# Sports team schedule

Table of Contents



## Background

There are N teams 1, 2, ..., N that need to be scheduled to play against each other in  a round-robin format: each team must meet all the remaining N - 1 teams twice (one at their home stadium and once at their opponents) in a period of 2N - 2 weeks, every week each team must play exactly 1 match. Distance from the stadium of team i to the stadium of team j is d(i, j). Make a schedule for N teams so that the total distance traveled by all teams is the shortest.

Input:

- Line 1: positive even integer N
- Line i + 1 (i = 1, 2, ..., N): i-th row of matrix d

## Understanding

Variables:

- N: the number of teams.
- Distance matrix d(i, j): weight matrix of undirected graph. It is a two-dimensional list, in which row i and column j is the distance from the stadium of team i to the stadium of team j.
- Schedule matrix x(i, j): a matrix stored by a two-dimensional list. The index of each row is the index of the home team, and the index of each column is the index of the guest team. Row i and column j is the week that the match between 2 teams is scheduled. The initial states of this matrix (week 0) in an empty matrix, and the final states of this matrix (last week) is the OUTPUT matrix.

Constraints:

- N: the number of teams is a positive even integer. The size of the distance matrix and the schedule matrix is N x N.
- The distance from the stadium of team i to itself d(i, i) is 0 or the main diagonal of the distance matrix is all 0.
- A team cannot play against itself. We set the week that a team plays against itself x(i, i) is week 0 or the main diagonal of the schedule matrix is all 0.
- Each team must meet all the remaining N - 1 teams twice, and every week each team must play exactly one match so the domain of the week in the schedule matrix is {0, 1, 2, ..., 2N - 2}. Except week 0, the number of matches each week is N / 2.
- Every week each team must play exactly one match. So all entries in row i (week that team i play at their home stadium) and column i (week that team i play at their opponents) are pairwise different.

Objective function:

- Total distance traveled by all teams is minimized

## Algorithms

Backtrack:

- Constructing a graph of the states of the tournament. Each node represents the schedule matrix and appropriate information of the tournament (as cost, current position of each team,etc...)
- Define a “get_sub_node” function to get all neighbors of a node. The main idea is we need to construct the schedule for next week of the tournament.

```
def backtrack_solve(input: list):
    N = len(input)
    queue = []
    # tracking the best value in the tree
    best_value = float('inf')

    best_config = None
    ##
    # initialize the root of the tree.
```

```python
root_matrix = [[None for i in range(N)] for i in range(N)]
for i in range(N):
    root_matrix[i][i] = 0
root_dis = 0
week = 0
root_tracking_pos = [i for i in range(N)]
root = Node(root_matrix, root_dis, week, root_tracking_pos)
queue.append(root)
while queue:
    temp = queue.pop()
    assert isinstance(temp, Node)

    current_week = temp.week
    if current_week == 2*N - 2:
        if temp.dis < best_value:
            best_config = temp.matrix
            best_value = temp.dis
            print(f"Found better configuration, the new best value is: {best_value}")
            print(f"The new best config is {best_config}")
            print()

    else:
        next_week = current_week + 1
        list_of_nodes = get_sub_node(input, temp)
        for item in list_of_nodes:
            sub_matrix, sub_pos, sub_dis = item
            if sub_dis < best_value:
                sub_Node = Node(sub_matrix, sub_dis, next_week, sub_pos)
                queue.append(sub_Node)
```

Beam search:

