)	Bi-objective covering tour location routing problem with replenishment at intermediate depots: Formulation and meta-heuristics	X	-	X	X

CHAPTER 3
MATHEMATICAL MODEL DEVELOPMENT

In this chapter, an information about mathematical model for OPHRSTW such as assumption and constraints toward the problem, model equation, and notation will be explained. Since tour planning process is hard and time consuming for tour agency or tourist to select and planned their trip to get the highest satisfaction score, so it is more efficient to transform this problem into mathematical model to let the computer solve and find solution for this kind of problem. The tour planning process could be seen as "Orienteering Problem with Hotel and Restaurants Selection with Time Window" (OPHRSTW).

In the OPHRSTW, there is a given set of all location $V = H \cup Att \cup RL \cup RD$ where H is a set of H+1 (i = 0,…,H) hotels, Att is a set of Att+1 (i = 0,…,Att) attractions, RL is a set of RL+1 (i = 0,…,RL) restaurants for lunch, and RD is a set of RD+1 (i = 0,…,RD) restaurants for Dinner. All location in V are assigned a score S_i, service/entrance fee EC_i, recommended amount of time to spend TS_i, opening time Opt_i, and closing time Clt_i. For all pairs of the locations, travelling time TrT_{ij} and cost TrC_{ij} required to travel from location i to j are given. In each trip d = 1,…,D, hotel must be starting and ending location except the first and the last trip that start in initial location and end in final location, respectively. The tour with D trip is limited to a given monetary budget Budget. The primary objective of the problem is to maximize total collected scores, while the secondary objective is to minimize the total cost of a tour to ensure that it doesn't exceed the available budget.

3.1 Objective

(1) Maximize a tour total score by selecting a combination of hotels, restaurants, and attractions.

3.2 Assumptions and Constraints

(1) Initial and final location of the tour must be fixed

(2) Each trip must start and end with hotel except initial and final location of the tour that will start and end at a specific location

(3) All attraction and restaurants can be visited only once

(4) Each hotel can be visited more than once

(5) Breakfast assumed to take place at the hotel

(6) There must be only 1 Restaurant for Lunch and Dinner in each trip

(7) Lunch time assumed to be 10:30 AM – 1:00 PM

(8) Dinner time assumed to be 4:00 PM – 6:00 PM

(8) Each location can be visited if the arrive time + time spend at the location less than its closing time (waiting time are allowed)

(9) Once each location is visited, it's associated score and cost would be included in total score and total cost, respectively.

(10) Budget used must be less than "B"

3.3 Notation

(1) Indices:

H = Index for Hotels

Att = Index for attractions

RL = Index for Restaurants (For Lunch)

RD = Index for Restaurants (For Dinner)

AnR = Set of attractions and restaurants (AnR = Att ∪ RL ∪ RD)

AnH = Set of attraction and hotel

AnL = Set of attraction and lunch

AnD = Set of attraction and dinner

V = Set of all locations (V = H ∪ Att ∪ RL ∪ RD)

d = range of day available (d = 1,…,D)

(2) Parameters:

D = Number of day available

B = Available budget for the whole tour

TS_i = recommended amount of time to spend at location i

OpT_i = Time that location i is open

ClT_i = Time that location i is close

EC_i = service/entrance fee at location i

S_i = assigned a score of location i

TrT_{ij} = Travelling time between location i and j

M = A large value (Big M)

(3) Decision Variables:

X_{ijd} = 1 if there is a travel from location i to j in day d,
0 otherwise

Y_{jd} = 1 if there is a travel from any location to attractions day d,
0 otherwise

Z_{jd} = 1 if there is a travel from any location in set AnR to Hotel in day d,
0 otherwise

LU_{jd} = 1 if there is a travel from any location in set AnR to Lunch day d,
0 otherwise

DI_{jd} = 1 if there is a travel from any location in set AnR to Dinner day d,
0 otherwise

$ArrT_{id}$ = Arrival time at each location i in set AnR of day d,

$ArrTW_{id}$ = Arrival tine at each hotel I in set H of day d,

3.4 Model

Objective:

Maximize: $\sum_{d=1}^{D}\sum_{j=0}^{Att} SjYjd + \sum_{d=1}^{D}\sum_{j=0}^{H} SjZjd + \sum_{d=1}^{D}\sum_{j=0}^{RL} SjLUjd + \sum_{d=1}^{D}\sum_{j=0}^{RD} SjDIjk$ (1)

Subjected to:

$\sum_{j=0}^{AnL} x_{1j1} = 1$ d = 1,…,D (2)

$\sum_{i=0}^{AnD} x_{i2D} = 1$ d = 1,…,D (3)

$\sum_{j=0}^{V}\sum_{i=0}^{H} X_{ijd} = 1$ d = 1,…,D (4)

$\sum_{i=0}^{V}\sum_{j=0}^{H} X_{ijd} = 1$ d = 1,…,D (5)

$\sum_{j=0,j\neq i}^{V} X_{ijd} = \sum_{j=0,i\neq j}^{V} X_{jid}$ d = 1,…,D-1, i = 1,…,AnR (6)

$$\sum_{i=0,i\neq j}^{V}\sum_{d=1}^{D} X_{ijd} \leq 1 \qquad j = 1,\ldots,AnR \qquad (7)$$

$$Y_{jd} = \sum_{i=0,i\neq j}^{V} X_{ijd} \qquad j = 1,\ldots,Att,\ d = 1,\ldots,D \qquad (8)$$

$$Z_{jd} = \sum_{i=0}^{AnD} X_{ijd} \qquad j = 1,\ldots,H,\ d = 1,\ldots,D \qquad (9)$$

$$\sum_{j=0}^{V} X_{jid} = \sum_{j=0}^{V} X_{ijd+1} \qquad i = 0,\ldots,H,\ d = 1,\ldots,D\text{-}1 \qquad (10)$$

$$LU_{jd} = \sum_{i=0}^{AnH} X_{ijd} \qquad j = 0,\ldots,RL,\ d = 1,\ldots,D \qquad (11)$$

$$\sum_{i=0}^{V}\sum_{j=0}^{RL} X_{ijd} = 1 \qquad d = 1,\ldots,D \qquad (12)$$

$$DI_{jd} = \sum_{i=0}^{AnL} X_{ijd} \qquad j = 0,\ldots,RD,\ d = 1,\ldots,D \qquad (13)$$

$$\sum_{i=0}^{V}\sum_{j=0}^{RD} X_{ijd} = 1 \qquad d = 1,\ldots,D \qquad (14)$$

$$ArrT_{id} - ArrT_{jd} + TrT_{ij} + TS_i \leq (1\text{-}X_{ijd})M \qquad i = 0,\ldots, V;\ j= 0,\ldots,AnR;\ d = 1,\ldots,D \qquad (15)$$

$$ArrT_{id} - ArrTW_{jd} + TrT_{ij} + TS_i \leq (1\text{-}X_{ijd})M \qquad i = 0,\ldots, AnR;\ j= 0,\ldots,H;\ d = 1,\ldots,D \qquad (16)$$

$$OpT_i \leq ArrT_{id} \leq ClT_i \qquad i = 0,\ldots,V,\ d = 1,\ldots,D \qquad (17)$$

$$OpT_i \leq ArrTW_{id} \leq ClT_i \qquad i = 0,\ldots,H,\ d = 1,\ldots,D \qquad (18)$$

$$\sum_{d=1}^{D}\sum_{j=0}^{Att} EC_j Y_{jd} + \sum_{d=1}^{D}\sum_{j=0}^{H} EC_j Z_{jd} + \sum_{d=1}^{D}\sum_{j=0}^{Rl} EC_j LU_{jd} + \sum_{d=1}^{D}\sum_{j=0}^{RD} EC_i DI_{jd} \leq B \qquad (19)$$

The objective (1) maximize total customer satisfaction score. Constraint (2) and (3) guarantee that the tour start and end at desired location. Constraint (4) and (5) ensure that each trip start and end in one of the available hotels. Constraint (6) ensure connectivity of the path. Constraint (7) ensure that attraction and restaurant can be visited at most once. Constraint (8) use for attraction score calculation. Constraint

(9) use for hotel score calculation. Constraint (10) ensure that the first trip and the second trip will end and start at the same hotel, respectively. Constraint (11) use to calculate score of restaurants for lunch. Constraint (12) ensure that each trip will have one and only one restaurant for lunch. Constraint (13) uses for restaurant for dinner score calculation. Constraint (14) ensures that each trip will have one and only one restaurant for dinner. Constraint (15) ensures timeline of the path for attraction and restaurant in each trip. Constraint (16) ensures timeline of the path for ending location/hotel in each trip. Constraint (17) limits that each attraction and restaurant must visited within time window. Constraint (18) limit that each ending location/hotel must visited within time window. The reason to separate constraint 17 and 18 is to allow the same hotel to be selected since combining these two constraints together would resulting in a constraint unbounded issue when number of day is more than number of available hotel in the list. Constraint (19) limit total cost of the whole tour to be less than available budget.

CHAPTER 4
PROPOSED ALGORITHM

4.1 Structure of Memetic Algorithm for OPHRSTW

Moscato (1999) was the first to introduced MA and defined it as a combination of genetic algorithm and local search technique such that have more capability to explore and exploit promising region of the search space. It was successfully applied to solve VRPs by Prins (2004). For the OP, MA was first applied by (Bouly, Dang, & Moukrim, 2010). Many study successfully applied MA to solve OP extension and related VRP such as A Memetic Algorithm for the Orienteering Problem with Hotel Selection (Divsalar et al., 2014), The Memetic algorithm for the optimization of urban transit network (Zhao, Xu, & Jiang, 2015), and A Memetic Algorithm for Orienteering Problem with Mandatory Visits and Exclusionary Constraints (Lu et al., 2018). The structure of MA for OPHRSTW compose of two parts: The first part, Population initialization which start the process by preprocessing step that calculate a score for each pair of hotel then used it to create initial population. After initial population is created, it will be processed through the second part called Main-loop. In Main-loop, there are two genetic operators named crossover and mutation that are used to increase population in the Solution Pool (*Pool*). This process will repeated until it reach Maximum iteration (*MaxIteration*). See **Figure 4.1** below for more detailed MA structure description.

1. **Population Initialization:**
   1.1. Pre-Processing Step (*Alg. 2*)
   1.2. Creating initial population (*Alg. 3*)
2. **Main-loop:**
   Iter = 1

   While (*Iter* <= *MaxIteration*) **Do:**

      1. Populate the *Pool (Alg. 4)*

      2. Sort the *Pool* according to solution quality

      3. Save the *Best-Found-Solution*

      4. Population management *(Alg. 5)*

      Iter += 1

   End

Output: *Best-Found-Solution*

Figure 4.1. Structure of Memetic Algorithm (*Alg. 1*)

4.2 Population Initialization

In population initialization, the estimated score for each pair of hotel is created in preprocessing step. After that it is used to generate the initial population. Note that the term "population" in this study refers to a tour which can consist of one trip or multiple trip. A detailed algorithm of the population initialization part and tour characteristics was described on **Figure 4.2** and **Figure 4.22**.

4.2.1 Tour characteristics and conditions:

1. Tour characteristics
   1.1. There must contains all type of the node in each trip of the tour
      1.1.1. There are 4 types of node as follows:
         1.1.1.1. Hotel/Depo node
         1.1.1.2. Tourist attraction node
         1.1.1.3. Restaurant for lunch node
         1.1.1.4. Restaurant for dinner node

1.2. Each trip must be start at the specified time and end less than or equals to specified time. In this study, the start time of each trip was set at 8 AM and the end time of each trip was set at 8 PM or 20

1.3. All of attraction/restaurant (lunch/dinner) node could be visited at most once

1.4. Each trip in tour would have the following value recorded

1.4.1. List of node in trip

1.4.2. List of leave time at each node (LT)

1.4.2.1. Leave time of each target node was calculated by using the following formula: *LT of previous adjacent node + traveling time from previous adjacent node to target node + time spent at target node*

1.4.3. List of compensation time (CoT)

1.4.3.1. Before the calculation occur, the comparison between LT and opening time of target node (OpT) should be made. If the LT less than the OpT then the CoT would be calculated. On the hand, if LT larger or equals to OpT then CoT equals to zero.

1.4.4. List of node position ratio (NPR)

1.4.4.1. Node position ratio (NPR) is the value to indicate the worthiness of visiting each target node (each node would have they own NPR value)

1.4.4.2. Node position ratio of each target node was calculated by using the following formula: *score of target node* divided by [*(traveling time from previous adjacent node to target node) + (traveling time from target node to next adjacent node) + (compensation time at target node) + Time Spent at each attraction node*]

2. There are 5 conditions to be checked for feasibility of the tour:
    2.1. Start/End node must be hotel/depo node
    2.2. There must be one and only one node of type lunch and dinner in each trip
    2.3. No duplication of attraction/restaurant node across the tour
    2.4. All type of node must be visited within specified time window (opening time (OpT) and closing time (ClT))
        2.4.1. For OpT, if the leave time at target node (LT) is less than OpT then the waiting time or called CoT (in this study) would be calculated, else CoT equals to zero.
        2.4.2. For ClT, LT would be compared with ClT. If LT less than or equals to ClT, a target node would be included to the trip. Else it would be ignored.
    2.5. Total cost of a tour must not exceed specified budget

Figure 4.2. shows an example of a one-day tour, which have only one trip in the tour. The trip is a complete trip that contains and satisfied tour characteristics and conditions as follows:

1. This trip has hotel as a start and end node
2. It have one and only lunch node in the trip
3. It have one and only dinner node in the trip
4. All time window conditions of all node is not violated

If there are the case where one tour contained more than one trip, the same characteristics and conditions must be satisfied.

Description	Trip Information							
Node in Trip	Hotel1	Att1	Att2	Lunch	Att3	Att4	Dinner	Hotel2
Time used (TU)	8	9.141389	10.24694	11.33694	12.33778	14.30889	16.27778	17.45083
Compensation Time (CoT)	0	0	0	0	0	0	0	0
Node Position Ratio (NPR)	0	15.95501	20.14773	33.02752	3.312947	1.860825	1.400597	0

* Att = Tourist Attraction

Figure 4.2 Example of expected result for a one-day tour planning problem

4.2.2 Matrices of pairs of hotels

The estimated score for each “trip” or called “pairs of hotels” is calculated by adapting greedy sub-OP heuristics proposed by (Divsalar et al., 2014). In this heuristic, each trip is solved independently such that attraction could be visited more than once in another trip. Each trip start and end in hotel, if a particular trip is not the first trip a start hotel must be the same as an end hotel of the previous trip. A possible end hotel for each trip are defined by the following constraints: 1. Opening time <= Leave time <= Closing time, 2. Total Cost + Cost of hotel <= Budget

There are four heuristics in greedy-sub OP algorithm which are Insert and Add, Replacement, Two-Opt, Move-best. These heuristic will apply differently to restaurant and attraction, and it would be running in a sequence as follows: Insert and Add, Replacement, Two-Opt, and Move-best. E.g. Insert and Add would be executed to formulate the initial trip. Then the initial trip (solution) would be sent to Replacement algorithm to improve the trip quality. If there are any improvement in trip quality it would be sent back to Insert and Add to check whether there are any possible insertion. But if there is no any improvement then the trip would be sent from Replacement to Two-Opt algorithm to improve its quality. Then after Two-Opt is executed the same condition as Replacement would be checked to see whether there are any improvement in solution quality. The same procedure would also goes to Move-best algorithm. Then after all of the algorithm are unable to improve quality of the trip then it would record that trip in a matrix according to

start and end hotel of the trip. The flowchart of the algorithm is show on below Figure 4.3.

Pre-Processing Step: (*Alg. 2*)
Iter = 1
MaxScore = 0
MaxNoImprovement = 4
While (*Iter* <= *MaxNoImprovement*) **Do:**
1. **If** *Iter=1* **Do:**
Insert and Add Algorithm
If *CurrentScoreFromAlgorithm* > *MaxScore* **Do:**
Iter=1
Else:
Iter=Iter+1
2. **If** *Iter=2* **Do:**
Replacement Algorithm
If *CurrentScoreFromAlgorithm* > *MaxScore* **Do:**
Iter=1
Else:
Iter=Iter+1
3. **If** *Iter=3* **Do:**
Two-Opt Algorithm
If *CurrentTripTime* < *OriginalTripTime* **Do:**
OriginalTripTime = *CurrentTripTime*
Iter=1
Else:
Iter=Iter+1
4. **If** *Iter*=**4 Do:**
Move-Best Algorithm
If *CurrentTripTime* < *OriginalTripTime* **Do:**
OriginalTripTime = *CurrentTripTime*
Iter=1
Else:
Iter=Iter+1
End
Output: *Best-Found-Solution*

Figure 4.3 Logic structure of pre-processing step

These four heuristics algorithm would process again and again until there is no improvement occurred. Below section would describe each of these four heuristics in details.

4.2.2.1 Insert and Add (a):

```
Insert and Add (a):
    BestTrip = OriginalTrip
     NoImprove = 0
     Lunch = 0
     Dinner = 0
     While (NoImprove < 2, Lunch < 1, Dinner < 1) Do:
            If 10.5 (10:30 AM) <= TW of last visited node <= 13 (1:00 PM) and Lunch < 1 Do:
                MaxRatio = 0
                For each lunch node:
                        Calculate TW of lunch node
                        If Opt (Lunch) <= TW of lunch node <=ClT (Lunch) and TotalCost + cost associated with each lunch
                        node <= MaxBudget Do:
                                Calculate NPR of each lunch node
                                If NPR > MaxRatio Do:
                                        If TW of all node in the trip is not violated Do:
                                                Insert lunch node to TemporaryTrip
                                                MaxRatio = NPR
                End
                If MaxRatio <> 0:
                        BestTrip = TemporaryTrip
                Else:
                        NoImprove = NoImprove + 1
            Else If 15.5 (3:30 PM) <= TW of last visited node <= 18 (6:00 PM) and Dinner < 1 Do:
                MaxRatio = 0
                For each dinner node:
                        Calculate TW of dinner node
                        If Opt (Dinner) <= TW of Dinner node <=ClT (Dinner) and TotalCost + cost associated with each
                        Dinner node <= MaxBudget Do:
                                Calculate NPR of each lunch node
                                If NPR > MaxRatio Do:
                                        If TW of all node in the trip is not violated Do:
                                                Insert dinner node to TemporaryTrip
                                                MaxRatio = NPR
                End
                If MaxRatio <> 0:
                        BestTrip = TemporaryTrip
                Else:
                        NoImprove = NoImprove + 1
            Else:
                For each attraction node:
                        For each position in trip:
                                Calculate TW of attraction node at the position
                                If Opt (attraction) <= TW of attraction node <=ClT (attraction) and TotalCost + cost
                                associated with each attraction node <= MaxBudget Do:
                                        Calculate NPR of each attraction node
                                        If NPR > MaxRatio Do:
                                                If TW of all node in the trip is not violated Do:
                                                        Insert attraction node to TemporaryTrip
                                                        MaxRatio = NPR
                        End
                End
                If MaxRatio <> 0:
                        BestTrip = TemporaryTrip
                Else:
                        NoImprove = NoImprove + 1

    End
Output: BestTrip
```

Figure 4.4 Logic structure of Insert and Add (a) algorithm

From Figure 4.4, The different between insert and add is that insert will determine the best position to insert, while add is just add a location into a free slot of the trip. In this study, Insert will be used to determine a position to insert attraction, while add will be used to add restaurant to a slot position. For each non-included attraction in the trip, the best position of insertion that have minimum association cost is determined. Every time that attraction is inserted, leave time will be calculated. Once leave time fall within the range of 10:30 AM – 1:00 PM (lunch time) and 4:00 – 6:00 PM (dinner time) the restaurant will be added. Note that an attraction and restaurant that have the maximum NPR and does not make the trip infeasible is inserted/added. The algorithm will process many iteration until it unable to insert/add a node to the trip.

Iteration	Trip/Leave time	Solution		
0	Trip	Hotel1	Hotel2	
	Leave time	8	20	

Figure 4.5 Example of input given to Insert and Add (a) algorithm

At the iteration zero, there would be only two nodes which are the start and end node (hotel/depo node) as shown on the Figure 4.5. The algorithm would check whether the leave time of the latest node, which is not the last node of the trip, fall within the range of 10:30 AM – 1:00 PM (lunch time) and 3:30 – 6:00 PM (dinner time). As shown on Figure 4.5 leave time of the latest node is 8:00 AM, so the algorithm would skip lunch and dinner inclusion process then start attraction insertion process. The attraction insertion process is to include an unvisited attraction node to the trip, using the following set of criteria to select the best attraction node to be inserted:

1. Target attraction node must have the highest NPR and if there are two attraction node that have the same NPR value, the first found node that would be selected
2. Leave time of target node must fall within time window
3. Total cost plus cost of target node must be less than specified budget
4. Time window of other node in the trip must feasible after including target node to the trip.

5. The leave time/arrival time of Hotel2 node in each trip must be less than or equals to a specified value which in this study used 20 or 8 P.M.

After the algorithm check all of the criteria for each target attraction node, then the NPR in iteration zero for each target node would be calculated and the result are show on Figure 4.6.

Attraction Node	Opening Time	Leave Time	Closing Time	Node Position Ratio
Att1	0	11.72444444	99999	0.650966473
Att2	9	10.79416667	19	1.003821656
Att3	7	9.521388889	12	1.754342813
Att4	8.5	9.42	17	3.038560411
Att5	7	18.35222222	18	N/A
Att6	10	11.28888889	17	0.713046688
Att7	9.5	9.937222222	18	3.221439927
Att8	8.5	11.30888889	16.5	0.981200453

Figure 4.6 Example of how algorithm select node to be included to the trip in iteration 1 of Insert and Add (a) algorithm

From Figure 4.6, it is show that all of the unvisited attraction node were pass the time window criteria except Att5, so NPR for Att5 would be and N/A because the algorithm will not calculate NPR if time window criteria is not satisfy. For the node that have NPR value calculated, only one node would be selected in each iteration based on their NPR. In this case Att7 node would be selected because it have the highest NPR value. The result of iteration 1 after algorithm selected Att7 node show on below Figure 4.7. From Figure 4.7, it is show that leave time of the latest node in the trip still not reach lunch/dinner time. The algorithm would try to insert another attraction to the trip. In iteration 2, there are 2 positions for algorithm to be considered to insert the attraction node to the trip as follows: 1. the position between Hotel1 and Att7, 2. the position between Att7 and Hotel2. The algorithm would calculated the NPR of each unvisited attraction node at each target position (1 and 2 in this case). The result of the algorithm calculation are show on below Figure 4.8.

Iteration	Trip/Leave time	Solution			
0	Trip	0	1		
	Leave time	8	20		
1	Trip	0	9	1	
	Leave time	8	9.937222	10.22306	

Position 1 Position 2

Figure 4.7 Example of output after 1st iteration of Insert and Add (a) algorithm

Position	Attraction Node	Opening Time	Leave Time	Closing Time	Node Position Ratio
1	Att1	0	11.72444444	99999	0.678396481
2	Att1	0	13.80861111	99999	0.634858266
1	Att2	9	10.79416667	19	1.067509596
2	Att2	9	12.87833333	19	0.967596698
1	Att3	7	9.521388889	12	2.015542211
2	Att3	7	11.91722222	12	1.399215302
1	Att4	8.5	9.42	17	3.707266074
2	Att4	8.5	11.56138889	17	2.625208218
1	Att5	7	18.35222222	18	N/A
2	Att5	7	21.14	18	N/A
1	Att6	9	9.418611111	17	3.361137441
2	Att6	9	11.64444444	17	2.324483776
1	Att8	8.5	11.30888889	16.5	0.934291876
2	Att8	8.5	12.40833333	16.5	1.270505426

Figure 4.8 Example of how algorithm select node to be included to the trip in iteration 2 of Insert and Add (a) algorithm

From Figure 4.8, the algorithm would select Att4 node and placed it at position 1 (position between Hotel1 and Att7). The result of iteration 2 show on below Figure 4.9. As show on Figure 4.9, leave time of the latest node is fall within the range 10:30 AM – 1:00 PM (lunch time) and also not exceed time limit of 20. So the algorithm in iteration 3 would switch logic from attraction node insertion to lunch node appending. In lunch node appending logic, the algorithm would calculate the NPR of each lunch node at the position between the first and the second node from the bottom of the trip. In this case the target position is position between Att7 and Hotel2 node. After algorithm calculated NPR for each lunch node then lunch node with the highest NPR value would be selected to be appended to the trip. However, if there are multiple lunch node that have the same NPR value, only the first found lunch node that would be selected. The result of the 3rd iteration show on below Figure 4.10.

Iteration	Trip/Leave time	Solution							
0	Trip	Hotel1	Hotel2						
	Leave time	8	20						
1	Trip	Hotel1	Att7	Hotel2					
	Leave time	8	9.937222	10.22306					
2	Trip	Hotel1	Att4	Att7	Hotel2				
	Leave time	8	9.42	11.06278	11.34861				

Figure 4.9 Example of output after 2nd iteration of Insert and Add (a) algorithm

Iteration	Trip/Leave time	Solution							
0	Trip	Hotel1	Hotel2						
	Leave time	8	20						
1	Trip	Hotel1	Att7	Hotel2					
	Leave time	8	9.937222	10.22306					
2	Trip	Hotel1	Att4	Att7	Hotel2				
	Leave time	8	9.42	11.06278	11.34861				
3	Trip	Hotel1	Att4	Att7	Lunch	Hotel2			
	Leave time	8	9.42	11.06278	12.435	13.11917			

Figure 4.10 Example of newly created trip with restaurant adding logic in 3rd iteration of Insert and Add (a) algorithm

The same algorithm would apply in the case of how to append dinner node to the trip but the only different is only the range of leave time which lunch node have 10:30 AM – 1:00 PM as an algorithm ticker, while dinner node have 4:00 – 6:00 PM as an algorithm ticker. To create a complete trip, the process would continue to over and over again until leave time/arrival time of Hotel2 node reach the time limit per day of 20 or 8 PM. Below Figure 4.11 is an example of a complete trip that generated by Insert and Add algorithm.

Iteration	Trip/Leave time	Solution							
0	Trip	Hotel1	Hotel2						
	Leave time	8	20						
1	Trip	Hotel1	Att7	Hotel2					
	Leave time	8	9.937222	10.22306					
2	Trip	Hotel1	Att4	Att7	Hotel2				
	Leave time	8	9.42	11.06278	11.34861				
3	Trip	Hotel1	Att4	Att7	Lunch	Hotel2			
	Leave time	8	9.42	11.06278	12.435	13.11917			
4	Trip	Hotel1	Att4	Att7	Lunch	Att6	Hotel2		
	Leave time	8	9.42	11.06278	12.435	13.79194	14.77972		
5	Trip	Hotel1	Att4	Att7	Lunch	Att6	Att2	Hotel2	
	Leave time	8	9.42	11.06278	12.435	13.79194	16.53056	18.66139	
6	Trip	Hotel1	Att4	Att7	Lunch	Att6	Att2	Dinner	Hotel2
	Leave time	8	9.42	11.06278	12.435	13.79194	16.53056	18.01556	19.8375

Figure 4.11 Example of complete trip generated by Insert and Add (a) algorithm

After complete trip was created, it would be used as an input for the later algorithm for solution improvement process before it would be used in population creation and main-loop process.

4.2.2.2 Replacement:

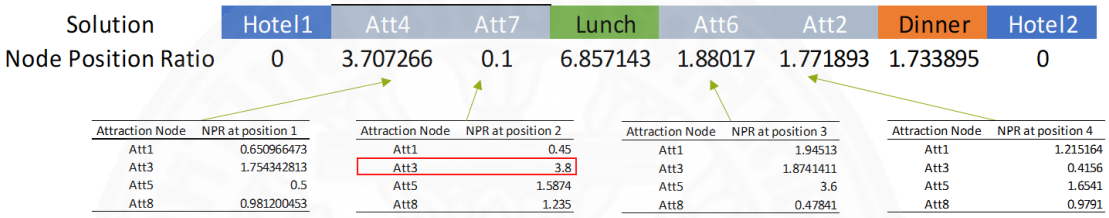
```
Replacement:
    BestTrip = OriginalTrip
    MaxRatio = 0
    For each non-included attraction node:
        For each position in the trip:
            Calculate TW of attraction node at the position
            If Opt (attraction) <= TW of attraction node <=ClT (attraction) and
            TotalCost + cost associated with each attraction node <= MaxBudget Do:
                Calculate NPR of each attraction node
                If NPR > MaxRatio Do:
                    If TW of all node in the trip is not violated Do:
                        Insert attraction node to TemporaryTrip
                        MaxRatio = NPR
        End
    End
    If MaxDiff <> 0 Do:
        BestTrip = TemporaryTrip
Output: BestTrip
```

Figure 4.12 Logic structure of Replacement algorithm

For all non-included attraction, the NPR of each position in the trip is calculated to check whether there are any position that non-included attraction node have a larger NPR than the attraction node in the trip. If the algorithm found that there are at least one non-included attraction node have larger NPR value than attraction node in the trip (could be in any position), then it is to be checked whether replacing a target non-included attraction node to the trip would make the trip feasible. If replacement still leads to feasible solution then the replacement would be executed, otherwise no replacement executed. Moreover, if there are more than one non-included attraction that have NPR value larger than an included attraction node then the non-included attraction node with the largest NPR value would be

selected in replacement process. Figure 4.12 shows flow of the algorithm in details and a detailed description of Replacement algorithm is provided.

As shows on Figure 4.13, after NPR for each non-included attraction node at each position from original trip were calculated then Att3 which is a non-included attraction node with NPR value larger than original included attraction node and have the largest NPR value among non-included attraction node would be selected to replace with Att7 at position 2 in the trip. Below Figure 4.14, show the result of the algorithm when node Att3 was selected to be replace with Att7.



Solution	Hotel1	Att4	Att7	Lunch	Att6	Att2	Dinner	Hotel2
Node Position Ratio	0	3.707266	0.1	6.857143	1.88017	1.771893	1.733895	0

Attraction Node	NPR at position 1
Att1	0.650966473
Att3	1.754342813
Att5	0.5
Att8	0.981200453

Attraction Node	NPR at position 2
Att1	0.45
Att3	3.8
Att5	1.5874
Att8	1.235

Attraction Node	NPR at position 3
Att1	1.94513
Att3	1.8741411
Att5	3.6
Att8	0.47841

Attraction Node	NPR at position 4
Att1	1.215164
Att3	0.4156
Att5	1.6541
Att8	0.9791

Figure 4.13 Example of how algorithm select node to be included to the trip in each iteration of Replacement algorithm

Solution	Hotel1	Att4	Att3	Lunch	Att6	Att2	Dinner	Hotel2
Node Position Ratio	0	3.927	3.8	7.12654	1.88017	1.771893	1.733895	0

Figure 4.14 Example of output generated by Replacement algorithm

In some to many cases, after replacement is executed the NPR of adjacent node of replaced node would increase because the new node that algorithm replace it in the trip have lower traveling time when compare to the original node. This cases also demonstrated on Figure 4.13 and Figure 4.14 that when Att4 node and Lunch node were connected to Att7 node it NPR value are 3.707266 and 6.857143, respectively. But when the algorithm replaced Att7 node with Att3 node the NPR of these 2 adjacent node were increase to 3.927 and 7.12654, respectively. This mean that the efficiency of the trip time usage is increase.

4.2.2.3 Two-Opt (a):

This heuristic determined the inversion in position of each pair of attraction in the trip that leads to highest overall trip time reduction without making the trip infeasible. On the other hand, if there is no solution that could reduce overall trip time or it leads to an infeasible solution then the original solution is returned. The purpose of this algorithm is to reduce the overall trip time so that it could create more room to insert a non-included attraction node to the trip to increase overall trip score. Below Figure 4.15 shows a logic structure of Two-Opt (a) algorithm.

```
Two-Opt (a):
        BestTrip = OriginalTrip
        MaxDiff = 0
        For each attraction node in trip:
                For each target node in trip:
                        Calculate Difference between old and new sequence of trip
                        If Difference > MaxDiff Do:
                                Calculate TW of attraction node
                                If Opt (attraction) <= TW of attraction node <=ClT (attraction) and
                                TotalCost + cost associated with each attraction node <= MaxBudget Do:
                                        If TW of all node in the trip is not violated Do:
                                                Execute Two-Opt (a) and save it as TemporaryTrip
                                                MaxDiff = Difference
                End
        End
        If MaxDiff <> 0 Do:
                BestTrip = TemporaryTrip
Output: BestTrip
```

Figure 4.15 Logic structure of Two-Opt (a) algorithm

The following formula is used to calculate the different in traveling between original trip sequence and new trip sequence generated by this algorithm by assuming that position of node of interested is i.

1. Original trip sequence overall trip time = [Traveling Time from i-1 to i] + [Traveling time from i to i+1] + [Traveling time from i+1 from the right to i+2]
2. New trip sequence overall trip time = [Traveling Time from i – 1 to i + 1] + [Traveling time from node i + 1 to node i] + [Traveling time from node i to node i + 2]
3. Different = Original trip sequence overall trip time – New trip sequence overall trip time

After different of each pair of attraction node was calculated then the algorithm would select the pair with the highest nonnegative/zero to be the pair for inversion process. Since positive different means that new trip sequence could reduce the overall trip time. Figure 4.16 show the different calculated for each pair of node.

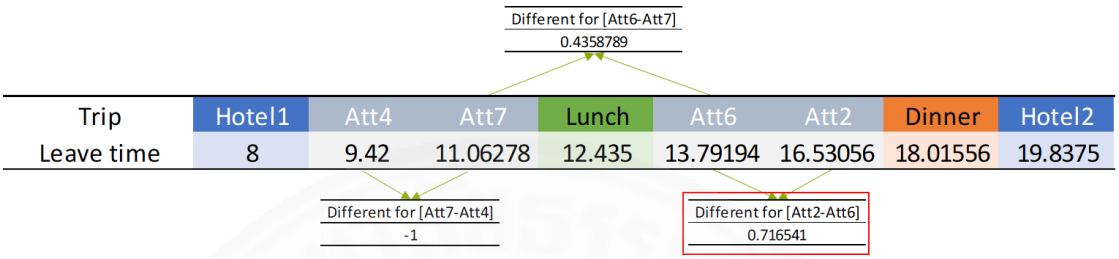


Figure 4.16 Example of how algorithm select node to be included to the trip in each iteration of Two-Opt (a) algorithm

As shows on Figure 4.16, there are 3 pair of attraction node as follows:

1. Pair between Att4 and Att7 with overall trip time after inversion -1
2. Pair between Att7 and Att6 with overall trip time after inversion 0.4358789
3. Pair between Att6 and Att2 with overall trip time after inversion 0.716541

This shows that the inversion of Att6 and Att2 node have the highest nonnegative/zero different with overall trip time saving of 0.716541 hour. Then the algorithm would select this pair of attraction node be processed through node inversion process and the result is shows on below Figure 4.17. As shows on Figure 4.17, the node inversion leads to the saving in leave time through overall trip when compare result from Figure 4.16 to Figure 4.17.

Solution	Hotel1	Att4	Att7	Lunch	Att2	Att6	Dinner	Hotel2
Leave Time	8	9.42	11.06278	12.435	13.39194	16.13056	17.29902	19.12096

Figure 4.17 Example of output generated by Two-Opt (a) algorithm

4.2.2.4 Move-best (a):

```
Move-best (a):
        BestTrip = OriginalTrip
        MaxDiff = 0
        For each attraction node in trip:
                For each target node in trip:
                        Calculate Difference between old and new sequence of trip
                        If Difference > MaxDiff Do:
                                Calculate TW of attraction node
                                If Opt (attraction) <= TW of attraction node <=ClT (attraction) and
                                TotalCost + cost associated with each attraction node <= MaxBudget Do:
                                        If TW of all node in the trip is not violated Do:
                                                Execute Move-best (a) and save it as TemporaryTrip
                                                MaxDiff = Difference
                End
        End
        If MaxDiff <> 0 Do:
                BestTrip = TemporaryTrip
Output: BestTrip
```

Figure 4.18 Logic structure of Move-best (a) algorithm

From Figure 4.18, each attraction would be checked that if move this attraction to another position in the trip will resulting in overall trip time reduction, and feasible solution. If overall trip time is reduced and the solution is feasible then the solution would be selected as candidate, after move-best algorithm is process through all of the attraction node then the solution that have the most time reduction would be selected to be moved to a target position within the same trip to create more room for inserting a non-included attraction node. However, if there is no solution that could reduce overall trip time or leads to an infeasible solution then the original solution would be selected. For each attraction node (i) in the trip when moving it to target position within the trip, the following formula would be used to calculate the "different" or "trip time saving":

4.1. Original trip sequence overall trip time = [Traveling Time from i-1 to i] + [Traveling time from I to i+1] + [Traveling time from j-1 from the right to j]

4.2. New trip sequence overall trip time = [Traveling Time from i-1 to i+1] + [Traveling time from j-1 to i] + [Traveling time from i from the right to j]

4.3. Different = Original trip sequence overall trip time – New trip sequence overall trip time

Where i is node of interested and j is target node to be swapped. The algorithm would calculated different for each included attraction node when moved to each target position in the trip and selected attraction node with the largest nonnegative/zero different to be processed through Move-best process. Figure 4.19 shows the input and result of the different calculation process for each included attraction node when moving to target position within the trip.

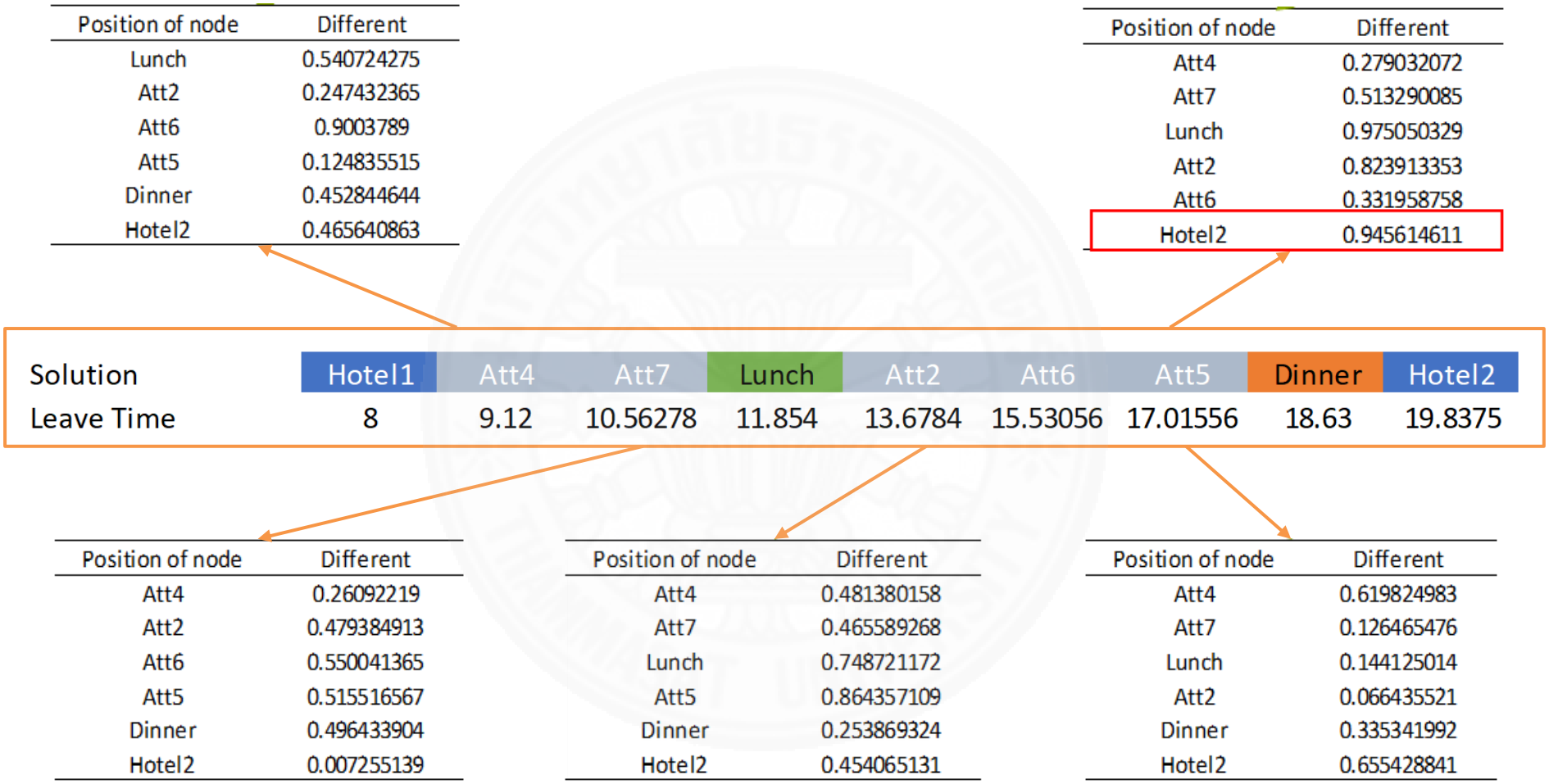


Figure 4.19 Example of how algorithm select node to be included to the trip in each iteration of Move-best (a) algorithm

From Figure 4.19, it is shows that Att5 node have the largest different when moved to position of Dinner node, thus the algorithm would select the Att5 to be moved to position of Dinner node. Below Figure 4.20 show the result of the Move-best algorithm after move Att5 node to the position of Hotel2 node.

Solution	Hotel1	Att4	Att7	Lunch	Att2	Att6	Dinner	Att5	Hotel2
Leave Time	8	9.12	10.56278	11.854	13.6784	15.53056	16.81556	18.21	18.89189

Figure 4.20 Example of output generated by Move-best (a) algorithm

Figure 4.20 shows that the arrival time of Hotel2 node was reduced by 0.945614611 as the Att5 node moved from its own position to position of Hotel2. Also, Dinner node was visited earlier than the original trip sequence. Because Att5 node was moved to the later position. The time that were saved by this algorithm might create enough time slot for insertion of an unvisited attraction node to the trip to increase the overall trip score.

After the preprocessing step is completed matrices of pair of hotel shows on below Figure 4.21 could be could be obtained and would be used as an input in initial population creation steps.

From/To	Hotel1	Hotel2	Hotel3	Hotel4	Hotel5	Hotel6	Hotel7	Hotel8	Hotel9	Hotel10
Hotel1	Hotel1-1	Hotel1-2	Hotel1-3	Hotel1-4	Hotel1-5	Hotel1-6	Hotel1-7	Hotel1-8	Hotel1-9	Hotel1-10
Hotel2	Hotel2-1	Hotel2-2	Hotel2-3	Hotel2-4	Hotel2-5	Hotel2-6	Hotel2-7	Hotel2-8	Hotel2-9	Hotel2-10
Hotel3	Hotel3-1	Hotel3-2	Hotel3-3	Hotel3-4	Hotel3-5	Hotel3-6	Hotel3-7	Hotel3-8	Hotel3-9	Hotel3-10
Hotel4	Hotel4-1	Hotel4-2	Hotel4-3	Hotel4-4	Hotel4-5	Hotel4-6	Hotel4-7	Hotel4-8	Hotel4-9	Hotel4-10
Hotel5	Hotel5-1	Hotel5-2	Hotel5-3	Hotel5-4	Hotel5-5	Hotel5-6	Hotel5-7	Hotel5-8	Hotel5-9	Hotel5-10
Hotel6	Hotel6-1	Hotel6-2	Hotel6-3	Hotel6-4	Hotel6-5	Hotel6-6	Hotel6-7	Hotel6-8	Hotel6-9	Hotel6-10
Hotel7	Hotel7-1	Hotel7-2	Hotel7-3	Hotel7-4	Hotel7-5	Hotel7-6	Hotel7-7	Hotel7-8	Hotel7-9	Hotel7-10
Hotel8	Hotel8-1	Hotel8-2	Hotel8-3	Hotel8-4	Hotel8-5	Hotel8-6	Hotel8-7	Hotel8-8	Hotel8-9	Hotel8-10
Hotel9	Hotel9-1	Hotel9-2	Hotel9-3	Hotel9-4	Hotel9-5	Hotel9-6	Hotel9-7	Hotel9-8	Hotel9-9	Hotel9-10
Hotel10	Hotel10-1	Hotel10-2	Hotel10-3	Hotel10-4	Hotel10-5	Hotel10-6	Hotel10-7	Hotel10-8	Hotel10-9	Hotel10-10

Figure 4.21 Example of a matrices of pair of hotel

4.2.3 Creating initial population

This process create an initial population by appending sequence of hotel to the tour one by one based on probability which proportioned to their score. For the tour with more than 1 day long, the end hotel of each day and a start hotel of the next day must be the same. If it makes an infeasible solution, the algorithm will try to improve the solution using **feasibility improvement algorithm** to make it feasible. If feasibility improvement algorithm was applied but the solution still infeasible then it would be discarded and the next solution would be created. After each individual population is created, it is improved by a local search technique. The whole process is repeated to create solution equals to the specified limit *PopSize*. Below Figure 4.22 show an algorithm for Population Initialization.

Creating Initial Population:
$k = 1$
While ($k < PopSize$) **Do:**
G_k = Generate an initial solution
Apply solution feasibility improvement algorithm on G_k to ensure feasibility of the solution
If initial solution is a feasible solution **then**:
Apply local search on G_k to improve solution quality
k += 1
End
Current Population = Initial Population
Calculate and save the *Best-Found-Solution*

Figure 4.22 Algorithm for Population Initialization

4.2.4 Example of how the algorithm generated initial solution

An example of how the algorithm generated initial solution for 1 day tour and 3 days tour (multiple day tour) are explained in details on the following sections.

4.2.4.1 Example of how the algorithm generate initial solution for 1 day tour

In this example a specified start node is Hotel1/Depo1 and the specified end node is Hotel2/Depo2, thus the algorithm select the SolutionHotel1-2 as an initial solution for this example. Below Figure 4.23 and Figure 4.24 shows how a solution was selected from a matrices of pair of hotel and initial solution, respectively.

From/To	Hotel1	Hotel2	Hotel3	Hotel4	Hotel5
Hotel1	SolutionHotel1-1	SolutionHotel1-2	SolutionHotel1-3	SolutionHotel1-4	SolutionHotel1-5
Hotel2	SolutionHotel2-1	SolutionHotel2-2	SolutionHotel2-3	SolutionHotel2-4	SolutionHotel2-5
Hotel3	SolutionHotel3-1	SolutionHotel3-2	SolutionHotel3-3	SolutionHotel3-4	SolutionHotel3-5
Hotel4	SolutionHotel4-1	SolutionHotel4-2	SolutionHotel4-3	SolutionHotel4-4	SolutionHotel4-5
Hotel5	SolutionHotel5-1	SolutionHotel5-2	SolutionHotel5-3	SolutionHotel5-4	SolutionHotel5-5

Figure 4.23 Example of how Population Initialization algorithm select a pair of hotel to generate an initial solution for 1 day tour

Figure 4.24 Initial solution generated by Population Initialization algorithm for 1 day tour

4.2.4.2 Example of how the algorithm generate initial solution for 3 days tour

In this example a specified start node is Hotel1/Depo1 and the specified end node is Hotel8/Depo8, then a probability of each pair of hotel would be calculated by using its score divided by sum of a score of all pair of hotel. After probability of each pair of hotel was calculated. Then the algorithm would randomly select one solution that have specified or previous day end hotel as start node, while the end hotel is randomly selected based on a probability of each end hotel. These process would be repeated until the maximum number of day in tour is reached. Below Figure 4.25 and Figure 4.26 shows were shown how the algorithm generate a 3 days long tour (3 trips).

