**Report**

Balanced Staff Routing for Maintenance

**I. Problem description**

There exist N customers where customer i resides in location i (i= 1,2,3,...,N) that demand Internet maintenance service. The maintenance service of the customer i lasts for d[i] (time unit). There are K employees departing from one single depot (denoted by 0). The traveling time between two locations i and j (i,j = 0,1,...,N) is t[i,j]. The objective is to schedule a plan in which the maximum working time (equals to the traveling time plus the maintenance time) among the employees is minimized.

**II.Modelling**

* Denote:

- **D** = [d[1], d[2], … , d[N]] is a vector containing the maintenance time of every customer

- **T =** [T[i,j] for i, j in {0,1,...,N}] is a 2D-array that represent the traveling time between location i and location j.

* Decision variable:

- **x[i,k]** = 1, nếu khách hàng i có trong lịch trình bảo trì của nhân viên k; **X[i,k]** = 0, nếu ngược lại.

(*với 1 <= i <= N; 1 <= k <= K)*

- **y[i,j,k]** = 1, nếu quãng đường di chuyển từ i đến j có trong lịch trình bảo trì của nhân viên k; **Y[i,j,k]** = 0, nếu ngược lại.

(*với 0 <= i,j <= N ; 1 <= k <= K*)

- **c[k]:** Thời gian làm việc của nhân viên k

(*với 1 <= k <= K*)

* Constraints

- Ràng buộc giữa x, y, c:

c[k] =

- K nhân viên đều phải xuất phát từ trụ sở 0:

- K nhân viên đều phải quay về trụ sở 0:

- Đảm bảo có đúng 1 cạnh vào nốt i và 1 cạnh ra khỏi nốt i trong một lịch trình:

- Mỗi khách hàng chỉ được bảo trì bởi đúng 1 nhân viên:

- Loại bỏ các hành trình con không hợp lệ:

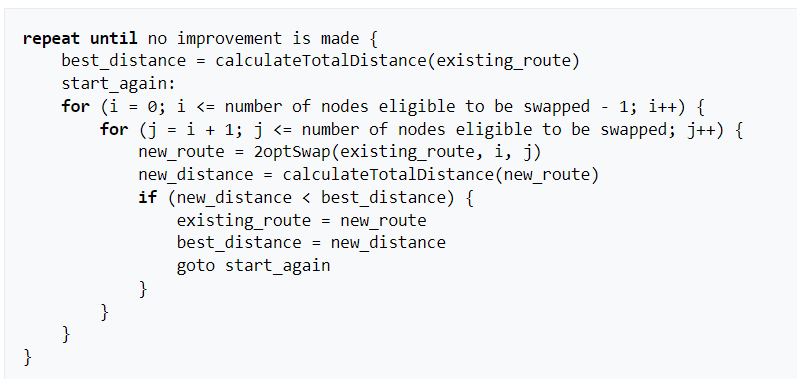
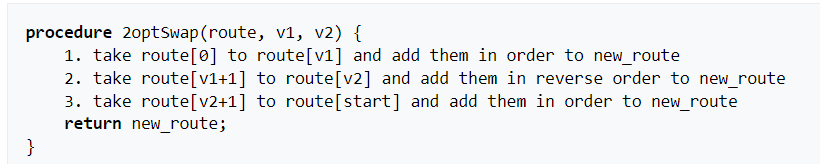
* Objective function:

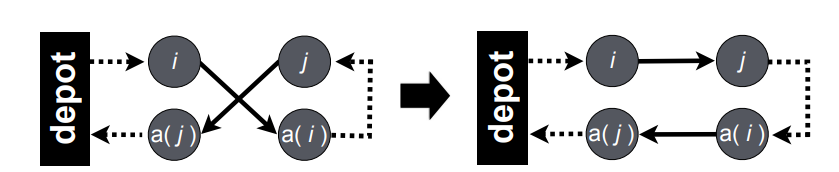
**III. Proposed Algorithms**

**Algorithm 1:** Greedy (Quý)

* The idea for the first algorithm is that we initialize K points from the set of N locations, except for the depot point 0, as centroids, then construct for each a return route between the point and the depot.