- Keeping k lowest nodes in terms of distance cost, and using same model as Backtrack (generalized Greedy)

```python
def beam_solve(input,k):
   N = len(input)
   queue = []
   ## tracking the best value in the tree
   c_min = find_min(input)

   ##
   ##initialize the root of the tree.
   root_matrix = [[None for i in range(N)] for i in range(N)]
   for i in range(N):
       root_matrix[i][i] = 0
   root_dis = 0
   week = 0
   root_tracking_pos = [i for i in range(N)]
   root = Node(root_matrix,root_dis,week,root_tracking_pos)
   queue.append(root)
   best = None

   while True:
       nodes = copy.deepcopy(queue)
       queue = []
       if nodes[0].week == 2*N - 2:
           for node in nodes:
               if best == None:
                   best = node.matrix,node.dis
               else:
                   if node.dis < best[1]:
                       best = node.matrix,node.dis
           break
       else:
           next_week = nodes[0].week + 1
           for node in nodes:
               lis = get_sub_node(input,node)
               sub_nodes = [Node(sub_matrix,sub_dis,next_week,sub_pos) for \
                   (sub_matrix,sub_pos,sub_dis) in lis]
               queue += sub_nodes
           queue = sorted(queue,key=lambda x: x.dis)[:k]

   return best
```

Hill climbing (Local search):

- Starting at a random complete assignment, defining a get_neighbors function to get all neighbor solutions of the current feasible solution, then keep moving if we can find better neighbor assignment

- The all neighbors of a solution is defined by:
  - Swapping the values of any 2 squares in the board, which accord to some teams,  and then check for any repetitions in those teams, if there are, fix those repetitions, and keep swapping and checking like that until no repetition is found.
  - Swapping 2 schedules of 2 teams in the board (swapping 2 crosses)

```
def local_solve(input):
    # random initial
    #input: distance matrix
    #return the schedule: matrix , cost
    N = len(input)
    random_matrix = generate(N)
    cost = compute_cost(input,random_matrix)
    initialize = random_matrix,cost
    flag = False
    while flag:
        neighbors = get_neighbors(initialize)
        for neighbor in neighbors:
            #neighbor: matrix:schedule,cost
            schedule,cost = neighbor
            flag = False
            if cost < initialize[1]:
                initialize = neighbor
                flag = True
    return initialize[0],initialize[1]
```

Iterated local search:

- Applying local search multiple times with random initializations.

```
def iterative_local_solve(input,K):
    best_config = None
    best_cost = float('inf')
    for i in range(K):
        matrix,cost = local_solve(input)
        if cost < best_cost:
            best_cost = cost
            best_config = matrix

    return best_config,best_cost
```

Gibbs sample:

- For each assignment, consider its neighbor assignments, then put the distance cost for the assignment. The higher cost, the lower probability we go into this assignment.

```
def gibb_sampling(input,K,factor):
    best_config = None
    best_cost = float('inf')
    #initialize start position
    mat = generate(len(input))
    current_postion = (mat,compute_cost(distance_matrix,mat))
    #

    for i in range(K):
        #choosing a random number:
        num = random.random()

        list_of_neighs = get_neighbors(current_postion) #mat,cost
        sum_of_cost = sum(1/factor**x[1] for x in list_of_neighs)
        accumulate = 0
        for i in range(len(list_of_neighs)):
            mat,cost = list_of_neighs[i]
            accumulate += 1/factor ** cost
            port = accumulate / sum_of_cost
            if port > num:
                current_postion = list_of_neighs[i-1]
                break
    return current_postion
```

OR-Tools:

- For this problem, we use the CP-SAT solver from OR-Tools. By adding intermediate Boolean variables for computation of the cost function, the function becomes a linear one with constant coefficients.
- In detail, if we use our normal way of computing the cost, that means finding the order of the coordinates in each team schedule that has according incremental values, and then based on this order, calculate the cost, we are hindered from implementing the problem into the OR-Tools solver. Therefore, we had to change our method of calculating the objective function.

- For each team, we ordered the coordinates (or boxes) in its schedule. The first one will be box (i,i), then all the boxes of the according row, from left to right, and finally all the boxes of the according column, from top to bottom. The first box represents the first week, the row ones represent away weeks, with the destination can be found as the column coordinate, and the column ones represent the home weeks. We indexed these boxes with values from 0 to 2*N - 1.
- And then we added in, for each of the teams, (2*N - 1)^2 intermediate variables, which can be represented as a square matrix Y with dimension (2*N - 1) x (2*N - 1). If the week_value of some box i is equal to some week_value of box j + 1, then Y[i][j] = 1, and else, for other cases, Y[i][j] = 0. Because of our coordinate ordering, the meaning of each of the boxes is predetermined, and therefore just by multiplying each intermediate variable with an according coefficient taken from the distance_matrix, and summing all of them, we have our objective function, calculated in a way that is suitable for implementing into the CP-SAT solver.