From/To	Hotel1	Hotel2	Hotel3	Hotel4	Hotel5	Hotel6	Hotel7	Hotel8
Hotel1	Hotel1-1	Hotel1-2	Hotel1-3	Hotel1-4	Hotel1-5	Hotel1-6	Hotel1-7	Hotel1-8
Hotel2	Hotel2-1	Hotel2-2	Hotel2-3	Hotel2-4	Hotel2-5	Hotel2-6	Hotel2-7	Hotel2-8
Hotel3	Hotel3-1	Hotel3-2	Hotel3-3	Hotel3-4	Hotel3-5	Hotel3-6	Hotel3-7	Hotel3-8
Hotel4	Hotel4-1	Hotel4-2	Hotel4-3	Hotel4-4	Hotel4-5	Hotel4-6	Hotel4-7	Hotel4-8
Hotel5	Hotel5-1	Hotel5-2	Hotel5-3	Hotel5-4	Hotel5-5	Hotel5-6	Hotel5-7	Hotel5-8
Hotel6	Hotel6-1	Hotel6-2	Hotel6-3	Hotel6-4	Hotel6-5	Hotel6-6	Hotel6-7	Hotel6-8
Hotel7	Hotel7-1	Hotel7-2	Hotel7-3	Hotel7-4	Hotel7-5	Hotel7-6	Hotel7-7	Hotel7-8
Hotel8	Hotel8-1	Hotel8-2	Hotel8-3	Hotel8-4	Hotel8-5	Hotel8-6	Hotel8-7	Hotel8-8

Figure 4.25 Example of how Population Initialization algorithm select a pair of hotel to generate an initial solution for 3 days tour

As shows on Figure 4.25, the first trip of the tour is SolutionHotel1-2 which started from a specified start node (Hotel1) and randomly ended in Hotel2. Then the second trip must started with Hotel2 but randomly ended in Hotel4 (SolutionHotel2-4). Then the start node of the third trip (last trip) is automatically be Hotel4. Since the number of day in tour is 3 the end hotel of this trip must be a specified end hotel/depo which is Hotel8 for this example, thus the SolutionHotel4-8 was selected for the last trip. Below Figure 4.26 is an example of initial solution generated by the algorithm described above.

Hotel1	Att4	Att7	Lunch31	Att2	Att6	Dinner20	Att5	Hotel2
Hotel2	Att11	Att35	Lunch5	Att18	Att2	Dinner9	Att21	Hotel4
Hotel4	Att14	Att16	Lunch16	Att24	Att30	Dinner28	Hotel8	

Figure 4.26 Initial solution generated by Population Initialization algorithm for 3 days tour

4.2.5 Feasibility Improvement

After an initial solution generated then it would be processed through the solution feasibility improvement algorithm to check and improve its feasibility. The following conditions would be checked to ensure the feasibility of the solution.

1. Duplication of attraction and restaurant(lunch/dinner) node across the tour
2. Check number of restaurant whether it less than or exceed 1 per type (lunch/dinner)
3. Time window of an included node in each trip of the tour
4. Total cost must lower than specified budget

On **Figure 4.27** the algorithm to check feasibility of the above condition was shown and would be described on below section.

Feasibility Improvement (Attraction/Restaurant):

For each *trip* **in** *tour*:

If number of attraction/ restaurant node *duplications* > 0 **then:**
Remove duplication across the tour
If *number of lunch/dinner node in trip* < 1 **then:**
Insert lunch/dinner node into the trip
If lunch/dinner node insertion couldn't process **then**:
Replace attraction with lowest ratio with lunch/dinner node
If *number of lunch/dinner node in trip* > 1 **then:**
Remove lunch/dinner node with the lowest NPR value from the trip
Improve solution time window

End

Figure 4.27 Logic structure of feasibility improvement algorithm

1. Remove duplication in attraction and restaurant node across the tour: the algorithm would check whether there are any duplication in attraction and restaurant node across the tour. If duplication occurred the node with the lowest NPR value would be removed from the tour.

2. Checking number of restaurant node (lunch/dinner): this is an algorithm to check whether it equals to 1 per trip across the tour. If number of restaurant (lunch/dinner) in some trip is less than 1 then the algorithm would insert a restaurant node of type that were missing to the trip. But if the restaurant insertion could not be processed due to the time limit of each trip (assumed to be 20/8PM in this study) then the algorithm would remove some attraction node from the tour to create time slot for restaurant node to be inserted by taking time window into account. On the other hand, if the number of restaurant node is more than 1 then the algorithm would remove the restaurant node that have the lowest NPR value from the trip.

If the solution still infeasible after processed through the above algorithm then this solution would be discarded and the next solution would be selected to process through this algorithm. On the other hand, if the solution is feasible after

processed through this algorithm then it would be passed to local search to improve its quality.

4.2.6 Local Search

The VND with seven local search heuristics is used to improve attractions and restaurants between each pair of hotel.

Best = the current incumbent solution
Set of four greedy moves (Gr_k):
Insert (Att) and Add (Res), Replacement (Both), Two-Opt (Both), Move-best (Att), Swap (Res) and Swap-Best (Att), Extract-Insert, Extract2-Insert
NoI = 0

While (*NoI<4*) **Do:**
    Best' = Best after applied local search Gr_k on
    If *X' > X* **then**
        Best = Best'
        NoI = 0
    Else
        NoI += 1
    End

Figure 4.28 Logic structure of Variable Neighborhood Descent with seven local search heuristics

4.2.6.1 Insert and Add (b):

Main logic of this algorithm is the same as Insert and Add algorithm in pre-processing step please see **Figure 4.4**. The only different is that this algorithm considered to insert and add node to a multiple trip within the tour, while the algorithm in pre-processing step considered only one trip. This algorithm would iterated through each trip in tour to check whether Insert and Add algorithm (b) could be executed.

4.2.6.2. Two-Opt (b):

Main logic of this algorithm is the same as Two-Opt (a) please see **Figure 4.15**. The only different is that this algorithm would try to do node inversion in every trip within the tour, while the algorithm in pre-processing step could be used with only one trip. This algorithm would iterated through each trip in tour to check whether Two-Opt (b) should be executed.

4.2.6.3. Move-best (b):

Main logic of this algorithm is the same as Move-best (a) in pre-processing step as show on **Figure 4.18**. The only different is that this algorithm could be used to find the best target position across multiple trip in the tour for each attraction node to be moved to, while the algorithm in pre-processing step considered only target node in one and only trip. Assumed that solution in Figure 4.29 is given to **Move-best (b)** *algorithm*, then each attraction node in each trip of the tour (3 trips in 1 tour) would be able to move across different trip in the tour, so Att4 from trip1 could be moved to position of node Att35 from trip2 or Att24 from trip3. When compare to Move-best (a) Att4 could only move within trip1.

Hotel1	Att4	Att7	Lunch31	Att2	Att6	Dinner20	Att5	Hotel2
Hotel2	Att11	Att35	Lunch5	Att18	Att2	Dinner9	Att21	Hotel4
Hotel4	Att14	Att16	Lunch16	Att24	Att30	Dinner28	Hotel8	

Figure 4.29 Example of input that could be processed by Move-best (b) algorithm

4.2.6.4. Swap-Best:

Swap-best:
BestTrip = *OriginalTrip*
MaxDiff = 0
For each trip in tour:
For each attraction node in trip:
For each target node in trip:
Calculate *Difference* between old and new sequence of trip
If *Difference* > *MaxDiff* **Do:**
Calculate *TW of attraction node*
If *Opt (attraction)* <= *TW of attraction node* <=*ClT (attraction)* and *TotalCost* + *cost associated with each attraction node* <= *MaxBudget* **Do:**
If TW of all node in the trip is not violated **Do:**
Execute **Swap-best** and save it as *TemporaryTrip*
MaxDiff = *Difference*
End
End
End
If *MaxDiff* <> 0 **Do:**
BestTrip = *TemporaryTrip*
Output: *BestTrip*

Figure 4.30 Logic structure of Swap-Best algorithm

From Figure 4.30, the purpose of Swap-Best algorithm is to reduce overall trip time. *Swap-Best* will be used to determine the best pair of attraction node to be swapped. It could be pair of attraction node from the same trip or pair of attraction node from different trip that could provide the most reduction on overall tour time. To check whether there are any improvement in solution by checking the different calculated from this algorithm whether it is a nonnegative/zero value. If different is a nonnegative/zero value (improvement exist) then the algorithm would proceed to find a pair of attraction with the highest different and select it as a pair to be swapped to create new solution. If any improvement occurred then a solution from this algorithm would be send back to the *Insert and Add* Algorithm to check whether a time that were reduced from this algorithm could create enough time slot to insert an unvisited attraction node. Note that feasibility also checked for every possible pair of attraction node before any swapping occurred. The logic structure of this algorithm was shown on Figure 4.30. From Figure 4.31, the input solution for *Swap-Best algorithm* have three trips in one tour. When this algorithm is executed, it would go through each attraction node in each trip then calculate different between original tour sequence and the tour sequence after swapping occurred. There are two main different calculation scenarios in this algorithm that each attraction node in the tour would go through: the first scenario is swapping within the same trip. The second scenario is swapping between two different trips. The details formula would be explained below for each scenario.

Hotel1	Att4	Att7	Lunch31	Att2	Att6	Dinner20	Att5	Hotel2
Hotel2	Att11	Att35	Lunch5	Att18	Att2	Dinner9	Att21	Hotel4
Hotel4	Att14	Att16	Lunch16	Att24	Att30	Dinner28	Hotel8	

Figure 4.31 Example of input that could be processed by Swap-Best algorithm

4.2.6.4.1 For Swapping within the same trip scenario

There are four sub scenarios and the following formula would be used to calculate the different in each sub scenarios. Assuming that i is a position of node of interested and j is a position of node to be swapped.

4.2.6.4.1.1 The first sub scenario is the scenario where i and j is an adjacent node of each other but i visited before j (j = i+1). The following

formula would be used to calculate the different (time saved) of the swapping:

1. Original trip sequence overall trip time = [Traveling Time from i-1 to i] + [Traveling time from i to i+1] + [Traveling time from j from the right to j+1]
2. New trip sequence overall trip time = [Traveling Time from i-1 to j] + [Traveling time from j to i] + [Traveling time from i from the right to j+1]
3. Different = Original trip sequence overall trip time – New trip sequence overall trip time

4.2.6.4.1.2 The second sub scenario is the scenario where i and j is an adjacent node of each other but i visited after j (j = i-1). The following formula would be used to calculate the different (time saved) of the swapping:

1. Original trip sequence overall trip time = [Traveling Time from j-1 to j] + [Traveling time from j to i] + [Traveling time from i from the right to i+1]
2. New trip sequence overall trip time = [Traveling Time from j-1 to i] + [Traveling time from i to j] + [Traveling time from j from the right to i+1]
3. Different = Original trip sequence overall trip time – New trip sequence overall trip time

4.2.6.4.1.3 And 4.2.6.4.1.4 the third and fourth sub scenario are the scenario where i visited before j and I visited after j, respectively. The following formula would be used to calculate the different (time saved) of the swapping:

1. Original trip sequence overall trip time = [Traveling Time from j-1 to j] + [Traveling time from j to j+1] + [Traveling time from i-1 from the right to i] + [Traveling time from i from the right to i+1]

2. New trip sequence overall trip time = [Traveling Time from j-1 to i] + [Traveling time from i to j] + [Traveling time from j from the right to i+1]

3. Different = Original trip sequence overall trip time – New trip sequence overall trip time

4.2.6.4.2 For Swapping between two different trip scenarios.

The following formula would be used to calculate the different for swapping between two attraction nodes form different trips:

1. Original trip sequence overall trip time = [Traveling Time from i-1 to i] + [Traveling time from i to i+1] + [Traveling time from j-1 from the right to j] + [Traveling time from j from the right to j+1]

2. New trip sequence overall trip time = [Traveling Time from i-1 to j] + [Traveling time from j to i+1] + [Traveling time from j-1 from the right to i] + [Traveling time from i from the right to j+1]

3. Different = Original trip sequence overall trip time – New trip sequence overall trip time

Where i is a position of node of interested and j is a position of node to be swapped. After different were calculated for each attraction node in the tour the pair of attraction node with the highest different (saved time) would be selected.

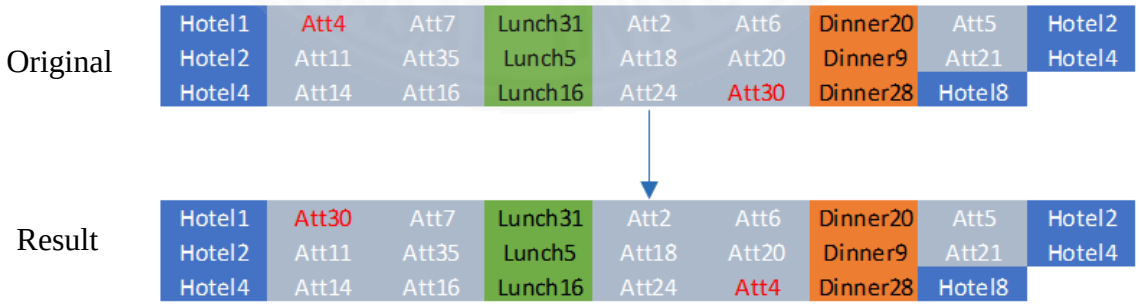


Figure 4.32 Structure of solution before and after processed by Swap-Best algorithm

Figure 4.32 shows an example of solution before and after processed through *Swap-Best algorithm*. In this example, it is assumed that Att4 and Att30 have the highest different when compared with other pairs of attraction node, so the algorithm would swap the position between Att4 and Att30.

4.2.6.5. Extract-Insert-Replace:

```
Extract-Insert-Replace:
    BestTrip = OriginalTrip
    MaxDiff = 0
    For each trip in tour:
        For each node in trip (exception hotel):
            If node is attraction node Do:
                Remove attraction node and all it associated value from the trip
                NoImprove = 0
                While NoImprove < 1
                    For each non-included attraction node:
                        For each position in trip:
                            Calculate TW of attraction node at the position
                              If Opt (attraction) <= TW of attraction node <=ClT (attraction) and TotalCost
                              + cost associated with each attraction node <= MaxBudget Do:
                                  Calculate NPR of each attraction node
                                  If NPR > MaxRatio Do:
                                      If TW of all node in the trip is not violated Do:
                                          Insert attraction node to TemporaryTrip
                                          MaxRatio = NPR
                        End
                    End
                    If MaxRatio <> 0:
                        BestTrip = TemporaryTrip
                    Else:
                        NoImprove = NoImprove + 1
                End
            Else If node is lunch node Do:
                Remove lunch node and all it associated value from the trip
                NoImprove = 0
                While NoImprove < 1
                    For each non-included lunch node:
                        For each position in trip:
                            Calculate TW of lunch node at the position
                              If Opt (lunch) <= TW of lunch node <=ClT (lunch) and TotalCost + cost
                              associated with each lunch node <= MaxBudget Do:
                                  Calculate NPR of each attraction node
                                  If NPR > MaxRatio Do:
                                      If TW of all node in the trip is not violated Do:
                                          Insert lunch node to TemporaryTrip
                                          MaxRatio = NPR
                        End
                    End
                    If MaxRatio <> 0:
                        BestTrip = TemporaryTrip
                        NoImprove = NoImprove + 1
                    Else:
                        NoImprove = NoImprove + 1
                End
            Else:
                Remove dinner node and all it associated value from the trip
                NoImprove = 0
                While NoImprove < 1
                    For each non-included dinner node:
                        For each position in trip:
                            Calculate TW of dinner node at the position
                              If Opt (dinner) <= TW of dinner node <=ClT (dinner) and TotalCost + cost
                              associated with each dinner node <= MaxBudget Do:
                                  Calculate NPR of each attraction node
                                  If NPR > MaxRatio Do:
                                      If TW of all node in the trip is not violated Do:
                                          Insert dinner node to TemporaryTrip
                                          MaxRatio = NPR
                        End
                    End
                    If MaxRatio <> 0:
                        BestTrip = TemporaryTrip
                        NoImprove = NoImprove + 1
                    Else:
                        NoImprove = NoImprove + 1

                End
        End
    End
    If MaxDiff <> 0 Do:
        BestTrip = TemporaryTrip
Output: BestTrip
```

Figure 4.33 Logic structure of Extract-Inset-Replace algorithm

From Figure 4.33, this algorithm would be applied to 2 types of node which are attraction and restaurant node. For attraction node, the algorithm would remove one attraction node from each trip in the tour then insert other unvisited attraction node as much as possible. Only attraction node with the highest NPR value that would be selected in each round of the algorithm. When the algorithm is executed, it will start to remove the first attraction node then try to insert non-included attraction node with highest NPR value in each loop as much as possible. The algorithm would continue this process until it reached the last attraction node in each trip. If there are any improvement in total score of a solution then the score would be saved and considered as master solution, otherwise original solution would still be considered as master solution. The same approach also applied to each restaurant in each trip of the tour, but the different is that only one node of type restaurant for lunch/dinner that would be insert to the trip. The algorithm would try to remove restaurant for lunch/dinner node then insert one of an unvisited restaurant for lunch/dinner node. In each type of restaurant for lunch and dinner, only node that have the highest NPR value would be selected to be inserted to the trip. If the total score of a solution after insertion is higher than the original total score, this algorithm applied. Noted that the excluded attractions and restaurant node are not considered again for insertion.

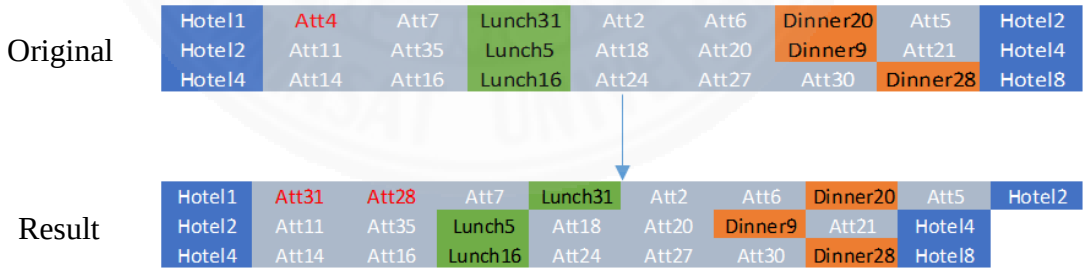


Figure 4.34 Structure of solution before and after processed by Extract-Insert-Replace algorithm

Figure 4.34 shows an example of solution before and after processed through Extract-Insert-Replace algorithm. In this example, Att4 in trip 1 of the tour was assumed to have be the best node to be removed because deleting this would resulting in the best total score improvement since its lead to insertion of Att31 and Att28 (these two nodes are non-included attraction node) which assumed to have

the best improvement in total tour score value when compared to other included attraction node in the tour.

4.3 Main-loop

In this part, the current population will be applied two types of genetic operator which are crossover and mutation. After each genetic operator applied, the local search will be applied to improve quality of the solution. Then list of the solution is sorted based on their quality and the best solution is saved as the best found solution. After that the population management is processed to increase diversity of a solution pool. Below section explain algorithm with a more detailed

4.3.1 Selection

The selection method used in both crossover and mutation process is the roulette wheel. This method assigns a probability P_i to each solution in the population, where probability calculated from their score over total score (Talbi, 2009):

$P_i = f_i / \sum_{i=1}^{V} fi$

4.3.2 Termination

In this study, *MaxIterations* is used to limit maximum generation created by this algorithm, this parameter need to be adjusted to balance the solution quality and calculation time usage more details would be discussed in Chapter 5.

4.3.3 Crossover

Crossover method used in this study is a one-point crossover method, two parent solutions is selected for crossover process to create two offspring solutions (O_1 and O_2). The algorithm randomly selected one trip in the tour and then randomly select a time in the selected trip to be used as a chromosome cut point for crossover process. This method would be called "*trip and time cut*" method and a target node that would be cut called "*node to be cut*". In this study, range of time is ranged from 8.00 to 20.00 and decimal number is allowed. The decimal is ranged from 0.00 to 0.99 (percentage in one hour). Then feasibility improvement and local search technique will be applied to both offspring to improve solution quality and ensure its feasibility. If a solution is infeasible after crossover process then the algorithm

will re-random new trip and time to be new cut point. After that the solution is transferred to the *Pool*. There are 2 parameters that used to control the number of crossover process as follows: 1. Current Population size (*PopSize*) 2. CrossOver rate (*Cr*). These 2 value would be discussed in details in Chapter 5. The crossover algorithm will be explained below and show in **Figure 3**.

1. The crossover will be randomly occur at a specific trip and time in the first parent solution P_1 and second parent solution P_2.

1.1 If a crossover point is not a start (leave time of previous node) or end time (leave time of interested node) of the location, it will calculate whether crossover point occurred near the start or end time of target node in the trip. If crossover point near start time, so it cut the tour from the first node of the tour up to the node before target node. Otherwise, the algorithm cut the tour from the first node of the tour up to the target node.

1.2 If a crossover point is a start or end time of the target node then the algorithm will cut the after that time.

2. Then the algorithm check whether the crossover is feasible.

2.1 If it feasible then the algorithm will take a solution of the first parent up to crossover point then attached it with solution of the second parent after the crossover point.

2.2 If it infeasible then the algorithm to improve solution feasibility (***Alg.3***) will be executed. If it still resulting in an infeasible solution the algorithm will re-random a new crossover point.

Note that the same approach also applied to P_2, so the crossover point can be different from P_1.

Figure 4.35 shows an example of crossover algorithm - the algorithm randomly selected trip 2 and time at 14.5 as a cut point to create one offspring (new solution) from Parent 1 and 2. After trip and time was selected the algorithm, the algorithm navigated to trip 2 and compare cut time of 14.5 with leave time of each node to find node with the nearest leave time. In this case node that have the nearest leave time with 14.5 in Parent 1 is Att19 in trip2. For Parent2, a node with the

nearest leave time with 14.5 is Att20 in trip2. After node to be cut in each trip is identified, the algorithm would take a solution from parent 1 starting from Hotel1 in the 1st trip up to Att19 in the second trip then combined it with solution from parent 2 which starting from Att20 up to Hotel8 of the last trip to create a complete initial offspring solution. After initial offspring solution was created then it would be sent to feasibility improvement and local search technique to improve feasibility and quality of the initial solution before sending it to the pool.

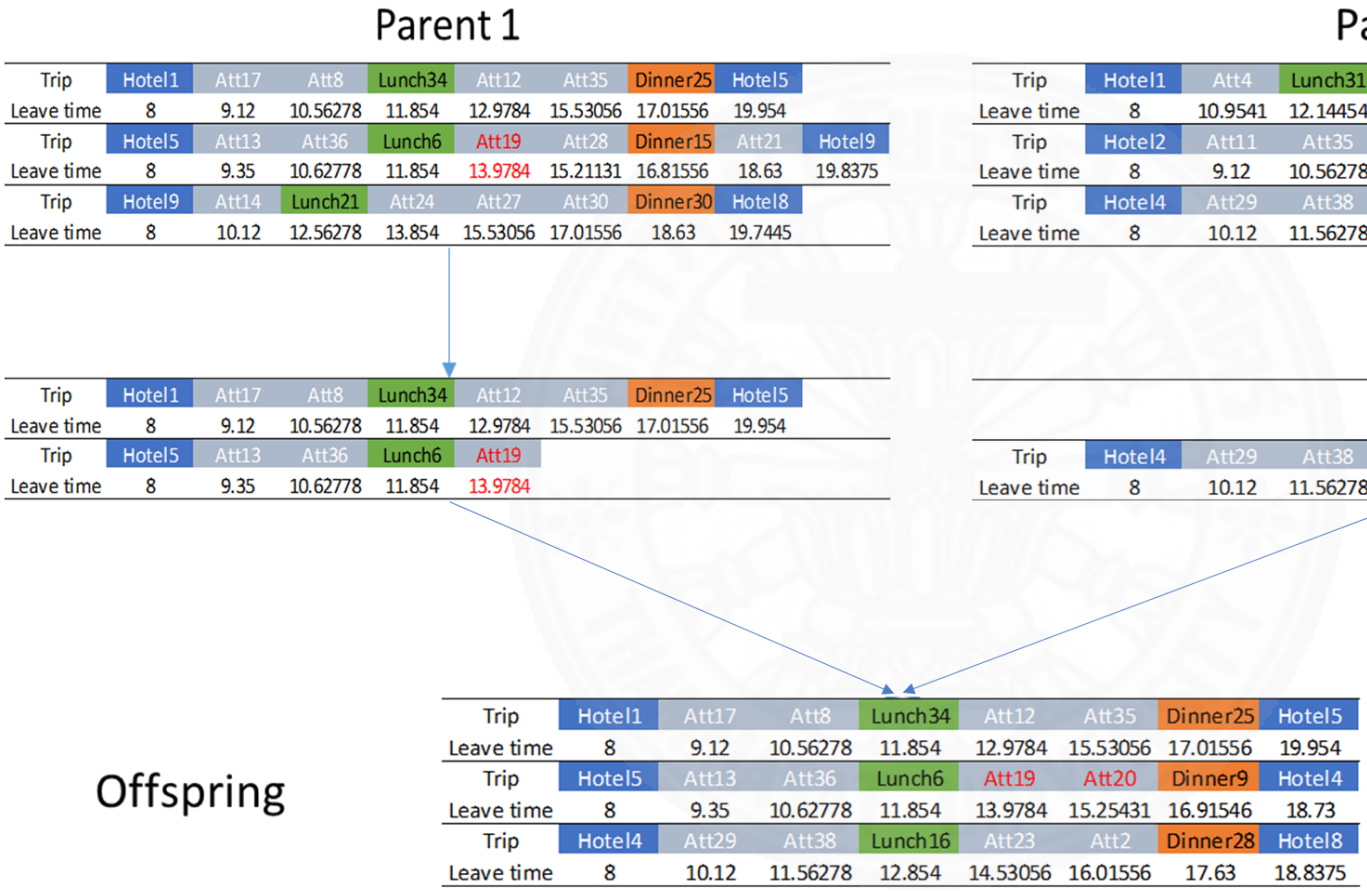


Figure 4.35 Example of how crossover algorithm create new offspring

4.3.4 Mutation

In mutation process, one parent solution is randomly selected and mutated to create one offspring. There are two main cases for mutation algorithm as follow: The first scenario, multiple trip in one tour - One trip in the parent solution with multiple trip will randomly selected and the last hotel of it will be replaced with a new end hotel from matrices pairs of hotels based on their score. If new end hotel is not matches with the start hotel of the next trip then the algorithm would also randomly selected a new start hotel of the next trip but fixed the end hotel of the next trip to be the same hotel. To conduct mutation process the following steps must be considered:

1. If selecting this new end hotel end up with infeasible solution then the algorithm to improve solution feasibility will be executed.

2. If by selecting this end hotel make step 1 impossible to be processed, the next possible end hotel will be selected.

3. If step 1 and 2 cannot be processed, then no mutation occurred.

The second scenario, one trip in one tour (one day tour) – One trip in the parent solution will randomly selected then *trip and time cut* would be executed to split a parent solution into 2 parts then combined it with another solution from matrices of pair of hotel that also split into 2 parts by the same method. In this second scenario, there are 2 ways that mutation would occurred. 1. The algorithm would keep first part of the parent solution and combined it with second part of a solution from a matrices of pair of hotel. 2. The algorithm would take the first part from a solution from a matrices of pair of hotel then combined it with second part of a parent solution. The aim of this method is to increase the diversity of the solution, so that new sequence and combination of node in the trip could be obtained.

After mutation has processed, a feasibility improvement and local search algorithm will be applied to improve a feasibility and quality of the offspring solution. If it resulting in an infeasible solution then no mutation occurred. There are 2 parameters that used to control the number of crossover process as follows: 1. Current Population size (*PopSize*) 2. Mutation rate (*Mut*). These 2 value would be

discussed in details in Chapter 5. The mutation algorithm will be explained below and show in **Figure 4.36**.

OffSpringCount = 1

While (*Cr* <= *CrP* * *Popsize*) **Do (Time cross over):**

Select solution P_1 and P_2 from the Current Population (Roulette Wheel)

Apply Feasibility Improvement on O_1 and O_2

Apply Local Search on O_1 and O_2

Add O_1 and O_2 to the Pool

OffSpringCount += 1

End

OffSpringCount = 1

While (*Mut* <= *(Mut** *Popsize*) **Do:**

Select solution P_1' from the Current Population (Roulette Wheel)

O_1' = Apply Mutation on P_1'

Apply Feasibility Improvement on O_1'

Apply Local Search on O_1'

Add O_1' to the *Pool*

OffSpringCount += 1

End

Figure 4.36 Algorithm for Crossover and Mutation

Below Figure 4.37 shows and example of mutation algorithm – the algorithm randomly select trip 2 from parent to be a mutated. Then the algorithm would randomly select new hotel for trip 2 by fixed start hotel as shown. As a result Hotel7 was selected as a new end hotel but it not match with a start hotel of the next trip (trip 3) which is Hotel4. So the algorithm will select new trip 3 by fixed a start hotel as Hotel7 and end hotel as Hotel8 to make end hotel of trip 3 align with the

original sequence. After the mutation process is completed, a solution would send to feasibility improvement and local search technique to improve feasibility and quality of the solution before sending it to the pool.

Parent

Trip	Hotel1	Att17	Att8	Lunch34	Att12	Att35	Dinner25	Hotel5
Leave time	8	9.12	10.56278	11.854	12.9784	15.53056	17.01556	19.954
Trip	Hotel5	Att13	Att36	Lunch6	Att19	Att20	Dinner9	Hotel4
Leave time	8	9.35	10.62778	11.854	13.9784	15.53056	16.91546	18.73
Trip	Hotel4	Att29	Att38	Lunch16	Att23	Att2	Dinner28	Hotel8
Leave time	8	10.12	11.56278	12.854	14.53056	16.01556	17.63	18.8375

From/To	Hotel1	Hotel2	Hotel3	Hotel4	Hotel5	Hotel6	Hotel7	Hotel8	Hotel9	Hotel10
Hotel1	Hotel1-1	Hotel1-2	Hotel1-3	Hotel1-4	Hotel1-5	Hotel1-6	Hotel1-7	Hotel1-8	Hotel1-9	Hotel1-10
Hotel2	Hotel2-1	Hotel2-2	Hotel2-3	Hotel2-4	Hotel2-5	Hotel2-6	Hotel2-7	Hotel2-8	Hotel2-9	Hotel2-10
Hotel3	Hotel3-1	Hotel3-2	Hotel3-3	Hotel3-4	Hotel3-5	Hotel3-6	Hotel3-7	Hotel3-8	Hotel3-9	Hotel3-10
Hotel4	Hotel4-1	Hotel4-2	Hotel4-3	Hotel4-4	Hotel4-5	Hotel4-6	Hotel4-7	Hotel4-8	Hotel4-9	Hotel4-10
Hotel5	Hotel5-1	Hotel5-2	Hotel5-3	Hotel5-4	Hotel5-5	Hotel5-6	Hotel5-7	Hotel5-8	Hotel5-9	Hotel5-10
Hotel6	Hotel6-1	Hotel6-2	Hotel6-3	Hotel6-4	Hotel6-5	Hotel6-6	Hotel6-7	Hotel6-8	Hotel6-9	Hotel6-10
Hotel7	Hotel7-1	Hotel7-2	Hotel7-3	Hotel7-4	Hotel7-5	Hotel7-6	Hotel7-7	Hotel7-8	Hotel7-9	Hotel7-10
Hotel8	Hotel8-1	Hotel8-2	Hotel8-3	Hotel8-4	Hotel8-5	Hotel8-6	Hotel8-7	Hotel8-8	Hotel8-9	Hotel8-10
Hotel9	Hotel9-1	Hotel9-2	Hotel9-3	Hotel9-4	Hotel9-5	Hotel9-6	Hotel9-7	Hotel9-8	Hotel9-9	Hotel9-10
Hotel10	Hotel10-1	Hotel10-2	Hotel10-3	Hotel10-4	Hotel10-5	Hotel10-6	Hotel10-7	Hotel10-8	Hotel10-9	Hotel10-10

From/To	Hotel1	Hotel2	Hotel3	Hotel4	Hotel5	Hotel6	Hotel7	Hotel8	Hotel9	Hotel10
Hotel1	Hotel1-1	Hotel1-2	Hotel1-3	Hotel1-4	Hotel1-5	Hotel1-6	Hotel1-7	Hotel1-8	Hotel1-9	Hotel1-10
Hotel2	Hotel2-1	Hotel2-2	Hotel2-3	Hotel2-4	Hotel2-5	Hotel2-6	Hotel2-7	Hotel2-8	Hotel2-9	Hotel2-10
Hotel3	Hotel3-1	Hotel3-2	Hotel3-3	Hotel3-4	Hotel3-5	Hotel3-6	Hotel3-7	Hotel3-8	Hotel3-9	Hotel3-10
Hotel4	Hotel4-1	Hotel4-2	Hotel4-3	Hotel4-4	Hotel4-5	Hotel4-6	Hotel4-7	Hotel4-8	Hotel4-9	Hotel4-10
Hotel5	Hotel5-1	Hotel5-2	Hotel5-3	Hotel5-4	Hotel5-5	Hotel5-6	Hotel5-7	Hotel5-8	Hotel5-9	Hotel5-10
Hotel6	Hotel6-1	Hotel6-2	Hotel6-3	Hotel6-4	Hotel6-5	Hotel6-6	Hotel6-7	Hotel6-8	Hotel6-9	Hotel6-10
Hotel7	Hotel7-1	Hotel7-2	Hotel7-3	Hotel7-4	Hotel7-5	Hotel7-6	Hotel7-7	Hotel7-8	Hotel7-9	Hotel7-10
Hotel8	Hotel8-1	Hotel8-2	Hotel8-3	Hotel8-4	Hotel8-5	Hotel8-6	Hotel8-7	Hotel8-8	Hotel8-9	Hotel8-10
Hotel9	Hotel9-1	Hotel9-2	Hotel9-3	Hotel9-4	Hotel9-5	Hotel9-6	Hotel9-7	Hotel9-8	Hotel9-9	Hotel9-10
Hotel10	Hotel10-1	Hotel10-2	Hotel10-3	Hotel10-4	Hotel10-5	Hotel10-6	Hotel10-7	Hotel10-8	Hotel10-9	Hotel10-10

Offspring

Trip	Hotel1	Att17	Att8	Lunch34	Att12	Att35	Dinner25	Hotel5
Leave time	8	9.12	10.56278	11.854	12.9784	15.53056	17.01556	19.954
Trip	Hotel5	Att13	Att36	Lunch6	Att19	Att20	Dinner9	Hotel7
Leave time	8	9.35	10.62778	11.854	13.9784	15.53056	16.91546	18.73
Trip	Hotel7	Att29	Att38	Lunch16	Att23	Att2	Dinner28	Hotel8
Leave time	8	10.12	11.56278	12.854	14.53056	16.01556	17.63	18.8375

Figure 4.37 Example of how mutation algorithm create new offspring

4.3.5 Population Management

In this study, population management algorithm will randomly select solution based roulette wheel by using top-down and bottom-up approach in each iteration of the algorithm. Top-down approach is the approach that a solution with higher score tent to have more chance to be selected. On the other hand, Bottom-up approach is the approach that a solution with lower score would have more chance to be selected. For top-down approach, each solution would divided by sum of score of all solution, so a solution with a higher score would to have a higher percent chance to be selected to the *Pool*. On the other hand, a bottom-up approach would favor a solution with lowest score by using the following calculation method to calculate percent chance for each solution in the population pool (*Pool*): [(-1 * Score of each solution) divided by sum of score of all solution in the *Pool*)]. By using this formula a solution with a lower score would have a higher percent chance to be selected to the *Pool*.

Empty the Current Generation
Ind_{c1} = 1
While (Ind_{c1} <= PopSize) **Do:**
    i = Ind_{c1}
    Select the i^{th} solution from the *top* of the Pool and add it to the Current Population (Roulette Wheel; top-down)
    Select the i^{th} solution from the *bottom* of the Pool and add it to the Current Population (Roulette Wheel: bottom-up)
    Ind_{c1} += 2
End

Figure 4.38 Algorithm for Population Management

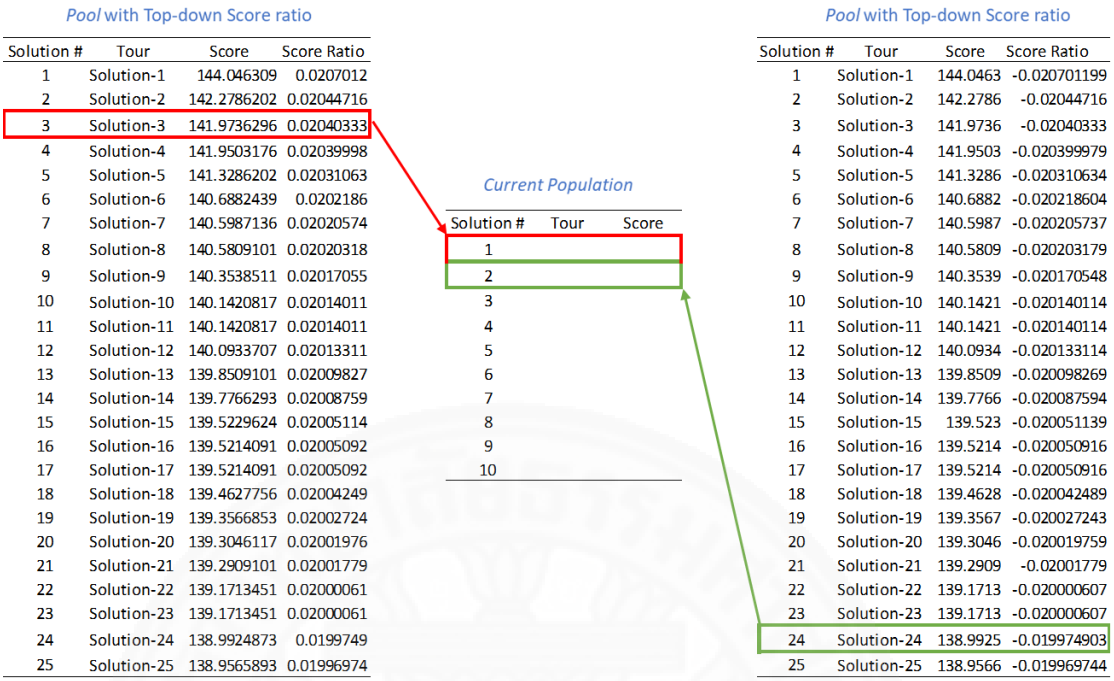


Figure 4.39 Example of how Population Management algorithm select solution to be included in the current population

Figure 4.39 – the algorithm randomly select solution from the *pool* by using top-down and bottom-up approach roulette wheel repeatedly until maximum number of current population reached. The purpose of this algorithm is to increase population diversity in current population.

CHAPTER 5
EXPERIMENTS AND RESULTS

5.1 Dataset Explanation

A review/customer satisfaction score was gathered from Agoda.com, Wongnai.com, and TripAdvisor.com. The reason to select these three websites is that these website is a creditable source of information based on number of users provided a review on their website. Data collection for destination in the model of this study was selected from the central province region of Thailand which are Kanchanaburi, Nakhon Pathom, and Ratchburi. These province have many natural and historical tourist attraction, which could represent cultures and history of Thailand, so that tourist could learn and enjoy Thai culture and history. In this study the following data were collected for hotel in each province: 1. Hotel name, 2. Price per night, 3. Facilities, 4. Check in time, 5. Satisfaction score (the selected hotel must have rating start between 3-5). The hotel with score less than 3 is not selected in this study because it is assumed that score lower than 3 is not an acceptable level. This assumption also applied when collecting a data for restaurant and tourist attractions. For restaurant data collection, the following data would be collected: 1. Restaurant name, 2. Price per person, 3. Satisfaction score (the selected restaurant must have score between 3-5). For tourist attraction, the following data would be collected: Attraction name, 2. Entrance fee per person, 3. Satisfaction score (the selected attraction must have score between 3-5), 4. Attraction opening and closing time.

After all other information were gathered, the distance between each place is calculated by using amount of duration times (without traffic factor) between places to another were calculated by collecting the latitude and longitude of each site using Google map and apply excel function ‘GetDuration’ for all 181 sites including 39 hotels, 48 lunch restaurants, 49 dinner restaurants, and 45 tour sites. For more detail please see Appendix A, B, C, and D for Hotel, Restaurant for Lunch, Restaurant for Dinner, and Attraction, respectively. For traveling time between each node, please check test data in a link provided in Appendix E or F.

5.2 Example of Solution

First, OPHRSTW is solved by using mixed integer linear programming (MILP) approach to solve the problem to optimality using an optimization software called "IBM ILOG CPLEX 12.10.0". The computer specification that is used are as follows: Operating System – Window 10 64-bit OS; Processor – Intel® Core ™ i7 – 6700HQ CPU @ 2.60GHz; RAM 16.00 GB. Table 5.1 shows the optimal OPHRSTW solution for 2 days tour with score of 46.016095. The computational time is 87 sec.

Table 5.1 Optimal OPHRSTW solution (Optimization Approach)

Number of day	Route	Score	Budget	
			Limit	Total
1	Depo → Att7 → Att1 → Lunch1 → Att3 → Att5 → Dinner1 → Hotel1	46.16095	6,000	1,845.5
2	Hotel1 → Att4 → Att2 → Lunch2 → Att6 → Dinner2 → Depo			

The same OPHRSTW problem is then solved using memetic algorithm, which coded in Python, run on the same computer. The following parameter and condition are as follows: *PopSize* = 40, *Cr* = 40%, *Mut* = 30%, termination condition (*MaxIterations)* = 5 iterations. The result is displayed in Table 5.2 with computational time of 3.51 sec.

Table 5.2 Optimal OPHRSTW solution (Memetic Algorithm Approach)

Number of day	Route	Score	Budget	
			Limit	Total
1	Depo → Att7 → Att3 → Att4 → Lunch1 → Att5 → Dinner1 → Hotel1	46.16095	6,000	1,845.5
2	Hotel1 → Att1 → Att2 → Lunch2 → Att6 → Dinner2 → Depo			

The result shows that both Optimization and Memetic Algorithm approach provided different solution but giving the same objective value. Below Figure 5.1 show in depth comparison between both two approaches.

MA Solution		
Day	Node number	Leave Time
1	Depo	8.00
1	Att7	9.03
1	Att3	11.16
1	Att4	13.26
1	Lunch1	14.27
1	Att5	16.82
1	Dinner1	19.51
1	Hotel1	19.69
2	Hotel1	8.00
2	Att1	9.14
2	Att2	11.08
2	Lunch2	13.84
2	Att6	15.54
2	Dinner2	17.65
2	Depo	18.86

Optimization Solution		
Day	Node number	Leave Time
1	Depo	8.00
1	Att7	9.03
1	Att1	10.16
1	Lunch1	11.34
1	Att3	13.45
1	Att5	17.12
1	Dinner1	19.81
1	Hotel1	20.00
2	Hotel1	8.00
2	Att4	9.11
2	Att2	11.75
2	Lunch2	14.50
2	Att6	16.21
2	Dinner2	18.80
2	Depo	20.00

Figure 5.1 Comparison between example solution of MA and Optimization approach for 2 days tour

Figure 5.1 shows that the solution provided by both approaches seems to have the same the same node contained in the tour but the sequence of how the node are visited is different and also the leave time of MA tend to be smaller than Optimization approach. The reason could be that the MA approach would try to reduce overall trip time, so that it could put more node to the tour, while this condition was not present in the optimization approach at the first place.

5.3 Result and Discussion

To test effectiveness and efficiency of the memetic algorithm, an experiment of computation was conducted. There are 51 test cases were created. Each of test case would have different number of node in each type assigned. There are 3 types of node as follows: Hotel, Restaurant (Lunch and Dinner), and Attractions. Each of the test case was solved using the ILOG CPLEX and the memetic algorithm (Coded in Python) running on the same computer with specifications described earlier.

Table 5.3 Input to be used to run both Optimization and MA approach

Test case	Budget Limit	Number of each node type			Total Node	Number of day
		Hotel	Lunch/Dinner	Attraction		
1	15000	3	2/2	8	15	1
2	15000	3	2/2	8	15	1
3	15000	3	2/2	8	15	1
4	15000	4	3/3	10	20	1
5	15000	4	3/3	10	20	1
6	15000	4	4/4	10	22	1
7	15000	4	4/4	10	22	1
8	15000	4	3/3	14	24	1
9	15000	4	3/3	14	24	1
10	15000	5	3/3	14	25	1

Test case	Limit	Number of each node type			Total Node	Number of day
		Hotel	Lunch/Dinner	Attraction		
11	15000	5	3/3	14	25	1
12	15000	5	3/3	14	25	1
13	15000	8	13/14	15	50	1
14	15000	22	25/30	23	100	1
15	15000	32	40/41	37	150	1
16	15000	39	48/49	45	181	1
17	15000	3	2/2	8	15	2
18	15000	3	2/2	8	15	2
19	15000	3	2/2	8	15	2
20	15000	4	3/3	10	20	2
21	15000	4	3/3	10	20	2
22	15000	4	4/4	10	22	2
23	15000	4	4/4	10	22	2

Test case	Limit	Number of each node type			Total Node	Number of day
		Hotel	Lunch/Dinner	Attraction		
24	15000	4	3/3	14	24	2
25	15000	4	3/3	14	24	2
26	15000	5	3/3	14	25	2
27	15000	5	3/3	14	25	2
28	15000	5	3/3	14	25	2
29	15000	8	13/14	15	50	2
30	15000	22	25/30	23	100	2
31	15000	32	40/41	37	150	2
32	15000	39	48/49	45	181	2
33	15000	4	3/3	10	20	3
34	15000	4	3/3	10	20	3
35	15000	4	4/4	10	22	3
36	15000	4	4/4	10	22	3

Test case	Limit	Number of each node type			Total Node	Number of day
		Hotel	Lunch/Dinner	Attraction		
37	15000	4	3/3	14	24	3
38	15000	4	3/3	14	24	3
39	15000	5	3/3	14	25	3
40	15000	5	3/3	14	25	3
41	15000	5	3/3	14	25	3
42	15000	8	13/14	15	50	3
43	15000	22	25/30	23	100	3
44	15000	32	40/41	37	150	3
45	15000	39	48/49	45	181	3
46	15000	4	4/4	10	22	4
47	15000	4	4/4	10	22	4
48	15000	8	13/14	15	50	4
49	15000	22	25/30	23	100	4
50	15000	32	40/41	37	150	4
51	15000	39	48/49	45	181	4

Table 5.4 Range of numbers of each node type and number of day in tour in test case

Number of				Number of day in tour
Hotel	Attraction	Lunch	Dinner	
3-39	8-45	2-48	2-49	1-4

Among 51 test cases, the ILOG CPLEX could solve only 14 test cases. The computation times ranged from 10.00 to 746.20 seconds. For test cases that CPLEX unable to solve was due to it reached time limit which is a pre-set time limit of 28,800 seconds (8 hours).

On the other hand, the memetic algorithm (MA) could solve all 35 test cases. The longest computation was only about 10 seconds with a difference in total score range from 0 to 6.34%. Some test case both optimization and MA approach provided the same solution but MA take much lower time to solve the test case. The next section would provide a more details of the test result.