* Next, consider in turn each unrouted point and insert it in a route that produces the least increase of the MinMax objective function, while updating the lists of the remaining unrouted points.
* Right after each insertion of a node, we applied a 2-opt procedure to optimize the traveling cost of each tour. The 2-opt procedure could be briefly described as follow





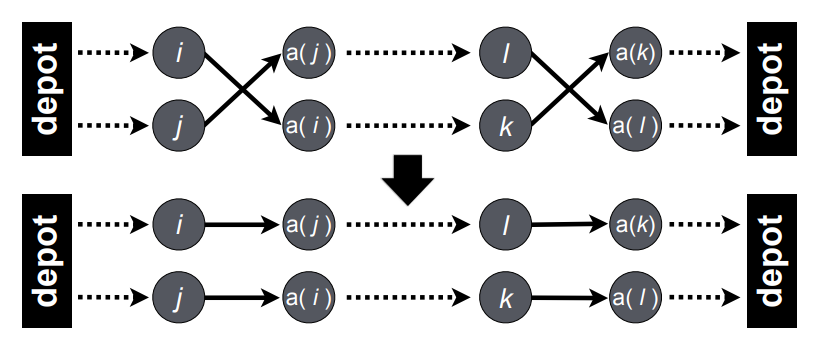
*2-opt procedure*

* Both the insertion and post-optimization process were executed sequentially and the process stopped when all points in the unrouted point list were routed.
* We tried processing the algorithm with the unsorted and sorted list of unrouted point and noticed that the the sorted by maintenance time in decreasing order returned the least bias among the values of the objective function of each iteration.
* Computational result indicates that this algorithm require impressively low execution time and return good or even optimal solution when N and K are small.

**Algorithm 2:** Local Search (Minh)

- After trying several times with greedy, we realized that greedy has some weakness. In more detail, there is the existence of bad solutions. So, we came up with the local search algorithm as another way to look for a better solution.

- The 2-opt algorithm exchanges two paths for other two paths in the same tour. In addition, the CROSS-exchange randomly exchanges two cities from 2 tours in one tour and a partial tour of the other tours.



*CROSS-exchange procedure*

- Local search methods move from a current solution to a better solution in neighborhoods of the current solution until an optimal or a local optimal solution is found. In the solution, we create two operators: Move (Move a city from the tour has maximum cost to the tour has minimum cost) and Swap (Swap randomly two cities from maximum tours and minimum tours) to find the best neighborhood. However, in general, local search methods cannot find optimal solutions due to the local minimum problem.

**Algorithm 3**: Integer Linear Programming (Quyên)

The Branch and Bound algorithm is a tree search algorithm used for solving optimization problems. It is a combination of depth-first search and backtracking. The algorithm works by dividing the problem into smaller sub-problems and finding their optimal solution. The algorithm starts by defining the root node of the tree which represents the original problem. The algorithm then generates children nodes by branching on the decision variables. The optimal solution is found by visiting the nodes in the tree in a depth-first manner and backtracking when a solution cannot be found in a particular node.

The problem can be formulated as an Integer Linear Programming (ILP) problem and solved using Branch and Bound algorithms. The objective function is to minimize the maximum working time among the employees, which can be written as:

minimize z = max {Σd[i]\*x[i,k] + Σt[i,j]\*y[i,j,k]}

In the above formula:

* x[i,k] = 1 if customer i is in the maintenance schedule of employee k; x[i,k] = 0 otherwise. (with 1 <= i <= N; 1 <= k <= K)
* y[i,j,k] = 1 if the travel distance from i to j is in the maintenance schedule of employee k; y[i,j,k] = 0 otherwise. (with 0 <= i,j <= N ; 1 <= k <= K)
* **d[i]** is a vector containing the maintenance time of every customer
* t[i,j] is a 2D-array that represents the traveling time between location i and location j.

In addition, to find the optimal solution, the solver employs a combination of branching, cutting plane algorithms, and heuristics. Branching is used to divide the problem into smaller subproblems, while cutting plane algorithms and heuristics help to refine the solution.

**IV. Experiments**

We ran tests with data instances with gradually increasing N, K and the average number of customers A = [N / K], each ran 50 times.

\* Exact algorithms:

| **N = 5, K = 2** | | | **N = 10, K = 2** | | |
| --- | --- | --- | --- | --- | --- |
|  | f | t(s) |  | f | t(s) |
| ILP | 360 | 0.03 | ILP | 550 | 6.49 |
| **N = 50, K = 5** | | | **N = 50, K = 10** | | |
|  | f | t(s) |  | f | t(s) |
| ILP | 804 | 32.89 | ILP | 528 | 41.23 |
| **N = 100, K = 10** | | | **N = 100, K = 20** | | |
|  | f | t(s) |  | f | t(s) |
| ILP | TiME OUT | TIME OUT | ILP | TIME OUT | TIME OUT |
| **N = 200, K = 10** | | | **N = 200, K = 20** | | |
|  | f | t(s) |  | f | t(s) |
| ILP | TIME OUT | TIME OUT | ILP | TIME OUT | TIME OUT |