## Results

We experiment our algorithms with cases of N = 4 to N = 18. Our results are shown below:

| **N = 4** | | Exact Algorithms | | | | | | Heuristic Algorithms | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | | | Meta-heuristic Algorithms | | | | | | | |
| | | Backtracking | | Beam Search (K = 10) | | OR-Tools (time_limit = 30) | | Local Search | | | | | Iterative Local Search (K = 20) | | Gibb Sampling (K = 20, factor = 10) | | | | | |
| No. | Input (distance_matrix) | Cost | Time | Cost | Time | Cost | Time | Cost_min | Cost_max | Cost_avg | SD | Time_avg | Cost | Time | Cost_min | Cost_max | Cost_avg | SD | Time_avg |
| 1 | [[0, 14, 8, 10], [14, 0, 2, 16 | **130** | 1.337977 | **130** | 0.022821 | **130** | 0.252533437 | 147 | 175 | 161.5 | 8.834591 | **0.005652** | 135 | 0.116170112 | 142 | 178 | 165.8 | 12.65543 | 0.075056 |
| 2 | [[0, 11, 6, 8], [11, 0, 13, 15 | **120** | 1.001075 | **120** | 0.015474 | **120** | 0.326545889 | 131 | 157 | 142.3 | 6.972087 | **0.006046** | **120** | 0.110396001 | 142 | 172 | 158 | 8.5557 | 0.075752 |
| 3 | [[0, 2, 10, 3], [2, 0, 1, 1], [1 | **46** | 0.335697 | 55 | 0.016643 | **46** | 0.130370849 | 60 | 94 | 76.8 | 11.241 | **0.006147** | 56 | 0.115875799 | 73 | 109 | 92.6 | 13.41044 | 0.075326 |
| 4 | [[0, 6, 19, 6], [6, 0, 6, 6], [1 | **115** | 0.628411 | 145 | 0.015848 | **115** | 0.224878667 | **115** | 166 | 145.3 | 16.45023 | **0.006464** | 121 | 0.11391803 | 139 | 207 | 171.7 | 20.1 | 0.074667 |
| 5 | [[0, 8, 11, 18], [8, 0, 6, 12], | **106** | 0.632101 | 108 | 0.016006 | **106** | 0.290010993 | 135 | 155 | 141.9 | 7.049113 | **0.006841** | **106** | 0.122274279 | 133 | 175 | 153.9 | 13.31503 | 0.074361 |
| 6 | [[0, 19, 0, 3], [19, 0, 15, 19 | **129** | 0.936859 | 135 | 0.015913 | **129** | 0.274870287 | 133 | 145 | 137.4 | 3.746999 | **0.006334** | **129** | 0.118231739 | 134 | 204 | 160.9 | 20.77715 | 0.073388 |
| 7 | [[0, 6, 19, 17], [6, 0, 14, 0], | **115** | 0.601652 | 149 | 0.016072 | **115** | 0.291891062 | 126 | 166 | 150.4 | 12.15895 | **0.006413** | 121 | 0.11198837 | 144 | 195 | 172.3 | 14.21302 | 0.07422 |
| 8 | [[0, 7, 13, 4], [7, 0, 3, 3], [1 | **77** | 0.582724 | 94 | 0.015744 | **77** | 0.485994128 | **77** | 125 | 100.2 | 15.92985 | **0.006235** | **77** | 0.200260892 | 89 | 132 | 117.4 | 12.72164 | 0.073296 |
| 9 | [[0, 8, 11, 10], [8, 0, 5, 7], [ | **125** | 0.893962 | 127 | 0.015564 | **125** | 0.142331376 | **125** | 169 | 145.6 | 12.45151 | **0.006063** | 137 | 0.198194362 | 158 | 188 | 172 | 10.45945 | 0.076564 |
| 10 | [[0, 14, 7, 0], [14, 0, 10, 0], | **54** | 0.619548 | 77 | 0.01556 | **54** | 0.261213993 | 65 | 87 | 75.4 | 7.2 | **0.005745** | 57 | 0.204744692 | **54** | 88 | 76.8 | 8.930845 | 0.075748 |