Table 5.5 Comparison of OPHRSTW solution by optimization and MA approaches

Test case	Approach	Score	Budget		Computation time (second)	Number of day	Number of constraints (Cplex)	Variable (Cplex)	
			Limit	Total				Binary	Other
	Optimization	24.32		475.50	10.00				
1	MA	23.40	15000	470.00	3.30	1	281	240	18
	% difference	3.78			6.70				
	Optimization	25.54		620.50	33.00				
2	MA	24.77	15000	900.50	2.50	1	281	240	18
	% difference	3.01			30.50				
	Optimization	24.70		951.00	29.00				
3	MA	24.70	15000	951.00	2.90	1	281	240	18
	% difference	0.00			26.10				
	Optimization	26.32		375.00	351.00				
4	MA	24.65	15000	670.00	3.70	1	468	420	24
	% difference	6.34			347.30				
	Optimization	27.98		1,160.00	47.57				
5	MA	27.98	15000	1,160.00	3.64	1	468	420	24
	% difference	0.00			43.93				
	Optimization	28.65		595.50	746.20				
6	MA	28.48	15000	675.50	3.55	1	560	506	26
	% difference	0.59			742.65				
	Optimization	27.98		825.00	66.27				
7	MA	27.98	15000	825.00	3.09	1	560	506	26
	% difference	0.00			63.18				
	Optimization	-		-	-				
8	MA	30.26	15000	480.00	5.27	1	660	600	28
	% difference								
	Optimization	-		-	-				
9	MA	28.70	15000	990.00	4.60	1	660	600	28
	% difference								
	Optimization	-		-	-				
10	MA	30.26	15000	480.00	5.62	1	703	650	30
	% difference								

Test case	Approach	Score	Budget		Computation time (second)	Number of day	Number of constraints (Cplex)	Variable (Cplex)	
			Limit	Total				Binary	Other
	Optimization	-		-	-				
11	MA	30.26	15000	480.00	5.64	1	703	650	30
	% difference								
	Optimization	-		-	-				
12	MA	25.82	15000	951.00	5.19	1	703	650	30
	% difference								
	Optimization	-		-	-				
13	MA	31.42	15000	685.50	5.10	1	2636	2550	58
	% difference								
	Optimization	-		-	-				
14	MA	33.31	15000	310.00	11.33	1	9902	10100	122
	% difference								
	Optimization	-		-	-				
15	MA	36.14	15000	780.00	16.89	1	22052	22650	182
	% difference								
	Optimization	-		-	-				
16	MA	36.14	15000	780.00	22.25	1	31933	32942	220
	% difference								
	Optimization	46.16		1,845.50	87.00				
17	MA	46.16	15000	1,845.50	3.70	2	551	480	36
	% difference	0.00			83.30				
	Optimization	44.58		2,681.00	25.00				
18	MA	44.05	15000	2,601.00	1.47	2	551	480	36
	% difference	1.19			23.53				
	Optimization	43.55		3,386.50	364.00				
19	MA	43.55	15000	3,386.50	2.24	2	551	480	36
	% difference	0.00			361.76				
	Optimization	-		-	-				
20	MA	49.22	15000	1,945.50	3.41	2	922	840	48
	% difference								
	Optimization	-		-	-				
21	MA	51.30	15000	2,726.00	3.71	2	922	840	48
	% difference								
	Optimization	-		-	-				
22	MA	50.40	15000	3,471.50	5.13	2	1104	1012	52
	% difference								
	Optimization	-		-	-				
23	MA	51.30	15000	2,726.00	3.33	2	1104	1012	52
	% difference								
	Optimization	-		-	-				
24	MA	56.31	15000	2,980.50	6.13	2	1302	1200	56
	% difference								
	Optimization	-		-	-				
25	MA	54.25	15000	2,976.00	4.13	2	1302	1200	56
	% difference								
	Optimization	-		-	-				
26	MA	56.02	15000	3,455.50	5.69	2	1389	1300	60
	% difference								
	Optimization	-		-	-				
27	MA	55.71	15000	2,536.50	5.74	2	1389	1300	60
	% difference								
	Optimization	-		-	-				
28	MA	52.09	15000	2,491.00	5.91	2	1389	1300	60
	% difference								
	Optimization	-		-	-				
29	MA	60.00	15000	3,416.00	6.52	2	5236	5100	116
	% difference								
	Optimization	-		-	-				
30	MA	65.54	15000	1,865.00	13.34	2	19746	20200	244
	% difference								

Test case	Approach	Score	Budget		Computation	Number of	Number of constraints	Variable (Cplex)	
			Limit	Total	time (second)	day	(Cplex)	Binary	Other
	Optimization	-		-	-				
31	MA	76.05	15000	2,937.00	24.75	2	44016	45300	364
	% difference								
	Optimization	-		-	-				
32	MA	76.49	15000	3,336.00	35.94	2	63761	65884	440
	% difference								
	Optimization	67.45		3,160.50	136.00				
33	MA	63.51	15000	3,120.50	6.00	3	1376	1260	72
	% difference	5.84			130.00				
	Optimization	66.68		4,221.00	71.55				
34	MA	66.37	15000	4,441.00	5.33	3	1376	1260	72
	% difference	0.46			66.22				
	Optimization	-		-	-				
35	MA	64.68	15000	2,986.50	7.47	3	1648	1518	78
	% difference								
	Optimization	-		-	-				
36	MA	63.74	15000	4,181.00	4.01	3	1648	1518	78
	% difference								
	Optimization	-		-	-				
37	MA	77.10	15000	4,906.50	7.84	3	1944	1800	84
	% difference								
	Optimization	-		-	-				
38	MA	74.77	15000	5,531.00	9.67	3	1944	1800	84
	% difference								
	Optimization	-		-	-				
39	MA	77.76	15000	4,250.50	8.00	3	2075	1950	90
	% difference								
	Optimization	-		-	-				
40	MA	77.08	15000	3,450.50	8.01	3	2075	1950	90
	% difference								
	Optimization	-		-	-				
41	MA	73.97	15000	3,766.50	9.51	3	2075	1950	90
	% difference								
	Optimization	-		-	-				
42	MA	82.97	15000	4,867.00	12.59	3	7836	7650	174
	% difference								
	Optimization	-		-	-				
43	MA	96.55	15000	10,155.50	28.52	3	29590	30300	366
	% difference								
	Optimization	-		-	-				
44	MA	109.82	15000	4,713.00	54.45	3	65980	67950	546
	% difference								
	Optimization	-		-	-				
45	MA	112.81	15000	5,247.00	74.17	3	95589	98826	660
	% difference								
	Optimization	78.97		4,236.50	10.75				
46	MA	78.97	15000	4,236.50	8.17	4	2192	2024	104
	% difference	0.00			2.58				
	Optimization	79.12		5,476.00	301.86				
47	MA	78.31	15000	7,016.00	4.76	4	2192	2024	104
	% difference	1.02			297.10				
	Optimization	-		-	-				
48	MA	98.16	15000	6,319.00	13.59	4	10436	10200	232
	% difference								
	Optimization	-		-	-				
49	MA	123.20	15000	5,966.00	32.69	4	39434	40400	488
	% difference								
	Optimization	-		-	-				
50	MA	143.18	15000	5,622.50	75.55	4	87944	90600	728
	% difference								
	Optimization	-		-	-				
51	MA	144.04	15000	7,509.50	98.96	4	127417	131768	880
	% difference								

As shown on Table 5.5, sample 1-6 all of them could be solved by CPLEX. But for sample 7-12 after increasing in hotel and attraction node, the optimization approach could solve 1 day and 3 days tour while 2 days tour, which the day on the middle between 1 day and 3 days, could be solved by optimization approach. Then the further testing was conducted in sample 13-20. In sample 13-20, each sample would have 4 lunch and 4 dinner nodes (increased by 1 from sample 7-12), then testing with the same method. And the result also show that only sample with day in tour on lower and upper bound that CPLEX could solve, while the node on the middle between 1 and 4, which are 2 and 3 days, were unable to be solved. Then the testing on sample 21-26 are conducted. These sample are the sample that have the same number of node as sample 7-12 exception number of attraction. These sample have 4 more attraction nodes when compare to sample 7-12. The reason to create these sample is to check the impact of number of attraction node. The result show that no sample that could be solved by CPLEX and also the same result for sample 27-35. No sample that could be solved by CPLEX. On the other hand, MA could solve all of the sample within a very short period of time when compare to CPLEX. Thus, MA would be used to solve the bigger dataset as on the table (sample 36 – 51). From sample 36 – 51**,** it is shows that the computation time would increase when number of node/choices was increased. However, the largest computation time is only 98.96 seconds for problem of size 181 nodes/choices which larger than most real world problem, so it could be summarized that the MA could solve OPHRSTW efficiently by using a combination of genetic algorithm, feasibility improvement and local search (nearest neighbor search technique) to develop and improve solution quality in each iteration. The MA could solve a problem which bigger than most real world case where there are many choice of places to be selected and also provide a solution that is optimal or close to optimal point within a very short period of time by using the real data gathered from a creditable websites. Thus, MA in this study could be used as a tour planner for agency or group of/individual tourist to maximize a total customer satisfaction score of the tour, which could leads to a tourist revisiting and a recommendation to visit Thailand.

Based on the above result, it could be summarized that the factor that mainly impact the optimization approach are as follows: number of attraction node and day in tour. CPLEX seems to be unable to solve the real world problem since the size of problem (number of choices) that CPLEX could solve is very small (15 – 22 nodes). On the other hands, MA approach seems to be able to use to solve a real world tour planning problem since both computation time and performance are in an acceptable level. A detailed performances comparison were shown on the following tables.

Computation Time (in second)			
Algorithm/Value	Min	Avg	Max
Optimization	10.00	162.80	746.20
MA	1.47	3.88	9.51
Saving in computation time	8.53	158.92	736.69
Ratio of saving in computation time	1.32	47.83	210.20

Figure 5.2 Summary of performance comparison

% Gap between Optimization and MA Approach			
Min	S.D.	Avg	Max
0.00	2.24	1.59	6.34

Figure 5.3 Summary of %gap between Optimization and MA approach

From both Figure 5.2 and Figure 5.3, it shown that saving in computation time when using MA approach on average is 158.92 seconds, while the maximum is 736.69 seconds. And for %gap in total tour score on average is 1.59 with standard deviation of 2.24. As a summary, the MA approach could solve OPHRSTW efficiently since both saving in computation time and %gap are in a good shape and well-balanced.

5.3 Conclusions

As travel & tourism is one of Thailand main industry that contributed 19.7% of GDP. So Thailand should focused to increase or at least maintained the same current level of tourism. So we should find the way to encourage tourist intension to revisit the country and recommend a tour to their relatives. Many studies shoes that the overall satisfaction score of the tour is directly affect the intension to revisit and recommend the tour of tourist. Tour satisfaction could be measured by a survey or reviews by other tourist who used to visit each place. This study collected data from Agoda, Trip

Advisor, and Wongnai which are the most creditable websites based on number of users in Thailand. Since tour planning problem is one of the vehicle routing problem (VRP) family called “orienteering problem with hotel and restaurant selection and time window” (OPHRSTW), so after data collected then the mathematical model (optimization approach) was created for this problem. The purpose is to use it as a tour planner that could create the best tour which provided maximum overall tour satisfaction. However, the real world data is too big for optimization approach to create a tour plan within a reasonable time, so it not practical for all kind of user to waiting for a very long time until the tour created. Thus, a study to find another way to create a tour plan within a reasonable time should be conduct. This study select memetic algorithm (MA) as an alternative way to solve OPHRSTW. MA is a combination between genetic algorithm (GA) and local search technique. GA is one of a machine learning algorithm which approach was inspired by the natural selection rules. Each solution would have a value called “fitness” and this value is a parameter that indicate the chance that this solution would be selected to create a new generation of solution. A solution with a higher fitness value tent to have a higher chance to be selected as a parent this method is called “Selection”. In GA, a new solution (offspring solution) could be created by two method: the first method is called “Crossover”. This method create new solution by combined some part (chromosome) of two existing solution (parent solution) together. The second method called “Mutation”, this method create offspring by changing some part of one parent using part of solution or node outside of the population pool (*Pool*). After each new solution was created, it would be processed through feasibility improvement and local search technique to ensure the feasibility and quality. Then a new solution would be send to a *Pool* which is a place where all solution are stored and waiting to be selected in selection process. After running and testing MA, the result shows that the MA could solve OPHRSTW efficiently as shown on **Table 4 and Table 5**. The MA could provide a solution that is optimal or close to optimal point within a reasonable runtime when compared to optimization approach. Thus, MA is a tool that would satisfy users in terms of quality and time usage in solving OPHRSTW and its related problem with avg. %gap and avg. saving in computation time of 158.92 seconds and 1.59%, respectively. In the future study, this algorithm could be used to solve a more complex tour planning problem such as a problem with mode of

transportation selection, number of attraction per trip, and etc. Moreover, the MA could be combined with a data analytic process to gathered customer insight and set it into the MA parameter to create a personalized tour planning for individual tourist or group of tourist which could leads to more satisfaction which in turn leads to more intension to revisit and recommend the tour to Thailand.

CHAPTER 6
SIGNIFICANT OF THE STUDY

This study will improve tour planning process by providing a tools that allows tour agency to create a better tour plan for individual tourist/group of tourist to obtain a high tourist overall tour satisfaction. Such that tourist could gain a good tour experiences that may leads to their intension to recommend the tour and revisit the region which in turn increase more income for Thai tourism business and other related business. On the other hand, this study also create a contribution to academic community in field of Tourism and Operation research in transportation by providing an efficient metaheuristic approach to solve OPHRSTW and its related problem with avg. %gap and avg. saving in computation time of 158.92 seconds and 1.59%, respectively.

REFERENCES

Bouly, H., Dang, D. C., & Moukrim, A. (2010). A memetic algorithm for the team orienteering problem. *4or*, *8*(1), 49–70. https://doi.org/10.1007/s10288-008-0094-4

Butt, S. E., & Cavalier, T. M. (1994). A heuristic for the multiple tour maximum collection problem. *Computers & Operations Research*, *21*(1), 101–111. https://doi.org/10.1016/0305-0548(94)90065-5

Chao, I. M., Golden, B. L., & Wasil, E. A. (1996). A fast and effective heuristic for the orienteering problem. *European Journal of Operational Research*, *88*(3), 475–489. https://doi.org/10.1016/0377-2217(95)00035-6

Divsalar, A., Vansteenwegen, P., & Cattrysse, D. (2013). A variable neighborhood search method for the orienteering problem with hotel selection. *International Journal of Production Economics*, *145*(1), 150–160. https://doi.org/10.1016/j.ijpe.2013.01.010

Divsalar, A., Vansteenwegen, P., Sörensen, K., & Cattrysse, D. (2014). A memetic algorithm for the orienteering problem with hotel selection. *European Journal of Operational Research*, *237*(1), 29–49. https://doi.org/10.1016/j.ejor.2014.01.001

Gendreau, M., Laporte, G., & Semet, F. (1998). A Branch-and-Cut Algorithm for the Undirected Selective Traveling Salesman Problem. *European Journal of Operational Research*, *106*(2), 539–545.

Golden, B. L., Levy, L., & Vohra, R. (1987). The orienteering problem. *Naval Research Logistics (NRL)*, *34*(3), 307–318. https://doi.org/10.1002/1520-6750(198706)34:3<307::AID-NAV3220340302>3.0.CO;2-D

Gunawan, A., Lau, H. C., & Vansteenwegen, P. (2016). Orienteering Problem: A survey of recent variants, solution approaches and applications. *European Journal of Operational Research*, *255*(2), 315–332. https://doi.org/10.1016/j.ejor.2016.04.059

Kantor, M. G., & Rosenwein, M. B. (1992). The Orienteering Problem with Time Windows. *The Journal of the Operational Research Society*, *43*(6), 629–635.

Kataoka, S., & Morito, S. (1988). AN ALGORITHM FOR SINGLE CONSTRAINT

MAXIMUM COLLECTION PROBLEM. *Journal of the Operations Research Society of Japan*, *31*(4). Retrieved from http://www.orsj.or.jp/~archive/pdf/e_mag/Vol.31_04_515.pdf

Kobeaga, G., Merino, M., & Lozano, J. A. (2018). An efficient evolutionary algorithm for the orienteering problem. *Computers & Operations Research*, *90*, 42–59. https://doi.org/10.1016/J.COR.2017.09.003

Laporte, G., & Martello, S. (1990). The selective travelling salesman problem. *Discrete Applied Mathematics*, *26*(2–3), 193–207. https://doi.org/10.1016/0166-218X(90)90100-Q

Liao, Z., & Zheng, W. (2018). Using a heuristic algorithm to design a personalized day tour route in a time-dependent stochastic environment. Tourism Management 68: 284–300.

Liu, Z. and Park, S. (2015). What makes a useful online review? Implication for travel product websites. Tourism Management, 47, pp.140-151.

Lu, Y., Benlic, U., & Wu, Q. (2018). A Memetic Algorithm for the Orienteering Problem with Mandatory Visits and Exclusionary Constraints. *European Journal of Operational Research*, *268*(1), 54–69. https://doi.org/10.1016/j.ejor.2018.01.019

Marin, J. A., & Taberner, J. G. (2008). Satisfaction and dissatisfaction with destination attributes: Influence on overall satisfaction and the intention to return. *Retrieved December*, *18*, 2011. Retrieved from http://www.esade.edu/cedit/pdfs/papers/pdf6.pdf

Moscato, P. (1999). Memetic algorithms: A short introduction. In D. Corne, M. Dorigo, & F. Glover (Eds.), New Ideas in Optimization (pp. 219–234). McGrw-Hill

Nedjati, Arman, Gokhan Izbirak, and Jamal Arkat. 2017. Bi-objective covering tour location routing problem with replenishment at intermediate depots: Formulation and meta-heuristics. Computers & Industrial Engineering 110: 191–206.

Prins, C. (2004). A simple and effective evolutionary algorithm for the vehicle routing problem. *Computers & Operations Research*, *31*(12), 1985–2002. https://doi.org/10.1016/S0305-0548(03)00158-8

Talbi, E.-G. (2009). *Metaheuristic: from design to implementation*. *John Wiley &*

Sons.

Valle, P. O. D., Silva, J. a., Mendes, J., & Guerreiro, M. (2006). Tourist satisfaction and destination loyalty intention : A structural and categorical analysis. *International Journal of Business Science and Applied Management*, *1*(1), 25–44. Retrieved from http://business-and-management.org/library/2006/1_1--25-44--Oom_do_Valle,Silva,Mendes,Guerreiro.pdf

Vansteenwegen, P., Souffriau, W., & Oudheusden, D. Van. (2011). The orienteering problem: A survey. *European Journal of Operational Research*, *209*(1), 1–10. https://doi.org/10.1016/j.ejor.2010.03.045

Vlachos, G. (2012). Online travel statistics. Info Graphics Mania. Available at: http://infographicsmania.com/online-travel-statistics-2012/

Wu, X., Guan, H., Yan, H., & Ma, J. (2017). A tour route planning model for tourism experience utility maximization. Advances in Mechanical Engineering 9: 1–8.

Xiao, Z., Li, S., Fan, Y., Liu, B., Zhang, B., & Li, B. (2017). Tourism Route Decision Support Based on Neural Net Buffer Analysis. Procedia Computer Science 107: 243–47.

Yan, H., Hongzhi, G., & Duan, J. (2014). Tour Route Multiobjective Optimization Design Based on the Tourist Satisfaction Discrete Dynamics in Nature and Society 2014: 603494.

Ye, Q., Law, R., Gu, B. and Chen, W. (2011). The influence of user-generated content on traveler behavior: An empirical investigation on the effects of e-word-of-mouth to hotel online bookings. Computers in Human Behavior, 27, pp. 634-639.

Yoon, Y., & Uysal, M. (2005). An examination of the effects of motivation and satisfaction on destination loyalty: A structural model. *Tourism Management*, *26*(1), 45–56. https://doi.org/10.1016/j.tourman.2003.08.016

Zhao, H., Xu, W. (Ato), & Jiang, R. (2015). The Memetic algorithm for the optimization of urban transit network. *Expert Systems with Applications*, *42*(7), 3760–3773. https://doi.org/10.1016/J.ESWA.2014.11.056

Zhu, C., Hu, J. Q., Wang, F., Xu, Y., & Cao, R. (2012). On the tour planning problem. *Annals of Operations Research*, *192*(1), 67–86. https://doi.org/10.1007/s10479-010-0763-5

APPENDICES

APPENDIX A

DATA OF HOTEL USED IN THIS STUDY

ID	Star	Name	Price	Sum Score	Province
1	3	Royal River Kwai Resort & Spa	1750	4.63	Kanchanaburi
2	3	Sky Resort Kanchanaburi	800	5.65	Kanchanaburi
3	3	Sabai @ kan Resort	1456	4.80	Kanchanaburi
4	3	River Kwai Park & Resort	1200	4.83	Kanchanaburi
5	3	The Glory River Kwai Hotel	2300	5.18	Kanchanaburi
6	4	Felix River Kwai Resort	1750	4.53	Kanchanaburi
7	4	U Inchantree Kanchanaburi Hotel	800	4.84	Kanchanaburi
8	4	River Kwai Bridge & Resort	1456	4.82	Kanchanaburi
9	4	Baannommaew Retreat Kanchanaburi	1200	5.16	Kanchanaburi
10	4	The Vista Pool Villa	2300	4.92	Kanchanaburi
11	5	X2 River Kwai Resort	6400	5.48	Kanchanaburi
12	5	Dheva Mantra Resort	8400	4.68	Kanchanaburi
13	3	Tai-Shan Suites	700	6.25	Ratchaburi
14	3	Vanilla River Ratchaburi	600	6.10	Ratchaburi
15	3	Space59 Hotel	800	5.23	Ratchaburi
16	3	The Great Hotel & Resort	980	5.55	Ratchaburi
17	3	Damnoen Maikaew	1800	4.76	Ratchaburi
18	3	Phanya Chaley Suanphueng	2400	5.85	Ratchaburi
19	3	Molloft Resort	1900	5.55	Ratchaburi
20	3	Baan Proud Fha	1920	5.51	Ratchaburi
21	3	Ozono Resort	1920	5.21	Ratchaburi
22	4	S.Swiss Hotel Ratchaburi	1000	6.05	Ratchaburi
23	4	Navela Hotel and Banquet	1500	5.95	Ratchaburi
24	4	La Toscana Resort	4061	5.51	Ratchaburi
25	4	Villa Moreeda	2688	5.04	Ratchaburi
26	4	La Provence Suan Phung Hotel	2688	5.38	Ratchaburi
27	4	Tammarene Villa	2196	5.68	Ratchaburi
28	3	The september salaya	1000	4.98	Nakhon Pathom
29	3	The Pround Exclusive Hotel	880	5.41	Nakhon Pathom
30	3	Suwan Golf & Country Club	1379	4.76	Nakhon Pathom
31	3	Parn Dhevi Riverside	1793	4.70	Nakhon Pathom
32	3	Xen Hotel Nakhon Pathom	900	5.43	Nakhon Pathom
33	3	The Royal Gem Glof Club	1700	4.18	Nakhon Pathom
34	4	Sampran Riverside	2300	4.44	Nakhon Pathom
35	4	Sun love Resort & Spa Grand view	1200	5.23	Nakhon Pathom
36	4	Sun love Resort & Spa Royal view	1300	5.11	Nakhon Pathom
37	4	Silver Woods Hotel	1152	4.42	Nakhon Pathom
38	4	Uniland Golf & Resort	1100	4.70	Nakhon Pathom
39	5	Mida Dhavaravati Grand hotel	2300	3.28	Nakhon Pathom

APPENDIX B

DATA OF RESTAURANT USED IN THIS STUDY

ID	Star	Price	Avg Price	Name	Open-Close Time	Province
1	3	100	100	ซุ่นเฮง ผัดไทยไร้เส้น (Soon Heng)	Every Day 10:00am-16:00pm	Kanchanaburi
2	3	250	250	แพอาหารโฟลทติ้ง (Floating Raft Restaurant)	Every Day 9:00am-22:00pm	Kanchanaburi
3	3	250	175	ร้านอาหารบ้านเรา (Ban Rao Restaurant)	Every Day 7:00am-20:00pm	Kanchanaburi
4	3	100	100	ครัวริมน้ำ คุณสุริยา (Khrua Rim Nam Khun Suriya Resturant)	Every Day 11:00am-21:00pm	Kanchanaburi
5	3	100	100	ยิ้มไอศครีม (Yim ICECREAM)	Every Day 10:00am-21:00pm	Kanchanaburi
6	3	250	175	ร้านอาหารครัวหัวนา (KHRUA HUA NA RESTAURANT)	Every Day 8:00am-23:00pm	Kanchanaburi
7	3	250	175	Mangosteen Garden Restaurant	Every Day 9:30am-22:00pm	Kanchanaburi
8	3	250	175	ครัวกาญ (Krua Karn)	Every Day 10:00am-22:00pm	Kanchanaburi
9	3	100	100	ก๋วยเตี๋ยวเรือ เป๋าปาก (Pak Pao)	Every Day 10:00am-22:00pm	Kanchanaburi
10	4	246	246	Kan Buri	Every Day 10:00am-22:00pm	Kanchanaburi
11	4	106-246	176	Nut's Restaurant	Every Day 8:00am-12:00am	Kanchanaburi
12	4	100	100	Prik Kaeng Restaurant	Every Day 11:00am-21:00pm	Kanchanaburi
13	4	387-633	510	Good Times Restaurant	Every Day 10:00am-22:00pm	Kanchanaburi
14	4	100-250	175	ร้านอาหาร คีรีธารา (Kere Tara)	Every Day 11:00am-23:00pm	Kanchanaburi
15	4	100	100	อบอุ่น เฮ้าส์ (Ob Oon Home Bakery)	Every Day 11:00am-23:00pm	Kanchanaburi
16	4	100-250	175	Library Cafe' (ไลบรารี่ คาเฟ่)	Every Day 9:00am-23:30pm	Kanchanaburi
17	4	100-250	175	ไทยเสรี (Thaiseri)	Every Day 9:30am-21:00pm	Kanchanaburi
18	4	250-500	375	ครัวชุกโดน (Krua Chuka Done)	Every Day 10:30am-22:30pm	Kanchanaburi
19	5	70	70	On's Thai Issan	Every Day 10:00am-22:00pm	Kanchanaburi
20	3	0 - 100	50	ก๋วยเตี๋ยวไข่เมืองราชบุรี คุณแหม่ม (Egg Noodle Khun Mam)	Every Day 10.00am - 21.00pm	Ratchaburi
21	3	0 - 100	50	เวียดนามเฮ้าส์ (Vietnam House)	Every Day 10.00am - 21.00pm	Ratchaburi
22	3	101 - 250	175.5	มุมโปรด @สวนผึ้ง (Moom Prod)	Every Day 10.00am - 21.00pm	Ratchaburi
23	3	0 - 100	50	ดงยาง บ้านโป่ง (Dong Yang)	Every Day 10.00am - 21.00pm	Ratchaburi
24	3	101 - 250	175.5	บ้านริมน้ำ (Ban Rim Nam)	Every Day 9.00am - 20.00pm	Ratchaburi
25	4	251 - 500	375.5	ครัวจัดจ้าน (judjarn)	Every Day 11.00am - 23.00pm	Ratchaburi
26	4	101 - 250	175.5	ครัวม่อนไข่ (Krua Mon Khai)	Every Day 9.00am - 20.00pm	Ratchaburi
27	4	101 - 250	175.5	ป.ปลาเผา (Tha Rap)	Every Day 11.00am - 22.00pm	Ratchaburi
28	4	0 - 100	50	ครัวลุงด้อ (khrua-lung-do-ahan-pa)	Every Day 9.00am - 17.00pm	Ratchaburi
29	5	100 - 999	549.5	Inlaya Bar and Grill	Every Day 9.00am - 20.00pm	Ratchaburi
30	5	101 - 250	175.5	ไหมไทย (MaïThaiRestaurant)	Every Day 12.00am - 24.00pm	Ratchaburi
31	5	501 - 1000	750.5	Alpaca Ratchaburi	Every Day 11.00am - 22.00pm	Ratchaburi
32	5	101 - 250	175.5	ร้านเจ๊ออน (OrnRestaurant)	Every Day 10.00am - 20.00pm	Ratchaburi
33	5	50 - 300	175	New land	Every Day 9.00am - 19.00pm	Ratchaburi
34	3	100	100	เป็ดพะโล้ นายหนับ (Nai Nub Ped Palo) ดอนหวาย	Every Day 7:00am-16:00pm	Nakhon Pathom
35	3	250-500	375	Tama Yakiniku (ทามายากินิคุ)	Every Day 9.00am - 20.00pm	Nakhon Pathom
36	3	100-250	175	แพพิณเทอง (Pae Pin Thong)	Every Day 10:00am-22:00pm	Nakhon Pathom
37	3	100-250	175	ติ๊กโภชนา (Tik Restaurant)	Every Day 10:00am-22:00pm	Nakhon Pathom
38	3	100-250	175	แพริมน้ำชวัล (Chaval River Side)	Every Day 10:30am-22:00pm	Nakhon Pathom
39	4	251 - 500	375.5	บ้านริมบึง (Ban Rim Bueng)	Every Day 11.00am - 23.00pm	Nakhon Pathom
40	4	0 - 100	50	เฉินหลงข้าวต้มเงินล้าน (Cheng Long)	Every Day 10.00am - 23.30pm	Nakhon Pathom
41	4	101 - 250	175.5	little Tree Garden	Every Day 9.00am - 18.00pm	Nakhon Pathom
42	4	501 - 1000	750.5	กุ้งอบภูเขาไฟ (Volcano Prawn Restaurant)	Every Day 10.00am - 22.00pm	Nakhon Pathom
43	4	251 - 500	375.5	Riva Floting Café	Every Day 9.00am - 20.00pm	Nakhon Pathom
44	5	251-500	375.5	Anya's Place	Every Day 11.00am - 23.00pm	Nakhon Pathom
45	5	251-500	375.5	D'Eiffle	Every Day 10:00am-22:00pm	Nakhon Pathom
46	5	0 - 100	50	ข้าวขาหมูบางหว้า นายต่อง (Khao Kha Moo Nai Tong)	Every Day 6.00am - 16.30pm	Nakhon Pathom
47	5	251-500	375.5	หมูกระป๋องบุฟเฟต์ (Mookrapong Buffet)	Every Day 16.00pm - 22.00pm	Nakhon Pathom
48	5	251-500	375.5	อู่ข้าว อู่ปลา (U Kwow U Pla)	Every Day 10:00am-22:00pm	Nakhon Pathom

APPENDIX C

DATA OF RESTAURANT FOR DINNER USED IN THIS STUDY

ID	Star	Price	Name	Open-Close Time	Province
1	3	250	แพอาหารโฟลทติ้ง (Floating Raft Restaurant)	Everyday 9:00am-22:00pm	Kanchanaburi
2	3	175	ร้านอาหารบ้านเรา (Ban Rao Restaurant)	Everyday 7:00am-20:00pm	Kanchanaburi
3	3	100	ครัวริมน้ำ คุณสุริยา (Khrua Rim Nam Khun Suriya Resturant)	Everyday 11:00am-21:00pm	Kanchanaburi
4	3	100	ยิ่มไอศครีม (Yim ICECREAM)	Everyday 10:00am-21:00pm	Kanchanaburi
5	3	175	ร้านอาหารครัวหัวนา (KHRUA HUA NA RESTAURANT)	Everyday 8:00am-23:00pm	Kanchanaburi
6	3	175	Mangosteen Garden Restaurant	Everyday 9:30am-22:00pm	Kanchanaburi
7	3	100	บายเดอริเวอร์ (By de River)	Everyday 17:00pm-23:00pm	Kanchanaburi
8	3	175	ครัวกาญ (Krua Karn)	Everyday 10:00am-22:00pm	Kanchanaburi
9	3	100	ก๋วยเตี๋ยวเรือ เป๋าปาก (Pak Pao)	Everyday 10:00am-22:00pm	Kanchanaburi
10	4	176	Bell's Pizzeria	Everyday 16:00pm-12:00am	Kanchanaburi
11	4	246	Kan Buri	Everyday 11:00am-21:00pm	Kanchanaburi
12	4	100	Prik Kaeng Restaurant	Everyday 11:00am-21:00pm	Kanchanaburi
13	4	510	Good Times Restaurant	Everyday 16:00pm-12:00am	Kanchanaburi
14	4	175	ร้านอาหาร คีรีธารา (Kere Tara)	Everyday 11:00am-23:00pm	Kanchanaburi
15	4	100	อบอุ่น เฮ้าส์ (Ob Oon Home Bakery)	Everyday 10:00am-22:00pm	Kanchanaburi
16	4	175	Library Cafe' (ไลบรารี่ คาเฟ่)	Everyday 9:00am-23:30pm	Kanchanaburi
17	4	175	ไทยเสรี (Thaiseri)	Everyday 9:30am-21:00pm	Kanchanaburi
18	4	375	ครัวชุกโดน (Krua Chuka Done)	Everyday 10:30am-22:30pm	Kanchanaburi
19	5	70	On's Thai Issan	Everyday 10:00am-22:00pm	Kanchanaburi
20	3	50	ก๋วยเตี๋ยวไข่เมืองราชบุรี คุณแหม่ม (Egg Noodle Khun Mam)	Everyday 10:00am-22:00pm	Ratchaburi
21	3	50	เวียดนามเฮ้าส์ (Vietnam House)	Everyday 10.00am - 21.00pm	Ratchaburi
22	3	175.5	มุมโปรด @สวนผึ้ง (Moom Prod)	Everyday 8.30 - 20.30	Ratchaburi
23	3	50	ดงยาง บ้านโป่ง (Dong Yang)	Everyday 10.00am - 21.00pm	Ratchaburi
24	3	175.5	บ้านริมน้ำ (Ban Rim Nam)	Everyday 10:00am-22:00pm	Ratchaburi
25	4	375.5	ครัวจัดจ้าน (judjarn)	Everyday 11.00am - 23.00pm	Ratchaburi
26	4	175.5	ครัวม่อนไข่ (Krua Mon Khai)	Everyday 9.00am - 20.00pm	Ratchaburi
27	4	175.5	ป.ปลาเผา (Tha Rap)	Everyday 11.00am - 22.00pm	Ratchaburi
28	4	50	ครัวลุงด้อ (khrua-lung-do-ahan-pa)	Everyday 11.00am - 22.00pm	Ratchaburi
29	4	175.5	ลองชาม (Longcharm)	Everyday 10:00am-22:00pm	Ratchaburi
30	5	549.5	Inlaya Bar and Grill	Everyday 10:00am-22:00pm	Ratchaburi
31	5	175.5	ไหมไทย (MaiThaiRestaurant)	Everyday 12.00am - 24.00pm	Ratchaburi
32	5	750.5	Alpaca Ratchaburi	Everyday 11.00am - 22.00pm	Ratchaburi
33	5	175.5	ร้านเจ๊ออน (OrnRestaurant)	Everyday 10.00am - 20.00pm	Ratchaburi
34	5	175	New land	Everyday 11.00am - 22.00pm	Ratchaburi
35	3	100	เป็ดพะโล้ นายหนับ (Nai Nub Ped Palo) ดอนหวาย	Everyday 11.00am - 22.00pm	Nakhon Pathom
36	3	375	Tama Yakiniku (ทามายากินิคุ)	Everyday 10:00am-22:00pm	Nakhon Pathom
37	3	175	แพพิณทอง (Pae Pin Thong)	Everyday 10:00am-22:00pm	Nakhon Pathom
38	3	175	ติ๊กโภชนา (Tik Restaurant)	Everyday 11.00am - 23.00pm	Nakhon Pathom
39	3	175	แพริมน้ำชวัล (Chaval River Side)	Everyday 10:30am-22:00pm	Nakhon Pathom
40	4	375.5	บ้านริมบึง (Ban Rim Bueng)	Everyday 11.00am - 23.00pm	Nakhon Pathom
41	4	50	เฉินหลงข้าวต้มเงินล้าน (Cheng Long)	Everyday 10.00am - 23.30pm	Nakhon Pathom
42	4	175.5	little Tree Garden	Everyday 11.00am - 22.00pm	Nakhon Pathom
43	4	750.5	กุ้งอบภูเขาไฟ (Volcano Prawn Restaurant)	Everyday 10.00am - 22.00pm	Nakhon Pathom
44	4	375.5	Riva Floting Café	Everyday 10:00am-22:00pm	Nakhon Pathom
45	5	375.5	Anya's Place	Everyday 11.00am - 23.00pm	Nakhon Pathom
46	5	375.5	D'Eiffle	Everyday 11.00am - 23.00pm	Nakhon Pathom
47	5	50	ข้าวขาหมูบางหว้า นายต่อง (Khao Kha Moo Nai Tong)	Everyday 11.00am - 22.00pm	Nakhon Pathom
48	5	375.5	หมูกระป๋องบุฟเฟต์ (Mookrapong Buffet)	Everyday 16.00pm - 22.00pm	Nakhon Pathom
49	5	375.5	อู่ข้าว อู่ปลา (U Kwow U Pla)	Everyday 10:00am-22:00pm	Nakhon Pathom

APPENDIX D

DATA OF ATTRACTION USED IN THIS STUDY

ID	Star	Open	Close	Price	Name	Province
1	3	8.00	18.00	10	พิพิธภัณฑ์อักษะเชลยศึก (The JEATH War Museum)	Kanchanaburi
2	3	7.00	18.00	20	พิพิธภัณฑ์สงครามโลก (Arts Gallery and War Museum)	Kanchanaburi
3	3	7.00	18.00	0	วัดสระลงเรือ (Wat Sa Long Ruea)	Kanchanaburi
4	3	0.00	99999.00	200	น้ำตกไทรโยคใหญ่ (Sai Yok Yai Water Fall)	Kanchanaburi
5	3	6.00	18.00	100	เขื่อนศรีนครินทร์ (Srinagarind Dam)	Kanchanaburi
6	4	9.00	17.00	200	สวนสัตว์เปิด ซาฟารี ปาร์ค แอนด์ แคมป์ (Safari Park)	Kanchanaburi
7	4	8.00	18.30	0	สะพานข้ามแม่น้ำแคว (The Bridge of the River Kwai)	Kanchanaburi
8	4	8.00	17.00	120	สุสานทหารสัมพันธมิตรดอนรัก (Kanchanaburi War Cemetery)	Kanchanaburi
9	4	6.00	18.00	20	อุทยานแห่งชาติถ้ำธารลอด (Chaloem Rattanakosin National Park)	Kanchanaburi
10	4	7.00	22.00	0	วัดวังก์วิเวการาม (Wat Wang Wiwekaram)	Kanchanaburi
11	5	10.00	17.00	0	สวนสันติภาพชินโตะ (Santiphap Park)	Kanchanaburi
12	5	9.00	19.00	150	เมืองมัลลิกา ร.ศ.124 (Malika R.E.124)	Kanchanaburi
13	5	9.00	17.00	140	พิพิธภัณฑ์ทางรถไฟไทย-พม่า (Thailand–Burma Railway Centre)	Kanchanaburi
14	5	8.00	17.00	0	วัดทิพย์สุคนธาราม (wat-thipsukhontharam)	Kanchanaburi
15	5	0.00	99999.00	40	อุทยานแห่งชาติทองผาภูมิ (Thong Pha Phum National Park)	Kanchanaburi
16	3	7.00	12.00	0	ตลาดน้ำดำเนินสะดวก (Damnoen Saduak Floating Market)	Ratchaburi
17	3	8.00	17.00	0	หมู่บ้านช้างดำเนินสะดวก (Chang Damnoen Saduak Housing)	Ratchaburi
18	3	8.00	18.00	5	ธารน้ำร้อนบ่อคลึง (Hot Springs Resort)	Ratchaburi
19	3	8.00	18.00	50	เฟอราซ่าฟาร์ม (Ferraza Farm เฟอร์ราซ่าฟาร์ม)	Ratchaburi
20	3	8.30	17.00	50	อุทยานหุ่นขี้ผึ้งสยาม (siam-cultural-park)	Ratchaburi
21	4	9.00	16.00	150	สวนเสือดำเนินสะดวก (damnoensaduak-tigerzoo)	Ratchaburi
22	4	9.00	18.00	0	โคโรฟิลด์ (CORO Field)	Ratchaburi
23	4	8.30	18.30	50	เดอะซีนเนอรี่ วินเทจ ฟาร์ม (The Scenery Resort)	Ratchaburi
24	4	7.00	18.00	0	วัดมหาธาตุวรวิหาร (Wat Mahathat Worrawihan)	Ratchaburi
25	4	8.00	16.00	20	ถ้ำเขาบิน (Khao Bin Cave)	Ratchaburi
26	5	8.00	17.00	40	ถ้ำสาริกา (Wat Tham Sarika)	Ratchaburi
27	5	9.00	22.00	190	อัลปาก้า ฮิลล์ (alpacahill)	Ratchaburi
28	5	9.00	17.00	80	สุนทรีแลนด์ แดนตุ๊กตา (suntreelandofdolls)	Ratchaburi
29	5	8.00	18.00	0	ตลาดน้ำเหล่าตั๊กลัก (laotukluck)	Ratchaburi
30	5	8.30	17.00	0	วัดถ้ำน้ำ (Wat Tham Nam)	Ratchaburi
31	3	8.30	17.30	80	ลานแสดงช้างและฟาร์มจระเข้สามพราน (Yard Elephant and Crocodile Farm)	Nakhon Pathom
32	3	9.30	18.00	40	พิพิธภัณฑ์ศิลปะนกฮูก (Owl Art Museum)	Nakhon Pathom
33	3	9.30	18.00	290	สวนน้ำจูราสสิค (Jurassic Water Park)	Nakhon Pathom
34	3	8.30	16.30	0	วัดพระเมรุ (Wat Na Phra Men)	Nakhon Pathom
35	3	8.00	17.30	50	สวนนกยูงสามพราน (samphran-peacock-park)	Nakhon Pathom
36	4	8.00	20.00	0	ตลาดน้ำลำพญา (Lam Phaya Floating Market)	Nakhon Pathom
37	4	8.30	18.00	80	พิพิธภัณฑ์หุ่นขี้ผึ้งไทย (thaiwaxmuseum)	Nakhon Pathom
38	4	9.00	17.00	0	พิพิธภัณฑ์รถโบราณ เจษฎาเทคนิคมิวเซียม (Jesada Technik Museum)	Nakhon Pathom
39	4	0.00	99999.00	0	วัดไร่ขิง (Rai Khing temple)	Nakhon Pathom
40	4	0.00	99999.00	0	วัดสามพราน (Wat Samphran)	Nakhon Pathom
41	5	8.00	16.30	0	วัดพระปฐมเจดีย์ (Phra Pathommachedi)	Nakhon Pathom
42	5	9.30	17.00	300	วู้ดแลนด์เมืองไม้ (Woodland Museum & Resort)	Nakhon Pathom
43	5	9.00	16.00	30	พระราชวังสนามจันทร์ (Sanam Chandra Palace)	Nakhon Pathom
44	5	7.00	18.00	0	ตลาดน้ำวัดดอนหวาย (Don Wai Floating Market)	Nakhon Pathom
45	5	5.00	19.00	0	พุทธมณฑล (Phutthamonthon)	Nakhon Pathom

APPENDIX D

EXAMPLE OF DATA OF TRAVEL TIME BETWEEN EACH NODE

Please check https://drive.google.com/file/d/1I-gayTtNMt5a44URk15sGNiEZDeS3S7k/view?usp=sharing for traveling time data.

ID	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	0.001	0.24889	0.22056	0.76389	0.42083	0.001	0.24889	0.22056	0.76389	0.42083	0.42083	0.13111	1.66972	1.63528
2	0.24	0.001	0.10028	0.87361	0.31167	0.24	0.001	0.10028	0.87361	0.31167	0.31167	0.18139	1.57778	1.54333
3	0.19694	0.09639	0.001	0.83083	0.33389	0.19694	0.09639	0.001	0.83083	0.33389	0.33389	0.15222	1.61167	1.57722
4	0.73972	0.86833	0.83972	0.001	0.9725	0.73972	0.86833	0.83972	0.001	0.9725	0.9725	0.84333	2.27111	2.23667
5	0.41833	0.30056	0.33278	0.96833	0.001	0.41833	0.30056	0.33278	0.96833	0.001	0.1	0.29139	1.31194	1.31056
6	0.2	0.16528	0.13972	0.89944	0.31778	0.2	0.16528	0.13972	0.89944	0.31778	0.31778	0.07917	1.60194	1.60056
7	0.16222	0.15389	0.10472	0.83167	0.41833	0.16222	0.15389	0.10472	0.83167	0.41833	0.41833	0.18417	1.63611	1.60167
8	0.1925	0.11917	0.02278	0.82639	0.35667	0.1925	0.11917	0.02278	0.82444	0.35667	0.35667	0.175	1.62	1.58556
9	0.56861	0.45111	0.48333	1.11861	0.17889	0.56861	0.45111	0.48333	1.11861	0.17889	0.17889	0.44194	1.3325	1.33111
10	0.43278	0.33056	0.35694	1.05278	0.54278	0.43278	0.33056	0.35694	1.05278	0.54278	0.54278	0.45528	1.31417	1.27972
11	0.41833	0.30056	0.33278	0.96833	0.54278	0.41833	0.30056	0.33278	0.96833	0.001	0.1	0.29139	1.31194	1.31056
12	0.13972	0.17917	0.15361	0.83917	0.30167	0.13972	0.17917	0.15361	0.83917	0.30167	0.30167	0.001	1.58583	1.58444
13	1.7075	1.56889	1.59528	2.2575	1.31778	1.7075	1.56889	1.59528	2.2575	1.31778	1.31778	1.58056	0.001	0.15861
14	1.71278	1.57417	1.60056	2.26278	1.32306	1.71278	1.57417	1.60056	2.26278	1.32306	1.32306	1.58583	0.16556	0.001

APPENDIX F

CPLEX CODE FOR OPTIMIZATION APPROACH

Please check https://drive.google.com/file/d/1ooSs42bJ-4raluVRIWcBr50zSavrKZIg/view?usp=sharing CPLEX code and test data

```
{int} H = ...;
{int} Att = ...;
{int} RL = ...;
{int} RD = ...;
{int} AnR = Att union RL union RD;
{int} AnH = Att union H;
{int} AnL = Att union RL;
{int} AnD = Att union RD;
{int} V = H union Att union RL union RD;
int D = 4;
range d = 1..D;
int MaxL = 10;

float B = 100000;
float TS[V]=...;
float OpT[V]=...;
float ClT[V]=...;
float EC[V]=...;
float S[V]=...;
float TrT[V][V]=...;
float M = 99999999;

dvar boolean x[V][V][d];
dvar boolean y[Att][d];
dvar boolean z[H][d];
dvar boolean lu[RL][d];
dvar boolean di[RD][d];
dvar float ArrT[V][d];
dvar float ArrTW[H][d];

maximize sum(j in Att, k in d)S[j]*y[j][k]
        +sum(j in H, k in d: k != D)S[j]*z[j][k]
        +sum(j in RL, k in d)S[j]*lu[j][k]
        +sum(j in RD, k in d)S[j]*di[j][k];
```

```
subject to {

    //-----------------1.Path and connectivity constraint-------------
        CT1:sum(j in AnL)x[1][j][1] == 1;
        CT2:sum(i in AnD)x[i][2][D] == 1;
        CT3:forall(k in d){
            CT4:sum(i in V, j in H)x[i][j][k] == 1;}
        CT5:forall(k in d){
            CT6:sum(i in H, j in V)x[i][j][k] == 1;}
        CT7:forall(i in AnR, k in d){
            CT8:sum(j in V)x[i][j][k] == sum(j in V)x[j][i][k];}
        CT9:forall(j in AnR){
            CT10:sum(i in V, k in d)x[i][j][k] <= 1;}
    //-----------------2.Tourist Site Constraint----------------------
        CT11:forall(j in Att, k in d){
            CT12:y[j][k] == sum(i in V: i != j) x[i][j][k];}
        CT13:forall(k in d){
            CT14:sum(i in V, j in Att)x[i][j][k]+1 <= MaxL;}
    //-----------------3.Hotel Constraint-----------------------------
        CT15:forall(i in H,k in d: k != D){
            CT16:sum(j in V)x[i][j][k+1] == sum(j in V)x[j][i][k];}
        CT17:forall(j in H,k in d){
            CT18:z[j][k] == sum(i in AnD)x[i][j][k];}
    //----------------4.Restaurant Constraint (Lunch)----------------
        CT19:forall(k in d){
            CT20:sum(i in V, j in RL)x[i][j][k] == 1;}
        CT21:forall(j in RL, k in d){
            CT22:lu[j][k] == sum(i in AnH)x[i][j][k];}
    //----------------5.Restaurant Constraint (Dinner)---------------
        CT23:forall(k in d){
            CT24:sum(i in V, j in RD)x[i][j][k] == 1;}
        CT25:forall(j in RD, k in d){
            CT26:di[j][k] == sum(i in AnL)x[i][j][k];}
    //----------------6.Time Constraint------------------------------
        CT27:forall(i in V,j in AnR, k in d){
            CT28:ArrT[i][k] - ArrT[j][k] + TrT[i][j] + TS[j] <= (1-x[i][j][k])*M;}
        CT29:forall(i in AnR,j in H, k in d){
            CT30:ArrT[i][k] - ArrTW[j][k] + TrT[i][j] + TS[j] <= (1-x[i][j][k])*M;}
        CT31:forall(i in V, k in d){
            CT32:OpT[i] <= ArrT[i][k] <= ClT[i];}
        CT33:forall(i in H, k in d){
            CT34:OpT[i] <= ArrTW[i][k] <= ClT[i];}
    //----------------7.Buget Constraint-----------------------------
        CT35:forall(k in d){
            CT36:sum(j in Att)EC[j]*y[j][k]
            +sum(j in H)EC[j]*z[j][k]
            +sum(j in RL)EC[j]*lu[j][k]
            +sum(j in RD)EC[j]*di[j][k]<= B;}

}
```

```
execute
{
  var fileName = "set5-25Output"+D+".csv";
  var Output = new IloOplOutputFile(fileName);
  Output.writeln("Objective Value",cplex.getObjValue());
  Output.writeln("Go to",x);
  Output.writeln("Selected Place",y);
  Output.writeln("Selected Hotel",z);
  Output.writeln("Time Window",ArrT);
  Output.writeln("Selected Lunch",lu);
  Output.writeln("Selected Dinner",di);
}
```

APPENDIX G

MA CODE WRITTEN IN PYTHON PROGRAMMING LANGUAGE

Please check https://drive.google.com/file/d/1GpxcJG0Bg7Ro1-AMtWvtpTK7LBOmuBwv/view?usp=sharing for the code and test data.