\* Heuristic algorithms:

| **N = 10, K = 2** | | | | | |
| --- | --- | --- | --- | --- | --- |
|  | f\_min | f\_max | f\_avg | std\_dev | t\_avg (s) |
| Greedy | 550 | 580 | 564.3 | 5.4 | 0.0007 |
| Local search | 550 | 560 | 558 | 4.4 | 8.9 |

| **N = 50, K = 5** | | | | | |
| --- | --- | --- | --- | --- | --- |
|  | f\_min | f\_max | f\_avg | std\_dev | t\_avg (s) |
| Greedy | 812 | 914 | 864.1 | 18.3 | 0.004 |
| Local search | 841 | 860 | 852.1 | 6.84 | 33.75 |

| **N = 50, K = 10** | | | | | |
| --- | --- | --- | --- | --- | --- |
|  | f\_min | f\_max | f\_avg | std\_dev | t\_avg (s) |
| Greedy | 528 | 598 | 562.04 | 16.04 | 0.006 |
| Local search | 535 | 557 | 549.1 | 7.1 | 9.18 |

| N = 100, K = 10 | | | | | |
| --- | --- | --- | --- | --- | --- |
|  | f\_min | f\_max | f\_avg | std\_dev | t\_avg (s) |
| Greedy | 3390 | 4190 | 3738.1 | 68.01 | 0.022 |
| Local search | 3710 | 3890 | 3811.1 | 59.04 | 36.67 |

| N = 100, K = 20 | | | | | |
| --- | --- | --- | --- | --- | --- |
|  | f\_min | f\_max | f\_avg | std\_dev | t\_avg (s) |
| Greedy | 2231 | 2712 | 2417.4 | 89.67 | 0.04 |
| Local search | 2487 | 2565 | 2515 | 24.75 | 9.78 |

| N = 200, K = 10 | | | | | |
| --- | --- | --- | --- | --- | --- |
|  | f\_min | f\_max | f\_avg | std\_dev | t\_avg (s) |
| Greedy | 4700 | 5870 | 5114.6 | 184.49 | 0.063 |
| Local search | 5230 | 5380 | 5328.89 | 43.43 | 262.38 |

| N = 200, K = 20 | | | | | |
| --- | --- | --- | --- | --- | --- |
|  | f\_min | f\_max | f\_avg | std\_dev | t\_avg (s) |
| Greedy | 3600 | 4230 | 3881.6 | 102.82 | 0.107 |
| Local search | 3900 | 4100 | 3898 | 49.48 | 250.68 |

| N = 400, K = 40 | | | | | |
| --- | --- | --- | --- | --- | --- |
|  | f\_min | f\_max | f\_avg | std\_dev | t\_avg (s) |
| Greedy | 3500 | 4030 | 3729.5 | 98.96 | 0.793 |
| Local search | 3700 | 3910 | 3891.2 | 47.89 | 350.23 |

| N = 500, K = 50 | | | | | |
| --- | --- | --- | --- | --- | --- |
|  | f\_min | f\_max | f\_avg | std\_dev | t\_avg (s) |
| Greedy | 3530 | 4070 | 3767.3 | 110.58 | 1.617 |
| Local search | 3640 | 3810 | 3712.2 | 98.22 | 450.23 |

| N = 1000, K = 100 | | | | | |
| --- | --- | --- | --- | --- | --- |
|  | f\_min | f\_max | f\_avg | std\_dev | t\_avg (s) |
| Greedy | 3500 | 4000 | 3783.8 | 92.09 | 12.084 |
| Local search | 3620 | 3810 | 3798.1 | 56.66 | 650.23 |

| N = 1000, K = 200 | | | | | |
| --- | --- | --- | --- | --- | --- |
|  | f\_min | f\_max | f\_avg | std\_dev | t\_avg (s) |
| Greedy | 2780 | 3148 | 2914.7 | 45.62 | 33.74 |
| Local search | 2810 | 3120 | 2915.5 | 21.53 | 672.53 |

**V. Conclusion**

Overall, all the aforementioned algorithms could give rather acceptable or even optimal solutions. But the performance is different between these algorithms.

* The first algorithm gives feasible solutions in the shortest time and of quite good quality. The bias of this algorithm is that the convergence rate of this algorithm depends much on the random seed\_customer, and the failure chance gets higher when the average number of customers assigned to each employee gets larger.
* The second algorithm takes longer to return the feasible solution, but the difference among the objective values of each iteration is narrowed, and it could return the optimal solution given enough time. The weakness of this problem is that it depends highly on the quality of the initial feasible solution.
* The third algorithm: When N and K are small, the ILP algorithm returns the optimal solution in the shortest time. However, when N and K become enormous, it takes a long time to return to the solution. (explicitly, when N > 200 and K > 20, the algorithm

In conclusion, method using Integer Linear Programming gives the best performance when the data size is small. However, in reality, the method

using Greedy algorithm would be recommended due its fast performance and good feasible solution - even though they may not be optimal. The Local Search method