| **N = 6** | | Exact Algorithms | | | | Heuristic Algorithms | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | Meta-heuristic Algorithms | | | | | | |
| | | Beam Search (K = 10) | | OR-Tools (time_limit = 30) | | Local Search | | | | | Iterative Local Search (K = 20) | | Gibb Sampling (K = 20, factor = 2) | | | | |
| No. | Input (distance_matrix) | Cost | Time | Cost | Time | Cost_min | Cost_max | Cost_avg | SD | Time_avg | Cost | Time | Cost_min | Cost_max | Cost_avg | SD | Time_avg |
| 1 | [[0, 14, 11, 16, 0, 6], [14, 0 | 248 | 1.010156 | **209** | 30.10935119 | 274 | 352 | 323.5 | 21.32422 | **0.058649** | 289 | 2.382435296 | 311 | 378 | 339 | 16.76637 | 0.817953 |
| 2 | [[0, 12, 8, 5, 5, 9], [12, 0, 9 | 247 | 0.928515 | **239** | 30.10478216 | 312 | 402 | 368.8 | 24.25238 | **0.056628** | 330 | 1.831483955 | 345 | 426 | 386.9 | 24.59652 | 0.838233 |
| 3 | [[0, 2, 11, 19, 4, 5], [2, 0, 1 | 187 | 0.899488 | **154** | 30.1192878 | 252 | 323 | 284.1 | 21.2208 | **0.054073** | 245 | 1.402985274 | 276 | 347 | 302.9 | 24.42881 | 0.84939 |
| 4 | [[0, 15, 3, 4, 3, 3], [15, 0, 9 | 180 | 1.666258 | **172** | 30.11800174 | 235 | 300 | 272.8 | 20.66021 | **0.06176** | 237 | 1.266083146 | 262 | 317 | 288 | 19.78215 | 0.832203 |
| 5 | [[0, 19, 4, 11, 17, 3], [19, 0 | 197 | 1.576084 | **161** | 30.1061077 | 261 | 365 | 315.5 | 33.67244 | **0.134521** | 260 | 1.484359646 | 281 | 350 | 329.5 | 22.01136 | 0.830625 |
| 6 | [[0, 3, 19, 6, 10, 1], [3, 0, 1 | 253 | 0.923506 | **234** | 30.11793164 | 284 | 359 | 328.5 | 23.00845 | **0.17658** | 304 | 1.317068501 | 310 | 377 | 350.5 | 20.72706 | 0.838203 |
| 7 | [[0, 17, 5, 18, 6, 9], [17, 0, | 241 | 0.941948 | **239** | 30.09339636 | 324 | 408 | 372.8 | 24.97465 | **0.077387** | 329 | 1.270587156 | 360 | 505 | 411.2 | 47.84651 | 0.838135 |
| 8 | [[0, 12, 16, 14, 3, 15], [12, | 338 | 0.947344 | **331** | 30.10977547 | 439 | 517 | 477.2 | 24.37576 | **0.057718** | 414 | 1.620045844 | 438 | 523 | 488 | 30.37543 | 0.808251 |
| 9 | [[0, 9, 6, 7, 12, 6], [9, 0, 1, | **150** | 0.986155 | 151 | 30.11102091 | 225 | 264 | 249.4 | 12.34864 | **0.054407** | 206 | 2.605445335 | 225 | 272 | 247.4 | 13.75338 | 0.827346 |
| 10 | [[0, 0, 5, 19, 17, 16], [0, 0, | 204 | 0.950292 | **186** | 30.12871928 | 312 | 369 | 346.2 | 17.86244 | **0.067287** | 310 | 1.404102484 | 310 | 388 | 350.8 | 24.85871 | 0.816618 |