1CV2-Data Management.py

```
def InPut(dat):
    from openpyxl import load_workbook
    Data = load_workbook(dat)

    '''Start Chain'''
    Hotel = Data['Hotel']
    global h
    h=[]
    for i in range(0,len(Hotel['B'])-1):
        h.append(i)
    x=len(Hotel['B'])-1
    del Hotel

    Place = Data['Place']
    global p
    p=[]
    for i in range(x,x+len(Place['B'])-3):
        p.append(i)
    x+=len(Place['B'])-3
    del Place

    Lunch = Data['Lunch']
    global l
    l=[]
    for i in range(x,x+len(Lunch['B'])-1):
        l.append(i)
    x+=len(Lunch['B'])-1
    del Lunch

    Dinner = Data['Dinner']
    global d
    d=[]
    for i in range(x,x+len(Dinner['B'])-1):
        d.append(i)
    x+=len(Dinner['B'])-1
    del Dinner

    Score = Data['Score']
```

```
    global s
    s=[]
    for i in range(2,len(Score['B'])+1):
        s.append(Score.cell(row=i,column=3).value)
    del Score

    Price = Data['Price']
    global pr
    pr=[]
    for i in range(2,len(Price['B'])+1):
        pr.append(Price.cell(row=i,column=3).value)
    del Price

    TimeSpend = Data['TimeSpend']
    global t
    t=[]
    for i in range(2,len(TimeSpend['B'])+1):
        t.append(TimeSpend.cell(row=i,column=3).value)
    del TimeSpend

    TimeWin = Data['TimeWin']
    global o
    global c
    o=[]
    c=[]
    for i in range(2,len(TimeWin['B'])+1):
        o.append(TimeWin.cell(row=i,column=3).value)
        c.append(TimeWin.cell(row=i,column=4).value)
    del TimeWin

    global tt
    tt = Data['TrTime']
    del Data
    print('All of the data are imported')
```

CombineLocalSearch.py

```
def LocalSearchImprovement(Node,Budget,h,p,l,d,s,pr,t,o,c,tt,DayInTour,
OriginalTotalScore):
    Method=0
    tempNode=Node.copy()
    finalResult=Node.copy()
    tempTotalScore=0
    ScoreM1, ScoreM2, ScoreM3, ScoreM4, ScoreM5 = 0,0,0,0,0
    nodem1,nodem2,nodem3,nodem4,nodem5=[],[],[],[],[]
    tempTabu=[]
    improveFlag=False
```

```
   while (Method<5):
      if Method == 0:

nodem1,ScoreM1=Insertion(tempNode,Budget,h,p,l,d,s,pr,t,o,c,tt,DayInTour)#1. Real
Insertion
         if tempTotalScore>OriginalTotalScore:
            OriginalTotalScore=ScoreM1
            tempNode=nodem1
            finalResult=nodem1
            Method=0
         else:
            Method+=1
      if Method == 1:

nodem2,ScoreM2,improveFlag=TwoOpt(tempNode,Budget,h,p,l,d,s,pr,t,o,c,tt,DayIn
Tour)#2. Two-Opt between 2 trips, switch attraction with attraction and Res with Res
         if improveFlag:
            OriginalTotalScore=ScoreM2
            tempNode=nodem2
            finalResult=nodem2
            Method=0
         else:
            Method+=1
      if Method == 2:

nodem3,ScoreM3,improveFlag=MoveBest(tempNode,Budget,h,p,l,d,s,pr,t,o,c,tt,DayI
nTour)#3. Move best, try to move each attraction to the best position in trip or tour
         if improveFlag:
            OriginalTotalScore=ScoreM3
            tempNode=nodem3
            finalResult=nodem3
            Method=0
         else:
            Method+=1
      if Method == 3:

nodem4,ScoreM4,improveFlag=SwapNSwapBest(tempNode,Budget,h,p,l,d,s,pr,t,o,c,
tt,DayInTour)#4. Swap and Swap Best
         if improveFlag:
            OriginalTotalScore=ScoreM4
            tempNode=nodem4
            finalResult=nodem4
            Method=0
         else:
            Method+=1
      if Method == 4:
```

```
nodem5,ScoreM5,tabuM5=extractinsert(tempNode,Budget,h,p,l,d,s,pr,t,o,c,tt,DayInTour,tempTabu)#5. Extract-Insert
        if ScoreM5>OriginalTotalScore:
            OriginalTotalScore=ScoreM5
            tempNode=nodem5
            tempTabu=tabuM5
            finalResult=nodem5
            Method=0
        else:
            Method+=1
    return finalResult, tempNode, OriginalTotalScore, ScoreM1, ScoreM2, ScoreM3, ScoreM4, ScoreM5,nodem1,nodem2,nodem3,nodem4,nodem5
```

3. CreatingIniPop.py

```
import ast
from openpyxl import load_workbook
from random import choices
from ImproveGAOffspring import *
from Utility import *
from Recursive import *
from LocalSearchV2 import *
from CombineLocalSearch import *

"Ask for initial and final location"
Ini=int(input('Please Enter your initial location here: '))
Final=int(input('Please Enter your final location here: '))
DayInTour=int(input('How many day that you want to spend: '))
Budget=int(input('Please enter budget: '))
'''DayInTour-2'''

Data=load_workbook('PreProcessPair.xlsx')
PairDat=Data['Pair']
ScoreDat=Data['Score']
MaxPop=50
Round=0

while Round<MaxPop:
    Tour=[]
    passCount=0
    if DayInTour==1:
        Tour+=ast.literal_eval(PairDat.cell(row=Ini+2,column=Final+2).value)
    elif DayInTour==2:
        Total=0
        for i in range(1,len(ScoreDat['B'])):
```

```
Total+=float(ScoreDat.cell(row=Ini+2,column=i+1).value)+float(ScoreDat.cell(row=i
+1,column=Final+2).value)
      IndexList=[]
      for i in range(1,len(ScoreDat['B'])):

IndexList.append((float(ScoreDat.cell(row=Ini+2,column=i+1).value)+float(ScoreDat
.cell(row=i+1,column=Final+2).value))/Total)
      RanNumb=choices(range(1,len(ScoreDat['B'])),IndexList)

Tour+=ast.literal_eval(PairDat.cell(row=Ini+2,column=RanNumb[0]+1).value)+ast.li
teral_eval(PairDat.cell(row=RanNumb[0]+1,column=Final+2).value)
   else:
      for i in range(1,DayInTour):
         if i == DayInTour-1:
            Total=0
            dayIndex=0+10*(i-2)
            for i in range(1,len(ScoreDat['B'])):
               Total+=float(ScoreDat.cell(row=Tour[dayIndex][len(Tour[dayIndex])-
1]+2,column=i+1).value)+float(ScoreDat.cell(row=i+1,column=Final+2).value)
            IndexList=[]
            for i in range(1,len(ScoreDat['B'])):

IndexList.append((float(ScoreDat.cell(row=Tour[dayIndex][len(Tour[dayIndex])-
1]+2,column=i+1).value)+float(ScoreDat.cell(row=i+1,column=Final+2).value))/Tota
l)
            RanNumb=choices(range(1,len(ScoreDat['B'])),IndexList)

Tour+=ast.literal_eval(PairDat.cell(row=Tour[dayIndex][len(Tour[dayIndex])-
1]+2,column=RanNumb[0]+1).value)+ast.literal_eval(PairDat.cell(row=RanNumb[0]
+1,column=Final+2).value)
         elif i == 1:
            Total=0
            for i in range(1,len(ScoreDat['B'])):
               Total+=float(ScoreDat.cell(row=Ini+2,column=i+1).value)
            IndexList=[]
            for i in range(1,len(ScoreDat['B'])):

IndexList.append(float(ScoreDat.cell(row=Ini+2,column=i+1).value)/Total)
            RanNumb=choices(range(1,len(ScoreDat['B'])),IndexList)

Tour+=ast.literal_eval(PairDat.cell(row=Ini+2,column=RanNumb[0]+1).value)
         else:
            Total=0
            dayIndex=0+10*(i-2)
            for i in range(1,len(ScoreDat['B'])):
```

```
                Total+=float(ScoreDat.cell(row=Tour[dayIndex][len(Tour[dayIndex])-
1]+2,column=i+1).value)
            IndexList=[]
            for i in range(1,len(ScoreDat['B'])):

IndexList.append(float(ScoreDat.cell(row=Tour[dayIndex][len(Tour[dayIndex])-
1]+2,column=i+1).value)/Total)
            RanNumb=choices(range(1,len(ScoreDat['B'])),IndexList)

Tour+=ast.literal_eval(PairDat.cell(row=Tour[dayIndex][len(Tour[dayIndex])-
1]+2,column=RanNumb[0]+1).value)
   CombinedTour,ScoreRatio,RangeForEachDay=CombineAllTrip(Tour,DayInTour)

DuplicateNodeList,DuplicateNodeInfo,DuplicateNodeRatio=FindDuplicateNode(Co
mbinedTour,ScoreRatio,h,p,l,d,s,pr,t,o,c,tt)
   Tour=RemoveAttLowestRatioTour(Tour, DuplicateNodeList, DuplicateNodeInfo,
DuplicateNodeRatio, RangeForEachDay,h,p,l,d,s,pr,t,o,c,tt)
   Tour=CheckTWAndRemoveAttraction(Tour,h,p,l,d,s,pr,t,o,c,tt)
   for eachDay in range(0,DayInTour):

LCounter,DCounter,LNode,LRatio,DNode,DRatio,FocusTrip,FocusTripRatio=Count
ResNode(Tour,eachDay,l,d)

Tour,completeFlag=CheckNumberOfResNode(Tour,eachDay,LCounter,DCounter,L
Node,DNode,LRatio,DRatio,h,p,l,d,s,pr,t,o,c,tt,Budget,DayInTour)
      '''Check number of Restaurant'''
      #Data['Pool'].cell(row=dummyRowNumb,column=17).value=str(Offlist[OffS])

LCounter,DCounter,LNode,LRatio,DNode,DRatio,FocusTrip,FocusTripRatio=Count
ResNode(Tour,eachDay,l,d)

Tour,completeFlag=CheckNumberOfResNode(Tour,eachDay,LCounter,DCounter,L
Node,DNode,LRatio,DRatio,h,p,l,d,s,pr,t,o,c,tt,Budget,DayInTour)

LCounter,DCounter,LNode,LRatio,DNode,DRatio,FocusTrip,FocusTripRatio=Count
ResNode(Tour,eachDay,l,d)

Tour,completeFlag=CheckNumberOfResNode(Tour,eachDay,LCounter,DCounter,L
Node,DNode,LRatio,DRatio,h,p,l,d,s,pr,t,o,c,tt,Budget,DayInTour)
      if completeFlag:
         passCount=passCount+1
   if passCount==DayInTour:

CombinedTour,ScoreRatio,RangeForEachDay=CombineAllTrip(Tour,DayInTour)

DuplicateNodeList,DuplicateNodeInfo,DuplicateNodeRatio=FindDuplicateNode(Co
mbinedTour,ScoreRatio,h,p,l,d,s,pr,t,o,c,tt)
```

```
        Tour=RemoveAttLowestRatioTour(Tour, DuplicateNodeList,
DuplicateNodeInfo, DuplicateNodeRatio, RangeForEachDay,h,p,l,d,s,pr,t,o,c,tt)
        Tour=CheckTWAndRemoveAttraction(Tour,h,p,l,d,s,pr,t,o,c,tt)

Tour,InvalidTour,TotalScore=ImproveResTW(Tour,Budget,h,p,l,d,s,pr,t,o,c,tt,DayIn
Tour)
        enoughBudget=CostValidation(Tour,Budget,DayInTour)
        if InvalidTour==0 and enoughBudget:
            Tour,TotalScore=LocalInsertion(Tour,Budget,h,p,l,d,s,pr,t,o,c,tt,DayInTour)
            Count=0
            TotalScore=0
            Run=0
            while Count<DayInTour:
                TotalScore+=Tour[6+Run]
                Count+=1
                Run+=10
            Tour, TestResult, TotalScore, ScoreM1, ScoreM2, ScoreM3, ScoreM4,
ScoreM5,nodem1,nodem2,nodem3,nodem4,nodem5=
LocalSearchImprovement(Tour,Budget,h,p,l,d,s,pr,t,o,c,tt,DayInTour, TotalScore)
            Data['CurrPop'].cell(row=Round+2,column=2).value=str(Tour)
            Data['CurrPop'].cell(row=Round+2,column=3).value=TotalScore
            Data['Pool'].cell(row=Round+2,column=2).value=str(Tour)
            Data['Pool'].cell(row=Round+2,column=3).value=TotalScore
            Round+=1
Data.save('PreProcessPair.xlsx')
```

4. DefCrossOver.py

```
from openpyxl import *
from random import choices, randrange, uniform
from ImproveGAOffspring import *
from Utility import *
from Recursive import *
from LocalSearchV2 import *
import ast
from CombineLocalSearch import *
import time

def CrossOver(h,p,l,d,s,pr,t,o,c,tt,Budget,MaxPop,Rate,DayInTour):
    '''Used data from TestDatV2 to create list name tt and t'''
    Data=load_workbook('PreProcessPair.xlsx')
    CP=Data['CurrPop']
    Pool=Data['Pool']
    B=Pool['B']
    RowNumb=len(B)+1
    dummyRowNumb=len(B)+1
    #MaxPop=50
```

```
#Rate=0.5
OuterRound=0
Total=0
m1=0
m2=0
m3=0
m4=0
m5=0
nodem1=[]
nodem2=[]
nodem3=[]
nodem4=[]
nodem5=[]
TestResult=[]
NumbList=[]
startTime = time.time()
for i in range(1,len(CP['B'])):
   NumbList.append(i)
for i in NumbList:
   Total+=float(CP.cell(row=i+1,column=3).value)
IndexList=[]
for i in NumbList:
   IndexList.append(float(CP.cell(row=i+1,column=3).value)/Total)
endTime = time.time() - startTime
while OuterRound<Rate*MaxPop:
   startTime = time.time()
   RanTour=choices(NumbList,IndexList)
   NumbList.pop(RanTour[0]-1)
   TempInd=IndexList.pop(RanTour[0]-1)
   RanTour+=choices(NumbList,IndexList)
   NumbList.insert(RanTour[0]-1,RanTour[0])
   IndexList.insert(RanTour[0]-1,TempInd)
   P1=ast.literal_eval(CP.cell(row=RanTour[0]+1,column=2).value)
   P2=ast.literal_eval(CP.cell(row=RanTour[1]+1,column=2).value)
   OffS=0
   notFoundCount=0
   dummyPair=[[],[]]
   dummyPairScore=[[],[]]
   endTime+= (time.time() - startTime)
   while OffS<2 and notFoundCount<10:
      passCount=0
      startTime = time.time()
      Parlist=[P1,P2]
      Offlist=[[],[]]
      Crosslist=[0,0]
      RandDay=randrange(DayInTour)
      RandTime=uniform(8,20)
```

```
        PTourIndex=0+10*RandDay
        PTimeIndex=1+10*RandDay
        PScorelistIndex=2+10*RandDay
        PCostlistIndex=3+10*RandDay
        if RandTime>=Parlist[OffS][PTimeIndex][len(Parlist[OffS][PTimeIndex])-1]:
           Crosslist[OffS]=len(Parlist[OffS][PTimeIndex])-2
           Cut=0
        else:
           for i in Parlist[OffS][PTimeIndex]:
              if RandTime<=i:
                 HalfTimeDiff=(i-
Parlist[OffS][PTimeIndex][Parlist[OffS][PTimeIndex].index(i)-1])/2
                 if
RandTime<=Parlist[OffS][PTimeIndex][Parlist[OffS][PTimeIndex].index(i)-
1]+HalfTimeDiff:
                    Crosslist[OffS]=Parlist[OffS][PTimeIndex].index(i)-1
                    Cut=0
                 else:
                    '''Crosslist[OffS]=Parlist[OffS][PTimeIndex].index(i)
                    Cut=1'''
                    if not
(Parlist[OffS][PTourIndex][Parlist[OffS][PTimeIndex].index(i)] in h):
                       Crosslist[OffS]=Parlist[OffS][PTimeIndex].index(i)
                       Cut=1
                    else:
                       Crosslist[OffS]=Parlist[OffS][PTimeIndex].index(i)-1
                       Cut=0
                 break
        if RandTime>=Parlist[OffS-1][PTimeIndex][len(Parlist[OffS-
1][PTimeIndex])-1]:
           Crosslist[OffS-1]=len(Parlist[OffS-1][PTimeIndex])-1
        else:
           for i in Parlist[OffS-1][PTimeIndex]:
              if RandTime<=i:
                 if Cut == 0:
                    Crosslist[OffS-1]=Parlist[OffS-1][PTimeIndex].index(i)
                 else:
                    if Parlist[OffS-1][PTimeIndex].index(i)-1 not in h:
                       Crosslist[OffS-1]=Parlist[OffS-1][PTimeIndex].index(i)-1
                    else:
                       Crosslist[OffS-1]=Parlist[OffS-1][PTimeIndex].index(i)
                 break
        Offlist[OffS]+=Parlist[OffS][:min(1,RandDay)*(9+10*(RandDay-1)+1)]
        OffTrip=Parlist[OffS][PTourIndex][:Crosslist[OffS]+1]+Parlist[OffS-
1][PTourIndex][Crosslist[OffS-1]:]
        Offlist[OffS]+=[OffTrip]
        TimeWindow=[8]
```

```
        Compensatelist=[0]
        for i in range(1,len(OffTrip)):
            TW=TimeWindow[i-1]+tt.cell(row=5+OffTrip[i-
1],column=5+OffTrip[i]).value+t[OffTrip[i]]
            TW,twPassFlag,compensate=oneNodeTWCalculation(TW,o,c,OffTrip[i])
            TimeWindow.append(TW)
            Compensatelist.append(compensate)
        Offlist[OffS]+=[TimeWindow]
        Scorelist=Parlist[OffS][PScorelistIndex][:Crosslist[OffS]+1]+Parlist[OffS-
1][PScorelistIndex][Crosslist[OffS-1]:]
        Costlist=Parlist[OffS][PCostlistIndex][:Crosslist[OffS]+1]+Parlist[OffS-
1][PCostlistIndex][Crosslist[OffS-1]:]
        Offlist[OffS]+=[Scorelist]+[Costlist]
        Score=0
        for i in Scorelist:
            Score+=i
        Cost=0
        for i in Costlist:
            Cost+=i
        TotalRatio, Ratiolist = CalculateRatio(OffTrip,s,tt,Compensatelist)
        TotalCompensate=CalculateCompensate(Compensatelist)
        Offlist[OffS]+=[Ratiolist]+[TotalRatio]
        Offlist[OffS]+=[Score]
        Offlist[OffS]+=[Cost]
        Offlist[OffS]+=[Compensatelist]
        Offlist[OffS]+=[TotalCompensate]
        Offlist[OffS]+=Parlist[OffS-1][0+10*(RandDay+1):]
        '''Delete infeasible and duplicate node'''
        DuplicateNodeList, DuplicateNodeInfo,
DuplicateNodeRatio=FindDuplicateNode(Offlist[OffS][0+10*RandDay],
Offlist[OffS][4+10*RandDay],h,p,l,d,s,pr,t,o,c,tt)
        Offlist[OffS]=RemoveAttLowestRatioTrip(Offlist[OffS], DuplicateNodeList,
DuplicateNodeInfo, DuplicateNodeRatio,DayInTour,RandDay,h,p,l,d,s,pr,t,o,c,tt)
        '''Test'''

CombinedTour,ScoreRatio,RangeForEachDay=CombineAllTrip(Offlist[OffS],DayIn
Tour)

DuplicateNodeList,DuplicateNodeInfo,DuplicateNodeRatio=FindDuplicateNode(Co
mbinedTour,ScoreRatio,h,p,l,d,s,pr,t,o,c,tt)
        Offlist[OffS]=RemoveAttLowestRatioTour(Offlist[OffS], DuplicateNodeList,
DuplicateNodeInfo, DuplicateNodeRatio, RangeForEachDay,h,p,l,d,s,pr,t,o,c,tt)

Offlist[OffS]=CheckTWAndRemoveAttraction(Offlist[OffS],h,p,l,d,s,pr,t,o,c,tt)
        for eachDay in range(0,DayInTour):
```

```
LCounter,DCounter,LNode,LRatio,DNode,DRatio,FocusTrip,FocusTripRatio=Count
ResNode(Offlist[OffS],eachDay,l,d)

Offlist[OffS],completeFlag=CheckNumberOfResNode(Offlist[OffS],eachDay,LCoun
ter,DCounter,LNode,DNode,LRatio,DRatio,h,p,l,d,s,pr,t,o,c,tt,Budget,DayInTour)
            '''Check number of Restaurant'''

LCounter,DCounter,LNode,LRatio,DNode,DRatio,FocusTrip,FocusTripRatio=Count
ResNode(Offlist[OffS],eachDay,l,d)

Offlist[OffS],completeFlag=CheckNumberOfResNode(Offlist[OffS],eachDay,LCoun
ter,DCounter,LNode,DNode,LRatio,DRatio,h,p,l,d,s,pr,t,o,c,tt,Budget,DayInTour)

LCounter,DCounter,LNode,LRatio,DNode,DRatio,FocusTrip,FocusTripRatio=Count
ResNode(Offlist[OffS],eachDay,l,d)

Offlist[OffS],completeFlag=CheckNumberOfResNode(Offlist[OffS],eachDay,LCoun
ter,DCounter,LNode,DNode,LRatio,DRatio,h,p,l,d,s,pr,t,o,c,tt,Budget,DayInTour)
            '''Check whether complete flag is True to proceed futher'''
            if completeFlag:
               passCount=passCount+1
         Data['Pool'].cell(row=dummyRowNumb,column=19).value=str(Offlist[OffS])
         if passCount==DayInTour:
            '''Have to fix the code to define new range from lunch to dinner not after
dinner to hotel'''

Offlist[OffS],InvalidTour,TotalScore=ImproveResTW(Offlist[OffS],Budget,h,p,l,d,s,
pr,t,o,c,tt,DayInTour)
            EnoughBudget=CostValidation(Offlist[OffS],Budget,DayInTour)
            if (InvalidTour==0) and EnoughBudget:
               '''Improve Solution by Local Search'''

Offlist[OffS],TotalScore=LocalInsertion(Offlist[OffS],Budget,h,p,l,d,s,pr,t,o,c,tt,DayI
nTour)
               Offlist[OffS], TestResult, TotalScore,
m1,m2,m3,m4,m5,nodem1,nodem2,nodem3,nodem4,nodem5=
LocalSearchImprovement(Offlist[OffS],Budget,h,p,l,d,s,pr,t,o,c,tt,DayInTour,
TotalScore)
               Count=0
               TotalScore=0
               Run=0
               while Count<DayInTour:
                  tripScore=0
                  for nodeScore in Offlist[OffS][2+Run]:
                        tripScore+=nodeScore
                  TotalScore+=tripScore
```

```
                    Offlist[OffS][6+Run]=tripScore
                    Count+=1
                    Run+=10
                '''Save solution in the Pool'''
                dummyPair[OffS]=Offlist[OffS]
                dummyPairScore[OffS]=str(RandDay)+'Cross'
                dummyRowNumb+=1
                OffS+=1
            else:
                notFoundCount+=1
        else:
            notFoundCount+=1
    if OffS == 2:
        for child in range(OffS):
            '''Calculate Total Score before save the result'''
            Count=0
            TotalScore=0
            Run=0
            while Count<DayInTour:
                tripScore=0
                for nodeScore in dummyPair[child][2+Run]:
                    tripScore+=nodeScore
                TotalScore+=tripScore
                dummyPair[child][6+Run]=tripScore
                Count+=1
                Run+=10
            endTime+= (time.time() - startTime)
            Data['Pool'].cell(row=RowNumb,column=2).value=str(dummyPair[child])
            Data['Pool'].cell(row=RowNumb,column=3).value=TotalScore
            Data['Pool'].cell(row=RowNumb,column=4).value=dummyPairScore[child]
            RowNumb+=1
        OuterRound+=1
    Data.save('PreProcessPair.xlsx')
    return endTime
```

5. DefMutation.py

```
from openpyxl import *
from random import choices, randrange, uniform
from ImproveGAOffspring import *
from Utility import *
from Recursive import *
from LocalSearchV2 import *
import ast
from CombineLocalSearch import *
import time
```

```
def
Mutation(h,p,l,d,s,pr,t,o,c,tt,Budget,MaxPop,MutationRate,DayInTour,startPoint,endP
oint):
    Data=load_workbook('PreProcessPair.xlsx')
    CP=Data['CurrPop']
    Pool=Data['Pool']
    Pair=Data['Pair']
    B=Pair['B']
    B2=Pool['B']
    RowNumb=len(B2)+1
    Round=0
    MaxPop=50
    Total=0
    MutationRate=1
    m1=0
    m2=0
    m3=0
    m4=0
    m5=0
    nodem1=[]
    nodem2=[]
    nodem3=[]
    nodem4=[]
    nodem5=[]
    TestResult=[]
    NumbList=[]
    startTime = time.time()
    for i in range(1,len(CP['B'])):
        NumbList.append(i)
    for i in NumbList:
        Total+=float(CP.cell(row=i+1,column=3).value)
    IndexList=[]
    for i in NumbList:
        IndexList.append(float(CP.cell(row=i+1,column=3).value)/Total)
    endTime = time.time() - startTime
    while Round<MaxPop*MutationRate:
        passCount=0
        startTime = time.time()
        RanTour=choices(NumbList,IndexList)
        NumbList.pop(RanTour[0]-1)
        TempInd=IndexList.pop(RanTour[0]-1)
        RanTour+=choices(NumbList,IndexList)
        NumbList.insert(RanTour[0]-1,RanTour[0])
        IndexList.insert(RanTour[0]-1,TempInd)
        P1=ast.literal_eval(CP.cell(row=RanTour[0]+1,column=2).value)
        RandDay=randrange(DayInTour)
        if RandDay == DayInTour-1:
```

```
            if DayInTour == 1:
               O1=[]
               mutationMethod=randrange(1,3)
               if mutationMethod == 1:
                  From=randrange(2,len(B)+1)
                  P1=ast.literal_eval(CP.cell(row=RanTour[0]+1,column=2).value)
                  P2=ast.literal_eval(Pair.cell(row=From,column=endPoint+2).value)
               if mutationMethod == 2:
                  To=randrange(2,len(B)+1)
                  P1=ast.literal_eval(Pair.cell(row=startPoint+2,column=To).value)
                  P2=ast.literal_eval(CP.cell(row=RanTour[0]+1,column=2).value)
               Crosslist=[0,0]
               RandTime=uniform(8,20)
               PTourIndex=0+10*RandDay
               PTimeIndex=1+10*RandDay
               PScorelistIndex=2+10*RandDay
               PCostlistIndex=3+10*RandDay
               if RandTime>=P1[PTimeIndex][len(P1[PTimeIndex])-1]:
                  Crosslist[0]=len(P1[PTimeIndex])-2
                  Cut=0
               else:
                  for i in P1[PTimeIndex]:
                     if RandTime<=i:
                        HalfTimeDiff=(i-P1[PTimeIndex][P1[PTimeIndex].index(i)-1])/2
                        if RandTime<=P1[PTimeIndex][P1[PTimeIndex].index(i)-
1]+HalfTimeDiff:
                           Crosslist[0]=P1[PTimeIndex].index(i)-1
                           Cut=0
                        else:
                           '''Crosslist[OffS]=Parlist[OffS][PTimeIndex].index(i)
                           Cut=1'''
                           if not (P1[PTourIndex][P1[PTimeIndex].index(i)] in h):
                              Crosslist[0]=P1[PTimeIndex].index(i)
                              Cut=1
                           else:
                              Crosslist[0]=P1[PTimeIndex].index(i)-1
                              Cut=0
                        break
               if RandTime>=P2[PTimeIndex][len(P2[PTimeIndex])-1]:
                  Crosslist[-1]=len(P2[PTimeIndex])-1
               else:
                  for i in P2[PTimeIndex]:
                     if RandTime<=i:
                        if Cut == 0:
                           Crosslist[-1]=P2[PTimeIndex].index(i)
                        else:
                           if P2[PTimeIndex].index(i)-1 not in h:
```

```
                    Crosslist[-1]=P2[PTimeIndex].index(i)-1
                else:
                    Crosslist[-1]=P2[PTimeIndex].index(i)
                break
        O1+=P1[:min(1,RandDay)*(9+10*(RandDay-1)+1)]
        OffTrip=P1[PTourIndex][:Crosslist[0]+1]+P2[PTourIndex][Crosslist[-1]:]
        O1+=[OffTrip]
        TimeWindow=[8]
        Compensatelist=[0]
        for i in range(1,len(OffTrip)):
            TW=TimeWindow[i-1]+tt.cell(row=5+OffTrip[i-
1],column=5+OffTrip[i]).value+t[OffTrip[i]]

TW,twPassFlag,compensate=oneNodeTWCalculation(TW,o,c,OffTrip[i])
            TimeWindow.append(TW)
            Compensatelist.append(compensate)
        O1+=[TimeWindow]

Scorelist=P1[PScorelistIndex][:Crosslist[0]+1]+P2[PScorelistIndex][Crosslist[-1]:]
        Costlist=P1[PCostlistIndex][:Crosslist[0]+1]+P2[PCostlistIndex][Crosslist[-
1]:]
        O1+=[Scorelist]+[Costlist]
        Score=0
        for i in Scorelist:
            Score+=i
        Cost=0
        for i in Costlist:
            Cost+=i
        TotalRatio, Ratiolist = CalculateRatio(OffTrip,s,tt,Compensatelist)
        TotalCompensate=CalculateCompensate(Compensatelist)
        O1+=[Ratiolist]+[TotalRatio]
        O1+=[Score]
        O1+=[Cost]
        O1+=[Compensatelist]
        O1+=[TotalCompensate]
        O1+=P2[0+10*(RandDay+1):]
    else:
        From=randrange(2,len(B)+1)
        To=endPoint#P1[0][0]
        SmallTour=P1[0+10*(RandDay-1)]
        RandTrip=ast.literal_eval(Pair.cell(row=From,column=To+2).value)
        if RandTrip[0][0] == SmallTour[len(SmallTour)-1]:
            O1 = P1[:10+10*(RandDay-1)]
            for i in RandTrip:
                O1.append(i)
        else:
            SmallTour2=P1[0+10*(RandDay-2)]
```

```
                From=SmallTour2[len(SmallTour2)-1]
                To=RandTrip[0][0]
                RandTrip2=ast.literal_eval(Pair.cell(row=From+2,column=To+2).value)
                O1 = P1[:10+10*(RandDay-2)]
                O1+=RandTrip2
                for i in RandTrip:
                    O1.append(i)
        else:
            From=P1[0+10*RandDay][0]
            To=randrange(2,len(B)+1)
            RandTrip=ast.literal_eval(Pair.cell(row=From+2,column=To).value)
            if RandTrip[0][len(RandTrip[0])-1] == P1[10*(RandDay+1)][0]:
                O1=P1[:]
                for i in range(0+10*RandDay,10*(RandDay+1)):
                    O1.pop(i)
                    O1.insert(i,RandTrip[i-10*RandDay])
            else:
                From=RandTrip[0][len(RandTrip[0])-1]
                To=P1[10*(RandDay+1)][len(P1[10*(RandDay+1)])-1]
                SupportTrip=ast.literal_eval(Pair.cell(row=From+2,column=To+2).value)
                O1=P1[:]
                for i in range(0+10*RandDay,10*(RandDay+1)):
                    O1.pop(i)
                    O1.insert(i,RandTrip[i-10*RandDay])
                for i in range(10*(RandDay+1),10*(RandDay+2)):
                    O1.pop(i)
                    O1.insert(i,SupportTrip[i-10*(RandDay+1)])
        if O1[0][0] == startPoint:
            DuplicateNodeList, DuplicateNodeInfo,
DuplicateNodeRatio=FindDuplicateNode(O1[0+10*RandDay],
O1[4+10*RandDay],h,p,l,d,s,pr,t,o,c,tt)
            O1=RemoveAttLowestRatioTrip(O1, DuplicateNodeList, DuplicateNodeInfo,
DuplicateNodeRatio,DayInTour,RandDay,h,p,l,d,s,pr,t,o,c,tt)

CombinedTour,ScoreRatio,RangeForEachDay=CombineAllTrip(O1,DayInTour)

DuplicateNodeList,DuplicateNodeInfo,DuplicateNodeRatio=FindDuplicateNode(Co
mbinedTour,ScoreRatio,h,p,l,d,s,pr,t,o,c,tt)
            O1=RemoveAttLowestRatioTour(O1, DuplicateNodeList, DuplicateNodeInfo,
DuplicateNodeRatio, RangeForEachDay,h,p,l,d,s,pr,t,o,c,tt)
            O1=CheckTWAndRemoveAttraction(O1,h,p,l,d,s,pr,t,o,c,tt)
            '''Delete infeasible and duplicate node'''
            for eachDay in range(0,DayInTour):

LCounter,DCounter,LNode,LRatio,DNode,DRatio,FocusTrip,FocusTripRatio=Count
ResNode(O1,eachDay,l,d)
```

```
O1,completeFlag=CheckNumberOfResNode(O1,eachDay,LCounter,DCounter,LNod
e,DNode,LRatio,DRatio,h,p,l,d,s,pr,t,o,c,tt,Budget,DayInTour)
            '''Check number of Restaurant'''

LCounter,DCounter,LNode,LRatio,DNode,DRatio,FocusTrip,FocusTripRatio=Count
ResNode(O1,eachDay,l,d)

O1,completeFlag=CheckNumberOfResNode(O1,eachDay,LCounter,DCounter,LNod
e,DNode,LRatio,DRatio,h,p,l,d,s,pr,t,o,c,tt,Budget,DayInTour)

LCounter,DCounter,LNode,LRatio,DNode,DRatio,FocusTrip,FocusTripRatio=Count
ResNode(O1,eachDay,l,d)

O1,completeFlag=CheckNumberOfResNode(O1,eachDay,LCounter,DCounter,LNod
e,DNode,LRatio,DRatio,h,p,l,d,s,pr,t,o,c,tt,Budget,DayInTour)
            '''Check whether complete flag is True to proceed futher'''
            if completeFlag:
               passCount=passCount+1
         if passCount==DayInTour:

O1,InvalidTour,TotalScore=ImproveResTW(O1,Budget,h,p,l,d,s,pr,t,o,c,tt,DayInTour
)
            EnoughBudget=CostValidation(O1,Budget,DayInTour)
            if (InvalidTour==0) and EnoughBudget:
               Data['Pool'].cell(row=RowNumb,column=5).value=str(O1)
               O1,TotalScore=LocalInsertion(O1,Budget,h,p,l,d,s,pr,t,o,c,tt,DayInTour)
               Data['Pool'].cell(row=RowNumb,column=6).value=str(O1)

O1,TestResult,TotalScore,m1,m2,m3,m4,m5,nodem1,nodem2,nodem3,nodem4,node
m5=LocalSearchImprovement(O1,Budget,h,p,l,d,s,pr,t,o,c,tt,DayInTour, TotalScore)
               Data['Pool'].cell(row=RowNumb,column=7).value=str(O1)
               '''Calculate Total Score before save the result'''
               Count=0
               TotalScore=0
               Run=0
               while Count<DayInTour:
                  TotalScore+=O1[6+Run]
                  Count+=1
                  Run+=10
               '''Save solution in the Pool'''
               endTime+= (time.time() - startTime)
               Data['Pool'].cell(row=RowNumb,column=2).value=str(O1)
               Data['Pool'].cell(row=RowNumb,column=3).value=TotalScore
               Data['Pool'].cell(row=RowNumb,column=4).value=str(RandDay)+'Mut'
               RowNumb+=1
      Round+=1
```

```
    Data.save('PreProcessPair.xlsx')
    return endTime
```

