Name: Teh Shao Yin Title: ModeFair Assessment

**Task 1**

|  |  |
| --- | --- |
| Code |  |
| Sample Output/Result |  |

For task 1, I have chosen to implement a Depth First Search (DFS) tree that perform searches starting from the root and explores as far as possible along each branch until the deepest node is visited where there are no more sibling nodes. Then, backtracking is performed to move back to the parent node and explore its sibling nodes. During the traversal, filtering conditions in *if* statement aided in finding letter combinations that complied with all given rules. The filtering criteria is guided using two hashmaps: *link* and *skip.* Additionally, the code will check if these elements are seen before in the combination: [B, E, H, D, F]. If any one of them are seen, the code will update the *link* hashmap to allow skipping of used element. As explored, all returned patterns in the list complied with all rules stated in the requirements: 7 elements, “AIC” points, connectivity patterns, and no repeating elements.

**Task 2 (Objective 1)**

|  |  |
| --- | --- |
| Code |  |
| Sample Output/Result |  |

For task 2, some adjustments were made to these variables: *POP\_SIZE, SELECTION\_SIZE, ELITE\_SIZE, NUM\_GENS, MUTATION\_RATE, CROSSOVER\_RATE, and fitness.* The main reason for adjustment of these variables (except *fitness*) is to ensure genetic diversity and perseverance of best solutions to the next generation, preventing potential premature convergence, leading to best local optima instead of best global optima. On the other hand, *fitness* function is redefined by substituting the linear function to an exponential function, that could introduce higher selection pressure by creating a larger gap between high-performing and low-performing individuals. Upon the adjustment, it can be observed that all requirements are fulfilled. The cost of the best route is reduced to RM117.04, which is lesser than RM120. Hard constraints and soft constraints are satisfied, with compliance with the assumptions made.

**Task 2 (Objective 2)**

|  |  |
| --- | --- |
| Code |  |
| Output |  |

Since the original code does not work well with individuals that have overloaded demand, that part of the code is modified, by removing the original break statement to prevent infeasible solution. Instead, the code is modified to constantly check if there are still available capacity to handle the next demand. If no capacity is found, the *full* will be flagged as true and the for loop will move on to the next vehicle and continue with loading that vehicle with demand (refer to line 360 to 362). This could solve the problem faced by the original code, in which it can’t process overloaded demand. With this introduced into the code, it faces another problem, which is skipping too many demands/customers. This is because skipping more customers will yield lower costs, which is highly favourable and does not satisfy one of the requirements: **Satisfy as many demands as possible*.*** Hence, the *empty­\_space* list (line 335) is introduced to record the total amount of capacity within each vehicle that was unfilled. The values within the list are summed, multiplied by 5, and added to the *total\_cost* (line 383)*,* thereby punishing individuals with high amount of unfilled capacity/demand. Furthermore, to satisfy the requirements of only 2 vehicles (1 type A, 1 type B), the *num\_cars* variable is adjusted to 2. As shown in the output snippet, it shows that all vehicle satisfied the assumption made in the question, including two vehicles only. Furthermore, as many demands as possible is fulfilled by all vehicles without overshooting their available capacity. Finally, the output shows a solution that can fulfil higher number of demands with lesser costs.