| **N = 8** | | Exact Algorithms | | | | Heuristic Algorithms | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | Meta-heuristic Algorithms | | | | | | |
| | | Beam Search (K = 10) | | OR-Tools (time_limit = 60) | | Local Search | | | | | Iterative Local Search (K = 20) | | Gibb Sampling (K = 20, factor = 2) | | | | |
| No. | Input (distance_matrix) | Cost | Time | Cost | Time | Cost_min | Cost_max | Cost_avg | SD | Time_avg | Cost | Time | Cost_min | Cost_max | Cost_avg | SD | Time_avg |
| 1 | [[0, 0, 2, 0, 15, 14, 17, 11], | **313** | 125.4899 | 362 | 60.2479353 | 564 | 640 | 595.7 | 27.44712 | **0.346815** | 514 | 6.613484005 | 537 | 652 | 602 | 30.98028 | 4.723227 |
| 2 | [[0, 15, 14, 1, 6, 18, 7, 6], [ | **444** | 132.6496 | 449 | 60.2686597 | 678 | 765 | 727.2 | 29.70896 | **0.351719** | 619 | 7.657409222 | 672 | 829 | 732.8 | 53.3933 | 4.70142 |
| 3 | [[0, 1, 8, 12, 17, 17, 13, 14 | **344** | 125.9711 | 390 | 60.270849 | 611 | 711 | 661.7 | 35.22641 | **0.437301** | 617 | 6.867997543 | 639 | 738 | 686.5 | 41.1832 | 4.734649 |
| 4 | [[0, 15, 18, 19, 4, 0, 16, 19 | 425 | 125.5147 | **392** | 60.4682847 | 560 | 748 | 675.9 | 51.72459 | **0.327517** | 642 | 7.848738293 | 679 | 760 | 724.5 | 25.0211 | 4.69486 |
| 5 | [[0, 9, 4, 4, 10, 0, 17, 11], [ | **378** | 123.1026 | 399 | 60.4871099 | 656 | 729 | 697.4 | 26.77561 | **0.347123** | 595 | 6.987632892 | 635 | 744 | 699.3 | 31.09859 | 4.715565 |
| 6 | [[0, 13, 2, 3, 14, 4, 6, 2], [1 | **337** | 127.7918 | 342 | 60.2801782 | 601 | 724 | 666.3 | 34.51264 | **0.469154** | 584 | 7.068864354 | 617 | 778 | 662.4 | 56.47654 | 4.713051 |
| 7 | [[0, 3, 13, 4, 19, 1, 5, 19], [ | **366** | 133.4441 | 440 | 60.273295 | 770 | 820 | 792.7 | 15.88885 | **0.329167** | 726 | 8.195604503 | 724 | 880 | 803.7 | 49.7439 | 4.68765 |
| 8 | [[0, 0, 0, 8, 3, 11, 7, 14], [0 | **437** | 125.0109 | 453 | 60.2498519 | 679 | 797 | 732.7 | 32.51683 | **0.365796** | 697 | 6.944989925 | 689 | 801 | 758.5 | 33.23068 | 4.698643 |
| 9 | [[0, 13, 17, 4, 5, 18, 9, 7], [ | **338** | 126.5925 | 412 | 60.5750086 | 715 | 829 | 782.4 | 37.05312 | **0.35498** | 686 | 7.815878786 | 701 | 834 | 753.4 | 49.19169 | 4.672189 |
| 10 | [[0, 19, 8, 2, 5, 3, 1, 1], [19 | 402 | 122.4059 | **366** | 60.2692796 | 619 | 705 | 670 | 26.8121 | **0.51653** | 579 | 6.588294417 | 620 | 704 | 662.4 | 31.22036 | 4.920971 |