6. ImproveGAOffspring.py

```
from Recursive import *
from Utility import *
from PreProcessLib import *
from LocalSearchV2 import *

'''Used data from TestDatV2 to create list name tt and t'''
MaxTime=20


'''Remove attraction node, this function might be used when travelling time exceed the
limit'''
def CheckTWAndRemoveAttraction(Node,h,p,l,d,s,pr,t,o,c,tt):
    tempTrip=Node[:]
    for i in range(int(len(Node)/10)):
        multiplier=10*i
        TimeTrip=tempTrip[1+multiplier]
        Trip=tempTrip[0+multiplier][1:-1]
        ForRatioIndex=tempTrip[4+multiplier][1:-1]
        while TimeTrip[len(TimeTrip)-1]>MaxTime:
            RefIndex=9999
            RefIndex2=9999
            Index=LorD(Trip,ForRatioIndex,RefIndex,RefIndex2,l,d)
            tempTrip[0+multiplier].pop(Index+1)
            TempTrip=tempTrip[0+multiplier]
            Trip=tempTrip[0+multiplier][1:-1]
            TimeTrip=[8]
            Compensationlist=[0]
            for Round in range(len(tempTrip[0+multiplier])-1):

TW=TimeTrip[Round]+tt.cell(row=5+TempTrip[Round],column=5+TempTrip[Roun
d+1]).value+t[TempTrip[Round+1]]

TW,twPassFlag,compensate=oneNodeTWCalculation(TW,o,c,TempTrip[Round+1])
                TimeTrip.append(TW)
                Compensationlist.append(compensate)
            tempTrip[2+multiplier].pop(Index+1)
            tempTrip[3+multiplier].pop(Index+1)
            tempTrip[4+multiplier].pop(Index+1)
            ForRatioIndex=tempTrip[4+multiplier][1:-1]
            TotalRatio=0
            Score=0
            Cost=0
```

```
            for x,j,k in
zip(tempTrip[2+multiplier],tempTrip[3+multiplier],tempTrip[4+multiplier]):
                TotalRatio+=k
                Score+=x
                Cost+=j
            TotalCompensation=CalculateCompensate(Compensationlist)
            tempTrip[1+multiplier]=TimeTrip
            tempTrip[5+multiplier]=TotalRatio
            tempTrip[6+multiplier]=Score
            tempTrip[7+multiplier]=Cost
            tempTrip[8+multiplier]=Compensationlist
            tempTrip[9+multiplier]=TotalCompensation
    return tempTrip

'''New Insertion Method'''
def
InANew(Trip,TimeWindow,Scorelist,Costlist,Ratiolist,TotalRatio,Score,Cost,Compe
nsatelist,TotalCompensate,Budget,h,p,l,d,s,pr,t,o,c,tt,CombinedTour):
    '''Tabu was used to lock the position of current solution (Lock position after
Restaurant is adding)'''
    global Tabu
    Tabu=0
    Round=0
    Lu=0
    Di=0
    while (Round<2 or Lu<1 or Di<1):
        if (10.5<=TimeWindow[len(TimeWindow[:-2])]<=13 and Lu<1 and all(i not in l
for i in Trip)) or (Round>0 and Lu<1 and all(i not in l for i in Trip)):
            Ratio=0
            for i in l:
                if i not in CombinedTour:
                    MiniTime=TimeWindow[:-1]
                    MiniComp=Compensatelist[:-1]
                    TW = MiniTime[len(MiniTime[:-1])]+tt.cell(row=5+Trip[len(Trip[:-
2])],column=5+i).value+t[i]
                    TW,twPassFlag,compensate=oneNodeTWCalculation(TW,o,c,i)
                    if twPassFlag and Cost+pr[i]<=Budget:
                        x = s[i]/(tt.cell(row=5+Trip[len(Trip)-2],column=5+i).value +
tt.cell(row=5+i,column=5+Trip[len(Trip)-1]).value + compensate)
                        if Ratio < x:
                            MiniTime.append(TW)
                            MiniComp.append(compensate)
                            del TW
                            TempTrip=Trip[:]
                            TempScore=Scorelist[:]
                            TempCost=Costlist[:]
                            TempTrip.insert(len(Trip[:-1]),i)
```

```
                    TempScore.insert(len(Scorelist[:-1]),s[i])
                    TempCost.insert(len(Costlist[:-1]),pr[i])
                    TW = MiniTime[len(MiniTime[:-
1])]+tt.cell(row=5+TempTrip[len(TempTrip[:-
2])],column=5+TempTrip[len(TempTrip[:-1])]).value+t[TempTrip[len(TempTrip[:-
1])]]

TW,twPassFlag,compensate=oneNodeTWCalculation(TW,o,c,TempTrip[len(TempTr
ip[:-1])])
                    if twPassFlag:
                        MiniTime.append(TW)
                        MiniComp.append(compensate)
                        if len(TempTrip)==len(MiniTime):
                            MirrorTime=MiniTime
                            MirrorComp=MiniComp
                            MirrorTrip=TempTrip
                            MirrorScore=TempScore
                            MirrorCost=TempCost
                            Tabu=len(TempTrip[:-2])
                            Ratio = x
                            SaveScore=s[i]
                            SaveCost=pr[i]
                            Lu+=1
                        del TempTrip, TempScore, TempCost
                    del x
                del MiniTime
            if 'MirrorTrip' in locals():
                Trip=MirrorTrip
                TimeWindow=MirrorTime
                Compensatelist=MirrorComp
                Scorelist=MirrorScore
                Costlist=MirrorCost
                Score+=SaveScore
                Cost+=SaveCost
                del MirrorTime, MirrorTrip, MirrorScore, MirrorCost, SaveScore, SaveCost
            else:
                Lu+=1
                Di+=1
                Round+=4
                TotalCompensate=CalculateCompensate(Compensatelist)
                TotalRatio, Ratiolist = CalculateRatio(Trip,s,tt,Compensatelist)
        elif (15.5<=TimeWindow[len(TimeWindow[:-2])]<=18 and Di<1 and all(i not in
d for i in Trip)) or (Round>0 and Di<1 and all(i not in d for i in Trip)):
            Ratio=0
            for i in d:
                if i not in CombinedTour:
                    MiniTime=TimeWindow[:-1]
```

```
                MiniComp=Compensatelist[:-1]
                TW = MiniTime[len(MiniTime[:-1])]+tt.cell(row=5+Trip[len(Trip[:-
2])],column=5+i).value+t[i]
                TW,twPassFlag,compensate=oneNodeTWCalculation(TW,o,c,i)
                if twPassFlag and Cost+pr[i]<=Budget:
                    x = s[i]/(tt.cell(row=5+Trip[len(Trip)-2],column=5+i).value +
tt.cell(row=5+i,column=5+Trip[len(Trip)-1]).value + compensate)
                    if Ratio < x:
                        MiniTime.append(TW)
                        MiniComp.append(compensate)
                        del TW
                        TempTrip=Trip[:]
                        TempScore=Scorelist[:]
                        TempCost=Costlist[:]
                        TempTrip.insert(len(Trip[:-1]),i)
                        TempScore.insert(len(Scorelist[:-1]),s[i])
                        TempCost.insert(len(Costlist[:-1]),pr[i])
                        TW = MiniTime[len(MiniTime[:-
1])]+tt.cell(row=5+TempTrip[len(TempTrip[:-
2])],column=5+TempTrip[len(TempTrip[:-1])]).value+t[TempTrip[len(TempTrip[:-
1])]]

TW,twPassFlag,compensate=oneNodeTWCalculation(TW,o,c,TempTrip[len(TempTr
ip[:-1])])
                        if twPassFlag:
                            MiniTime.append(TW)
                            MiniComp.append(compensate)
                            if len(TempTrip)==len(MiniTime):
                                MirrorTime=MiniTime
                                MirrorComp=MiniComp
                                MirrorTrip=TempTrip
                                MirrorScore=TempScore
                                MirrorCost=TempCost
                                Tabu=len(TempTrip[:-2])
                                Ratio = x
                                SaveScore=s[i]
                                SaveCost=pr[i]
                                Di+=1
                            del TempTrip, TempScore, TempCost
                        del x
                    del MiniTime
            if 'MirrorTrip' in locals():
                Trip=MirrorTrip
                TimeWindow=MirrorTime
                Compensatelist=MirrorComp
                Scorelist=MirrorScore
                Costlist=MirrorCost
```

```
            Score+=SaveScore
            Cost+=SaveCost
            del MirrorTime, MirrorTrip, MirrorScore, MirrorCost, SaveScore, SaveCost
         else:
            Di+=1
            Round+=4
            TotalCompensate=CalculateCompensate(Compensatelist)
            TotalRatio, Ratiolist = CalculateRatio(Trip,s,tt,Compensatelist)
      else:
         Ratio = 0
         for i in p:
            if (i not in Trip) and (i not in CombinedTour):
               for j in range(Tabu,len(Trip)-1):
                  for k in range(Tabu+1,len(Trip)):
                     if j < k:
                        MiniTime=TimeWindow[:k]
                        MiniComp=Compensatelist[:k]
                        TW = MiniTime[k-1]+tt.cell(row=5+Trip[k-
1],column=5+i).value+t[i]
                        TW,twPassFlag,compensate=oneNodeTWCalculation(TW,o,c,i)
                        if twPassFlag and Cost+pr[i]<=Budget:
                           x = s[i]/(tt.cell(row=5+Trip[k-1],column=5+i).value +
tt.cell(row=5+i,column=5+Trip[k]).value + compensate)
                           if Ratio < x:
                              MiniTime.append(TW)
                              MiniComp.append(compensate)
                              del TW
                              TempTrip=Trip[:]
                              TempScore=Scorelist[:]
                              TempCost=Costlist[:]
                              TempTrip.insert(k,i)
                              TempScore.insert(k,s[i])
                              TempCost.insert(k,pr[i])
                              for RA in range(len(TempTrip[:k]),len(TempTrip)-1):
                                 TW =
MiniTime[RA]+tt.cell(row=5+TempTrip[RA],column=5+TempTrip[RA+1]).value+t[
TempTrip[RA+1]]

TW,twPassFlag,compensate=oneNodeTWCalculation(TW,o,c,TempTrip[RA+1])
                                 if twPassFlag:
                                    MiniTime.append(TW)
                                    MiniComp.append(compensate)
                                 else:
                                    break
                              if len(TempTrip)==len(MiniTime):
                                 MirrorTime=MiniTime
                                 MirrorComp=MiniComp
```

```
                                MirrorTrip=TempTrip
                                MirrorScore=TempScore
                                MirrorCost=TempCost
                                Ratio = x
                                SaveScore=s[i]
                                SaveCost=pr[i]
                            del TempTrip, TempScore, TempCost
                        del x
                    del MiniTime
                    break
        if 'MirrorTrip' in locals():
            IniTrip=MirrorTrip
            Trip=MirrorTrip
            TimeWindow=MirrorTime
            Compensatelist=MirrorComp
            Scorelist=MirrorScore
            Costlist=MirrorCost
            Score+=SaveScore
            Cost+=SaveCost
            del MirrorTime, MirrorTrip, MirrorScore, MirrorCost, SaveScore, SaveCost
        else:
            Round+=1
        if Round>1:
            if Lu<1 or Di<1:
                Lu+=1
                Di+=1
            TotalCompensate=CalculateCompensate(Compensatelist)
            TotalRatio, Ratiolist = CalculateRatio(Trip,s,tt,Compensatelist)
    return
[Trip,TimeWindow,Scorelist,Costlist,Ratiolist,TotalRatio,Score,Cost,Compensatelist,
TotalCompensate]

"'No time window condition, it would be improve by another function"'
def
AddMoreRes(Node,CrossOverTrip,NodeType,h,p,l,d,s,pr,t,o,c,tt,Budget,DayInTour):
#Completed
    combinedTour, ScoreRatio,
RangeForEachDay=CombineAllTrip(Node,DayInTour)
    tempTrip=Node[:]
    CrossOverPoint=10*CrossOverTrip
    CurrentTrip=tempTrip[0+CrossOverPoint][:]
    CurrentTimeWindow=tempTrip[1+CrossOverPoint][:]
    CurrentScorelist=tempTrip[2+CrossOverPoint][:]
    CurrentCostlist=tempTrip[3+CrossOverPoint][:]
    CurrentRatioList=tempTrip[4+CrossOverPoint][:]
    CurrentTotalRatio=tempTrip[5+CrossOverPoint]
    CurrentScore=tempTrip[6+CrossOverPoint]
```

```
    CurrentCost=tempTrip[7+CrossOverPoint]
    CurrentCompensatelist=tempTrip[8+CrossOverPoint]
    CurrentTotalCompensate=tempTrip[9+CrossOverPoint]
    nodeIndexCounter=1
    MaxCurrentNode=len(CurrentTimeWindow)
    grandMaxRatio=-999999 #Store maximum diff ratio of all node in p comparing
with resNode
    resNodeToReplaced=0#Store pair of node to be replaced (node in p would be
replaced by resNode with grandMaxRatio)
    indexToReplace=0#Store pair of node to be replaced (node in p would be replaced
by resNode with grandMaxRatio)

TripInfo=InANew(CurrentTrip,CurrentTimeWindow,CurrentScorelist,CurrentCostlist
,CurrentRatioList,CurrentTotalRatio,CurrentScore,CurrentCost,CurrentCompensatelis
t,CurrentTotalCompensate,Budget,h,p,l,d,s,pr,t,o,c,tt,combinedTour)
    CurrentTrip=TripInfo[0][:]
    CurrentTimeWindow=TripInfo[1][:]
    CurrentScorelist=TripInfo[2][:]
    CurrentCostlist=TripInfo[3][:]
    CurrentRatioList=TripInfo[4][:]
    CurrentTotalRatio=TripInfo[5]
    CurrentScore=TripInfo[6]
    CurrentCost=TripInfo[7]
    CurrentCompensatelist=TripInfo[8][:]
    CurrentTotalCompensate=TripInfo[9]
    if NodeType=="L":
        openTime=10.50
        closeTime=14.50
        Res=l[:]
    if NodeType=="D":
        openTime=16.00
        closeTime=20.00
        Res=d[:]
    while nodeIndexCounter<MaxCurrentNode:
        dispatchTime=CurrentTimeWindow[nodeIndexCounter]
        currentNode=CurrentTrip[nodeIndexCounter]
        currentRatio=CurrentRatioList[nodeIndexCounter]
        currentMaxRatio=-999999
        currentResNodeToReplaced=0
        if openTime<=dispatchTime<=closeTime:
            if currentNode in p: #focus only on attraction node this condition is to filter
out other type of node
                for resNode in Res: #Find Res node that have the biggest positive different
in ratio with node in p
                    if resNode not in combinedTour:
```

```
Ratio=CalculateRatioForOneNode(CurrentTrip,resNode,nodeIndexCounter,s,tt,Curre
ntCompensatelist)
                    diffRatio=Ratio-currentRatio
                    if diffRatio>currentMaxRatio:
                       currentMaxRatio=diffRatio
                       currentResNodeToReplaced=resNode
             if currentMaxRatio>grandMaxRatio: #Find the biggest positive different
among node in p to replaced it with its pair
                grandMaxRatio=currentMaxRatio
                indexToReplace=nodeIndexCounter
                resNodeToReplaced=currentResNodeToReplaced
       nodeIndexCounter+=1
   if resNodeToReplaced != 0:
       CurrentTrip.pop(indexToReplace)
       CurrentTrip.insert(indexToReplace,resNodeToReplaced)

CurrentTimeWindow,CurrentScore,CurrentCost,CurrentScoreList,CurrentCostList,Cu
rrentCompensateList=CalculateGeneral(CurrentTrip,CurrentTimeWindow,pr,s,tt,t,Cr
ossOverTrip,DayInTour,o,c)

CurrentTotalRatio,CurrentRatioList=CalculateRatio(CurrentTrip,s,tt,CurrentCompens
ateList)
   CurrentTotalCompensate=CalculateCompensate(CurrentCompensateList)
   tempTrip[0+CrossOverPoint]=CurrentTrip
   tempTrip[1+CrossOverPoint]=CurrentTimeWindow
   tempTrip[2+CrossOverPoint]=CurrentScoreList
   tempTrip[3+CrossOverPoint]=CurrentCostList
   tempTrip[4+CrossOverPoint]=CurrentRatioList
   tempTrip[5+CrossOverPoint]=CurrentTotalRatio
   tempTrip[6+CrossOverPoint]=CurrentScore
   tempTrip[7+CrossOverPoint]=CurrentCost
   tempTrip[8+CrossOverPoint]=CurrentCompensateList
   tempTrip[9+CrossOverPoint]=CurrentTotalCompensate
   return tempTrip
   '''CurrentTrip.pop(LorDNode[LorDRatio.index(min(LorDRatio))])'''

'''No time window condition, it would be improve by another function'''
def
RemoveExceedRes(Node,CrossOverTrip,LorDNode,LorDCounter,LorDRatio,h,p,l,d,
s,pr,t,o,c,tt):#Completed
   tempTrip=Node[:]
   CrossOverPoint=10*CrossOverTrip
   CurrentTrip=tempTrip[0+CrossOverPoint][:]
   CurrentTimeWindow=[8]
   lastIndex=len(CurrentTrip)-1
   '''Score Initialization'''
```

```
    endHotelScore=s[CurrentTrip[lastIndex]]
    CurrentScore=endHotelScore
    CurrentScoreList=[0,endHotelScore]
    '''Cost Initialization'''
    endHotelCost=pr[CurrentTrip[lastIndex]]
    CurrentCost=endHotelCost
    CurrentCostList=[0,endHotelCost]
    CurrentRatioList=[0,0]
    CurrentTotalRatio=0
    CurrentCompensatelist=[0]
    CurrentTotalCompensate=0
    IndexCounter=1
    CurrentTrip.pop(LorDNode[LorDRatio.index(min(LorDRatio))])
    for node in CurrentTrip[1:]:
        TW = CurrentTimeWindow[IndexCounter-
1]+tt.cell(row=5+CurrentTrip[IndexCounter-1],column=5+node).value+t[node]
        TW,twPassFlag,compensate=oneNodeTWCalculation(TW,o,c,node)
        CurrentTimeWindow.append(TW)
        CurrentCompensatelist.append(compensate)
        if IndexCounter!=len(CurrentTrip):
            CurrentScoreList.insert(IndexCounter,s[node])
            CurrentCostList.insert(IndexCounter,pr[node])
            CurrentScore+=s[node]
            CurrentCost+=pr[node]
        IndexCounter+=1

CurrentTotalRatio,CurrentRatioList=CalculateRatio(CurrentTrip,s,tt,CurrentCompens
atelist)
    CurrentTotalCompensate=CalculateCompensate(CurrentCompensatelist)
    tempTrip[0+CrossOverPoint]=CurrentTrip
    tempTrip[1+CrossOverPoint]=CurrentTimeWindow
    tempTrip[2+CrossOverPoint]=CurrentScoreList
    tempTrip[3+CrossOverPoint]=CurrentCostList
    tempTrip[4+CrossOverPoint]=CurrentRatioList
    tempTrip[5+CrossOverPoint]=CurrentTotalRatio
    tempTrip[6+CrossOverPoint]=CurrentScore
    tempTrip[7+CrossOverPoint]=CurrentCost
    tempTrip[8+CrossOverPoint]=CurrentCompensatelist
    tempTrip[9+CrossOverPoint]=CurrentTotalCompensate
    return tempTrip

'''This is a function to check number of Res node in the trip and improve the trip to
make it feasible'''
def
CheckNumberOfResNode(Node,CrossOverTrip,LCounter,DCounter,LNode,DNode,L
Ratio,DRatio,h,p,l,d,s,pr,t,o,c,tt,Budget,DayInTour):#Completed
    completeFlag=False
```

```
    if LCounter==0: #Add more Lunch Node

Node=AddMoreRes(Node,CrossOverTrip,"L",h,p,l,d,s,pr,t,o,c,tt,Budget,DayInTour)
    elif DCounter==0: #Add more Dinner Node

Node=AddMoreRes(Node,CrossOverTrip,"D",h,p,l,d,s,pr,t,o,c,tt,Budget,DayInTour)
    elif LCounter>1: #Remove Lunch Node

Node=RemoveExceedRes(Node,CrossOverTrip,LNode,LCounter,LRatio,h,p,l,d,s,pr,t
,o,c,tt)
    elif DCounter>1: #Remove Dinner Node

Node=RemoveExceedRes(Node,CrossOverTrip,DNode,DCounter,DRatio,h,p,l,d,s,pr,
t,o,c,tt)
    else:
        completeFlag=True
    return Node, completeFlag

'''Check and add Att Node before Res'''
def
CheckAttBeforeRes(Node,Trip,TimeWindow,Scorelist,Costlist,Ratiolist,TotalRatio,S
core,Cost,Compensatelist,TotalCompensate,Budget,h,p,l,d,s,pr,t,o,c,tt,DayInTour,curr
entDay):
    lunchFlag=False
    dinnerFlag=False
    lunchCount=0
    dinnerCount=0
    numberOfLunch=0
    numberOfDinner=0
    Tabu=[]
    #LunchIndex=0
    for node in Trip:
        nodeIndex=Trip.index(node)
        TW=TimeWindow[nodeIndex]
        if node in l:
            numberOfLunch+=1
            LunchIndex=nodeIndex
            if o[Trip[nodeIndex]]<=TW<=c[Trip[nodeIndex]]:
                lunchFlag=True
            else:
                #logic to extract and add p
                CombinedTour, ScoreRatio,
RangeForEachDay=CombineAllTrip(Node,DayInTour)

tempTrip,tempTimeWindow,tempScorelist,tempCostlist,tempRatiolist,tempTotalRati
o,tempScore,tempCost,tempCompensatelist,tempTotalCompensate,lunchFlag,LunchI
ndex=InsertionAttExtractLunch(Trip,TimeWindow,Scorelist,Costlist,Ratiolist,TotalR
```

```
atio,Score,Cost,Compensatelist,TotalCompensate,Budget,h,p,l,d,s,pr,t,o,c,tt,DayInTou
r,Tabu,CombinedTour,nodeIndex,currentDay)
            if lunchFlag == True:

Trip,TimeWindow,Scorelist,Costlist,Ratiolist,TotalRatio,Score,Cost,Compensatelist,
TotalCompensate=tempTrip,tempTimeWindow,tempScorelist,tempCostlist,tempRati
olist,tempTotalRatio,tempScore,tempCost,tempCost,tempCompensatelist,tempTotalC
ompensate
            else:

tempTrip,tempTimeWindow,tempScorelist,tempCostlist,tempRatiolist,tempTotalRati
o,tempScore,tempCost,tempCompensatelist,tempTotalCompensate,lunchFlag,LunchI
ndex=extractinsertAttLunch(Node,Trip,TimeWindow,Scorelist,Costlist,Ratiolist,Tota
lRatio,Score,Cost,Compensatelist,TotalCompensate,Budget,h,p,l,d,s,pr,t,o,c,tt,DayInT
our,nodeIndex,currentDay)
                if lunchFlag == True:

Trip,TimeWindow,Scorelist,Costlist,Ratiolist,TotalRatio,Score,Cost,Compensatelist,
TotalCompensate=tempTrip,tempTimeWindow,tempScorelist,tempCostlist,tempRati
olist,tempTotalRatio,tempScore,tempCost,tempCompensatelist,tempTotalCompensate
                else:
                    lunchCount+=1
            break
    for node in Trip:
        nodeIndex=Trip.index(node)
        TW=TimeWindow[nodeIndex]
        if node in d:
            numberOfDinner+=1
            if o[Trip[nodeIndex]]<=TW<=c[Trip[nodeIndex]]:
                dinnerFlag=True
            else:
                #logic to extract and add p
                CombinedTour, ScoreRatio,
RangeForEachDay=CombineAllTrip(Node,DayInTour)

tempTrip,tempTimeWindow,tempScorelist,tempCostlist,tempRatiolist,tempTotalRati
o,tempScore,tempCost,tempCompensatelist,tempTotalCompensate,dinnerFlag=Inserti
onAttExtractDinner(Trip,TimeWindow,Scorelist,Costlist,Ratiolist,TotalRatio,Score,C
ost,Compensatelist,TotalCompensate,Budget,h,p,l,d,s,pr,t,o,c,tt,DayInTour,Tabu,Com
binedTour,nodeIndex,LunchIndex,currentDay)
            if dinnerFlag == True:

Trip,TimeWindow,Scorelist,Costlist,Ratiolist,TotalRatio,Score,Cost,Compensatelist,
TotalCompensate=tempTrip,tempTimeWindow,tempScorelist,tempCostlist,tempRati
olist,tempTotalRatio,tempScore,tempCost,tempCompensatelist,tempTotalCompensate
            else:
```

```
tempTrip,tempTimeWindow,tempScorelist,tempCostlist,tempRatiolist,tempTotalRati
o,tempScore,tempCost,tempCompensatelist,tempTotalCompensate,dinnerFlag=extrac
tinsertAttDinner(Node,Trip,TimeWindow,Scorelist,Costlist,Ratiolist,TotalRatio,Score
,Cost,Compensatelist,TotalCompensate,Budget,h,p,l,d,s,pr,t,o,c,tt,DayInTour,LunchIn
dex,nodeIndex,currentDay)
                if dinnerFlag == True:

Trip,TimeWindow,Scorelist,Costlist,Ratiolist,TotalRatio,Score,Cost,Compensatelist,
TotalCompensate=tempTrip,tempTimeWindow,tempScorelist,tempCostlist,tempRati
olist,tempTotalRatio,tempScore,tempCost,tempCompensatelist,tempTotalCompensate
                else:
                    dinnerCount+=1
            break
    if numberOfLunch!=1:
        lunchCount+=1
    if numberOfDinner!=1:
        dinnerCount+=1
    return
Trip,TimeWindow,Scorelist,Costlist,Ratiolist,TotalRatio,Score,Cost,Compensatelist,
TotalCompensate,lunchCount,dinnerCount

'''For each Day in tour'''
def ImproveResTW(Node,Budget,h,p,l,d,s,pr,t,o,c,tt,DayInTour):
    invalidTour=1
    totalInvalid=0
    for day in range(DayInTour):
        multiplier=10*day
        Trip=Node[0+multiplier]# i would be used with day
        TimeWindow=Node[1+multiplier]
        Scorelist=Node[2+multiplier]
        Costlist=Node[3+multiplier]
        Ratiolist=Node[4+multiplier]
        TotalRatio=Node[5+multiplier]
        Score=Node[6+multiplier]
        Cost=Node[7+multiplier]
        Compensatelist=Node[8+multiplier]
        TotalCompensate=Node[9+multiplier]

Trip,TimeWindow,Scorelist,Costlist,Ratiolist,TotalRatio,Score,Cost,Compensatelist,
TotalCompensate,lunchCount,dinnerCount=CheckAttBeforeRes(Node,Trip,TimeWin
dow,Scorelist,Costlist,Ratiolist,TotalRatio,Score,Cost,Compensatelist,TotalCompensa
te,Budget,h,p,l,d,s,pr,t,o,c,tt,DayInTour,day)

TimeWindow,Score,Cost,Scorelist,Costlist,Compensatelist,TotalCompensate,Should
Apply,Response=CalculateGeneralWValidationAll(Trip,TimeWindow,pr,s,tt,t,o,c,day
,DayInTour)
```

```
        Node[0+multiplier]=Trip
        Node[1+multiplier]=TimeWindow
        Node[2+multiplier]=Scorelist
        Node[3+multiplier]=Costlist
        Node[4+multiplier]=Ratiolist
        Node[5+multiplier]=TotalRatio
        Node[6+multiplier]=Score
        Node[7+multiplier]=Cost
        Node[8+multiplier]=Compensatelist
        Node[9+multiplier]=TotalCompensate
        totalInvalid+=lunchCount+dinnerCount+Response
    if totalInvalid==0:
        invalidTour=0
    '''Calculate Total Score before save the result'''
    Count=0
    TotalScore=0
    Run=0
    while Count<DayInTour:
        TotalScore+=Node[6+Run]
        Count+=1
        Run+=10
    return Node,invalidTour,TotalScore
```

7. LocalSearchV2.py

```
from openpyxl import load_workbook
from Utility import *
from Recursive import *
from PreProcessLib import *
import ast

def
CheckResTimeWindow(Trip,TimeWindow,ScoreList,CostList,RatioList,TotalRatio,T
otalScore,TotalCost,Compensatelist,TotalCompensate,Budget,h,p,l,d,s,pr,t,o,c,tt):
    nodeCounter=0
    for Node in Trip:
        if (Node in l):
            if o[Node]<=TimeWindow[nodeCounter]<=c[Node]:
                LBreach=False
            else:
                LBreach=True
            LTabu=nodeCounter
        elif (Node in d):
            if o[Node]<=TimeWindow[nodeCounter]<=c[Node]:
                DBreach=False
            else:
                DBreach=True
```

```
            DTabu=nodeCounter
        nodeCounter+=1
    return LBreach, LTabu, DBreach, DTabu

'''Insertion - Insert more attraction to the trip'''
def LocalInsertion(Node,Budget,h,p,l,d,s,pr,t,o,c,tt,DayInTour):
    tempNode=Node.copy()

CombinedTour,ScoreRatio,RangeForEachDay=CombineAllTrip(tempNode,DayInTo
ur)
    GrandTotalScore=0
    multiply=10
    '''Check and Modify each trip'''
    for eachDay in range(0,DayInTour):
        Tabu=0
        Trip=tempNode[0+multiply*eachDay]
        TimeWindow=tempNode[1+multiply*eachDay]
        ScoreList=tempNode[2+multiply*eachDay]
        CostList=tempNode[3+multiply*eachDay]
        RatioList=tempNode[4+multiply*eachDay]
        TotalRatio=tempNode[5+multiply*eachDay]
        TotalScore=tempNode[6+multiply*eachDay]
        TotalCost=tempNode[7+multiply*eachDay]
        Compensatelist=tempNode[8+multiply*eachDay]
        TotalCompensate=tempNode[9+multiply*eachDay]
        LBreach, LTabu, DBreach,
DTabu=CheckResTimeWindow(Trip,TimeWindow,ScoreList,CostList,RatioList,Tota
lRatio,TotalScore,TotalCost,Compensatelist,TotalCompensate,Budget,h,p,l,d,s,pr,t,o,c
,tt)
        '''Add more att to make lunch TW feasible'''
        if LBreach:
            RoundCounter=0
            while LBreach and RoundCounter<3:
                Ratio=0
                for i in p:
                    if (i not in CombinedTour) and (i not in Trip):
                        for j in range(0,LTabu):
                            for k in range(1,LTabu+1):
                                if j < k:
                                    MiniTime=TimeWindow[:k]
                                    MiniComp=Compensatelist[:k]
                                    TW = MiniTime[k-1]+tt.cell(row=5+Trip[k-
1],column=5+i).value+t[i]

TW,twPassFlag,compensate=oneNodeTWCalculation(TW,o,c,i)
                                    if  twPassFlag and TotalCost+pr[i]<=Budget:
```

```
                        x = s[i]/(tt.cell(row=5+Trip[k-1],column=5+i).value +
tt.cell(row=5+i,column=5+Trip[k]).value + compensate)
                        if Ratio < x:
                            MiniTime.append(TW)
                            MiniComp.append(compensate)
                            del TW
                            TempTrip=Trip[:]
                            TempScore=ScoreList[:]
                            TempCost=CostList[:]
                            TempTrip.insert(k,i)
                            TempScore.insert(k,s[i])
                            TempCost.insert(k,pr[i])
                            for RA in range(len(TempTrip[:k]),LTabu+1):
                                TW =
MiniTime[RA]+tt.cell(row=5+TempTrip[RA],column=5+TempTrip[RA+1]).value+t[
TempTrip[RA+1]]

TW,twPassFlag,compensate=oneNodeTWCalculation(TW,o,c,TempTrip[RA+1])
                                if twPassFlag:
                                    MiniTime.append(TW)
                                    MiniComp.append(compensate)
                                else:
                                    break
                            if len(TempTrip)==len(MiniTime):
                                MirrorTime=MiniTime
                                MirrorTrip=TempTrip
                                MirrorComp=MiniComp
                                MirrorScore=TempScore
                                MirrorCost=TempCost
                                Ratio = x
                                SaveScore=s[i]
                                SaveCost=pr[i]
                                targetNode=i
                            del TempTrip, TempScore, TempCost
                        del MiniTime
                        del x
                    break
        if 'MirrorTrip' in locals():
            Trip=MirrorTrip
            TimeWindow=MirrorTime
            CombinedTour.append(targetNode)

TimeWindow,TotalScore,TotalCost,ScoreList,CostList,Compensatelist=CalculateGen
eral(Trip,TimeWindow,pr,s,tt,t,eachDay,DayInTour,o,c)
        TotalRatio, RatioList=CalculateRatio(Trip,s,tt,Compensatelist)
        TotalCompensate=CalculateCompensate(Compensatelist)
```

```
            LBreach, LTabu, DBreach,
DTabu=CheckResTimeWindow(Trip,TimeWindow,ScoreList,CostList,RatioList,Tota
lRatio,TotalScore,TotalCost,Compensatelist,TotalCompensate,Budget,h,p,l,d,s,pr,t,o,c
,tt)
            del MirrorTrip
          else:
            RoundCounter+=1
    '''Add more att to make dinner TW feasible'''
    if DBreach:
      RoundCounter=0
      while DBreach and RoundCounter<3:
        Ratio=0
        for i in p:
          if i not in CombinedTour and (i not in Trip):
            for j in range(LTabu,DTabu):
              for k in range(LTabu+1,DTabu+1):
                if j < k:
                  MiniTime=TimeWindow[:k]
                  MiniComp=Compensatelist[:k]
                  TW = MiniTime[k-1]+tt.cell(row=5+Trip[k-
1],column=5+i).value+t[i]

TW,twPassFlag,compensate=oneNodeTWCalculation(TW,o,c,i)
                  if  twPassFlag and TotalCost+pr[i]<=Budget:
                    x = s[i]/(tt.cell(row=5+Trip[k-1],column=5+i).value +
tt.cell(row=5+i,column=5+Trip[k]).value + compensate)
                    if Ratio < x:
                      MiniTime.append(TW)
                      MiniComp.append(compensate)
                      del TW
                      TempTrip=Trip[:]
                      TempScore=ScoreList[:]
                      TempCost=CostList[:]
                      TempTrip.insert(k,i)
                      TempScore.insert(k,s[i])
                      TempCost.insert(k,pr[i])
                      for RA in range(len(TempTrip[:k]),len(TempTrip)-1):
                        TW =
MiniTime[RA]+tt.cell(row=5+TempTrip[RA],column=5+TempTrip[RA+1]).value+t[
TempTrip[RA+1]]

TW,twPassFlag,compensate=oneNodeTWCalculation(TW,o,c,TempTrip[RA+1])
                        if twPassFlag:
                          MiniTime.append(TW)
                          MiniComp.append(compensate)
                        else:
                          break
```

```
                                    if len(TempTrip)==len(MiniTime):
                                        MirrorTime=MiniTime
                                        MirrorTrip=TempTrip
                                        MirrorComp=MiniComp
                                        MirrorScore=TempScore
                                        MirrorCost=TempCost
                                        Ratio = x
                                        SaveScore=s[i]
                                        SaveCost=pr[i]
                                        targetNode=i
                                    del TempTrip, TempScore, TempCost
                                del MiniTime
                                del x
                            break
                if 'MirrorTrip' in locals():
                    Trip=MirrorTrip
                    TimeWindow=MirrorTime
                    CombinedTour.append(targetNode)

TimeWindow,TotalScore,TotalCost,ScoreList,CostList,Compensatelist=CalculateGen
eral(Trip,TimeWindow,pr,s,tt,t,eachDay,DayInTour,o,c)
                    TotalRatio, RatioList=CalculateRatio(Trip,s,tt,Compensatelist)
                    TotalCompensate=CalculateCompensate(Compensatelist)
                    LBreach, LTabu, DBreach,
DTabu=CheckResTimeWindow(Trip,TimeWindow,ScoreList,CostList,RatioList,Tota
lRatio,TotalScore,TotalCost,Budget,h,p,l,d,s,pr,t,o,c,tt)
                    del MirrorTrip
                else:
                    RoundCounter+=1
        '''Randomly insert new att node to the trip if possible'''
        Round=0
        while (Round<1):
            Ratio = 0
            for i in p:
                if i not in CombinedTour and (i not in Trip):
                    for j in range(0,len(Trip)-1):
                        for k in range(1,len(Trip)):
                            if j < k:
                                MiniTime=TimeWindow[:k]
                                MiniComp=Compensatelist[:k]
                                TW = MiniTime[k-1]+tt.cell(row=5+Trip[k-
1],column=5+i).value+t[i]
                                TW,twPassFlag,compensate=oneNodeTWCalculation(TW,o,c,i)
                                if twPassFlag and TotalCost+pr[i]<=Budget:
                                    x = s[i]/(tt.cell(row=5+Trip[k-1],column=5+i).value +
tt.cell(row=5+i,column=5+Trip[k]).value + compensate)
                                if Ratio < x:
```

```
                                MiniTime.append(TW)
                                MiniComp.append(compensate)
                                del TW
                                TempTrip=Trip[:]
                                TempScore=ScoreList[:]
                                TempCost=CostList[:]
                                TempTrip.insert(k,i)
                                TempScore.insert(k,s[i])
                                TempCost.insert(k,pr[i])
                                for RA in range(len(TempTrip[:k]),len(TempTrip)-1):
                                   TW =
MiniTime[RA]+tt.cell(row=5+TempTrip[RA],column=5+TempTrip[RA+1]).value+t[
TempTrip[RA+1]]

TW,twPassFlag,compensate=oneNodeTWCalculation(TW,o,c,TempTrip[RA+1])
                                   if twPassFlag:
                                      MiniTime.append(TW)
                                      MiniComp.append(compensate)
                                   else:
                                      break
                                if len(TempTrip)==len(MiniTime):
                                   MirrorTime=MiniTime
                                   MirrorTrip=TempTrip
                                   MirrorComp=MiniComp
                                   MirrorScore=TempScore
                                   MirrorCost=TempCost
                                   Ratio = x
                                   SaveScore=s[i]
                                   SaveCost=pr[i]
                                   targetNode=i
                                del TempTrip, TempScore, TempCost
                             del MiniTime
                             del x
                          break
            if 'MirrorTrip' in locals():
               Trip=MirrorTrip
               TimeWindow=MirrorTime
               CombinedTour.append(targetNode)

TimeWindow,TotalScore,TotalCost,ScoreList,CostList,Compensatelist=CalculateGen
eral(Trip,TimeWindow,pr,s,tt,t,eachDay,DayInTour,o,c)
               TotalRatio, RatioList=CalculateRatio(Trip,s,tt,Compensatelist)
               del MirrorTime, MirrorTrip, MirrorScore, MirrorCost, SaveScore, SaveCost
            else:
               Round+=1
            if Round>0:
               TotalRatio, RatioList = CalculateRatio(Trip,s,tt,Compensatelist)
```

```
            TotalCompensate=CalculateCompensate(Compensatelist)
        tempNode[0+multiply*eachDay]=Trip
        tempNode[1+multiply*eachDay]=TimeWindow
        tempNode[2+multiply*eachDay]=ScoreList
        tempNode[3+multiply*eachDay]=CostList
        tempNode[4+multiply*eachDay]=RatioList
        tempNode[5+multiply*eachDay]=TotalRatio
        tempNode[6+multiply*eachDay]=TotalScore
        tempNode[7+multiply*eachDay]=TotalCost
        tempNode[8+multiply*eachDay]=Compensatelist
        tempNode[9+multiply*eachDay]=TotalCompensate
        GrandTotalScore+=TotalScore
    return tempNode,GrandTotalScore

'"1. Real Insertion"'
def Insertion(Node,Budget,h,p,l,d,s,pr,t,o,c,tt,DayInTour):
    tempNode=Node.copy()
    multiply=10

CombinedTour,ScoreRatio,RangeForEachDay=CombineAllTrip(tempNode,DayInTo
ur)
    for day in range(0,DayInTour):
        Trip=Node[0+multiply*day]
        TimeWindow=Node[1+multiply*day]
        Scorelist=Node[2+multiply*day]
        Costlist=Node[3+multiply*day]
        Ratiolist=Node[4+multiply*day]
        TotalRatio=Node[5+multiply*day]
        Score=Node[6+multiply*day]
        Cost=Node[7+multiply*day]
        Compensatelist=Node[8+multiply*day]
        TotalCompensate=Node[9+multiply*day]
        Round=0
        while (Round<1):
            Ratio = 0
            for i in p:
                if (i not in CombinedTour) and (i not in Trip):
                    for j in range(0,len(Trip)-1):
                        for k in range(1,len(Trip)):
                            if j < k:
                                MiniTime=TimeWindow[:k]
                                MiniComp=Compensatelist[:k]
                                TW = MiniTime[k-1]+tt.cell(row=5+Trip[k-
1],column=5+i).value+t[i]
                                TW,twPassFlag,compensate=oneNodeTWCalculation(TW,o,c,i)
                                if twPassFlag and Cost+pr[i]<=Budget:
```

```
                        x = s[i]/(tt.cell(row=5+Trip[k-1],column=5+i).value +
tt.cell(row=5+i,column=5+Trip[k]).value + compensate)
                        if Ratio < x:
                            MiniTime.append(TW)
                            MiniComp.append(compensate)
                            del TW
                            TempTrip=Trip[:]
                            TempScore=Scorelist[:]
                            TempCost=Costlist[:]
                            TempTrip.insert(k,i)
                            TempScore.insert(k,s[i])
                            TempCost.insert(k,pr[i])
                            for RA in range(len(TempTrip[:k]),len(TempTrip)-1):
                                TW =
MiniTime[RA]+tt.cell(row=5+TempTrip[RA],column=5+TempTrip[RA+1]).value+t[
TempTrip[RA+1]]

TW,twPassFlag,compensate=oneNodeTWCalculation(TW,o,c,TempTrip[RA+1])
                                if twPassFlag:
                                    MiniTime.append(TW)
                                    MiniComp.append(compensate)
                                else:
                                    break
                            if len(TempTrip)==len(MiniTime):
                                MirrorTime=MiniTime
                                MirrorTrip=TempTrip
                                MirrorComp=MiniComp
                                MirrorScore=TempScore
                                MirrorCost=TempCost
                                Ratio = x
                                SaveScore=s[i]
                                SaveCost=pr[i]
                                CombinedTour.append(i)
                            del TempTrip, TempScore, TempCost
                        del MiniTime
                        del x
                    break
        if 'MirrorTrip' in locals():
            Trip=MirrorTrip
            TimeWindow=MirrorTime

TimeWindow,Score,Cost,Scorelist,Costlist,Compensatelist=CalculateGeneral(Trip,Ti
meWindow,pr,s,tt,t,day,DayInTour,o,c)
            del MirrorTime, MirrorTrip, MirrorScore, MirrorCost, SaveScore, SaveCost
        else:
            Round+=1
        if Round>0:
```

```
            TotalRatio, RatioList = CalculateRatio(Trip,s,tt,Compensatelist)
            TotalCompensate=CalculateCompensate(Compensatelist)
        Node[0+multiply*day]=Trip
        Node[1+multiply*day]=TimeWindow
        Node[2+multiply*day]=Scorelist
        Node[3+multiply*day]=Costlist
        Node[4+multiply*day]=Ratiolist
        Node[5+multiply*day]=TotalRatio
        Node[6+multiply*day]=Score
        Node[7+multiply*day]=Cost
        Node[8+multiply*day]=Compensatelist
        Node[9+multiply*day]=TotalCompensate
    '''Calculate Total Score before save the result'''
    Count=0
    TotalScore=0
    Run=0
    while Count<DayInTour:
        TotalScore+=Node[6+Run]
        Count+=1
        Run+=10
    return Node, TotalScore

'''2. Two-Opt between 2 trips, switch attraction with attraction and Res with Res'''
def TwoOpt(Node,Budget,h,p,l,d,s,pr,t,o,c,tt,DayInTour):
    improveCount=0
    improveFlag=False
    multiply=10
    for day in range(DayInTour):
        Trip=Node[0+multiply*day]
        TimeWindow=Node[1+multiply*day]
        Scorelist=Node[2+multiply*day]
        Costlist=Node[3+multiply*day]
        Ratiolist=Node[4+multiply*day]
        TotalRatio=Node[5+multiply*day]
        Score=Node[6+multiply*day]
        Cost=Node[7+multiply*day]
        Compensatelist=Node[8+multiply*day]
        TotalCompensate=Node[9+multiply*day]
        LastTime=TimeWindow[len(TimeWindow)-1]

Trip,TimeWindow,Scorelist,Costlist,Ratiolist,TotalRatio,Score,Cost,Compensatelist,
TotalCompensate=TwO(Trip,TimeWindow,Scorelist,Costlist,Ratiolist,TotalRatio,Sco
re,Cost,Compensatelist,TotalCompensate,Budget,h,p,l,d,s,pr,t,o,c,tt)

TimeWindow,Score,Cost,Scorelist,Costlist,Compensatelist=CalculateGeneral(Trip,Ti
meWindow,pr,s,tt,t,day,DayInTour,o,c)
        TotalRatio,Ratiolist=CalculateRatio(Trip,s,tt,Compensatelist)
```

```
            TotalCompensate=CalculateCompensate(Compensatelist)
            Node[0+multiply*day]=Trip
            Node[1+multiply*day]=TimeWindow
            Node[2+multiply*day]=Scorelist
            Node[3+multiply*day]=Costlist
            Node[4+multiply*day]=Ratiolist
            Node[5+multiply*day]=TotalRatio
            Node[6+multiply*day]=Score
            Node[7+multiply*day]=Cost
            Node[8+multiply*day]=Compensatelist
            Node[9+multiply*day]=TotalCompensate
            if LastTime!=TimeWindow[len(TimeWindow)-1]:
                improveCount+=1
        if improveCount>0:
            improveFlag=True
        '''Calculate Total Score before save the result'''
        Count=0
        TotalScore=0
        Run=0
        while Count<DayInTour:
            TotalScore+=Node[6+Run]
            Count+=1
            Run+=10
        return Node, TotalScore, improveFlag

'''3. Move best, try to move each attraction to the best position in trip or tour'''
def MoveBest(Node,Budget,h,p,l,d,s,pr,t,o,c,tt,DayInTour):
    multiply=10
    MaxDiff=0
    DayFromSwitch=0
    DayToSwitch=0
    tempNode=Node.copy()
    improveFlag=False
    currentSolution=0
    totalDiff=0
    for day in range(DayInTour):
        Trip=Node[0+multiply*day]# i would be used with day
        TimeWindow=Node[1+multiply*day]
        Scorelist=Node[2+multiply*day]
        Costlist=Node[3+multiply*day]
        Ratiolist=Node[4+multiply*day]
        TotalRatio=Node[5+multiply*day]
        Score=Node[6+multiply*day]
        Cost=Node[7+multiply*day]
        Compensatelist=Node[8+multiply*day]
        TotalCompensate=Node[9+multiply*day]
        minCompensate=TotalCompensate
```

```
      for i in range(1,len(Trip[1:])):
         if Trip[i] not in l+d:
            for dayLoop in range(DayInTour):
               if day==dayLoop: #Pair with the same day move i to j
                  for j in range(1,len(Trip[1:])):
                     if Trip[j] not in l+d:
                        if (j!=i) and (j==i+1):# For i occurred before j but next to each
other
                           P = (tt.cell(row=5+Trip[i-1],column=5+Trip[j]).value +
tt.cell(row=5+Trip[j],column=5+Trip[i]).value +
tt.cell(row=5+Trip[i],column=5+Trip[j+1]).value)
                           Old = (tt.cell(row=5+Trip[i-1],column=5+Trip[i]).value +
tt.cell(row=5+Trip[i],column=5+Trip[j]).value +
tt.cell(row=5+Trip[j],column=5+Trip[j+1]).value)
                           Diff = Old - P
                           if Diff > MaxDiff:
                              MiniTime=TimeWindow[:i]
                              MiniComp=Compensatelist[:i]
                              TW = MiniTime[i-1] + tt.cell(row=5+Trip[i-
1],column=5+Trip[j]).value + t[Trip[j]]

TW,twPassFlag,compensate=oneNodeTWCalculation(TW,o,c,Trip[j])
                              if twPassFlag:
                                 MiniTime.append(TW)
                                 MiniComp.append(compensate)
                                 del TW
                                 TempTrip=Trip[:]
                                 TempScore=Scorelist[:]
                                 TempCost=Costlist[:]
                                 TempTrip[i],TempTrip[j] = TempTrip[j],TempTrip[i]
                                 TempScore[i],TempScore[j] =
TempScore[j],TempScore[i]
                                 TempCost[i],TempCost[j] = TempCost[j],TempCost[i]
                                 for RA in range(len(TempTrip[:i]),len(TempTrip)-1):
                                    TW =
MiniTime[RA]+tt.cell(row=5+TempTrip[RA],column=5+TempTrip[RA+1]).value+t[
TempTrip[RA+1]]

TW,twPassFlag,compensate=oneNodeTWCalculation(TW,o,c,TempTrip[RA+1])
                                    if twPassFlag:
                                       MiniTime.append(TW)
                                       MiniComp.append(compensate)
                                    else:
                                       break
                                 if len(TempTrip)==len(MiniTime):

tempMinCompensate=CalculateCompensate(MiniComp)
```

```
                                compensateDiff=TotalCompensate-
tempMinCompensate
                                tempTotalDiff=Diff+compensateDiff
                                if tempTotalDiff >= totalDiff:
                                    MirrorTime=MiniTime
                                    MirrorComp=MiniComp
                                    minCompensate=tempMinCompensate
                                    totalDiff=tempTotalDiff
                                    MirrorTrip=TempTrip
                                    MirrorScore=TempScore
                                    MirrorCost=TempCost
                                    MaxDiff = Diff
                                    DayFromSwitch = day
                                    #DayToSwitch = dayLoop
                                    SaveScore=Score
                                    SaveCost=Cost
                                    currentSolution=1
                                del TempTrip, TempScore, TempCost
                            del MiniTime
                        del P, Old
                    elif (j!=i) and (j==i-1):# For i occurred after j but next to each
other
                        P = (tt.cell(row=5+Trip[j-1],column=5+Trip[i]).value +
tt.cell(row=5+Trip[i],column=5+Trip[j]).value +
tt.cell(row=5+Trip[j],column=5+Trip[i+1]).value)
                        Old = (tt.cell(row=5+Trip[j-1],column=5+Trip[j]).value +
tt.cell(row=5+Trip[j],column=5+Trip[i]).value +
tt.cell(row=5+Trip[i],column=5+Trip[i+1]).value)
                        Diff = Old - P
                        if Diff > MaxDiff:
                            MiniTime=TimeWindow[:j]
                            MiniComp=Compensatelist[:j]
                            TW = MiniTime[j-1] + tt.cell(row=5+Trip[j-
1],column=5+Trip[i]).value + t[Trip[i]]

TW,twPassFlag,compensate=oneNodeTWCalculation(TW,o,c,Trip[i])
                            if twPassFlag:
                                MiniTime.append(TW)
                                MiniComp.append(compensate)
                                del TW
                                TempTrip=Trip[:]
                                TempScore=Scorelist[:]
                                TempCost=Costlist[:]
                                TempTrip[i],TempTrip[j] = TempTrip[j],TempTrip[i]
                                TempScore[i],TempScore[j] =
TempScore[j],TempScore[i]
                                TempCost[i],TempCost[j] = TempCost[j],TempCost[i]
```

```
                                for RA in range(len(TempTrip[:j]),len(TempTrip)-1):
                                    TW =
MiniTime[RA]+tt.cell(row=5+TempTrip[RA],column=5+TempTrip[RA+1]).value+t[
TempTrip[RA+1]]

TW,twPassFlag,compensate=oneNodeTWCalculation(TW,o,c,TempTrip[RA+1])
                                    if twPassFlag:
                                        MiniTime.append(TW)
                                        MiniComp.append(compensate)
                                    else:
                                        break
                                if len(TempTrip)==len(MiniTime):

tempMinCompensate=CalculateCompensate(MiniComp)
                                    compensateDiff=TotalCompensate-
tempMinCompensate
                                    tempTotalDiff=Diff+compensateDiff
                                    if tempTotalDiff >= totalDiff:
                                        MirrorTime=MiniTime
                                        MirrorTrip=TempTrip
                                        MirrorComp=MiniComp
                                        minCompensate=tempMinCompensate
                                        totalDiff=tempTotalDiff
                                        MirrorScore=TempScore
                                        MirrorCost=TempCost
                                        MaxDiff = Diff
                                        DayFromSwitch = day
                                        #DayToSwitch = dayLoop
                                        SaveScore=Score
                                        SaveCost=Cost
                                        currentSolution=1
                                    del TempTrip, TempScore, TempCost
                                del MiniTime
                            del P, Old
                        elif (j!=i) and (j>i+1):# For i occurred before j and distance more
than 1
                            P = (tt.cell(row=5+Trip[j-1],column=5+Trip[i]).value +
tt.cell(row=5+Trip[i],column=5+Trip[j]).value)
                            P2 = (tt.cell(row=5+Trip[i-1],column=5+Trip[i+1]).value)
                            Old = (tt.cell(row=5+Trip[i-1],column=5+Trip[i]).value +
tt.cell(row=5+Trip[i],column=5+Trip[i+1]).value)
                            Old2 = (tt.cell(row=5+Trip[j-1],column=5+Trip[j]).value)
                            Diff = (Old - Old2) - (P + P2)
                            if Diff > MaxDiff:
                                MiniTime=TimeWindow[:i]
                                MiniComp=Compensatelist[:i]
```

```
                        TW = MiniTime[i-1] + tt.cell(row=5+Trip[i-
1],column=5+Trip[i+1]).value + t[Trip[i+1]]

TW,twPassFlag,compensate=oneNodeTWCalculation(TW,o,c,Trip[i+1])
                        if twPassFlag:
                            MiniTime.append(TW)
                            MiniComp.append(compensate)
                            del TW
                            TempTrip=Trip[:]
                            TempScore=Scorelist[:]
                            TempCost=Costlist[:]
                            TempTrip.insert(j,TempTrip[i])
                            TempTrip.pop(i)
                            TempScore.insert(j,TempScore[i])
                            TempScore.pop(i)
                            TempCost.insert(i,TempCost[i])
                            TempCost.pop(i)
                            for RA in range(len(TempTrip[:i]),len(TempTrip)-1):
                                TW =
MiniTime[RA]+tt.cell(row=5+TempTrip[RA],column=5+TempTrip[RA+1]).value+t[
TempTrip[RA+1]]

TW,twPassFlag,compensate=oneNodeTWCalculation(TW,o,c,TempTrip[RA+1])
                                if twPassFlag:
                                    MiniTime.append(TW)
                                    MiniComp.append(compensate)
                                else:
                                    break
                            if len(TempTrip)==len(MiniTime):

tempMinCompensate=CalculateCompensate(MiniComp)
                                compensateDiff=TotalCompensate-
tempMinCompensate
                                tempTotalDiff=Diff+compensateDiff
                                if tempTotalDiff >= totalDiff:
                                    MirrorTime=MiniTime
                                    MirrorTrip=TempTrip
                                    MirrorScore=TempScore
                                    MirrorCost=TempCost
                                    MirrorComp=MiniComp
                                    minCompensate=tempMinCompensate
                                    totalDiff=tempTotalDiff
                                    MaxDiff = Diff
                                    DayFromSwitch = day
                                    #DayToSwitch = dayLoop
                                    SaveScore=Score
                                    SaveCost=Cost
```

```
                                currentSolution=1
                            del TempTrip, TempScore, TempCost
                        del MiniTime
                    del P, P2, Old, Old2
                elif (j!=i) and (j<i-1):# For i occurred after j and distance more
than 1
                    P = (tt.cell(row=5+Trip[j-1],column=5+Trip[i]).value +
tt.cell(row=5+Trip[i],column=5+Trip[j]).value)
                    P2 = (tt.cell(row=5+Trip[i-1],column=5+Trip[i+1]).value)
                    Old = (tt.cell(row=5+Trip[i-1],column=5+Trip[i]).value +
tt.cell(row=5+Trip[i],column=5+Trip[i+1]).value)
                    Old2 = (tt.cell(row=5+Trip[j-1],column=5+Trip[j]).value)
                    Diff = (Old + Old2) - (P + P2)
                    if Diff > MaxDiff:
                        MiniTime=TimeWindow[:j]
                        MiniComp=Compensatelist[:j]
                        TW = MiniTime[j-1] + tt.cell(row=5+Trip[j-
1],column=5+Trip[i]).value + t[Trip[i]]

TW,twPassFlag,compensate=oneNodeTWCalculation(TW,o,c,Trip[i])
                        if twPassFlag:
                            MiniTime.append(TW)
                            MiniComp.append(compensate)
                            del TW
                            TempTrip=Trip[:]
                            TempScore=Scorelist[:]
                            TempCost=Costlist[:]
                            TempTrip.insert(j,TempTrip[i])
                            TempTrip.pop(i+1)
                            TempScore.insert(j,TempScore[i])
                            TempScore.pop(i+1)
                            TempCost.insert(i,TempCost[i])
                            TempCost.pop(i+1)
                            for RA in range(len(TempTrip[:j]),len(TempTrip)-1):
                                TW =
MiniTime[RA]+tt.cell(row=5+TempTrip[RA],column=5+TempTrip[RA+1]).value+t[
TempTrip[RA+1]]

TW,twPassFlag,compensate=oneNodeTWCalculation(TW,o,c,TempTrip[RA+1])
                                if twPassFlag:
                                    MiniTime.append(TW)
                                    MiniComp.append(compensate)
                                else:
                                    break
                            if len(TempTrip)==len(MiniTime):

tempMinCompensate=CalculateCompensate(MiniComp)
```

```
                                compensateDiff=TotalCompensate-
tempMinCompensate
                                tempTotalDiff=Diff+compensateDiff
                                if tempTotalDiff >= totalDiff:
                                    MirrorTime=MiniTime
                                    MirrorTrip=TempTrip
                                    MirrorScore=TempScore
                                    MirrorCost=TempCost
                                    MirrorComp=MiniComp
                                    minCompensate=tempMinCompensate
                                    totalDiff=tempTotalDiff
                                    MaxDiff = Diff
                                    DayFromSwitch = day
                                    #DayToSwitch = dayLoop
                                    SaveScore=Score
                                    SaveCost=Cost
                                    currentSolution=1
                                del TempTrip, TempScore, TempCost
                            del MiniTime
                        del P, P2, Old, Old2
                else: #Pair with other day in tour move i to j
                    Trip2=Node[0+multiply*dayLoop]# j would be used with dayLoop
                    TimeWindow2=Node[1+multiply*dayLoop]
                    Scorelist2=Node[2+multiply*dayLoop]
                    Costlist2=Node[3+multiply*dayLoop]
                    Ratiolist2=Node[4+multiply*dayLoop]
                    TotalRatio2=Node[5+multiply*dayLoop]
                    Score2=Node[6+multiply*dayLoop]
                    Cost2=Node[7+multiply*dayLoop]
                    Compensatelist2=Node[8+multiply*dayLoop]
                    TotalCompensate2=Node[9+multiply*dayLoop]
                    minCompensate2=TotalCompensate2
                    for j in range(1,len(Trip2[1:])):
                        if Trip2[j] not in l+d:
                            P = (tt.cell(row=5+Trip2[j-1],column=5+Trip[i]).value +
tt.cell(row=5+Trip[i],column=5+Trip2[j]).value)# i must be moved from current
position, so i-1 will connected to i+1
                            P2 = (tt.cell(row=5+Trip[i-1],column=5+Trip[i+1]).value)# i will
be placed in place of j, so it would be placed between j-1 and j
                            Old = (tt.cell(row=5+Trip[i-1],column=5+Trip[i]).value +
tt.cell(row=5+Trip[i],column=5+Trip[i+1]).value)
                            Old2 = (tt.cell(row=5+Trip2[j-1],column=5+Trip2[j]).value)
                            Diff = (Old + Old2) - (P + P2)
                            if Diff > MaxDiff:
                                MiniTime=TimeWindow[:i]
                                MiniTime2=TimeWindow2[:j]
                                MiniComp=Compensatelist[:i]
```

```
                        MiniComp2=Compensatelist2[:j]
                        TW = MiniTime[i-1] + tt.cell(row=5+Trip[i-
1],column=5+Trip[i+1]).value + t[Trip[i+1]]
                        TW2 = MiniTime2[j-1] + tt.cell(row=5+Trip2[j-
1],column=5+Trip[i]).value + t[Trip[i]]

TW,twPassFlag,compensate=oneNodeTWCalculation(TW,o,c,Trip[i+1])

TW2,twPassFlag2,compensate2=oneNodeTWCalculation(TW2,o,c,Trip[i])
                        if twPassFlag and twPassFlag2:
                            MiniTime.append(TW)
                            MiniTime2.append(TW2)
                            MiniComp.append(compensate)
                            MiniComp2.append(compensate2)
                            del TW
                            TempTrip=Trip[:]
                            TempScore=Scorelist[:]
                            TempCost=Costlist[:]
                            #MiniTime=TimeWindow[:i+1]
                            TempTrip2=Trip2[:]
                            TempScore2=Scorelist2[:]
                            TempCost2=Costlist2[:]
                            TempTrip2.insert(j,Trip[i])
                            TempScore2.insert(j,Scorelist[i])
                            TempCost2.insert(j,Costlist[i])
                            TempTrip.pop(i)
                            TempScore.pop(i)
                            TempCost.pop(i)
                            #Already changed from i to i-1
                            for RA in range(len(TempTrip[:i]),len(TempTrip)-1):
                                TW =
MiniTime[RA]+tt.cell(row=5+TempTrip[RA],column=5+TempTrip[RA+1]).value+t[
TempTrip[RA+1]]

TW,twPassFlag,compensate=oneNodeTWCalculation(TW,o,c,TempTrip[RA+1])
                                if twPassFlag:
                                    MiniTime.append(TW)
                                    MiniComp.append(compensate)
                                else:
                                    break
                            for RA2 in range(len(TempTrip2[:j]),len(TempTrip2)-1):
                                TW =
MiniTime2[RA2]+tt.cell(row=5+TempTrip2[RA2],column=5+TempTrip2[RA2+1]).
value+t[TempTrip2[RA2+1]]

TW,twPassFlag,compensate2=oneNodeTWCalculation(TW,o,c,TempTrip2[RA2+1])
                                if twPassFlag:
```

```
                                MiniTime2.append(TW)
                                MiniComp2.append(compensate2)
                            else:
                                break
                        if len(TempTrip)==len(MiniTime) and
len(TempTrip2)==len(MiniTime2):
                            tempMinCompensate=CalculateCompensate(MiniComp)
                            compensateDiff=TotalCompensate-tempMinCompensate

tempMinCompensate2=CalculateCompensate(MiniComp2)
                            compensateDiff2=TotalCompensate2-
tempMinCompensate2
                            tempTotalDiff=Diff+compensateDiff+compensateDiff2
                            if tempTotalDiff >= totalDiff:
                                MirrorTime=MiniTime
                                MirrorComp=MiniComp
                                minCompensate=tempMinCompensate
                                MirrorTrip=TempTrip
                                MirrorScore=TempScore
                                MirrorCost=TempCost
                                SaveScore=Score-Scorelist[i]
                                SaveCost=Score-Costlist[i]
                                MaxDiff = Diff
                                DayFromSwitch = day
                                DayToSwitch = dayLoop
                                #Variable for second trip
                                MirrorTime2=MiniTime2
                                MirrorComp2=MiniComp2
                                minCompensate2=tempMinCompensate2
                                MirrorTrip2=TempTrip2
                                MirrorScore2=TempScore2
                                MirrorCost2=TempCost2
                                SaveScore2=Score2+Scorelist[i]
                                SaveCost2=Score2+Costlist[i]
                                currentSolution=2
                                totalDiff=tempTotalDiff
                        del TempTrip, TempScore, TempCost, TempTrip2,
TempScore2, TempCost2
                    del MiniTime2
                del P, P2, Old, Old2
    if currentSolution==1:

MirrorTime,SaveScore,SaveCost,MirrorScore,MirrorCost,MiniComp=CalculateGene
ral(MirrorTrip,MirrorTime,pr,s,tt,t,DayFromSwitch,DayInTour,o,c)
        TotalRatio, RatioList = CalculateRatio(MirrorTrip,s,tt,MiniComp)
        TotalCompensate=CalculateCompensate(MiniComp)
        tempNode[0+multiply*DayFromSwitch]=MirrorTrip
```

```
            tempNode[1+multiply*DayFromSwitch]=MirrorTime
            tempNode[2+multiply*DayFromSwitch]=MirrorScore
            tempNode[3+multiply*DayFromSwitch]=MirrorCost
            tempNode[4+multiply*DayFromSwitch]=Ratiolist
            tempNode[5+multiply*DayFromSwitch]=TotalRatio
            tempNode[6+multiply*DayFromSwitch]=SaveScore
            tempNode[7+multiply*DayFromSwitch]=SaveCost
            tempNode[8+multiply*DayFromSwitch]=MiniComp
            tempNode[9+multiply*DayFromSwitch]=TotalCompensate
            improveFlag=True
        elif currentSolution==2:

MirrorTime,SaveScore,SaveCost,MirrorScore,MirrorCost,MiniComp=CalculateGene
ral(MirrorTrip,MirrorTime,pr,s,tt,t,DayFromSwitch,DayInTour,o,c)

MirrorTime2,SaveScore2,SaveCost2,MirrorScore2,MirrorCost2,MiniComp2=Calcula
teGeneral(MirrorTrip2,MirrorTime2,pr,s,tt,t,DayToSwitch,DayInTour,o,c)
            TotalRatio, RatioList = CalculateRatio(MirrorTrip,s,tt,MiniComp)
            TotalRatio2, RatioList2 = CalculateRatio(MirrorTrip2,s,tt,MiniComp2)
            TotalCompensate=CalculateCompensate(MiniComp)
            TotalCompensate2=CalculateCompensate(MiniComp2)
            tempNode[0+multiply*DayFromSwitch]=MirrorTrip
            tempNode[1+multiply*DayFromSwitch]=MirrorTime
            tempNode[2+multiply*DayFromSwitch]=MirrorScore
            tempNode[3+multiply*DayFromSwitch]=MirrorCost
            tempNode[4+multiply*DayFromSwitch]=Ratiolist
            tempNode[5+multiply*DayFromSwitch]=TotalRatio
            tempNode[6+multiply*DayFromSwitch]=SaveScore
            tempNode[7+multiply*DayFromSwitch]=SaveCost
            tempNode[8+multiply*DayFromSwitch]=MiniComp
            tempNode[9+multiply*DayFromSwitch]=TotalCompensate
            tempNode[0+multiply*DayToSwitch]=MirrorTrip2
            tempNode[1+multiply*DayToSwitch]=MirrorTime2
            tempNode[2+multiply*DayToSwitch]=MirrorScore2
            tempNode[3+multiply*DayToSwitch]=MirrorCost2
            tempNode[4+multiply*DayToSwitch]=Ratiolist2
            tempNode[5+multiply*DayToSwitch]=TotalRatio2
            tempNode[6+multiply*DayToSwitch]=SaveScore2
            tempNode[7+multiply*DayToSwitch]=SaveCost2
            tempNode[8+multiply*DayToSwitch]=MiniComp2
            tempNode[9+multiply*DayToSwitch]=TotalCompensate2
            improveFlag=True
        else:
            dummyInt=0
        '''Calculate Total Score before save the result'''
        Count=0
        TotalScore=0
```

```
    Run=0
    while Count<DayInTour:
        TotalScore+=tempNode[6+Run]
        Count+=1
        Run+=10
    return tempNode, TotalScore, improveFlag

"'4. Swap and Swap Best"'
def SwapNSwapBest(Node,Budget,h,p,l,d,s,pr,t,o,c,tt,DayInTour):
    multiply=10
    whatCase=1
    MaxDiff=0
    DayFromSwitch=0
    DayToSwitch=0
    tempNode=Node.copy()
    improveFlag=False
    currentSolution=0
    totalDiff=0
    for day in range(DayInTour):
        Trip=Node[0+multiply*day]# i would be used with day
        TimeWindow=Node[1+multiply*day]
        Scorelist=Node[2+multiply*day]
        Costlist=Node[3+multiply*day]
        Ratiolist=Node[4+multiply*day]
        TotalRatio=Node[5+multiply*day]
        Score=Node[6+multiply*day]
        Cost=Node[7+multiply*day]
        Compensatelist=Node[8+multiply*day]
        TotalCompensate=Node[9+multiply*day]
        minCompensate=TotalCompensate
        for i in range(1,len(Trip[1:])):
            if Trip[i] not in l+d:
                for dayLoop in range(DayInTour):
                    if day==dayLoop: #Pair with the same day move i to j
                        for j in range(1,len(Trip[1:])):
                            if Trip[j] not in l+d:
                                if (j!=i) and (j==i+1):# For i occurred before j but next to each
other
                                    P = (tt.cell(row=5+Trip[i-1],column=5+Trip[j]).value +
tt.cell(row=5+Trip[j],column=5+Trip[i]).value +
tt.cell(row=5+Trip[i],column=5+Trip[j+1]).value)
                                    Old = (tt.cell(row=5+Trip[i-1],column=5+Trip[i]).value +
tt.cell(row=5+Trip[i],column=5+Trip[j]).value +
tt.cell(row=5+Trip[j],column=5+Trip[j+1]).value)
                                    Diff = Old - P
                                    if Diff > MaxDiff:
                                        MiniTime=TimeWindow[:i]
```

```
                        MiniComp=Compensatelist[:i]
                        TW = MiniTime[i-1] + tt.cell(row=5+Trip[i-
1],column=5+Trip[j]).value + t[Trip[j]]

TW,twPassFlag,compensate=oneNodeTWCalculation(TW,o,c,Trip[j])
                        if twPassFlag:
                            MiniTime.append(TW)
                            MiniComp.append(compensate)
                            del TW
                            TempTrip=Trip[:]
                            TempScore=Scorelist[:]
                            TempCost=Costlist[:]
                            TempTrip[i],TempTrip[j] = TempTrip[j],TempTrip[i]
                            TempScore[i],TempScore[j] =
TempScore[j],TempScore[i]
                            TempCost[i],TempCost[j] = TempCost[j],TempCost[i]
                            for RA in range(len(TempTrip[:i]),len(TempTrip)-1):
                                TW =
MiniTime[RA]+tt.cell(row=5+TempTrip[RA],column=5+TempTrip[RA+1]).value+t[
TempTrip[RA+1]]

TW,twPassFlag,compensate=oneNodeTWCalculation(TW,o,c,TempTrip[RA+1])
                                if twPassFlag:
                                    MiniTime.append(TW)
                                    MiniComp.append(compensate)
                                else:
                                    break
                            if len(TempTrip)==len(MiniTime):

tempMinCompensate=CalculateCompensate(MiniComp)
                                compensateDiff=TotalCompensate-
tempMinCompensate
                                tempTotalDiff=Diff+compensateDiff
                                if tempTotalDiff >= totalDiff:
                                    MirrorTime=MiniTime
                                    MirrorComp=MiniComp
                                    MirrorTrip=TempTrip
                                    MirrorScore=TempScore
                                    MirrorCost=TempCost
                                    minCompensate=tempMinCompensate
                                    totalDiff=tempTotalDiff
                                    MaxDiff = Diff
                                    DayFromSwitch = day
                                    #DayToSwitch = dayLoop
                                    SaveScore=Score
                                    SaveCost=Cost
                                    currentSolution=1
```

```
                            del TempTrip, TempScore, TempCost
                        del MiniTime
                    del P, Old
                elif (j!=i) and (j==i-1):# For i occurred after j but next to each
other
                    P = (tt.cell(row=5+Trip[j-1],column=5+Trip[i]).value +
tt.cell(row=5+Trip[i],column=5+Trip[j]).value +
tt.cell(row=5+Trip[j],column=5+Trip[i+1]).value)
                    Old = (tt.cell(row=5+Trip[j-1],column=5+Trip[j]).value +
tt.cell(row=5+Trip[j],column=5+Trip[i]).value +
tt.cell(row=5+Trip[i],column=5+Trip[i+1]).value)
                    Diff = Old - P
                    if Diff > MaxDiff:
                        MiniTime=TimeWindow[:j]
                        MiniComp=Compensatelist[:j]
                        TW = MiniTime[j-1] + tt.cell(row=5+Trip[j-
1],column=5+Trip[i]).value + t[Trip[i]]

TW,twPassFlag,compensate=oneNodeTWCalculation(TW,o,c,Trip[i])
                        if twPassFlag:
                            MiniTime.append(TW)
                            MiniComp.append(compensate)
                            del TW
                            TempTrip=Trip[:]
                            TempScore=Scorelist[:]
                            TempCost=Costlist[:]
                            TempTrip[i],TempTrip[j] = TempTrip[j],TempTrip[i]
                            TempScore[i],TempScore[j] =
TempScore[j],TempScore[i]
                            TempCost[i],TempCost[j] = TempCost[j],TempCost[i]
                            for RA in range(len(TempTrip[:j]),len(TempTrip)-1):
                                TW =
MiniTime[RA]+tt.cell(row=5+TempTrip[RA],column=5+TempTrip[RA+1]).value+t[
TempTrip[RA+1]]

TW,twPassFlag,compensate=oneNodeTWCalculation(TW,o,c,TempTrip[RA+1])
                                if twPassFlag:
                                    MiniTime.append(TW)
                                    MiniComp.append(compensate)
                                else:
                                    break
                            if len(TempTrip)==len(MiniTime):

tempMinCompensate=CalculateCompensate(MiniComp)
                                compensateDiff=TotalCompensate-
tempMinCompensate
                                tempTotalDiff=Diff+compensateDiff
```

```
                                    if tempTotalDiff >= totalDiff:
                                        MirrorTime=MiniTime
                                        MirrorTrip=TempTrip
                                        MirrorComp=MiniComp
                                        minCompensate=tempMinCompensate
                                        totalDiff=tempTotalDiff
                                        MirrorScore=TempScore
                                        MirrorCost=TempCost
                                        MaxDiff = Diff
                                        DayFromSwitch = day
                                        #DayToSwitch = dayLoop
                                        SaveScore=Score
                                        SaveCost=Cost
                                        currentSolution=1
                                    del TempTrip, TempScore, TempCost
                                del MiniTime
                            del P, Old
                        elif (j!=i) and (j>i+1):# For i occurred before j and distance more
than 1
                            P = (tt.cell(row=5+Trip[j-1],column=5+Trip[i]).value +
tt.cell(row=5+Trip[i],column=5+Trip[j+1]).value)
                            P2 = (tt.cell(row=5+Trip[i-1],column=5+Trip[j]).value +
tt.cell(row=5+Trip[j],column=5+Trip[i+1]).value)
                            Old = (tt.cell(row=5+Trip[i-1],column=5+Trip[i]).value +
tt.cell(row=5+Trip[i],column=5+Trip[i+1]).value)
                            Old2 = (tt.cell(row=5+Trip[j-1],column=5+Trip[j]).value +
tt.cell(row=5+Trip[j],column=5+Trip[j+1]).value)
                            Diff = (Old - Old2) - (P + P2)
                            if Diff > MaxDiff:
                                MiniTime=TimeWindow[:i]
                                MiniComp=Compenstaelist[:i]
                                TW = MiniTime[i-1] + tt.cell(row=5+Trip[i-
1],column=5+Trip[j]).value + t[Trip[j]]

TW,twPassFlag,compensate=oneNodeTWCalculation(TW,o,c,Trip[j])
                                if twPassFlag:
                                    MiniTime.append(TW)
                                    MiniComp.append(compensate)
                                    del TW
                                    TempTrip=Trip[:]
                                    TempScore=Scorelist[:]
                                    TempCost=Costlist[:]
                                    TempTrip[i],TempTrip[j] = TempTrip[j],TempTrip[i]
                                    TempScore[i],TempScore[j] =
TempScore[j],TempScore[i]
                                    TempCost[i],TempCost[j] = TempCost[j],TempCost[i]
                                    for RA in range(len(TempTrip[:i]),len(TempTrip)-1):
```

```
                                TW =
MiniTime[RA]+tt.cell(row=5+TempTrip[RA],column=5+TempTrip[RA+1]).value+t[
TempTrip[RA+1]]

TW,twPassFlag,compensate=oneNodeTWCalculation(TW,o,c,TempTrip[RA+1])
                                if twPassFlag:
                                   MiniTime.append(TW)
                                   MiniComp.append(compensate)
                                else:
                                   break
                             if len(TempTrip)==len(MiniTime):

tempMinCompensate=CalculateCompensate(MiniComp)
                                compensateDiff=TotalCompensate-
tempMinCompensate
                                tempTotalDiff=Diff+compensateDiff
                                if tempTotalDiff >= totalDiff:
                                   MirrorTime=MiniTime
                                   MirrorTrip=TempTrip
                                   MirrorScore=TempScore
                                   MirrorCost=TempCost
                                   MirrorComp=MiniComp
                                   minCompensate=tempMinCompensate
                                   totalDiff=tempTotalDiff
                                   MaxDiff = Diff
                                   DayFromSwitch = day
                                   #DayToSwitch = dayLoop
                                   SaveScore=Score
                                   SaveCost=Cost
                                   currentSolution=1
                                del TempTrip, TempScore, TempCost
                             del MiniTime
                          del P, P2, Old, Old2
                       elif (j!=i) and (j<i-1):# For i occurred after j and distance more
than 1
                          P = (tt.cell(row=5+Trip[j-1],column=5+Trip[i]).value +
tt.cell(row=5+Trip[i],column=5+Trip[j+1]).value)
                          P2 = (tt.cell(row=5+Trip[i-1],column=5+Trip[j]).value +
tt.cell(row=5+Trip[j],column=5+Trip[i+1]).value)
                          Old = (tt.cell(row=5+Trip[i-1],column=5+Trip[i]).value +
tt.cell(row=5+Trip[i],column=5+Trip[i+1]).value)
                          Old2 = (tt.cell(row=5+Trip[j-1],column=5+Trip[j]).value +
tt.cell(row=5+Trip[j],column=5+Trip[j+1]).value)
                          Diff = (Old + Old2) - (P + P2)
                          if Diff > MaxDiff:
                             MiniTime=TimeWindow[:j]
                             MiniComp=Compensatelist[:j]
```

```
                        TW = MiniTime[j-1] + tt.cell(row=5+Trip[j-
1],column=5+Trip[i]).value + t[Trip[i]]

TW,twPassFlag,compensate=oneNodeTWCalculation(TW,o,c,Trip[i])
                        if twPassFlag:
                            MiniTime.append(TW)
                            MiniComp.append(compensate)
                            del TW
                            TempTrip=Trip[:]
                            TempScore=Scorelist[:]
                            TempCost=Costlist[:]
                            TempTrip[i],TempTrip[j] = TempTrip[j],TempTrip[i]
                            TempScore[i],TempScore[j] =
TempScore[j],TempScore[i]
                            TempCost[i],TempCost[j] = TempCost[j],TempCost[i]
                            for RA in range(len(TempTrip[:j]),len(TempTrip)-1):
                                TW =
MiniTime[RA]+tt.cell(row=5+TempTrip[RA],column=5+TempTrip[RA+1]).value+t[
TempTrip[RA+1]]

TW,twPassFlag,compensate=oneNodeTWCalculation(TW,o,c,TempTrip[RA+1])
                                if twPassFlag:
                                    MiniTime.append(TW)
                                    MiniComp.append(compensate)
                                else:
                                    break
                            if len(TempTrip)==len(MiniTime):

tempMinCompensate=CalculateCompensate(MiniComp)
                                compensateDiff=TotalCompensate-
tempMinCompensate
                                tempTotalDiff=Diff+compensateDiff
                                if tempTotalDiff >= totalDiff:
                                    MirrorTime=MiniTime
                                    MirrorTrip=TempTrip
                                    MirrorScore=TempScore
                                    MirrorCost=TempCost
                                    MirrorComp=MiniComp
                                    minCompensate=tempMinCompensate
                                    totalDiff=tempTotalDiff
                                    MaxDiff = Diff
                                    DayFromSwitch = day
                                    #DayToSwitch = dayLoop
                                    SaveScore=Score
                                    SaveCost=Cost
                                    currentSolution=1
                            del TempTrip, TempScore, TempCost
```

```
                        del MiniTime
                    del P, P2, Old, Old2
            else: #Pair with other day in tour move i to j
                Trip2=Node[0+multiply*dayLoop]# j would be used with dayLoop
                TimeWindow2=Node[1+multiply*dayLoop]
                Scorelist2=Node[2+multiply*dayLoop]
                Costlist2=Node[3+multiply*dayLoop]
                Ratiolist2=Node[4+multiply*dayLoop]
                TotalRatio2=Node[5+multiply*dayLoop]
                Score2=Node[6+multiply*dayLoop]
                Cost2=Node[7+multiply*dayLoop]
                Compensatelist2=Node[8+multiply*dayLoop]
                TotalCompensate2=Node[9+multiply*dayLoop]
                minCompensate2=TotalCompensate2
                for j in range(1,len(Trip2[1:])):
                    if Trip2[j] not in l+d:
                        P = (tt.cell(row=5+Trip2[j-1],column=5+Trip[i]).value +
tt.cell(row=5+Trip[i],column=5+Trip2[j+1]).value)# i must be moved from current
position, so i-1 will connected to i+1
                        P2 = (tt.cell(row=5+Trip[i-1],column=5+Trip2[j]).value +
tt.cell(row=5+Trip2[j],column=5+Trip[i+1]).value)# i will be placed in place of j, so
it would be placed between j-1 and j
                        Old = (tt.cell(row=5+Trip[i-1],column=5+Trip[i]).value +
tt.cell(row=5+Trip[i],column=5+Trip[i+1]).value)
                        Old2 = (tt.cell(row=5+Trip2[j-1],column=5+Trip2[j]).value +
tt.cell(row=5+Trip2[j],column=5+Trip2[j+1]).value)
                        Diff = (Old + Old2) - (P + P2)
                        if Diff > MaxDiff:
                            MiniTime=TimeWindow[:i]
                            MiniTime2=TimeWindow2[:j]
                            MiniComp=Compensatelist[:i]
                            MiniComp2=Compensatelist2[:j]
                            TW = MiniTime[i-1] + tt.cell(row=5+Trip[i-
1],column=5+Trip2[j]).value + t[Trip2[j]]
                            TW2 = MiniTime2[j-1] + tt.cell(row=5+Trip2[j-
1],column=5+Trip[i]).value + t[Trip[i]]

TW,twPassFlag,compensate=oneNodeTWCalculation(TW,o,c,Trip2[j])

TW2,twPassFlag2,compensate2=oneNodeTWCalculation(TW2,o,c,Trip[i])
                        if twPassFlag and twPassFlag2:
                            MiniTime.append(TW)
                            MiniTime2.append(TW2)
                            MiniComp.append(compensate)
                            MiniComp2.append(compensate2)
                            del TW
                            del TW2
```

```
                        TempTrip=Trip[:]
                        TempScore=Scorelist[:]
                        TempCost=Costlist[:]
                        TempTrip2=Trip2[:]
                        TempScore2=Scorelist2[:]
                        TempCost2=Costlist2[:]
                        TempTrip[i],TempTrip2[j] = TempTrip2[j],TempTrip[i]
                        TempScore[i],TempScore2[j] =
TempScore2[j],TempScore[i]
                        TempCost[i],TempCost2[j] = TempCost2[j],TempCost[i]
                        for RA in range(len(TempTrip[:i]),len(TempTrip)-1):
                            TW =
MiniTime[RA]+tt.cell(row=5+TempTrip[RA],column=5+TempTrip[RA+1]).value+t[
TempTrip[RA+1]]

TW,twPassFlag,compensate=oneNodeTWCalculation(TW,o,c,TempTrip[RA+1])
                            if twPassFlag:
                                MiniTime.append(TW)
                                MiniComp.append(compensate)
                            else:
                                break
                        for RA2 in range(len(TempTrip2[:j]),len(TempTrip2)-1):
                            TW =
MiniTime2[RA2]+tt.cell(row=5+TempTrip2[RA2],column=5+TempTrip2[RA2+1]).
value+t[TempTrip2[RA2+1]]

TW,twPassFlag,compensate2=oneNodeTWCalculation(TW,o,c,TempTrip2[RA2+1])
                            if twPassFlag:
                                MiniTime2.append(TW)
                                MiniComp2.append(compensate2)
                            else:
                                break
                        if len(TempTrip)==len(MiniTime) and
len(TempTrip2)==len(MiniTime2):
                            tempMinCompensate=CalculateCompensate(MiniComp)
                            compensateDiff=TotalCompensate-tempMinCompensate

tempMinCompensate2=CalculateCompensate(MiniComp2)
                            compensateDiff2=TotalCompensate2-
tempMinCompensate2
                            tempTotalDiff=Diff+compensateDiff+compensateDiff2
                            if tempTotalDiff >= totalDiff:
                                whatCase=2
                                MirrorTime=MiniTime
                                MirrorComp=MiniComp
                                minCompensate=tempMinCompensate
                                MirrorTrip=TempTrip
```

```
                                    MirrorScore=TempScore
                                    MirrorCost=TempCost
                                    SaveScore=Score-Scorelist[i]+Scorelist2[j]
                                    SaveCost=Score-Costlist[i]+Costlist2[j]
                                    MaxDiff = Diff
                                    DayFromSwitch = day
                                    DayToSwitch = dayLoop
                                    #Variable for second trip
                                    MirrorTime2=MiniTime2
                                    MirrorComp2=MiniComp2
                                    minCompensate2=tempMinCompensate2
                                    MirrorTrip2=TempTrip2
                                    MirrorScore2=TempScore2
                                    MirrorCost2=TempCost2
                                    SaveScore2=Score2+Scorelist[i]-Scorelist2[j]
                                    SaveCost2=Score2+Costlist[i]-Costlist2[j]
                                    currentSolution=2
                                    totalDiff=tempTotalDiff
                            del TempTrip, TempScore, TempCost, TempTrip2,
TempScore2, TempCost2
                                del MiniTime
                            del P, P2, Old, Old2
    if currentSolution==1:

MirrorTime,SaveScore,SaveCost,MirrorScore,MirrorCost,MiniComp=CalculateGene
ral(MirrorTrip,MirrorTime,pr,s,tt,t,DayFromSwitch,DayInTour,o,c)
        TotalRatio, RatioList = CalculateRatio(MirrorTrip,s,tt,MiniComp)
        TotalCompensate=CalculateCompensate(MiniComp)
        tempNode[0+multiply*DayFromSwitch]=MirrorTrip
        tempNode[1+multiply*DayFromSwitch]=MirrorTime
        tempNode[2+multiply*DayFromSwitch]=MirrorScore
        tempNode[3+multiply*DayFromSwitch]=MirrorCost
        tempNode[4+multiply*DayFromSwitch]=Ratiolist
        tempNode[5+multiply*DayFromSwitch]=TotalRatio
        tempNode[6+multiply*DayFromSwitch]=SaveScore
        tempNode[7+multiply*DayFromSwitch]=SaveCost
        tempNode[8+multiply*DayFromSwitch]=MiniComp
        tempNode[9+multiply*DayFromSwitch]=TotalCompensate
        improveFlag=True
    elif currentSolution==2:

MirrorTime,SaveScore,SaveCost,MirrorScore,MirrorCost,MiniComp=CalculateGene
ral(MirrorTrip,MirrorTime,pr,s,tt,t,DayFromSwitch,DayInTour,o,c)

MirrorTime2,SaveScore2,SaveCost2,MirrorScore2,MirrorCost2,MiniComp2=Calcula
teGeneral(MirrorTrip2,MirrorTime2,pr,s,tt,t,DayToSwitch,DayInTour,o,c)
        TotalRatio, RatioList = CalculateRatio(MirrorTrip,s,tt,MiniComp)
```

```
        TotalRatio2, RatioList2 = CalculateRatio(MirrorTrip2,s,tt,MiniComp2)
        TotalCompensate=CalculateCompensate(MiniComp)
        TotalCompensate2=CalculateCompensate(MiniComp2)
        tempNode[0+multiply*DayFromSwitch]=MirrorTrip
        tempNode[1+multiply*DayFromSwitch]=MirrorTime
        tempNode[2+multiply*DayFromSwitch]=MirrorScore
        tempNode[3+multiply*DayFromSwitch]=MirrorCost
        tempNode[4+multiply*DayFromSwitch]=Ratiolist
        tempNode[5+multiply*DayFromSwitch]=TotalRatio
        tempNode[6+multiply*DayFromSwitch]=SaveScore
        tempNode[7+multiply*DayFromSwitch]=SaveCost
        tempNode[8+multiply*DayFromSwitch]=MiniComp
        tempNode[9+multiply*DayFromSwitch]=TotalCompensate
        tempNode[0+multiply*DayToSwitch]=MirrorTrip2
        tempNode[1+multiply*DayToSwitch]=MirrorTime2
        tempNode[2+multiply*DayToSwitch]=MirrorScore2
        tempNode[3+multiply*DayToSwitch]=MirrorCost2
        tempNode[4+multiply*DayToSwitch]=Ratiolist2
        tempNode[5+multiply*DayToSwitch]=TotalRatio2
        tempNode[6+multiply*DayToSwitch]=SaveScore2
        tempNode[7+multiply*DayToSwitch]=SaveCost2
        tempNode[8+multiply*DayToSwitch]=MiniComp2
        tempNode[9+multiply*DayToSwitch]=TotalCompensate2
        improveFlag=True
    else:
        dummyInt=0
    '''Calculate Total Score before save the result'''
    Count=0
    TotalScore=0
    Run=0
    while Count<DayInTour:
        TotalScore+=tempNode[6+Run]
        Count+=1
        Run+=10
    return tempNode, TotalScore, improveFlag

'''5.1 insert for extract'''
def
InsertionExtract(Trip,TimeWindow,Scorelist,Costlist,Ratiolist,TotalRatio,OriginalSco
re,Score,Cost,Compensatelist,TotalCompensate,Budget,h,p,l,d,s,pr,t,o,c,tt,DayInTour,
Tabu,CombinedTour,currentDay):
    Round=0
    SaveScore=0
    ShouldApply=False
    while (Round<1):
        Ratio = 0
        for i in p:
```

```
        if (i not in CombinedTour) and (i not in Tabu) and (i not in Trip):
            for j in range(0,len(Trip)-1):
                for k in range(1,len(Trip)):
                    if j < k:
                        MiniTime=TimeWindow[:k]
                        MiniComp=Compensatelist[:k]
                        TW = MiniTime[k-1]+tt.cell(row=5+Trip[k-
1],column=5+i).value+t[i]
                        TW,twPassFlag,compensate=oneNodeTWCalculation(TW,o,c,i)
                        if twPassFlag and Cost+pr[i]<=Budget:
                            x = s[i]/(tt.cell(row=5+Trip[k-1],column=5+i).value +
tt.cell(row=5+i,column=5+Trip[k]).value + compensate)
                            if Ratio < x:
                                MiniTime.append(TW)
                                MiniComp.append(compensate)
                                del TW
                                TempTrip=Trip[:]
                                TempScore=Scorelist[:]
                                TempCost=Costlist[:]
                                TempTrip.insert(k,i)
                                TempScore.insert(k,s[i])
                                TempCost.insert(k,pr[i])
                                for RA in range(len(TempTrip[:k]),len(TempTrip)-1):
                                    TW =
MiniTime[RA]+tt.cell(row=5+TempTrip[RA],column=5+TempTrip[RA+1]).value+t[
TempTrip[RA+1]]

TW,twPassFlag,compensate=oneNodeTWCalculation(TW,o,c,TempTrip[RA+1])
                                    if twPassFlag:
                                        MiniTime.append(TW)
                                        MiniComp.append(compensate)
                                    else:
                                        break
                                if len(TempTrip)==len(MiniTime):
                                    MirrorTime=MiniTime
                                    MirrorComp=MiniComp
                                    MirrorTrip=TempTrip
                                    MirrorScore=TempScore
                                    MirrorCost=TempCost
                                    Ratio = x
                                    SaveScore=Score+s[i]
                                    SaveCost=Cost+pr[i]
                                    CombinedTour.append(i)
                                del TempTrip, TempScore, TempCost
                            del MiniTime
                            del x
                        break
```

```
        if 'MirrorTrip' in locals():
            Trip=MirrorTrip

TimeWindow,Score,Cost,Scorelist,Costlist,Compensatelist=CalculateGeneral(Trip,Ti
meWindow,pr,s,tt,t,currentDay,DayInTour,o,c)
            ShouldApply=True
            del MirrorTime, MirrorTrip, MirrorScore, MirrorCost, SaveScore, SaveCost
        else:
            Round+=1
        if Round>0:
            TotalRatio, RatioList = CalculateRatio(Trip,s,tt,Compensatelist)
            TotalCompensate=CalculateCompensate(Compensatelist)
    return
Trip,TimeWindow,Scorelist,Costlist,Ratiolist,TotalRatio,Score,Cost,Compensatelist,
TotalCompensate,ShouldApply

'''5.2 insert for extract only before Res'''
def
InsertionAttExtractLunch(Trip,TimeWindow,Scorelist,Costlist,Ratiolist,TotalRatio,Sc
ore,Cost,Compensatelist,TotalCompensate,Budget,h,p,l,d,s,pr,t,o,c,tt,DayInTour,Tabu
,CombinedTour,indexNode,currentDay):
    Round=0
    SaveScore=0
    ShouldApply=False
    while (Round<1):
        Ratio = 0
        for j in range(0,indexNode):
            for k in range(1,indexNode+1):
                if k>j:
                    for i in p:
                        if (i not in CombinedTour) and (i not in Tabu) and (i not in Trip):
                            MiniTime=TimeWindow[:k]
                            MiniComp=Compensatelist[:k]
                            TW = MiniTime[k-1]+tt.cell(row=5+Trip[k-
1],column=5+i).value+t[i]
                            TW,twPassFlag,compensate=oneNodeTWCalculation(TW,o,c,i)
                            if twPassFlag and Cost+pr[i]<=Budget:
                                x = s[i]/(tt.cell(row=5+Trip[k-1],column=5+i).value +
tt.cell(row=5+i,column=5+Trip[k]).value + compensate)
                                if Ratio < x:
                                    MiniTime.append(TW)
                                    MiniComp.append(compensate)
                                    del TW
                                    TempTrip=Trip[:]
                                    TempScore=Scorelist[:]
                                    TempCost=Costlist[:]
                                    TempTrip.insert(k,i)
```

```
                        TempScore.insert(k,s[i])
                        TempCost.insert(k,pr[i])
                        for RA in range(len(TempTrip[:k]),len(TempTrip)-1):
                            TW =
MiniTime[RA]+tt.cell(row=5+TempTrip[RA],column=5+TempTrip[RA+1]).value+t[
TempTrip[RA+1]]

TW,twPassFlag,compensate=oneNodeTWCalculation(TW,o,c,TempTrip[RA+1])
                            if twPassFlag:
                                MiniTime.append(TW)
                                MiniComp.append(compensate)
                            else:
                                break
                        if len(TempTrip)==len(MiniTime):
                            MirrorTime=MiniTime
                            MirrorTrip=TempTrip
                            MirrorComp=MiniComp
                            MirrorScore=TempScore
                            MirrorCost=TempCost
                            Ratio = x
                            SaveScore=Score+s[i]
                            SaveCost=Cost+pr[i]
                            CombinedTour.append(i)
                            del TempTrip, TempScore, TempCost
                        del MiniTime
                    del x
                    break
    if 'MirrorTrip' in locals():
        Trip=MirrorTrip

TimeWindow,Score,Cost,Scorelist,Costlist,Compensatelist=CalculateGeneral(Trip,Ti
meWindow,pr,s,tt,t,currentDay,DayInTour,o,c)
        indexNode+=1
        ShouldApply=True
        del MirrorTime, MirrorTrip, MirrorScore, MirrorCost, SaveScore, SaveCost
    else:
        Round+=1
    if Round>0:
        TotalRatio, RatioList = CalculateRatio(Trip,s,tt,Compensatelist)
        TotalCompensate=CalculateCompensate(Compensatelist)
  return
Trip,TimeWindow,Scorelist,Costlist,Ratiolist,TotalRatio,Score,Cost,Compensatelist,
TotalCompensate,ShouldApply,indexNode

'''5.3 insert for extract only before Res'''
def
InsertionAttExtractDinner(Trip,TimeWindow,Scorelist,Costlist,Ratiolist,TotalRatio,S
```

```
core,Cost,Compensatelist,TotalCompensate,Budget,h,p,l,d,s,pr,t,o,c,tt,DayInTour,Tab
u,CombinedTour,indexNode,LunchIndex,currentDay):
   Round=0
   SaveScore=0
   ShouldApply=False
   while (Round<1):
      Ratio = 0
      for j in range(LunchIndex,indexNode-1):
         for k in range(LunchIndex+1,indexNode):
            if k>j:
               for i in p:
                  if (i not in CombinedTour) and (i not in Tabu) and (i not in Trip):
                     MiniTime=TimeWindow[:k]
                     MiniComp=Compensatelist[:k]
                     TW = MiniTime[k-1]+tt.cell(row=5+Trip[k-
1],column=5+i).value+t[i]
                     TW,twPassFlag,compensate=oneNodeTWCalculation(TW,o,c,i)
                     if twPassFlag and Cost+pr[i]<=Budget:
                        x = s[i]/(tt.cell(row=5+Trip[k-1],column=5+i).value +
tt.cell(row=5+i,column=5+Trip[k]).value + compensate)
                        if Ratio < x:
                           MiniTime.append(TW)
                           MiniComp.append(compensate)
                           del TW
                           TempTrip=Trip[:]
                           TempScore=Scorelist[:]
                           TempCost=Costlist[:]
                           TempTrip.insert(k,i)
                           TempScore.insert(k,s[i])
                           TempCost.insert(k,pr[i])
                           for RA in range(len(TempTrip[:k]),len(TempTrip)-1):
                              TW =
MiniTime[RA]+tt.cell(row=5+TempTrip[RA],column=5+TempTrip[RA+1]).value+t[
TempTrip[RA+1]]

TW,twPassFlag,compensate=oneNodeTWCalculation(TW,o,c,TempTrip[RA+1])
                              if twPassFlag:
                                 MiniTime.append(TW)
                                 MiniComp.append(compensate)
                              else:
                                 break
                           if len(TempTrip)==len(MiniTime):
                              MirrorTime=MiniTime
                              MirrorComp=MiniComp
                              MirrorTrip=TempTrip
                              MirrorScore=TempScore
                              MirrorCost=TempCost
```

```
                                Ratio = x
                                SaveScore=Score+s[i]
                                SaveCost=Cost+pr[i]
                                CombinedTour.append(i)
                                del TempTrip, TempScore, TempCost
                            del MiniTime
                        del x
                        break
        if 'MirrorTrip' in locals():
            Trip=MirrorTrip

TimeWindow,Score,Cost,Scorelist,Costlist,Compensatelist=CalculateGeneral(Trip,Ti
meWindow,pr,s,tt,t,currentDay,DayInTour,o,c)
            ShouldApply=True
            del MirrorTime, MirrorTrip, MirrorScore, MirrorCost, SaveScore, SaveCost
        else:
            Round+=1
        if Round>0:
            TotalRatio, RatioList = CalculateRatio(Trip,s,tt,Compensatelist)
            TotalCompensate=CalculateCompensate(Compensatelist)
    return
Trip,TimeWindow,Scorelist,Costlist,Ratiolist,TotalRatio,Score,Cost,Compensatelist,
TotalCompensate,ShouldApply

'''5.ก insert for extract'''
def
InsertionExtractLunch(Trip,TimeWindow,Scorelist,Costlist,Ratiolist,TotalRatio,Origi
nalScore,Score,Cost,Compensatelist,TotalCompensate,Budget,h,p,l,d,s,pr,t,o,c,tt,DayI
nTour,Tabu,CombinedTour,currentDay):
    Round=0
    SaveScore=0
    ShouldApply=False
    while (Round<1):
        Ratio = 0
        for i in l:
            if (i not in CombinedTour) and (i not in Tabu) and (i not in Trip):
                for j in range(0,len(Trip)-1):
                    for k in range(1,len(Trip)):
                        if j < k:
                            MiniTime=TimeWindow[:k]
                            MiniComp=Compensatelist[:k]
                            TW = MiniTime[k-1]+tt.cell(row=5+Trip[k-
1],column=5+i).value+t[i]
                            TW,twPassFlag,compensate=oneNodeTWCalculation(TW,o,c,i)
                            if twPassFlag and Cost+pr[i]<=Budget:
                                x = s[i]/(tt.cell(row=5+Trip[k-1],column=5+i).value +
tt.cell(row=5+i,column=5+Trip[k]).value + compensate)
```

```
                    if Ratio < x:
                        MiniTime.append(TW)
                        MiniComp.append(compensate)
                        del TW
                        TempTrip=Trip[:]
                        TempScore=Scorelist[:]
                        TempCost=Costlist[:]
                        TempTrip.insert(k,i)
                        TempScore.insert(k,s[i])
                        TempCost.insert(k,pr[i])
                        for RA in range(len(TempTrip[:k]),len(TempTrip)-1):
                            TW =
MiniTime[RA]+tt.cell(row=5+TempTrip[RA],column=5+TempTrip[RA+1]).value+t[
TempTrip[RA+1]]

TW,twPassFlag,compensate=oneNodeTWCalculation(TW,o,c,TempTrip[RA+1])
                            if twPassFlag:
                                MiniTime.append(TW)
                                MiniComp.append(compensate)
                            else:
                                break
                        if len(TempTrip)==len(MiniTime):
                            MirrorTime=MiniTime
                            MirrorComp=MiniComp
                            MirrorTrip=TempTrip
                            MirrorScore=TempScore
                            MirrorCost=TempCost
                            Ratio = x
                            SaveScore=Score+s[i]
                            SaveCost=Cost+pr[i]
                            CombinedTour.append(i)
                        del TempTrip, TempScore, TempCost
                    del MiniTime
                    del x
                break
    if 'MirrorTrip' in locals():
        Trip=MirrorTrip

TimeWindow,Score,Cost,Scorelist,Costlist,Compensatelist=CalculateGeneral(Trip,Ti
meWindow,pr,s,tt,t,currentDay,DayInTour,o,c)
        ShouldApply=True
        del MirrorTime, MirrorTrip, MirrorScore, MirrorCost, SaveScore, SaveCost
        Round+=1
    else:
        Round+=1
    if Round>0:
        TotalRatio, RatioList = CalculateRatio(Trip,s,tt,Compensatelist)
```

```
        TotalCompensate=CalculateCompensate(Compensatelist)
    return
Trip,TimeWindow,Scorelist,Costlist,Ratiolist,TotalRatio,Score,Cost,Compensatelist,
TotalCompensate,ShouldApply

'''5.5 insert for extract'''
def
InsertionExtractDinner(Trip,TimeWindow,Scorelist,Costlist,Ratiolist,TotalRatio,Orig
inalScore,Score,Cost,Compensatelist,TotalCompensate,Budget,h,p,l,d,s,pr,t,o,c,tt,Day
InTour,Tabu,CombinedTour,currentDay):
    Round=0
    SaveScore=0
    ShouldApply=False
    while (Round<1):
        Ratio = 0
        for i in d:
            if (i not in CombinedTour) and (i not in Tabu) and (i not in Trip):
                for j in range(0,len(Trip)-1):
                    for k in range(1,len(Trip)):
                        if j < k:
                            MiniTime=TimeWindow[:k]
                            MiniComp=Compensatelist[:k]
                            TW = MiniTime[k-1]+tt.cell(row=5+Trip[k-
1],column=5+i).value+t[i]
                            TW,twPassFlag,compensate=oneNodeTWCalculation(TW,o,c,i)
                            if twPassFlag and Cost+pr[i]<=Budget:
                                x = s[i]/(tt.cell(row=5+Trip[k-1],column=5+i).value +
tt.cell(row=5+i,column=5+Trip[k]).value + compensate)
                                if Ratio < x:
                                    MiniTime.append(TW)
                                    MiniComp.append(compensate)
                                    del TW
                                    TempTrip=Trip[:]
                                    TempScore=Scorelist[:]
                                    TempCost=Costlist[:]
                                    TempTrip.insert(k,i)
                                    TempScore.insert(k,s[i])
                                    TempCost.insert(k,pr[i])
                                    for RA in range(len(TempTrip[:k]),len(TempTrip)-1):
                                        TW =
MiniTime[RA]+tt.cell(row=5+TempTrip[RA],column=5+TempTrip[RA+1]).value+t[
TempTrip[RA+1]]

TW,twPassFlag,compensate=oneNodeTWCalculation(TW,o,c,TempTrip[RA+1])
                                        if twPassFlag:
                                            MiniTime.append(TW)
                                            MiniComp.append(compensate)
```

```
                        else:
                            break
                    if len(TempTrip)==len(MiniTime):
                        MirrorTime=MiniTime
                        MirrorComp=MiniComp
                        MirrorTrip=TempTrip
                        MirrorScore=TempScore
                        MirrorCost=TempCost
                        Ratio = x
                        SaveScore=Score+s[i]
                        SaveCost=Cost+pr[i]
                        CombinedTour.append(i)
                    del TempTrip, TempScore, TempCost
                del MiniTime
                del x
            break
    if 'MirrorTrip' in locals():
        Trip=MirrorTrip

TimeWindow,Score,Cost,Scorelist,Costlist,Compensatelist=CalculateGeneral(Trip,Ti
meWindow,pr,s,tt,t,currentDay,DayInTour,o,c)
        ShouldApply=True
        del MirrorTime, MirrorTrip, MirrorScore, MirrorCost, SaveScore, SaveCost
        Round+=1
    else:
        Round+=1
    if Round>0:
        TotalRatio, RatioList = CalculateRatio(Trip,s,tt,Compensatelist)
        TotalCompensate=CalculateCompensate(Compensatelist)
  return
Trip,TimeWindow,Scorelist,Costlist,Ratiolist,TotalRatio,Score,Cost,Compensatelist,
TotalCompensate,ShouldApply

'''5. Extract-Insert'''
def extractinsert(Node,Budget,h,p,l,d,s,pr,t,o,c,tt,DayInTour,Tabu):
  ShouldApply=False
  tempNode=Node[:]

CombinedTour,ScoreRatio,RangeForEachDay=CombineAllTrip(tempNode,DayInTo
ur)
  for day in range(DayInTour):
    multiplier=10*day
    Trip=Node[0+multiplier]# i would be used with day
    TimeWindow=Node[1+multiplier]
    Scorelist=Node[2+multiplier]
    Costlist=Node[3+multiplier]
    Ratiolist=Node[4+multiplier]
```

```
        TotalRatio=Node[5+multiplier]
        Score=Node[6+multiplier]
        Cost=Node[7+multiplier]
        Compensatelist=Node[8+multiplier]
        TotalCompensate=Node[9+multiplier]
        tempTrip=Trip[:]
        tempTimeWindow=TimeWindow[:]
        tempScorelist=Scorelist[:]
        tempCostlist=Costlist[:]
        tempRatiolist=Ratiolist[:]
        tempTotalRatio=TotalRatio
        tempScore=Score
        tempCost=Cost
        tempCompensatelist=Compensatelist[:]
        tempTotalCompensate=TotalCompensate
        tempTrip2=Trip[:]
        tempTimeWindow2=TimeWindow[:]
        tempScorelist2=Scorelist[:]
        tempCostlist2=Costlist[:]
        tempRatiolist2=Ratiolist[:]
        tempTotalRatio2=TotalRatio
        tempCompensatelist2=Compensatelist[:]
        tempTotalCompensate2=TotalCompensate
        tempScore2=Score
        tempCost2=Cost
        for i in Trip:
            if i not in l+d+h:
                Tabu.append(i)
                tempTrip.pop(tempTrip.index(i))

tempTimeWindow,tempScore,tempCost,tempScorelist,tempCostlist,tempCompensate
list,tempTotalCompensate,ShouldApply,Response=CalculateGeneralWValidationAll(
tempTrip,tempTimeWindow,pr,s,tt,t,o,c,day,DayInTour)
                if ShouldApply:

tempTotalRatio,tempRatiolist=CalculateRatio(tempTrip,s,tt,tempCompensatelist)

tempTrip,tempTimeWindow,tempScorelist,tempCostlist,tempRatiolist,tempTotalRati
o,tempScore,tempCost,tempCompensatelist,tempTotalCompensate,ShouldApply2=In
sertionExtract(tempTrip,tempTimeWindow,tempScorelist,tempCostlist,tempRatiolist,
tempTotalRatio,Score,tempScore,tempCost,tempCompensatelist,tempTotalCompensa
te,Budget,h,p,l,d,s,pr,t,o,c,tt,3,Tabu,CombinedTour,day)

tempTimeWindow,tempScore,tempCost,tempScorelist,tempCostlist,tempCompensate
list,tempTotalCompensate,ShouldApply,Response=CalculateGeneralWValidationAll(
tempTrip,tempTimeWindow,pr,s,tt,t,o,c,day,DayInTour)
                if ShouldApply:
```

```
            if ShouldApply2:
                if tempTotalRatio2>=tempTotalRatio:
                    tempTrip=tempTrip2[:]
                    tempTimeWindow=tempTimeWindow2[:]
                    tempScorelist=tempScorelist2[:]
                    tempCostlist=tempCostlist2[:]
                    tempRatiolist=tempRatiolist2[:]
                    tempTotalRatio=tempTotalRatio2
                    tempScore=tempScore2
                    tempCost=tempCost2
                    tempCompensatelist=tempCompensatelist2[:]
                    tempTotalCompensate=tempTotalCompensate2
                if tempTotalRatio2<tempTotalRatio:
                    tempTrip2=tempTrip[:]
                    tempTimeWindow2=tempTimeWindow[:]
                    tempScorelist2=tempScorelist[:]
                    tempCostlist2=tempCostlist[:]
                    tempRatiolist2=tempRatiolist[:]
                    tempTotalRatio2=tempTotalRatio
                    tempScore2=tempScore
                    tempCost2=tempCost
                    tempCompensatelist2=tempCompensatelist[:]
                    tempTotalCompensate2=tempTotalCompensate
            else:
                tempTrip=tempTrip2[:]
                tempTimeWindow=tempTimeWindow2[:]
                tempScorelist=tempScorelist2[:]
                tempCostlist=tempCostlist2[:]
                tempRatiolist=tempRatiolist2[:]
                tempTotalRatio=tempTotalRatio2
                tempScore=tempScore2
                tempCost=tempCost2
                tempCompensatelist=tempCompensatelist2[:]
                tempTotalCompensate=tempTotalCompensate2
        else:
            tempTrip=tempTrip2[:]
            tempTimeWindow=tempTimeWindow2[:]
            tempScorelist=tempScorelist2[:]
            tempCostlist=tempCostlist2[:]
            tempRatiolist=tempRatiolist2[:]
            tempTotalRatio=tempTotalRatio2
            tempScore=tempScore2
            tempCost=tempCost2
            tempCompensatelist=tempCompensatelist2[:]
            tempTotalCompensate=tempTotalCompensate2
    elif i in l:
        Tabu.append(i)
```

```
            tempTrip.pop(tempTrip.index(i))

tempTimeWindow,tempScore,tempCost,tempScorelist,tempCostlist,tempCompensate
list,tempTotalCompensate,ShouldApply,Response=CalculateGeneralWValidationAll(
tempTrip,tempTimeWindow,pr,s,tt,t,o,c,day,DayInTour)
            if ShouldApply:

tempTotalRatio,tempRatiolist=CalculateRatio(tempTrip,s,tt,tempCompensatelist)

tempTrip,tempTimeWindow,tempScorelist,tempCostlist,tempRatiolist,tempTotalRati
o,tempScore,tempCost,tempCompensatelist,tempTotalCompensate,ShouldApply2=In
sertionExtractLunch(tempTrip,tempTimeWindow,tempScorelist,tempCostlist,tempRa
tiolist,tempTotalRatio,Score,tempScore,tempCost,tempCompensatelist,tempTotalCo
mpensate,Budget,h,p,l,d,s,pr,t,o,c,tt,3,Tabu,CombinedTour,day)

tempTimeWindow,tempScore,tempCost,tempScorelist,tempCostlist,tempCompensate
list,tempTotalCompensate,ShouldApply,Response=CalculateGeneralWValidationAll(
tempTrip,tempTimeWindow,pr,s,tt,t,o,c,day,DayInTour)
            if ShouldApply:
                if ShouldApply2:
                    if tempTotalRatio2>=tempTotalRatio:
                        tempTrip=tempTrip2[:]
                        tempTimeWindow=tempTimeWindow2[:]
                        tempScorelist=tempScorelist2[:]
                        tempCostlist=tempCostlist2[:]
                        tempRatiolist=tempRatiolist2[:]
                        tempTotalRatio=tempTotalRatio2
                        tempScore=tempScore2
                        tempCost=tempCost2
                        tempCompensatelist=tempCompensatelist2[:]
                        tempTotalCompensate=tempTotalCompensate2
                    if tempTotalRatio2<tempTotalRatio:
                        tempTrip2=tempTrip[:]
                        tempTimeWindow2=tempTimeWindow[:]
                        tempScorelist2=tempScorelist[:]
                        tempCostlist2=tempCostlist[:]
                        tempRatiolist2=tempRatiolist[:]
                        tempTotalRatio2=tempTotalRatio
                        tempScore2=tempScore
                        tempCost2=tempCost
                        tempCompensatelist2=tempCompensatelist[:]
                        tempTotalCompensate2=tempTotalCompensate
                    else:
                        tempTrip=tempTrip2[:]
                        tempTimeWindow=tempTimeWindow2[:]
                        tempScorelist=tempScorelist2[:]
                        tempCostlist=tempCostlist2[:]
```

```
                tempRatiolist=tempRatiolist2[:]
                tempTotalRatio=tempTotalRatio2
                tempScore=tempScore2
                tempCost=tempCost2
                tempCompensatelist=tempCompensatelist2[:]
                tempTotalCompensate=tempTotalCompensate2
            else:
                tempTrip=tempTrip2[:]
                tempTimeWindow=tempTimeWindow2[:]
                tempScorelist=tempScorelist2[:]
                tempCostlist=tempCostlist2[:]
                tempRatiolist=tempRatiolist2[:]
                tempTotalRatio=tempTotalRatio2
                tempScore=tempScore2
                tempCost=tempCost2
                tempCompensatelist=tempCompensatelist2[:]
                tempTotalCompensate=tempTotalCompensate2
        elif i in d:
            Tabu.append(i)
            tempTrip.pop(tempTrip.index(i))

tempTimeWindow,tempScore,tempCost,tempScorelist,tempCostlist,tempCompensate
list,tempTotalCompensate,ShouldApply,Response=CalculateGeneralWValidationAll(
tempTrip,tempTimeWindow,pr,s,tt,t,o,c,day,DayInTour)
            if ShouldApply:

tempTotalRatio,tempRatiolist=CalculateRatio(tempTrip,s,tt,tempCompensatelist)

tempTrip,tempTimeWindow,tempScorelist,tempCostlist,tempRatiolist,tempTotalRati
o,tempScore,tempCost,tempCompensatelist,tempTotalCompensate,ShouldApply2=In
sertionExtractDinner(tempTrip,tempTimeWindow,tempScorelist,tempCostlist,tempR
atiolist,tempTotalRatio,Score,tempScore,tempCost,tempCompensatelist,tempTotalCo
mpensate,Budget,h,p,l,d,s,pr,t,o,c,tt,3,Tabu,CombinedTour,day)

tempTimeWindow,tempScore,tempCost,tempScorelist,tempCostlist,tempCompensate
list,tempTotalCompensate,ShouldApply,Response=CalculateGeneralWValidationAll(
tempTrip,tempTimeWindow,pr,s,tt,t,o,c,day,DayInTour)
            if ShouldApply:
                if ShouldApply2:
                    if tempTotalRatio2>=tempTotalRatio:
                        tempTrip=tempTrip2[:]
                        tempTimeWindow=tempTimeWindow2[:]
                        tempScorelist=tempScorelist2[:]
                        tempCostlist=tempCostlist2[:]
                        tempRatiolist=tempRatiolist2[:]
                        tempTotalRatio=tempTotalRatio2
                        tempScore=tempScore2
```

```
                    tempCost=tempCost2
                    tempCompensatelist=tempCompensatelist2[:]
                    tempTotalCompensate=tempTotalCompensate2
                if tempTotalRatio2<tempTotalRatio:
                    tempTrip2=tempTrip[:]
                    tempTimeWindow2=tempTimeWindow[:]
                    tempScorelist2=tempScorelist[:]
                    tempCostlist2=tempCostlist[:]
                    tempRatiolist2=tempRatiolist[:]
                    tempTotalRatio2=tempTotalRatio
                    tempScore2=tempScore
                    tempCost2=tempCost
                    tempCompensatelist2=tempCompensatelist[:]
                    tempTotalCompensate2=tempTotalCompensate
                else:
                    tempTrip=tempTrip2[:]
                    tempTimeWindow=tempTimeWindow2[:]
                    tempScorelist=tempScorelist2[:]
                    tempCostlist=tempCostlist2[:]
                    tempRatiolist=tempRatiolist2[:]
                    tempTotalRatio=tempTotalRatio2
                    tempScore=tempScore2
                    tempCost=tempCost2
                    tempCompensatelist=tempCompensatelist2[:]
                    tempTotalCompensate=tempTotalCompensate2
            else:
                tempTrip=tempTrip2[:]
                tempTimeWindow=tempTimeWindow2[:]
                tempScorelist=tempScorelist2[:]
                tempCostlist=tempCostlist2[:]
                tempRatiolist=tempRatiolist2[:]
                tempTotalRatio=tempTotalRatio2
                tempScore=tempScore2
                tempCost=tempCost2
                tempCompensatelist=tempCompensatelist2[:]
                tempTotalCompensate=tempTotalCompensate2

tempTimeWindow2,tempScore2,tempCost2,tempScorelist2,tempCostlist2,tempComp
ensatelist2=CalculateGeneral(tempTrip2,tempTimeWindow2,pr,s,tt,t,day,DayInTour,
o,c)

tempTotalRatio2,tempRatiolist2=CalculateRatio(tempTrip2,s,tt,tempCompensatelist2
)
        tempTotalCompensate2=CalculateCompensate(tempCompensatelist2)
        tempNode[0+multiplier]=tempTrip2
        tempNode[1+multiplier]=tempTimeWindow2
        tempNode[2+multiplier]=tempScorelist2
```

```
        tempNode[3+multiplier]=tempCostlist2
        tempNode[4+multiplier]=tempRatiolist2
        tempNode[5+multiplier]=tempTotalRatio2
        tempNode[6+multiplier]=tempScore2
        tempNode[7+multiplier]=tempCost2
        tempNode[8+multiplier]=tempCompensatelist2
        tempNode[9+multiplier]=tempTotalCompensate2
        '''Calculate Total Score before save the result'''
        Count=0
        TotalScore=0
        Run=0
        while Count<DayInTour:
            TotalScore+=tempNode[6+Run]
            Count+=1
            Run+=10
    return tempNode, TotalScore,Tabu

'''Special Extract-Insert Att'''
def
extractinsertAttLunch(Node,Trip,TimeWindow,Scorelist,Costlist,Ratiolist,TotalRatio,
Score,Cost,Compensatelist,TotalCompensate,Budget,h,p,l,d,s,pr,t,o,c,tt,DayInTour,N
odeIndex,currentDay):
    Tabu=[]
    ShouldApply=False
    ShouldApply2=False
    CombinedTour,ScoreRatio,RangeForEachDay=CombineAllTrip(Node,DayInTour)
    tempTrip=Trip[:]
    tempTimeWindow=TimeWindow[:]
    tempScorelist=Scorelist[:]
    tempCostlist=Costlist[:]
    tempRatiolist=Ratiolist[:]
    tempTotalRatio=TotalRatio
    tempScore=Score
    tempCost=Cost
    tempCompensatelist=Compensatelist[:]
    tempTotalCompensate=TotalCompensate
    tempTrip2=Trip[:]
    tempTimeWindow2=TimeWindow[:]
    tempScorelist2=Scorelist[:]
    tempCostlist2=Costlist[:]
    tempRatiolist2=Ratiolist[:]
    tempTotalRatio2=TotalRatio
    tempScore2=Score
    tempCost2=Cost
    tempCompensatelist2=Compensatelist[:]
    tempTotalCompensate2=TotalCompensate
    indexNode=NodeIndex
```

```
    combinedResNode=l+d+h
    for i in range(NodeIndex+1,len(tempTrip)):
        if not (tempTrip[i] in combinedResNode):
            Tabu.append(i)
            tempTrip.pop(i)

tempTimeWindow,tempScore,tempCost,tempScorelist,tempCostlist,tempCompensate
list,tempTotalCompensate,ShouldApply,Response=CalculateGeneralWValidationAll(
Trip,TimeWindow,pr,s,tt,t,o,c,currentDay,DayInTour)
            if ShouldApply:

tempTotalRatio,tempRatiolist=CalculateRatio(tempTrip,s,tt,tempCompensatelist)

tempTrip,tempTimeWindow,tempScorelist,tempCostlist,tempRatiolist,tempTotalRati
o,tempScore,tempCost,tempCompensatelist,tempTotalCompensate,ShouldApply2,ind
exNode=InsertionAttExtractLunch(tempTrip,tempTimeWindow,tempScorelist,tempC
ostlist,tempRatiolist,tempTotalRatio,Score,Cost,tempCompensatelist,tempTotalComp
ensate,Budget,h,p,l,d,s,pr,t,o,c,tt,DayInTour,Tabu,CombinedTour,NodeIndex,current
Day)

tempTimeWindow,tempScore,tempCost,tempScorelist,tempCostlist,tempCompensate
list,tempTotalCompensate,ShouldApply,Response=CalculateGeneralWValidationAll(
Trip,TimeWindow,pr,s,tt,t,o,c,currentDay,DayInTour)
            if ShouldApply:
                if ShouldApply2:
                    if tempTotalRatio2>=tempTotalRatio:
                        tempTrip=tempTrip2[:]
                        tempTimeWindow=tempTimeWindow2[:]
                        tempScorelist=tempScorelist2[:]
                        tempCostlist=tempCostlist2[:]
                        tempRatiolist=tempRatiolist2[:]
                        tempTotalRatio=tempTotalRatio2
                        tempScore=tempScore2
                        tempCost=tempCost2
                        tempCompensatelist=tempCompensatelist2[:]
                        tempTotalCompensate=tempTotalCompensate2
                    if tempTotalRatio2<tempTotalRatio:
                        tempTrip2=tempTrip[:]
                        tempTimeWindow2=tempTimeWindow[:]
                        tempScorelist2=tempScorelist[:]
                        tempCostlist2=tempCostlist[:]
                        tempRatiolist2=tempRatiolist[:]
                        tempTotalRatio2=tempTotalRatio
                        tempScore2=tempScore
                        tempCost2=tempCost
                        tempCompensatelist2=tempCompensatelist[:]
                        tempTotalCompensate2=tempTotalCompensate
```

```
            else:
                tempTrip=tempTrip2[:]
                tempTimeWindow=tempTimeWindow2[:]
                tempScorelist=tempScorelist2[:]
                tempCostlist=tempCostlist2[:]
                tempRatiolist=tempRatiolist2[:]
                tempTotalRatio=tempTotalRatio2
                tempScore=tempScore2
                tempCost=tempCost2
                tempCompensatelist=tempCompensatelist2[:]
                tempTotalCompensate=tempTotalCompensate2
        else:
            tempTrip=tempTrip2[:]
            tempTimeWindow=tempTimeWindow2[:]
            tempScorelist=tempScorelist2[:]
            tempCostlist=tempCostlist2[:]
            tempRatiolist=tempRatiolist2[:]
            tempTotalRatio=tempTotalRatio2
            tempScore=tempScore2
            tempCost=tempCost2
            tempCompensatelist=tempCompensatelist2[:]
            tempTotalCompensate=tempTotalCompensate2

tempTimeWindow2,tempScore2,tempCost2,tempScorelist2,tempCostlist2,tempComp
ensatelist2=CalculateGeneral(tempTrip2,tempTimeWindow2,pr,s,tt,t,currentDay,Day
InTour,o,c)

tempTotalRatio2,tempRatiolist2=CalculateRatio(tempTrip2,s,tt,tempCompensatelist2
)
    tempTotalCompensate2=CalculateCompensate(tempCompensatelist2)
    return tempTrip2, tempTimeWindow2, tempScorelist2, tempCostlist2,
tempRatiolist2, tempTotalRatio2, tempScore2, tempCost2, tempCompensatelist2,
tempTotalCompensate2, ShouldApply2, indexNode

'''Special Extract-Insert Att'''
def
extractinsertAttDinner(Node,Trip,TimeWindow,Scorelist,Costlist,Ratiolist,TotalRatio
,Score,Cost,Compensatelist,TotalCompensate,Budget,h,p,l,d,s,pr,t,o,c,tt,DayInTour,L
unchIndex,NodeIndex,currentDay):
    Tabu=[]
    ShouldApply=False
    ShouldApply2=False
    CombinedTour,ScoreRatio,RangeForEachDay=CombineAllTrip(Node,DayInTour)
    tempTrip=Trip[:]
    tempTimeWindow=TimeWindow[:]
    tempScorelist=Scorelist[:]
    tempCostlist=Costlist[:]
```

```
    tempRatiolist=Ratiolist[:]
    tempTotalRatio=TotalRatio
    tempScore=Score
    tempCost=Cost
    tempCompensatelist=Compensatelist[:]
    tempTotalCompensate=TotalCompensate
    tempTrip2=Trip[:]
    tempTimeWindow2=TimeWindow[:]
    tempScorelist2=Scorelist[:]
    tempCostlist2=Costlist[:]
    tempRatiolist2=Ratiolist[:]
    tempTotalRatio2=TotalRatio
    tempScore2=Score
    tempCost2=Cost
    tempCompensatelist2=Compensatelist[:]
    tempTotalCompensate2=TotalCompensate
    combinedResNode=l+d+h
    for i in range(1,LunchIndex+1):
        if not (tempTrip[i] in combinedResNode):
            Tabu.append(i)
            tempTrip.pop(i)

tempTimeWindow,tempScore,tempCost,tempScorelist,tempCostlist,tempCompensate
list,tempTotalCompensate,ShouldApply=CalculateGeneralWValidation(tempTrip,Ti
meWindow,pr,s,tt,t,o,c,LunchIndex,currentDay,DayInTour)
            if ShouldApply:

tempTotalRatio,tempRatiolist=CalculateRatio(tempTrip,s,tt,tempCompensatelist)

tempTrip,tempTimeWindow,tempScorelist,tempCostlist,tempRatiolist,tempTotalRati
o,tempScore,tempCost,tempCompensatelist,tempTotalCompensate,ShouldApply=Ins
ertionAttExtractDinner(tempTrip,tempTimeWindow,tempScorelist,tempCostlist,temp
Ratiolist,tempTotalRatio,Score,Cost,tempCompensatelist,tempTotalCompensate,Bud
get,h,p,l,d,s,pr,t,o,c,tt,DayInTour,Tabu,CombinedTour,NodeIndex,LunchIndex,curren
tDay)

tempTimeWindow,tempScore,tempCost,tempScorelist,tempCostlist,tempCompensate
list,tempTotalCompensate,ShouldApply2,Response=CalculateGeneralWValidationAl
l(tempTrip,TimeWindow,pr,s,tt,t,o,c,currentDay,DayInTour)
            if ShouldApply:
                if ShouldApply2:
                    if tempTotalRatio2>=tempTotalRatio:
                        tempTrip=tempTrip2[:]
                        tempTimeWindow=tempTimeWindow2[:]
                        tempScorelist=tempScorelist2[:]
                        tempCostlist=tempCostlist2[:]
                        tempRatiolist=tempRatiolist2[:]
```

```
                    tempTotalRatio=tempTotalRatio2
                    tempScore=tempScore2
                    tempCost=tempCost2
                    tempCompensatelist=tempCompensatelist2[:]
                    tempTotalCompensate=tempTotalCompensate2
                if tempTotalRatio2<tempTotalRatio:
                    tempTrip2=tempTrip[:]
                    tempTimeWindow2=tempTimeWindow[:]
                    tempScorelist2=tempScorelist[:]
                    tempCostlist2=tempCostlist[:]
                    tempRatiolist2=tempRatiolist[:]
                    tempTotalRatio2=tempTotalRatio
                    tempScore2=tempScore
                    tempCost2=tempCost
                    tempCompensatelist2=tempCompensatelist[:]
                    tempTotalCompensate2=tempTotalCompensate
                else:
                    tempTrip=tempTrip2[:]
                    tempTimeWindow=tempTimeWindow2[:]
                    tempScorelist=tempScorelist2[:]
                    tempCostlist=tempCostlist2[:]
                    tempRatiolist=tempRatiolist2[:]
                    tempTotalRatio=tempTotalRatio2
                    tempScore=tempScore2
                    tempCost=tempCost2
                    tempCompensatelist=tempCompensatelist2[:]
                    tempTotalCompensate=tempTotalCompensate2
            else:
                tempTrip=tempTrip2[:]
                tempTimeWindow=tempTimeWindow2[:]
                tempScorelist=tempScorelist2[:]
                tempCostlist=tempCostlist2[:]
                tempRatiolist=tempRatiolist2[:]
                tempTotalRatio=tempTotalRatio2
                tempScore=tempScore2
                tempCost=tempCost2
                tempCompensatelist=tempCompensatelist2[:]
                tempTotalCompensate=tempTotalCompensate2
    if not ShouldApply2:
        for i in range(NodeIndex+1,len(tempTrip)):
            if not (tempTrip[i] in combinedResNode):
                Tabu.append(i)
                tempTrip.pop(i)

tempTimeWindow,tempScore,tempCost,tempScorelist,tempCostlist,tempCompensate
list,tempTotalCompensate,ShouldApply=CalculateGeneralWValidation(tempTrip,Ti
meWindow,pr,s,tt,t,o,c,LunchIndex,currentDay,DayInTour)
```

```
            if ShouldApply:

tempTotalRatio,tempRatiolist=CalculateRatio(tempTrip,s,tt,tempCompensatelist)

tempTrip,tempTimeWindow,tempScorelist,tempCostlist,tempRatiolist,tempTotalRati
o,tempScore,tempCost,tempCompensatelist,tempTotalCompensate,ShouldApply=Ins
ertionAttExtractDinner(tempTrip,tempTimeWindow,tempScorelist,tempCostlist,temp
Ratiolist,tempTotalRatio,Score,Cost,tempCompensatelist,tempTotalCompensate,Bud
get,h,p,l,d,s,pr,t,o,c,tt,DayInTour,Tabu,CombinedTour,NodeIndex,LunchIndex,curren
tDay)

tempTimeWindow,tempScore,tempCost,tempScorelist,tempCostlist,tempCompensate
list,tempTotalCompensate,ShouldApply2,Response=CalculateGeneralWValidationAl
l(tempTrip,TimeWindow,pr,s,tt,t,o,c,currentDay,DayInTour)
            if ShouldApply:
               if ShouldApply2:
                  tempTrip2=tempTrip[:]
                  tempTimeWindow2=tempTimeWindow[:]
                  tempScorelist2=tempScorelist[:]
                  tempCostlist2=tempCostlist[:]
                  tempRatiolist2=tempRatiolist[:]
                  tempTotalRatio2=tempTotalRatio
                  tempScore2=tempScore
                  tempCost2=tempCost
                  tempCompensatelist2=tempCompensatelist[:]
                  tempTotalCompensate2=tempTotalCompensate
                  break
               else:
                  tempTrip=tempTrip2[:]
                  tempTimeWindow=tempTimeWindow2[:]
                  tempScorelist=tempScorelist2[:]
                  tempCostlist=tempCostlist2[:]
                  tempRatiolist=tempRatiolist2[:]
                  tempTotalRatio=tempTotalRatio2
                  tempScore=tempScore2
                  tempCost=tempCost2
                  tempCompensatelist=tempCompensatelist2[:]
                  tempTotalCompensate=tempTotalCompensate2
            else:
               tempTrip=tempTrip2[:]
               tempTimeWindow=tempTimeWindow2[:]
               tempScorelist=tempScorelist2[:]
               tempCostlist=tempCostlist2[:]
               tempRatiolist=tempRatiolist2[:]
               tempTotalRatio=tempTotalRatio2
               tempScore=tempScore2
               tempCost=tempCost2
```

```
            tempCompensatelist=tempCompensatelist2[:]
            tempTotalCompensate=tempTotalCompensate2

tempTimeWindow2,tempScore2,tempCost2,tempScorelist2,tempCostlist2,tempComp
ensatelist2=CalculateGeneral(tempTrip2,tempTimeWindow2,pr,s,tt,t,currentDay,Day
InTour,o,c)

tempTotalRatio2,tempRatiolist2=CalculateRatio(tempTrip2,s,tt,tempCompensatelist2
)
   tempTotalCompensate2=CalculateCompensate(tempCompensatelist2)
   return tempTrip2, tempTimeWindow2, tempScorelist2, tempCostlist2,
tempRatiolist2, tempTotalRatio2, tempScore2, tempCost2, tempCompensatelist2,
tempTotalCompensate2, ShouldApply2
```