| **N = 10** | | Exact Algorithms | | Heuristic Algorithms | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | Meta-heuristic Algorithms | | | | | | |
| | | OR-Tools (time_limit = 120) | | Local Search | | | | | Iterative Local Search (K = 20) | | Gibb Sampling (K = 20, factor = 1.5) | | | | |
| No. | Input (distance_matrix) | Cost | Time | Cost_min | Cost_max | Cost_avg | SD | Time_avg | Cost | Time | Cost_min | Cost_max | Cost_avg | SD | Time_avg |
| 1 | [[0, 19, 6, 12, 14, 0, 4, 18, | **854** | 120.5396489 | 1191 | 1330 | 1269 | 44.22669 | **1.324301** | 1207 | 50.65744615 | 1248 | 1398 | 1313.2 | 47.29059 | 18.94755 |
| 2 | [[0, 17, 17, 8, 0, 18, 0, 16, | **739** | 120.5897861 | 1107 | 1210 | 1150.4 | 28.9029 | **1.352366** | 1058 | 41.92160708 | 1054 | 1216 | 1135.5 | 54.86195 | 18.83109 |
| 3 | [[0, 15, 13, 11, 11, 13, 11, | **988** | 120.5168863 | 1419 | 1596 | 1517.9 | 57.88773 | **1.382398** | 1412 | 42.81292845 | 1430 | 1620 | 1530.7 | 53.83937 | 18.82224 |
| 4 | [[0, 13, 12, 2, 14, 18, 19, 1 | **884** | 120.6480511 | 1283 | 1448 | 1359.1 | 48.47783 | **1.325354** | 1264 | 42.208624 | 1278 | 1414 | 1347.1 | 38.23451 | 18.67534 |
| 5 | [[0, 13, 17, 0, 1, 14, 3, 15, | **714** | 120.525107 | 1032 | 1240 | 1141.1 | 67.79126 | **1.328594** | 1089 | 41.20659101 | 1097 | 1308 | 1201.1 | 66.39854 | 18.79935 |
| 6 | [[0, 4, 6, 12, 5, 5, 8, 17, 11, | **908** | 120.9153499 | 1308 | 1458 | 1370.4 | 49.25489 | **1.358444** | 1293 | 41.92893219 | 1362 | 1480 | 1419.8 | 32.20352 | 18.42685 |
| 7 | [[0, 7, 15, 0, 19, 4, 11, 13, | **920** | 120.9027252 | 1320 | 1427 | 1385.3 | 35.8486 | **1.38664** | 1283 | 41.8134089 | 1256 | 1459 | 1349.3 | 64.49126 | 18.59945 |
| 8 | [[0, 0, 18, 5, 8, 15, 7, 16, 1 | **963** | 120.5460515 | 1343 | 1562 | 1441.1 | 66.23938 | **1.383823** | 1360 | 42.03538163 | 1322 | 1506 | 1409.5 | 52.80415 | 18.84176 |
| 9 | [[0, 19, 5, 9, 13, 14, 3, 7, 1, | **913** | 120.5844218 | 1224 | 1349 | 1279.9 | 46.64869 | **1.334151** | 1166 | 42.48218277 | 1227 | 1368 | 1303.1 | 43.44972 | 18.9689 |
| 10 | [[0, 2, 13, 6, 11, 18, 19, 7, | **767** | 120.528994 | 1223 | 1349 | 1293.5 | 40.47564 | **1.329337** | 1201 | 43.86289846 | 1207 | 1350 | 1297.6 | 41.69785 | 18.39978 |