8. LoopRunFile.py

```
from DefCrossOver import *
from DefMutation import *
from PopulationManagement import *
from openpyxl import load_workbook
import time


Budget=15000
loopRound = 0
maxRowIndex=49
CrossOverMaxPop=40
CrossOverRatio=0.4
MutationMaxPop=40
MutationRatio=0.30

#Setting
DayInTour=4
setNumber=1
maxLoop=5
start_time = time.time()
startPoint=0
endPoint=1
TotalRunTime=0

while loopRound < maxLoop:
   crossEndTime =
CrossOver(h,p,l,d,s,pr,t,o,c,tt,Budget,CrossOverMaxPop,CrossOverRatio,DayInTour)
   mutEndTime =
Mutation(h,p,l,d,s,pr,t,o,c,tt,Budget,MutationMaxPop,MutationRatio,DayInTour,start
Point,endPoint)
```

```
    fileName =
'C:\\Users\\Zeds\\opl\\Test1\\Output\\Set'+str(setNumber)+'\\Output'+str(loopRound)+
'.xlsx'
    popEndTime = PopPulationManagment(loopRound,fileName)
    TotalRunTime+= popEndTime + crossEndTime + mutEndTime
    loopRound += 1
print("--- %s seconds ---" % (TotalRunTime))

Data['Runtime'].cell(row=2,column=2).value=TotalRunTime
```

9. PopulationManagement.py

```
import pandas as pd
from openpyxl import load_workbook
from random import choices, randrange, uniform
import time

def PopPulationManagment(Loop,fileName):
    xl = pd.ExcelFile("PreProcessPair.xlsx")
    df = xl.parse("Pool")
    df = df.sort_values('Score')
    df2 = df.sort_values('Score',ascending=False)
    Data=load_workbook('PreProcessPair.xlsx')
    maxCurrentPop=50
    roundCount=0
    Total=0
    NumbList=[]
    startTime = time.time()
    for i in range(0,len(df)):
        NumbList.append(i)
        Total+=float(df.iloc[i]['Score'])
    ascIndexList=[]
    descIndexList=[]
    for i in NumbList:
        ascIndexList.append(float(df.iloc[i]['Score'])/Total)
        descIndexList.append((-1*float(df.iloc[i]['Score'])/Total))
    while roundCount<maxCurrentPop:
        ascRanTour=choices(NumbList,ascIndexList)
        TourSol=df.iloc[ascRanTour[0]]['Tour']
        TotalScore=df.iloc[ascRanTour[0]]['Score']
        Data['CurrPop'].cell(row=roundCount+2,column=2).value=str(TourSol)
        Data['CurrPop'].cell(row=roundCount+2,column=3).value=TotalScore
        descRanTour=choices(NumbList,descIndexList)
        roundCount+=1
        TourSol=df.iloc[descRanTour[0]]['Tour']
        TotalScore=df.iloc[descRanTour[0]]['Score']
        Data['CurrPop'].cell(row=roundCount+2,column=2).value=str(TourSol)
```

```
        Data['CurrPop'].cell(row=roundCount+2,column=3).value=TotalScore
        roundCount+=1
    endTime = time.time() - startTime
    Data.save('PreProcessPair.xlsx')
    writer = pd.ExcelWriter(fileName)
    df2.to_excel(writer,sheet_name='Sheet'+str(Loop)+",header=True,index=False)
    writer.save()
    return endTime
```

10. PreProcessCombine.py

```
from PreProcessLib import *
from openpyxl import load_workbook
from tqdm import tqdm
PrePro = load_workbook('PreProcessPair.xlsx')

for i in tqdm(h,desc='Progress'):
    for j in h:
        Trip=[i,j]
        TimeWindow=[8,20]
        Scorelist=[0,s[j]]
        Costlist=[0,pr[j]]
        Ratiolist=[0,0]
        TotalRatio=0
        Score=s[j]
        Cost=pr[j]
        Compensatelist=[0,0]
        TotalCompensate=0

TempStorage=[Trip,TimeWindow,Scorelist,Costlist,Ratiolist,TotalRatio,Score,Cost,C
ompensatelist,TotalCompensate]
        Budget=30000
        Method=0
        Storage=[]
        MaxScore=Score
        while (Method<4):
            if Method == 0:
                TempStorage =
InA(TempStorage[0],TempStorage[1],TempStorage[2],TempStorage[3],TempStorage
[4],TempStorage[5],TempStorage[6],TempStorage[7],TempStorage[8],TempStorage[
9],Budget,h,p,l,d,s,pr,t,o,c,tt)
                if TempStorage[6] > MaxScore:
                    Method=0
                    MaxScore = TempStorage[6]
                else:
                    Method+=1
            elif Method == 1:
```

```
            TempStorage =
InR(TempStorage[0],TempStorage[1],TempStorage[2],TempStorage[3],TempStorage
[4],TempStorage[5],TempStorage[6],TempStorage[7],TempStorage[8],TempStorage[
9],Budget,h,p,l,d,s,pr,t,o,c,tt)
            if TempStorage[6] > MaxScore:
               Method=0
               MaxScore = TempStorage[6]
            else:
               Method+=1
               IniTime = TempStorage[1][-1:]
         elif Method == 2:
            TempStorage =
TwO(TempStorage[0],TempStorage[1],TempStorage[2],TempStorage[3],TempStorag
e[4],TempStorage[5],TempStorage[6],TempStorage[7],TempStorage[8],TempStorage
[9],Budget,h,p,l,d,s,pr,t,o,c,tt)
            if TempStorage[1][-1:] < IniTime:
               Method=0
               IniTime = TempStorage[1][-1:]
            else:
               Method+=1
         else:
            TempStorage =
MB(TempStorage[0],TempStorage[1],TempStorage[2],TempStorage[3],TempStorage
[4],TempStorage[5],TempStorage[6],TempStorage[7],TempStorage[8],TempStorage[
9],Budget,h,p,l,d,s,pr,t,o,c,tt)
            if TempStorage[1][-1:] < IniTime:
               Method=0
               IniTime = TempStorage[1][-1:]
            else:
               Method+=1
      PrePro['Pair'].cell(row=i+2,column=j+2).value=str(TempStorage)
      PrePro['Score'].cell(row=i+2,column=j+2).value=str(TempStorage[6])
PrePro.save('PreProcessPair.xlsx')
```

11. PreProcessLib.py

```
'''Insert and Add'''
def
InA(Trip,TimeWindow,Scorelist,Costlist,Ratiolist,TotalRatio,Score,Cost,Compensate
list,TotalCompensate,Budget,h,p,l,d,s,pr,t,o,c,tt):
   '''Tabu was used to lock the position of current solution (Lock position after
Restaurant is adding)'''
   global Tabu
   Tabu=0
   Round=0
   Lu=0
   Di=0
```

```
   while (Round<2 or Lu<1 or Di<1):
      if (10.5<=TimeWindow[len(TimeWindow[:-2])]<=13 and Lu<1 and all(i not in l
for i in Trip)) or (Round>0 and Lu<1 and all(i not in l for i in Trip)):
         Ratio=0
         for i in l:
            MiniTime=TimeWindow[:-1]
            MiniComp=Compensatelist[:-1]
            TW = MiniTime[len(MiniTime[:-1])]+tt.cell(row=5+Trip[len(Trip[:-
2])],column=5+i).value+t[i]
            TW,twPassFlag,compensate=oneNodeTWCalculation(TW,o,c,i)
            if twPassFlag and Cost+pr[i]<=Budget:
               x = s[i]/(tt.cell(row=5+Trip[len(Trip)-2],column=5+i).value +
tt.cell(row=5+i,column=5+Trip[len(Trip)-1]).value + compensate)
               if Ratio < x:
                  MiniTime.append(TW)
                  MiniComp.append(compensate)
                  del TW
                  TempTrip=Trip[:]
                  TempScore=Scorelist[:]
                  TempCost=Costlist[:]
                  TempTrip.insert(len(Trip[:-1]),i)
                  TempScore.insert(len(Scorelist[:-1]),s[i])
                  TempCost.insert(len(Costlist[:-1]),pr[i])
                  TW = MiniTime[len(MiniTime[:-
1])]+tt.cell(row=5+TempTrip[len(TempTrip[:-
2])],column=5+TempTrip[len(TempTrip[:-1])]).value+t[TempTrip[len(TempTrip[:-
1])]]

TW,twPassFlag,compensate=oneNodeTWCalculation(TW,o,c,TempTrip[len(TempTr
ip[:-1])])
                  if twPassFlag:
                     MiniTime.append(TW)
                     MiniComp.append(compensate)
                     if len(TempTrip)==len(MiniTime):
                        MirrorTime=MiniTime
                        MirrorComp=MiniComp
                        MirrorTrip=TempTrip
                        MirrorScore=TempScore
                        MirrorCost=TempCost
                        Tabu=len(TempTrip[:-2])
                        Ratio = x
                        SaveScore=s[i]
                        SaveCost=pr[i]
                        Lu+=1
                     del TempTrip, TempScore, TempCost
                  del x
               del MiniTime
```

```
        if 'MirrorTrip' in locals():
            Trip=MirrorTrip
            TimeWindow=MirrorTime
            Compensatelist=MirrorComp
            Scorelist=MirrorScore
            Costlist=MirrorCost
            Score+=SaveScore
            Cost+=SaveCost
            del MirrorTime, MirrorTrip, MirrorScore, MirrorCost, SaveScore, SaveCost
        else:
            Lu+=1
            Di+=1
            Round+=4
            TotalCompensate=CalculateCompensate(Compensatelist)
            TotalRatio, Ratiolist = CalculateRatio(Trip,s,tt,Compensatelist)
    elif (15.5<=TimeWindow[len(TimeWindow[:-2])]<=18 and Di<1 and all(i not in
d for i in Trip)) or (Round>0 and Di<1 and all(i not in d for i in Trip)):
        Ratio=0
        for i in d:
            MiniTime=TimeWindow[:-1]
            MiniComp=Compensatelist[:-1]
            TW = MiniTime[len(MiniTime[:-1])]+tt.cell(row=5+Trip[len(Trip[:-
2])],column=5+i).value+t[i]
            TW,twPassFlag,compensate=oneNodeTWCalculation(TW,o,c,i)
            if twPassFlag and Cost+pr[i]<=Budget:
                x = s[i]/(tt.cell(row=5+Trip[len(Trip)-2],column=5+i).value +
tt.cell(row=5+i,column=5+Trip[len(Trip)-1]).value + compensate)
                if Ratio < x:
                    MiniTime.append(TW)
                    MiniComp.append(compensate)
                    del TW
                    TempTrip=Trip[:]
                    TempScore=Scorelist[:]
                    TempCost=Costlist[:]
                    TempTrip.insert(len(Trip[:-1]),i)
                    TempScore.insert(len(Scorelist[:-1]),s[i])
                    TempCost.insert(len(Costlist[:-1]),pr[i])
                    TW = MiniTime[len(MiniTime[:-
1])]+tt.cell(row=5+TempTrip[len(TempTrip[:-
2])],column=5+TempTrip[len(TempTrip[:-1])]).value+t[TempTrip[len(TempTrip[:-
1])]]

TW,twPassFlag,compensate=oneNodeTWCalculation(TW,o,c,TempTrip[len(TempTr
ip[:-1])])
                    if twPassFlag:
                        MiniTime.append(TW)
                        MiniComp.append(compensate)
```

```
                    if len(TempTrip)==len(MiniTime):
                        MirrorTime=MiniTime
                        MirrorComp=MiniComp
                        MirrorTrip=TempTrip
                        MirrorScore=TempScore
                        MirrorCost=TempCost
                        Tabu=len(TempTrip[:-2])
                        Ratio = x
                        SaveScore=s[i]
                        SaveCost=pr[i]
                        Di+=1
                    del TempTrip, TempScore, TempCost
                del x
            del MiniTime
        if 'MirrorTrip' in locals():
            Trip=MirrorTrip
            TimeWindow=MirrorTime
            Compensatelist=MirrorComp
            Scorelist=MirrorScore
            Costlist=MirrorCost
            Score+=SaveScore
            Cost+=SaveCost
            del MirrorTime, MirrorTrip, MirrorScore, MirrorCost, SaveScore, SaveCost
        else:
            Di+=1
            Round+=4
            TotalCompensate=CalculateCompensate(Compensatelist)
            TotalRatio, Ratiolist = CalculateRatio(Trip,s,tt,Compensatelist)
    else:
        Ratio = 0
        for i in p:
            if i not in Trip:
                for j in range(Tabu,len(Trip)-1):
                    for k in range(Tabu+1,len(Trip)):
                        if j < k:
                            MiniTime=TimeWindow[:k]
                            MiniComp=Compensatelist[:k]
                            TW = MiniTime[k-1]+tt.cell(row=5+Trip[k-
1],column=5+i).value+t[i]
                            TW,twPassFlag,compensate=oneNodeTWCalculation(TW,o,c,i)
                            if twPassFlag and Cost+pr[i]<=Budget:
                                x = s[i]/(tt.cell(row=5+Trip[k-1],column=5+i).value +
tt.cell(row=5+i,column=5+Trip[k]).value + compensate)
                                if Ratio < x:
                                    MiniTime.append(TW)
                                    MiniComp.append(compensate)
                                    del TW
```

```
                                TempTrip=Trip[:]
                                TempScore=Scorelist[:]
                                TempCost=Costlist[:]
                                TempTrip.insert(k,i)
                                TempScore.insert(k,s[i])
                                TempCost.insert(k,pr[i])
                                for RA in range(len(TempTrip[:k]),len(TempTrip)-1):
                                    TW =
MiniTime[RA]+tt.cell(row=5+TempTrip[RA],column=5+TempTrip[RA+1]).value+t[
TempTrip[RA+1]]

TW,twPassFlag,compensate=oneNodeTWCalculation(TW,o,c,TempTrip[RA+1])
                                    if twPassFlag:
                                        MiniTime.append(TW)
                                        MiniComp.append(compensate)
                                    else:
                                        break
                                if len(TempTrip)==len(MiniTime):
                                    MirrorTime=MiniTime
                                    MirrorComp=MiniComp
                                    MirrorTrip=TempTrip
                                    MirrorScore=TempScore
                                    MirrorCost=TempCost
                                    Ratio = x
                                    SaveScore=s[i]
                                    SaveCost=pr[i]
                                del TempTrip, TempScore, TempCost
                            del x
                        del MiniTime
                        break
            if 'MirrorTrip' in locals():
                IniTrip=MirrorTrip
                Trip=MirrorTrip
                TimeWindow=MirrorTime
                Compensatelist=MirrorComp
                Scorelist=MirrorScore
                Costlist=MirrorCost
                Score+=SaveScore
                Cost+=SaveCost
                del MirrorTime, MirrorTrip, MirrorScore, MirrorCost, SaveScore, SaveCost
            else:
                Round+=1
            if Round>1:
                if Lu<1 or Di<1:
                    Lu+=1
                    Di+=1
                TotalCompensate=CalculateCompensate(Compensatelist)
```

```
            TotalRatio, Ratiolist = CalculateRatio(Trip,s,tt,Compensatelist)
    return
[Trip,TimeWindow,Scorelist,Costlist,Ratiolist,TotalRatio,Score,Cost,Compensatelist,
TotalCompensate]

'''Insert and Replace'''
def
InR(Trip,TimeWindow,Scorelist,Costlist,Ratiolist,TotalRatio,Score,Cost,Compensate
list,TotalCompensate,Budget,h,p,l,d,s,pr,t,o,c,tt):
    Round=0
    OpenReplace=0
    while (Round<1):
        Ratio = 0
        for i in p:
            if i not in Trip:
                for j in range(0,len(Trip)-1):
                    for k in range(1,len(Trip)):
                        if j < k:
                            MiniTime=TimeWindow[:k]
                            MiniComp=Compensatelist[:k]
                            TW = MiniTime[k-1]+tt.cell(row=5+Trip[k-
1],column=5+i).value+t[i]
                            TW,twPassFlag,compensate=oneNodeTWCalculation(TW,o,c,i)
                            if twPassFlag and Cost+pr[i]<=Budget:
                                x = s[i]/(tt.cell(row=5+Trip[k-1],column=5+i).value +
tt.cell(row=5+i,column=5+Trip[k]).value + compensate)
                                if Ratio < x:
                                    MiniTime.append(TW)
                                    MiniComp.append(compensate)
                                    del TW
                                    TempTrip=Trip[:]
                                    TempScore=Scorelist[:]
                                    TempCost=Costlist[:]
                                    TempTrip.insert(k,i)
                                    TempScore.insert(k,s[i])
                                    TempCost.insert(k,pr[i])
                                    for RA in range(len(TempTrip[:k]),len(TempTrip)-1):
                                        TW =
MiniTime[RA]+tt.cell(row=5+TempTrip[RA],column=5+TempTrip[RA+1]).value+t[
TempTrip[RA+1]]

TW,twPassFlag,compensate=oneNodeTWCalculation(TW,o,c,TempTrip[RA+1])
                                    if twPassFlag:
                                        MiniTime.append(TW)
                                        MiniComp.append(compensate)
                                    else:
                                        break
```

```
                        if len(TempTrip)==len(MiniTime):
                            MirrorTime=MiniTime
                            MirrorComp=MiniComp
                            MirrorTrip=TempTrip
                            MirrorScore=TempScore
                            MirrorCost=TempCost
                            Ratio = x
                            SaveScore=s[i]
                            SaveCost=pr[i]
                        del TempTrip, TempScore, TempCost
                    del x
                del MiniTime
                break
        if 'MirrorTrip' in locals():
            OpenReplace+=1
            Trip=MirrorTrip
            TimeWindow=MirrorTime
            Compensatelist=MirrorComp
            Scorelist=MirrorScore
            Costlist=MirrorCost
            Score+=SaveScore
            Cost+=SaveCost
            del MirrorTime, MirrorTrip, MirrorScore, MirrorCost, SaveScore, SaveCost
        else:
            Round+=1
        if Round>0:
            TotalCompensate=CalculateCompensate(Compensatelist)
            TotalRatio, Ratiolist = CalculateRatio(Trip,s,tt,Compensatelist)
    if OpenReplace<1:
        del OpenReplace
        for j in Trip[1:-1]:
            if j not in l+d:
                Ratio=0
                for i in p:
                    if i not in Trip:
                        Index=Trip.index(j)
                        MiniTime=TimeWindow[:Index]
                        MiniComp=Compensatelist[:Index]
                        TW = MiniTime[Index-1]+tt.cell(row=5+Trip[Index-
1],column=5+i).value+t[i]
                        TW,twPassFlag,compensate=oneNodeTWCalculation(TW,o,c,i)
                        if twPassFlag and Cost+pr[i]-Costlist[Index]<=Budget:
                            x = s[i]/(tt.cell(row=5+Trip[Index-1],column=5+i).value +
tt.cell(row=5+i,column=5+Trip[Index+1]).value + compensate)
                        Champ = Ratiolist[Index]
                        if Champ < x and Ratio < x:
                            MiniTime.append(TW)
```

```
                        MiniComp.append(compensate)
                        del TW
                        TempTrip=Trip[:]
                        TempScore=Scorelist[:]
                        TempCost=Costlist[:]
                        TempTrip.pop(Index)
                        TempScore.pop(Index)
                        TempCost.pop(Index)
                        TempTrip.insert(Index,i)
                        TempScore.insert(Index,s[i])
                        TempCost.insert(Index,pr[i])
                        for RA in range(len(TempTrip[:Index]),len(TempTrip)-1):
                            TW =
MiniTime[RA]+tt.cell(row=5+TempTrip[RA],column=5+TempTrip[RA+1]).value+t[
TempTrip[RA+1]]

TW,twPassFlag,compensate=oneNodeTWCalculation(TW,o,c,TempTrip[RA+1])
                            if twPassFlag:
                                MiniTime.append(TW)
                                MiniComp.append(compensate)
                            else:
                                break
                        if len(TempTrip)==len(MiniTime):
                            MirrorTime=MiniTime
                            MirrorComp=MiniComp
                            MirrorTrip=TempTrip
                            MirrorScore=TempScore
                            MirrorCost=TempCost
                            Ratio=x
                            SaveScore=s[i]-s[Index]
                            SaveCost=pr[i]-Costlist[Index]
                            del TempTrip, TempScore, TempCost
                    del x
                del Index, MiniTime
    if 'MirrorTrip' in locals():
        Trip=MirrorTrip
        TimeWindow=MirrorTime
        Compensatelist=MirrorComp
        Scorelist=MirrorScore
        Costlist=MirrorCost
        Score+=SaveScore
        Cost+=SaveCost
        TotalCompensate=CalculateCompensate(Compensatelist)
        TotalRatio, Ratiolist = CalculateRatio(Trip,s,tt,Compensatelist)
        del MirrorTime, MirrorTrip, MirrorScore, MirrorCost, SaveScore, SaveCost
```

```
    return
[Trip,TimeWindow,Scorelist,Costlist,Ratiolist,TotalRatio,Score,Cost,Compensatelist,
TotalCompensate]

'''Two-Opt'''
def
TwO(Trip,TimeWindow,Scorelist,Costlist,Ratiolist,TotalRatio,Score,Cost,Compensat
elist,TotalCompensate,Budget,h,p,l,d,s,pr,t,o,c,tt):
    MaxDiff=0
    totalDiff=0
    minCompensate=TotalCompensate
    for i in Trip[1:-1]:
        if i not in l+d:
            Index=Trip.index(i)
            if Trip[Index+1] in l or (Trip[Index+1] in d and (not Trip[Index+2] in h)):
                LD = (tt.cell(row=5+Trip[Index+1],column=5+i).value +
tt.cell(row=5+i,column=5+Trip[Index+3]).value)
                LD2 = (tt.cell(row=5+Trip[Index-1],column=5+Trip[Index+2]).value +
tt.cell(row=5+Trip[Index+2],column=5+Trip[Index+1]).value)
                Old = (tt.cell(row=5+Trip[Index-1],column=5+i).value +
tt.cell(row=5+i,column=5+Trip[Index+1]).value)
                Old2 =
(tt.cell(row=5+Trip[Index+1],column=5+Trip[Index+2]).value+tt.cell(row=5+Trip[In
dex+2],column=5+Trip[Index+3]).value)
                Diff = (Old + Old2) - (LD + LD2)
                if Diff > MaxDiff:
                    MiniTime=TimeWindow[:Index]
                    MiniComp=Compensatelist[:Index]
                    TW = MiniTime[Index-1]+tt.cell(row=5+Trip[Index-
1],column=5+Trip[Index+2]).value+t[Trip[Index+2]]

TW,twPassFlag,compensate=oneNodeTWCalculation(TW,o,c,Trip[Index+2])
                    if twPassFlag:
                        MiniTime.append(TW)
                        MiniComp.append(compensate)
                        del TW
                        TempTrip=Trip[:]
                        TempScore=Scorelist[:]
                        TempCost=Costlist[:]
                        TempTrip[Index],TempTrip[Index+2] =
TempTrip[Index+2],TempTrip[Index]
                        TempScore[Index],TempScore[Index+2] =
TempScore[Index+2],TempScore[Index]
                        TempCost[Index],TempCost[Index+2] =
TempCost[Index+2],TempCost[Index]
                        for RA in range(len(TempTrip[:Index-1]),len(TempTrip)-1):
```

```
                    TW =
MiniTime[RA]+tt.cell(row=5+TempTrip[RA],column=5+TempTrip[RA+1]).value+t[
TempTrip[RA+1]]

TW,twPassFlag,compensate=oneNodeTWCalculation(TW,o,c,TempTrip[RA+1])
                    if twPassFlag:
                       MiniTime.append(TW)
                       MiniComp.append(compensate)
                    else:
                       break
                 if len(TempTrip)==len(MiniTime):
                    tempMinCompensate=CalculateCompensate(MiniComp)
                    compensateDiff=TotalCompensate-tempMinCompensate
                    tempTotalDiff=Diff+compensateDiff
                    if tempTotalDiff >= totalDiff:
                       MirrorTime=MiniTime
                       MirrorComp=MiniComp
                       MirrorTrip=TempTrip
                       MirrorScore=TempScore
                       MirrorCost=TempCost
                       minCompensate=tempMinCompensate
                       totalDiff=tempTotalDiff
                       MaxDiff = Diff
                 del TempTrip, TempScore, TempCost
              del MiniTime
           del Index
        elif Trip[Index+1] in p:
           P = (tt.cell(row=5+Trip[Index-1],column=5+Trip[Index+1]).value +
tt.cell(row=5+Trip[Index+1],column=5+i).value +
tt.cell(row=5+i,column=5+Trip[Index+2]).value)
           Old = (tt.cell(row=5+Trip[Index-1],column=5+i).value +
tt.cell(row=5+i,column=5+Trip[Index+1]).value +
tt.cell(row=5+Trip[Index+1],column=5+Trip[Index+2]).value)
           Diff = Old - P
           if Diff > MaxDiff:
              MiniTime=TimeWindow[:Index]
              MiniComp=Compensatelist[:Index]
              TW = MiniTime[Index-1]+tt.cell(row=5+Trip[Index-
1],column=5+Trip[Index+1]).value+t[Trip[Index+1]]

TW,twPassFlag,compensate=oneNodeTWCalculation(TW,o,c,Trip[Index+1])
              if twPassFlag:
                 MiniTime.append(TW)
                 MiniComp.append(compensate)
                 del TW
                 TempTrip=Trip[:]
                 TempScore=Scorelist[:]
```

```
                TempCost=Costlist[:]
                TempTrip[Index],TempTrip[Index+1] =
TempTrip[Index+1],TempTrip[Index]
                TempScore[Index],TempScore[Index+1] =
TempScore[Index+1],TempScore[Index]
                TempCost[Index],TempCost[Index+1] =
TempCost[Index+1],TempCost[Index]
                for RA in range(len(TempTrip[:Index-1]),len(TempTrip)-1):
                   TW =
MiniTime[RA]+tt.cell(row=5+TempTrip[RA],column=5+TempTrip[RA+1]).value+t[
TempTrip[RA+1]]

TW,twPassFlag,compensate=oneNodeTWCalculation(TW,o,c,TempTrip[RA+1])
                   if twPassFlag:
                      MiniTime.append(TW)
                      MiniComp.append(compensate)
                   else:
                      break
                if len(TempTrip)==len(MiniTime):
                   tempMinCompensate=CalculateCompensate(MiniComp)
                   compensateDiff=TotalCompensate-tempMinCompensate
                   tempTotalDiff=Diff+compensateDiff
                   if tempTotalDiff >= totalDiff:
                      MirrorTime=MiniTime
                      MirrorComp=MiniComp
                      MirrorTrip=TempTrip
                      MirrorScore=TempScore
                      MirrorCost=TempCost
                      minCompensate=tempMinCompensate
                      totalDiff=tempTotalDiff
                      MaxDiff = Diff
                del TempTrip, TempScore, TempCost
             del MiniTime
          del Index
       else:
          del Index
          break
   if 'MirrorTrip' in locals():
      Trip=MirrorTrip
      TimeWindow=MirrorTime
      Compensatelist=MirrorComp
      TotalCompensate=minCompensate
      Scorelist=MirrorScore
      Costlist=MirrorCost
      TotalRatio, Ratiolist = CalculateRatio(Trip,s,tt,Compensatelist)
```

```
    return
[Trip,TimeWindow,Scorelist,Costlist,Ratiolist,TotalRatio,Score,Cost,Compensatelist,
TotalCompensate]

'''MoveBest Needed to be collect'''
'''def
MBOld(Trip,TimeWindow,Scorelist,Costlist,Ratiolist,TotalRatio,Score,Cost,Budget,
h,p,l,d,s,pr,t,o,c,tt):
    MaxDiff=0
    for i in range(1,len(Trip[1:-1])):
        if Trip[i] not in l+d:
            for j in range(i+1,len(Trip[1:])):
                if Trip[j] not in l+d:
                    if j==i+1:
                        P = (tt.cell(row=5+Trip[i-1],column=5+Trip[j]).value +
tt.cell(row=5+Trip[j],column=5+Trip[i]).value +
tt.cell(row=5+Trip[i],column=5+Trip[i+2]).value)
                        Old = (tt.cell(row=5+Trip[i-1],column=5+Trip[i]).value +
tt.cell(row=5+Trip[i],column=5+Trip[j]).value +
tt.cell(row=5+Trip[j],column=5+Trip[i+2]).value)
                        Diff = Old - P
                        if Diff > MaxDiff:
                            MiniTime=TimeWindow[:i]
                            TW = MiniTime[i-1] + tt.cell(row=5+Trip[i-
1],column=5+Trip[j]).value + t[Trip[j]]
                            if o[Trip[j]]<=TW<=c[Trip[j]]:
                                MiniTime.append(TW)
                                del TW
                                TempTrip=Trip[:]
                                TempScore=Scorelist[:]
                                TempCost=Costlist[:]
                                TempTrip[i],TempTrip[j] = TempTrip[j],TempTrip[i]
                                TempScore[i],TempScore[j] = TempScore[j],TempScore[i]
                                TempCost[i],TempCost[j] = TempCost[j],TempCost[i]
                                for RA in range(len(TempTrip[:i]),len(TempTrip)-1):
                                    TW =
MiniTime[RA]+tt.cell(row=5+TempTrip[RA],column=5+TempTrip[RA+1]).value+t[
TempTrip[RA+1]]
                                    if o[TempTrip[RA+1]]<=TW<=c[TempTrip[RA+1]]:
                                        MiniTime.append(TW)
                                    else:
                                        break
                                if len(TempTrip)==len(MiniTime):
                                    MirorTime=MiniTime
                                    MirorTrip=TempTrip
                                    MirorScore=TempScore
                                    MirorCost=TempCost
```

```
                        MaxDiff = Diff
                    del TempTrip, TempScore, TempCost
                del MiniTime
            del P, Old
        else:
            P = (tt.cell(row=5+Trip[i-1],column=5+Trip[j]).value +
tt.cell(row=5+Trip[j],column=5+Trip[i+1]).value)
            P2 = (tt.cell(row=5+Trip[j-1],column=5+Trip[i]).value +
tt.cell(row=5+Trip[i],column=5+Trip[j+1]).value)
            Old = (tt.cell(row=5+Trip[i-1],column=5+Trip[i]).value +
tt.cell(row=5+Trip[i],column=5+Trip[i+1]).value)
            Old2 = (tt.cell(row=5+Trip[j-1],column=5+Trip[j]).value +
tt.cell(row=5+Trip[j],column=5+Trip[j+1]).value)
            Diff = (Old + Old2) - (P + P2)
            if Diff > MaxDiff:
                MiniTime=TimeWindow[:i]
                TW = MiniTime[i-1] + tt.cell(row=5+Trip[i-
1],column=5+Trip[j]).value + t[Trip[j]]
                if o[Trip[j]]<=TW<=c[Trip[j]]:
                    MiniTime.append(TW)
                    del TW
                    TempTrip=Trip[:]
                    TempScore=Scorelist[:]
                    TempCost=Costlist[:]
                    TempTrip[i],TempTrip[j] = TempTrip[j],TempTrip[i]
                    TempScore[i],TempScore[j] = TempScore[j],TempScore[i]
                    TempCost[i],TempCost[j] = TempCost[j],TempCost[i]
                    for RA in range(len(TempTrip[:i]),len(TempTrip)-1):
                        TW =
MiniTime[RA]+tt.cell(row=5+TempTrip[RA],column=5+TempTrip[RA+1]).value+t[
TempTrip[RA+1]]
                        if o[TempTrip[RA+1]]<=TW<=c[TempTrip[RA+1]]:
                            MiniTime.append(TW)
                        else:
                            break
                    if len(TempTrip)==len(MiniTime):
                        MirrorTime=MiniTime
                        MirrorTrip=TempTrip
                        MirrorScore=TempScore
                        MirrorCost=TempCost
                        MaxDiff = Diff
                    del TempTrip, TempScore, TempCost
                del MiniTime
            del P, P2, Old, Old2
    if 'MirrorTrip' in locals():
        Trip=MirrorTrip
        TimeWindow=MirrorTime
```

```
        Scorelist=MirrorScore
        Costlist=MirrorCost
        Ratiolist=[0,0]
        TotalRatio=0
        for RB in Trip[1:-1]:
            Index=Trip.index(RB)
            Champ = s[RB]/(tt.cell(row=5+Trip[Index-1],column=5+RB).value +
tt.cell(row=5+RB,column=5+Trip[Index+1]).value)
            TotalRatio+=Champ
            Ratiolist.insert(Index,Champ)
            del Index, Champ
    return [Trip,TimeWindow,Scorelist,Costlist,Ratiolist,TotalRatio,Score,Cost]'"

'"MB"'
def
MB(Trip,TimeWindow,Scorelist,Costlist,Ratiolist,TotalRatio,Score,Cost,Compensate
list,TotalCompensate,Budget,h,p,l,d,s,pr,t,o,c,tt):
    MaxDiff=0
    totalDiff=0
    minCompensate=TotalCompensate
    for i in range(1,len(Trip[1:])):
        if Trip[i] not in l+d:
            for j in range(1,len(Trip[1:])):
                if Trip[j] not in l+d:
                    if (j!=i) and (j==i+1):# For i occurred before j but next to each other
                        P = (tt.cell(row=5+Trip[i-1],column=5+Trip[j]).value +
tt.cell(row=5+Trip[j],column=5+Trip[i]).value +
tt.cell(row=5+Trip[i],column=5+Trip[j+1]).value)
                        Old = (tt.cell(row=5+Trip[i-1],column=5+Trip[i]).value +
tt.cell(row=5+Trip[i],column=5+Trip[j]).value +
tt.cell(row=5+Trip[j],column=5+Trip[j+1]).value)
                        Diff = Old - P
                        if Diff > MaxDiff:
                            MiniTime=TimeWindow[:i]
                            MiniComp=Compensatelist[:i]
                            TW = MiniTime[i-1] + tt.cell(row=5+Trip[i-
1],column=5+Trip[j]).value + t[Trip[j]]

TW,twPassFlag,compensate=oneNodeTWCalculation(TW,o,c,Trip[j])
                            if twPassFlag:
                                MiniTime.append(TW)
                                MiniComp.append(compensate)
                                del TW
                                TempTrip=Trip[:]
                                TempScore=Scorelist[:]
                                TempCost=Costlist[:]
                                TempTrip[i],TempTrip[j] = TempTrip[j],TempTrip[i]
```

```
                        TempScore[i],TempScore[j] = TempScore[j],TempScore[i]
                        TempCost[i],TempCost[j] = TempCost[j],TempCost[i]
                        for RA in range(len(TempTrip[:i]),len(TempTrip)-1):
                            TW =
MiniTime[RA]+tt.cell(row=5+TempTrip[RA],column=5+TempTrip[RA+1]).value+t[
TempTrip[RA+1]]

TW,twPassFlag,compensate=oneNodeTWCalculation(TW,o,c,TempTrip[RA+1])
                            if twPassFlag:
                                MiniTime.append(TW)
                                MiniComp.append(compensate)
                            else:
                                break
                        if len(TempTrip)==len(MiniTime):
                            tempMinCompensate=CalculateCompensate(MiniComp)
                            compensateDiff=TotalCompensate-tempMinCompensate
                            tempTotalDiff=Diff+compensateDiff
                            if tempTotalDiff >= totalDiff:
                                MirorTime=MiniTime
                                MirrorComp=MiniComp
                                MirorTrip=TempTrip
                                MirorScore=TempScore
                                MirorCost=TempCost
                                minCompensate=tempMinCompensate
                                totalDiff=tempTotalDiff
                                MaxDiff = Diff
                        del TempTrip, TempScore, TempCost
                    del MiniTime
                del P, Old
            elif (j!=i) and (j==i-1):# For i occurred after j but next to each other
                P = (tt.cell(row=5+Trip[j-1],column=5+Trip[i]).value +
tt.cell(row=5+Trip[i],column=5+Trip[j]).value +
tt.cell(row=5+Trip[j],column=5+Trip[i+1]).value)
                Old = (tt.cell(row=5+Trip[j-1],column=5+Trip[j]).value +
tt.cell(row=5+Trip[j],column=5+Trip[i]).value +
tt.cell(row=5+Trip[i],column=5+Trip[i+1]).value)
                Diff = Old - P
                if Diff > MaxDiff:
                    MiniTime=TimeWindow[:j]
                    MiniComp=Compensatelist[:j]
                    TW = MiniTime[j-1] + tt.cell(row=5+Trip[j-
1],column=5+Trip[i]).value + t[Trip[i]]

TW,twPassFlag,compensate=oneNodeTWCalculation(TW,o,c,Trip[i])
                if twPassFlag:
                    MiniTime.append(TW)
                    MiniComp.append(compensate)
```

```
                    del TW
                    TempTrip=Trip[:]
                    TempScore=Scorelist[:]
                    TempCost=Costlist[:]
                    TempTrip[i],TempTrip[j] = TempTrip[j],TempTrip[i]
                    TempScore[i],TempScore[j] = TempScore[j],TempScore[i]
                    TempCost[i],TempCost[j] = TempCost[j],TempCost[i]
                    for RA in range(len(TempTrip[:j]),len(TempTrip)-1):
                       TW =
MiniTime[RA]+tt.cell(row=5+TempTrip[RA],column=5+TempTrip[RA+1]).value+t[
TempTrip[RA+1]]

TW,twPassFlag,compensate=oneNodeTWCalculation(TW,o,c,TempTrip[RA+1])
                       if twPassFlag:
                          MiniTime.append(TW)
                          MiniComp.append(compensate)
                       else:
                          break
                    if len(TempTrip)==len(MiniTime):
                       tempMinCompensate=CalculateCompensate(MiniComp)
                       compensateDiff=TotalCompensate-tempMinCompensate
                       tempTotalDiff=Diff+compensateDiff
                       if tempTotalDiff >= totalDiff:
                          MirorTime=MiniTime
                          MirrorComp=MiniComp
                          MirorTrip=TempTrip
                          MirorScore=TempScore
                          MirorCost=TempCost
                          minCompensate=tempMinCompensate
                          totalDiff=tempTotalDiff
                          MaxDiff = Diff
                    del TempTrip, TempScore, TempCost
                 del MiniTime
              del P, Old
           elif (j!=i) and (j>i+1):# For i occurred before j and distance more than 1
              P = (tt.cell(row=5+Trip[j-1],column=5+Trip[i]).value +
tt.cell(row=5+Trip[i],column=5+Trip[j]).value)
              P2 = (tt.cell(row=5+Trip[i-1],column=5+Trip[i+1]).value)
              Old = (tt.cell(row=5+Trip[i-1],column=5+Trip[i]).value +
tt.cell(row=5+Trip[i],column=5+Trip[i+1]).value)
              Old2 = (tt.cell(row=5+Trip[j-1],column=5+Trip[j]).value)
              Diff = (Old - Old2) - (P + P2)
              if Diff > MaxDiff:
                 MiniTime=TimeWindow[:i]
                 MiniComp=Compensatelist[:i]
                 TW = MiniTime[i-1] + tt.cell(row=5+Trip[i-
1],column=5+Trip[i+1]).value + t[Trip[i+1]]
```

```
TW,twPassFlag,compensate=oneNodeTWCalculation(TW,o,c,Trip[i+1])
                    if twPassFlag:
                        MiniTime.append(TW)
                        MiniComp.append(compensate)
                        del TW
                        TempTrip=Trip[:]
                        TempScore=Scorelist[:]
                        TempCost=Costlist[:]
                        TempTrip.insert(j,TempTrip[i])
                        TempTrip.pop(i)
                        TempScore.insert(j,TempScore[i])
                        TempScore.pop(i)
                        TempCost.insert(i,TempCost[i])
                        TempCost.pop(i)
                        for RA in range(len(TempTrip[:i]),len(TempTrip)-1):
                            TW =
MiniTime[RA]+tt.cell(row=5+TempTrip[RA],column=5+TempTrip[RA+1]).value+t[
TempTrip[RA+1]]

TW,twPassFlag,compensate=oneNodeTWCalculation(TW,o,c,TempTrip[RA+1])
                            if twPassFlag:
                                MiniTime.append(TW)
                                MiniComp.append(compensate)
                            else:
                                break
                        if len(TempTrip)==len(MiniTime):
                            tempMinCompensate=CalculateCompensate(MiniComp)
                            compensateDiff=TotalCompensate-tempMinCompensate
                            tempTotalDiff=Diff+compensateDiff
                            if tempTotalDiff >= totalDiff:
                                MirrorTime=MiniTime
                                MirrorComp=MiniComp
                                MirrorTrip=TempTrip
                                MirrorScore=TempScore
                                MirrorCost=TempCost
                                minCompensate=tempMinCompensate
                                totalDiff=tempTotalDiff
                                MaxDiff = Diff
                            del TempTrip, TempScore, TempCost
                        del MiniTime
                    del P, P2, Old, Old2
                elif (j!=i) and (j<i-1):# For i occurred after j and distance more than 1
                    P = (tt.cell(row=5+Trip[j-1],column=5+Trip[i]).value +
tt.cell(row=5+Trip[i],column=5+Trip[j]).value)
                    P2 = (tt.cell(row=5+Trip[i-1],column=5+Trip[i+1]).value)
```

```
                Old = (tt.cell(row=5+Trip[i-1],column=5+Trip[i]).value +
tt.cell(row=5+Trip[i],column=5+Trip[i+1]).value)
                Old2 = (tt.cell(row=5+Trip[j-1],column=5+Trip[j]).value)
                Diff = (Old + Old2) - (P + P2)
                if Diff > MaxDiff:
                   MiniTime=TimeWindow[:j]
                   MiniComp=Compensatelist[:j]
                   TW = MiniTime[j-1] + tt.cell(row=5+Trip[j-
1],column=5+Trip[i]).value + t[Trip[i]]

TW,twPassFlag,compensate=oneNodeTWCalculation(TW,o,c,Trip[i])
                   if twPassFlag:
                      MiniTime.append(TW)
                      MiniComp.append(compensate)
                      del TW
                      TempTrip=Trip[:]
                      TempScore=Scorelist[:]
                      TempCost=Costlist[:]
                      TempTrip.insert(j,TempTrip[i])
                      TempTrip.pop(i+1)
                      TempScore.insert(j,TempScore[i])
                      TempScore.pop(i+1)
                      TempCost.insert(i,TempCost[i])
                      TempCost.pop(i+1)
                      for RA in range(len(TempTrip[:j]),len(TempTrip)-1):
                         TW =
MiniTime[RA]+tt.cell(row=5+TempTrip[RA],column=5+TempTrip[RA+1]).value+t[
TempTrip[RA+1]]

TW,twPassFlag,compensate=oneNodeTWCalculation(TW,o,c,TempTrip[RA+1])
                         if twPassFlag:
                            MiniTime.append(TW)
                            MiniComp.append(compensate)
                         else:
                            break
                      if len(TempTrip)==len(MiniTime):
                         tempMinCompensate=CalculateCompensate(MiniComp)
                         compensateDiff=TotalCompensate-tempMinCompensate
                         tempTotalDiff=Diff+compensateDiff
                         if tempTotalDiff >= totalDiff:
                            MirrorTime=MiniTime
                            MirrorComp=MiniComp
                            MirrorTrip=TempTrip
                            MirrorScore=TempScore
                            MirrorCost=TempCost
                            minCompensate=tempMinCompensate
                            totalDiff=tempTotalDiff
```

```
                        MaxDiff = Diff
                    del TempTrip, TempScore, TempCost
                del MiniTime
            del P, P2, Old, Old2
    if 'MirrorTrip' in locals():
        Trip=MirrorTrip
        TimeWindow=MirrorTime
        Compensatelist=MirrorComp
        TotalCompensate=minCompensate
        Scorelist=MirrorScore
        Costlist=MirrorCost
        TotalRatio, Ratiolist = CalculateRatio(Trip,s,tt,Compensatelist)
    return
[Trip,TimeWindow,Scorelist,Costlist,Ratiolist,TotalRatio,Score,Cost,Compensatelist,
TotalCompensate]
```

12. Recursive.py

```
from collections import Counter
from Utility import *

'''Used data from TestDatV2 to create list named l and d, Must be sure that there will
be no multiple l and d in the trip'''
'''Remove Attraction with lowest Ratio used in ImproveGAOffspring'''
def LorD(Trip,ForRatioIndex,RefIndex,RefIndex2,l,d):
    MinRatio=min(ForRatioIndex)
    Index=ForRatioIndex.index(MinRatio)
    if Trip[Index] in l or Trip[Index] in d:
        if Index >= RefIndex:
            ForRatioIndex.pop(Index)
            Trip.pop(Index)
            RefIndex2=Index
            Result=LorD(Trip,ForRatioIndex,RefIndex,RefIndex2,l,d)
        else:
            ForRatioIndex.pop(Index)
            Trip.pop(Index)
            RefIndex=Index
            Result=LorD(Trip,ForRatioIndex,RefIndex,RefIndex2,l,d)
    else:
        if Index < RefIndex:
            Result=Index
            '''Migth have to change elif to else and remove current else condition'''
        elif Index >= RefIndex and Index < RefIndex2:
            Result=Index+1
            '''Else may not be unused'''
        else:
            Result=Index+2
```

```
    return Result

'''----- Used to check feasibility of the trip after cross-over state -----'''
''' This is a function to count number of restaurant in CrossOvered trip'''
def CountResNode(Node,Randay,l,d):
    LCounter=0
    DCounter=0
    IndexCounter=0
    LNode=[]
    LRatio=[]
    DNode=[]
    DRatio=[]
    FocusTrip=Node[0+10*Randay]
    FocusTripRatio=Node[4+10*Randay]
    for EachNode in FocusTrip:
        if EachNode in l:
            LCounter = LCounter+1
            LNode.append(IndexCounter)
            LRatio.append(FocusTripRatio[IndexCounter])
        elif EachNode in d:
            DCounter = DCounter+1
            DNode.append(IndexCounter)
            DRatio.append(FocusTripRatio[IndexCounter])
        IndexCounter+=1
    return LCounter, DCounter, LNode, LRatio, DNode, DRatio, FocusTrip,
FocusTripRatio

'''Find attraction before ResFound'''
def FindAttractionBefore(FocusTrip,ResFound,l,d):
    AttIndex=ResFound-1
    AttNode=FocusTrip[AttIndex]
    if AttNode not in l and AttNode not in d:
        print("FindAttractionBefore")
    else:
        AttNode,AttIndex=FindAttractionBefore(FocusTrip,AttIndex,l,d)
    return AttNode, AttIndex

'''Find attraction after ResFound'''
def FindAttractionAfter(FocusTrip,ResFound,l,d):
    AttIndex=ResFound+1
    AttNode=FocusTrip[AttIndex]
    if AttNode not in l and AttNode not in d:
        print("FindAttractionAfter")
    else:
        AttNode,AttIndex=FindAttractionAfter(FocusTrip,AttIndex,l,d)
    return AttNode, AttIndex
```

```
'''----- Used to check feasibility of the tour after cross-over state -----'''
'''Combine all trip in tour'''
def CombineAllTrip(Node,DayInTour):
    CombinedTour=[]
    ScoreRatio=[]
    RangeForEachDay=[]
    UpperBound=0
    for Day in range(DayInTour):
        if Day == 0:
            LowerBound=0
        else:
            LowerBound=UpperBound+1
        UpperBound=LowerBound+len(Node[0+10*Day])-1
        RangeForEachDay.append([LowerBound,UpperBound])   #Set Boundary for
each Day: LowerBound could be used to deduct and find the real location of the node
from index of CombinedTour
        CombinedTour=CombinedTour+Node[0+10*Day][:]
        ScoreRatio=ScoreRatio+Node[4+10*Day][:]
    return CombinedTour, ScoreRatio, RangeForEachDay

'''Find duplicate attraction'''
def FindDuplicateNode(CombinedTour, ScoreRatio,h,p,l,d,s,pr,t,o,c,tt):
    DuplicateNodeList=[]
    DuplicateNodeInfo=[]
    DuplicateNodeRatio=[]
    nodeCount=0
    roundCounter=0
    cominedAllExceptionNode=h
    DuplicateNodeList=[NodeNumber for NodeNumber,Occurence in
Counter(CombinedTour).items() if Occurence>1]
    for DupNode in DuplicateNodeList:
        DuplicateNodeInfo.append([index for index, value in enumerate(CombinedTour)
if value == DupNode])
    for Ratio in DuplicateNodeInfo:
        TempRatio=[]
        for RatioNumber in Ratio:
            TempRatio=TempRatio+[ScoreRatio[RatioNumber]]
        DuplicateNodeRatio=DuplicateNodeRatio+[TempRatio]
    #tempDuplicateNode=DuplicateNodeList[:]
    maxDuplicate=len(DuplicateNodeList)
    '''for Node in tempDuplicateNode:
        if (Node in l) or (Node in d):
            DuplicateNodeList.pop(nodeCount)
            DuplicateNodeInfo.pop(nodeCount)
            DuplicateNodeRatio.pop(nodeCount)
        elif Node in h:
            DuplicateNodeList.pop(nodeCount)
```

```
            DuplicateNodeInfo.pop(nodeCount)
            DuplicateNodeRatio.pop(nodeCount)
        elif Node in p:
            nodeCount+=1'''
    while roundCounter<maxDuplicate:
        if DuplicateNodeList[nodeCount] in cominedAllExceptionNode:
            DuplicateNodeList.pop(nodeCount)
            DuplicateNodeInfo.pop(nodeCount)
            DuplicateNodeRatio.pop(nodeCount)
        else:
            nodeCount+=1
        roundCounter+=1
    return DuplicateNodeList, DuplicateNodeInfo, DuplicateNodeRatio

'''Remove Att with Lowest score in tour'''
def RemoveAttLowestRatioTour(Node, DuplicateNodeList, DuplicateNodeInfo,
DuplicateNodeRatio, RangeForEachDay,h,p,l,d,s,pr,t,o,c,tt):
    IndexCounter=0
    ReducedTour=Node[:]
    for Att in DuplicateNodeList:
        LocationToBeDeleted=[]
        DayToBeDeleted=[]
        DayIndex=[]
        TabuList=[]
        DuplicateNodeInfo[IndexCounter]
        DuplicateNodeRatio[IndexCounter]
        maxNumberOfDupNode=len(DuplicateNodeRatio[IndexCounter])
        numberOfRemovedNode=0
        '''Find location in combined tour'''
        while (maxNumberOfDupNode-numberOfRemovedNode)!=1:
            RatioIndexCounter=0
            MinRatio=9999999
            MinRatioLocation=9999999
            for Ratio in DuplicateNodeRatio[IndexCounter]:
                if Ratio<MinRatio and (not RatioIndexCounter in TabuList):
                    MinRatio=Ratio
                    MinRatioLocation=RatioIndexCounter
                RatioIndexCounter+=1
            currentLocation=DuplicateNodeInfo[IndexCounter][MinRatioLocation]
            LocationToBeDeleted.append(currentLocation)
            TabuList.append(MinRatioLocation)
            numberOfRemovedNode+=1
        '''Find start node for each tirp in combined tour and day index'''
        for Location in LocationToBeDeleted:
            dayIndexCount=0
            for dayRange in RangeForEachDay:
                if dayRange[0]<Location<=dayRange[1]:
```

```
                pointToDeduct=dayRange[0]
                DayToBeDeleted.append(pointToDeduct)
                DayIndex.append(dayIndexCount)
                break
            dayIndexCount+=1
        '''Remove each node in main tour node'''
        dayInTour=len(RangeForEachDay)
        for index in DayIndex:
            CrossOverPoint=10*index
            CurrentTrip=ReducedTour[0+CrossOverPoint][:]
            CurrentTrip.pop(CurrentTrip.index(Att))
            CurrentTimeWindow=ReducedTour[1+CrossOverPoint][:]

CurrentTimeWindow,CurrentScore,CurrentCost,CurrentScoreList,CurrentCostList,Cu
rrentCompensateList=CalculateGeneral(CurrentTrip,CurrentTimeWindow,pr,s,tt,t,ind
ex,dayInTour,o,c)

CurrentTotalRatio,CurrentRatioList=CalculateRatio(CurrentTrip,s,tt,CurrentCompens
ateList)
            CurrentTotalCompensate=CalculateCompensate(CurrentCompensateList)
            ReducedTour[0+CrossOverPoint]=CurrentTrip
            ReducedTour[1+CrossOverPoint]=CurrentTimeWindow
            ReducedTour[2+CrossOverPoint]=CurrentScoreList
            ReducedTour[3+CrossOverPoint]=CurrentCostList
            ReducedTour[4+CrossOverPoint]=CurrentRatioList
            ReducedTour[5+CrossOverPoint]=CurrentTotalRatio
            ReducedTour[6+CrossOverPoint]=CurrentScore
            ReducedTour[7+CrossOverPoint]=CurrentCost
            ReducedTour[8+CrossOverPoint]=CurrentCompensateList
            ReducedTour[9+CrossOverPoint]=CurrentTotalCompensate
        IndexCounter+=1
    return ReducedTour

'''Remove Att with Lowest score in trip'''
def RemoveAttLowestRatioTrip(Node, DuplicateNodeList, DuplicateNodeInfo,
DuplicateNodeRatio,DayInTour,DayIndex,h,p,l,d,s,pr,t,o,c,tt):
    IndexCounter=0
    ReducedTour=Node[:]
    for Att in DuplicateNodeList:
        LocationToBeDeleted=[]
        TabuList=[]
        DuplicateNodeInfo[IndexCounter]
        DuplicateNodeRatio[IndexCounter]
        maxNumberOfDupNode=len(DuplicateNodeRatio[IndexCounter])
        numberOfRemovedNode=0
        '''Find location in combined tour'''
        while (maxNumberOfDupNode-numberOfRemovedNode)!=1:
```

```
            RatioIndexCounter=0
            MinRatio=9999999
            MinRatioLocation=9999999
            for Ratio in DuplicateNodeRatio[IndexCounter]:
               if Ratio<MinRatio and (not RatioIndexCounter in TabuList):
                  MinRatio=Ratio
                  MinRatioLocation=RatioIndexCounter
               RatioIndexCounter+=1
            currentLocation=DuplicateNodeInfo[IndexCounter][MinRatioLocation]
            LocationToBeDeleted.append(currentLocation)
            TabuList.append(MinRatioLocation)
            numberOfRemovedNode+=1
         '''Remove each node in main tour node'''
         for index in LocationToBeDeleted:
            CrossOverPoint=10*DayIndex
            CurrentTrip=ReducedTour[0+CrossOverPoint][:]
            CurrentTrip.pop(CurrentTrip.index(Att))
            CurrentTimeWindow=ReducedTour[1+CrossOverPoint][:]

CurrentTimeWindow,CurrentScore,CurrentCost,CurrentScoreList,CurrentCostList,Cu
rrentCompensateList=CalculateGeneral(CurrentTrip,CurrentTimeWindow,pr,s,tt,t,Da
yIndex,DayInTour,o,c)

CurrentTotalRatio,CurrentRatioList=CalculateRatio(CurrentTrip,s,tt,CurrentCompens
ateList)
            CurrentTotalCompensate=CalculateCompensate(CurrentCompensateList)
            ReducedTour[0+CrossOverPoint]=CurrentTrip
            ReducedTour[1+CrossOverPoint]=CurrentTimeWindow
            ReducedTour[2+CrossOverPoint]=CurrentScoreList
            ReducedTour[3+CrossOverPoint]=CurrentCostList
            ReducedTour[4+CrossOverPoint]=CurrentRatioList
            ReducedTour[5+CrossOverPoint]=CurrentTotalRatio
            ReducedTour[6+CrossOverPoint]=CurrentScore
            ReducedTour[7+CrossOverPoint]=CurrentCost
            ReducedTour[8+CrossOverPoint]=CurrentCompensateList
            ReducedTour[9+CrossOverPoint]=CurrentTotalCompensate
         IndexCounter+=1
      return ReducedTour
```

13. Utility.py

```
def oneNodeTWCalculation(TW,o,c,i):
   if TW >= o[i]:
      compensate=0
   else:
      compensate=o[i]-TW
      TW = o[i]
```

```
    if TW<=c[i]:
        twPassFlag=True
    else:
        twPassFlag=False
    return TW,twPassFlag,compensate

def CalculateRatio(Trip,Score,TravelTime,Compensatelist):
    TotalRatio=0
    RatioList=[0,0]
    for Node in Trip[1:-1]:
        Index=Trip.index(Node)
        Champ = Score[Node]/(TravelTime.cell(row=5+Trip[Index-
1],column=5+Node).value +
TravelTime.cell(row=5+Node,column=5+Trip[Index+1]).value +
Compensatelist[Index])
        TotalRatio+=Champ
        RatioList.insert(Index,Champ)
    return TotalRatio, RatioList

def CalculateCompensate(Compensatelist):
    TotalCompensate=0
    for comepnsate in Compensatelist:
        TotalCompensate+=comepnsate
    return TotalCompensate

def CalculateRatioForOneNode(Trip,Node,Index,Score,TravelTime,Compensatelist):
    Ratio = Score[Node]/(TravelTime.cell(row=5+Trip[Index-
1],column=5+Node).value +
TravelTime.cell(row=5+Node,column=5+Trip[Index+1]).value +
Compensatelist[Index])
    return Ratio

def PrintOutLoud(TextMessage,Value):
    print('----------------------',TextMessage,'value is',Value)

def
CalculateGeneral(Trip,TimeWindow,Price,ReferenceScore,TravelTime,TimeSpend,c
urrentDay,dayInTour,o,c):
    tempTimeWindow=[TimeWindow[0]]
    tempCompensatelist=[0]
    if currentDay != (dayInTour-1):
        lastIndex=len(Trip)-1
        '''Score Initialization'''
        endHotelScore=ReferenceScore[Trip[lastIndex]]
        Score=endHotelScore
        ScoreList=[0,endHotelScore]
        '''Cost Initialization'''
```

```
        endHotelCost=Price[Trip[lastIndex]]
        Cost=endHotelCost
        CostList=[0,endHotelCost]
    else:
        '''Score Initialization'''
        Score=0
        ScoreList=[0,0]
        '''Cost Initialization'''
        Cost=0
        CostList=[0,0]
    IndexCounter=1
    for node in Trip[1:]:
        TW = tempTimeWindow[IndexCounter-
1]+TravelTime.cell(row=5+Trip[IndexCounter-
1],column=5+node).value+TimeSpend[node]
        TW,twPassFlag,compensate = oneNodeTWCalculation(TW,o,c,node)
        tempTimeWindow.append(TW)
        tempCompensatelist.append(compensate)
        if IndexCounter!=len(Trip)-1:
            ScoreList.insert(IndexCounter,ReferenceScore[node])
            CostList.insert(IndexCounter,Price[node])
            Score+=ReferenceScore[node]
            Cost+=Price[node]
        IndexCounter+=1
    return tempTimeWindow,Score,Cost,ScoreList,CostList,tempCompensatelist

def
CalculateGeneralWValidation(Trip,TimeWindow,Price,ReferenceScore,TravelTime,
TimeSpend,o,c,LunchIndex,currentDay,dayInTour):
    tempTimeWindow=[TimeWindow[0]]
    tempCompensatelist=[0]
    if currentDay != (dayInTour-1):
        lastIndex=len(Trip)-1
        '''Score Initialization'''
        endHotelScore=ReferenceScore[Trip[lastIndex]]
        Score=endHotelScore
        ScoreList=[0,endHotelScore]
        '''Cost Initialization'''
        endHotelCost=Price[Trip[lastIndex]]
        Cost=endHotelCost
        CostList=[0,endHotelCost]
    else:
        '''Score Initialization'''
        Score=0
        ScoreList=[0,0]
        '''Cost Initialization'''
        Cost=0
```

```
        CostList=[0,0]
    IndexCounter=1
    ShouldApply=False
    for node in Trip[1:LunchIndex+1]:
        TW = tempTimeWindow[IndexCounter-
1]+TravelTime.cell(row=5+Trip[IndexCounter-
1],column=5+node).value+TimeSpend[node]
        TW,twPassFlag,compensate = oneNodeTWCalculation(TW,o,c,node)
        if twPassFlag:
            tempTimeWindow.append(TW)
            tempCompensatelist.append(compensate)
        else:
            break
        if IndexCounter!=len(Trip)-1:
            ScoreList.insert(IndexCounter,ReferenceScore[node])
            CostList.insert(IndexCounter,Price[node])
            Score+=ReferenceScore[node]
            Cost+=Price[node]
        IndexCounter+=1
    tempTotalCompensate=CalculateCompensate(tempCompensatelist)
    if len(Trip)==len(tempTimeWindow):
        ShouldApply=True
    return
tempTimeWindow,Score,Cost,ScoreList,CostList,tempCompensatelist,tempTotalCom
pensate,ShouldApply

def
CalculateGeneralWValidationAll(Trip,TimeWindow,Price,ReferenceScore,TravelTi
me,TimeSpend,o,c,currentDay,dayInTour):
    tempTimeWindow=[TimeWindow[0]]
    tempCompensatelist=[0]
    if currentDay != (dayInTour-1):
        lastIndex=len(Trip)-1
        '''Score Initialization'''
        endHotelScore=ReferenceScore[Trip[lastIndex]]
        Score=endHotelScore
        ScoreList=[0,endHotelScore]
        '''Cost Initialization'''
        endHotelCost=Price[Trip[lastIndex]]
        Cost=endHotelCost
        CostList=[0,endHotelCost]
    else:
        '''Score Initialization'''
        Score=0
        ScoreList=[0,0]
        '''Cost Initialization'''
        Cost=0
```

```
        CostList=[0,0]
    IndexCounter=1
    Response=1
    ShouldApply=False
    for node in Trip[1:]:
        TW = tempTimeWindow[IndexCounter-
1]+TravelTime.cell(row=5+Trip[IndexCounter-
1],column=5+node).value+TimeSpend[node]
        TW,twPassFlag,compensate = oneNodeTWCalculation(TW,o,c,node)
        if twPassFlag:
            tempTimeWindow.append(TW)
            tempCompensatelist.append(compensate)
        else:
            break
        if IndexCounter!=len(Trip)-1:
            ScoreList.insert(IndexCounter,ReferenceScore[node])
            CostList.insert(IndexCounter,Price[node])
            Score+=ReferenceScore[node]
            Cost+=Price[node]
        IndexCounter+=1
    tempTotalCompensate=CalculateCompensate(tempCompensatelist)
    if len(Trip)==len(tempTimeWindow):
        ShouldApply=True
        Response=0
    return
tempTimeWindow,Score,Cost,ScoreList,CostList,tempCompensatelist,tempTotalCom
pensate,ShouldApply,Response

def CostValidation(Tour,Budget,dayInTour):
    GrandTotalCost=0
    EnoughBudget=False
    for day in range(0,dayInTour):
        TotalCost=Tour[7+10*day]
        GrandTotalCost+=TotalCost
    if GrandTotalCost<=Budget:
        EnoughBudget=True
    return EnoughBudget

'''def timeWindowCalculation(TW,Compensatelist,o,c,i):
    twPassFlag=False
    if TW >= o[i]:
        Compensatelist.append(0)
    else:
        Compensatelist.append(o[i]-TW)
        TW = o[i]
    if TW<=c[i]:
        twPassFlag=True
```

```
    else:
        twPassFlag=False
    return TW,twPassFlag,Compensatelist'''

def
timeWindowCalculation(startTime,startIndex,targetIndex,startvalue,targetValue,Trip,t
t,t,IsStart):
    if IsStart:
        TW=startTime+tt.cell(row=5+Trip[startIndex-
1],column=5+startvalue).value+t[startvalue]
    elif startIndex == targetIndex-1:

TW=startTime+tt.cell(row=5+Trip[startIndex],column=5+targetValue).value+t[target
Value]
    else:

TW=startTime+tt.cell(row=5+Trip[startIndex],column=5+Trip[startIndex+1]).value+t
[Trip[startIndex+1]]
    startIndex=startIndex+1
    if startIndex<targetIndex:

TW=timeWindowCalculation(TW,startIndex,targetIndex,startvalue,targetValue,Trip,t
t,t,False)
    return TW

def preCalPreProInA(Compensatelist):
    MiniTime=TimeWindow[:-1]
    TW = MiniTime[len(MiniTime[:-1])]+tt.cell(row=5+Trip[len(Trip[:-
2])],column=5+i).value+t[i]
    twPassFlag,Compensatelist=timeWindowCalculation(TW,Compensatelist,o,c,i)
    '''x = s[i]/(tt.cell(row=5+Trip[len(Trip)-2],column=5+i).value +
tt.cell(row=5+i,column=5+Trip[len(Trip)-1]).value)
        if Ratio < x:
            MiniTime=TimeWindow[:-1]
            TW = MiniTime[len(MiniTime[:-1])]+tt.cell(row=5+Trip[len(Trip[:-
2])],column=5+i).value+t[i]
            if o[i]<=TW<=c[i] and Cost+pr[i]<=Budget:'''
```

APPENDIX H
COMPARISON BETWEEN TOP-TOP AND TOP-BOTTOM POPULATION MANAGEMENT APPROACH

Test case	Approach	Score	Budget		Computation time	Number of day	Number of	Variable (Cplex)	
			Limit	Total				Binary	Other
1	Top-Top	30.12	15000	670.5	5.05	1	2636	2550	58
	Top-Bottom	31.42		685.50	5.10				
	% difference								
2	Top-Top	34.31	15000	310.00	12.02	1	9902	10100	122
	Top-Bottom	33.31		310.00	11.33				
	% difference								
3	Top-Top	36.14	15000	780.00	17.00	1	22052	22650	182
	Top-Bottom	36.14		780.00	16.89				
	% difference								
4	Top-Top	35.14	15000	730.00	23.45	1	31933	32942	220
	Top-Bottom	36.14		780.00	22.25				
	% difference								
5	Top-Top	58.45	15000	3,516.00	6.89	2	5236	5100	116
	Top-Bottom	60.00		3,416.00	6.52				
	% difference								
6	Top-Top	65.54	15000	1,865.00	14.58	2	19746	20200	244
	Top-Bottom	65.54		1,865.00	13.34				
	% difference								
7	Top-Top	78.50	15000	3,100.00	23.58	2	44016	45300	364
	Top-Bottom	76.05		2,937.00	24.75				
	% difference								
8	Top-Top	72.89	15000	3,500.00	37.85	2	63761	65884	440
	Top-Bottom	76.49		3,336.00	35.94				
	% difference								
9	Top-Top	84.79	15000	5,545.00	15.85	3	7836	7650	174
	Top-Bottom	82.97		4,867.00	12.59				
	% difference								
10	Top-Top	93.48	15000	9,550.50	25.49	3	29590	30300	366
	Top-Bottom	96.55		10,155.50	28.52				
	% difference								
11	Top-Top	112.64	15000	5,455.25	51.48	3	65980	67950	546
	Top-Bottom	109.82		4,713.00	54.45				
	% difference								
12	Top-Top	107.14	15000	4,958.00	72.28	3	95589	98826	660
	Top-Bottom	112.81		5,247.00	74.17				
	% difference								
13	Top-Top	95.87	15000	6,115.00	14.87	4	10436	10200	232
	Top-Bottom	98.16		6,319.00	13.59				
	% difference								
14	Top-Top	128.50	15000	6,251.00	35.45	4	39434	40400	488
	Top-Bottom	123.20		5,966.00	32.69				
	% difference								
15	Top-Top	135.48	15000	5,945.55	68.47	4	87944	90600	728
	Top-Bottom	143.18		5,622.50	75.55				
	% difference								
16	Top-Top	145.04	15000	7,800.50	100.20	4	127417	131768	880
	Top-Bottom	144.04		7,509.50	98.96				
	% difference								

BIOGRAPHY

Name	Mr. Apisit Cheng
Date of Birth	October 1, 1993
Education	2016: Bachelor of Science (Engineering Management) Sirindhorn International Institute of Technology Thammasat University