| **N = 12** | | Exact Algorithms | | Heuristic Algorithms | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | Meta-heuristic Algorithms | | | | | | |
| | | OR-Tools (time_limit = 150) | | Local Search | | | | | Iterative Local Search (K = 20) | | Gibb Sampling (K = 20, factor = 1.25) | | | | |
| No. | Input (distance_matrix) | Cost | Time | Cost_min | Cost_max | Cost_avg | SD | Time_avg | Cost | Time | Cost_min | Cost_max | Cost_avg | SD | Time_avg |
| 1 | [[0, 3, 4, 13, 9, 11, 2, 1, 11, | **1062** | 150.9121057 | 1582 | 1847 | 1733 | 70.3199 | **4.274181** | 1594 | 136.1658315 | 1643 | 2029 | 1791.3 | 108.7761 | 58.98845 |
| 2 | [[0, 17, 9, 18, 17, 5, 5, 10, | **1383** | 150.9176916 | 1795 | 1970 | 1882.6 | 47.56796 | **4.209649** | 1768 | 128.913341 | 1808 | 2002 | 1873 | 66.9245 | 59.48496 |
| 3 | [[0, 8, 16, 6, 3, 14, 12, 17, | **1280** | 150.9812279 | 1706 | 1840 | 1760.5 | 39.32274 | **4.173995** | 1621 | 128.66995 | 1740 | 1916 | 1815.5 | 67.90884 | 59.85225 |
| 4 | [[0, 10, 6, 14, 17, 1, 8, 12, | **1369** | 150.9684659 | 1824 | 2059 | 1961.6 | 62.55877 | **4.165353** | 1844 | 128.4017569 | 1889 | 2080 | 1998.1 | 70.30955 | 59.15007 |
| 5 | [[0, 13, 5, 16, 12, 7, 19, 4, | **1072** | 151.6469485 | 1495 | 1684 | 1584.9 | 66.45374 | **4.273927** | 1518 | 129.0815929 | 1472 | 1786 | 1626.3 | 105.8438 | 59.27507 |
| 6 | [[0, 0, 17, 2, 3, 11, 16, 2, 2, | **1245** | 151.6277501 | 1838 | 2049 | 1931.7 | 69.78387 | **4.233463** | 1750 | 129.3147388 | 1789 | 1962 | 1892.4 | 60.34199 | 59.13124 |
| 7 | [[0, 7, 3, 16, 1, 0, 7, 2, 19, | **1311** | 151.5670786 | 1768 | 2040 | 1869 | 79.69804 | **4.179716** | 1699 | 130.3951209 | 1762 | 1980 | 1832.5 | 63.03835 | 59.54058 |
| 8 | [[0, 3, 12, 10, 5, 1, 5, 9, 2, | **1163** | 150.9558213 | 1525 | 1664 | 1607 | 46.03863 | **4.257278** | 1546 | 128.1993256 | 1572 | 1743 | 1643.2 | 57.20684 | 59.03462 |
| 9 | [[0, 14, 1, 1, 6, 10, 10, 16, | **1432** | 151.70122 | 1889 | 2098 | 1985.9 | 66.67075 | **4.146802** | 1835 | 130.1112017 | 1840 | 2040 | 1958.9 | 63.63167 | 57.98654 |
| 10 | [[0, 16, 10, 3, 0, 18, 18, 17, | **1247** | 150.959891 | 1656 | 1898 | 1772.4 | 80.59942 | **4.145123** | 1656 | 127.177093 | 1750 | 1926 | 1826.7 | 62.33966 | 58.50409 |

| N = 14 | | Exact Algorithms | | Heuristic Algorithms | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | Meta-heuristic Algorithms | | | | | | | |
| | | OR-Tools (time_limit = 150) | | Local Search | | | | | Iterative Local Search (K = 20) | | Gibb Sampling (K = 20, factor = 1.125) | | | | | |
| No. | Input (distance_matrix) | Cost | Time | Cost_min | Cost_max | Cost_avg | SD | Time_avg | Cost | Time | Cost_min | Cost_max | Cost_avg | SD | Time_avg |
| 1 | [[0, 11, 6, 19, 13, 6, 10, 15, | **1604** | 151.52007 | 2043 | 2309 | 2186.9 | 83.62542 | **11.20461** | 1992 | 347.2234184 | 2041 | 2324 | 2219.3 | 85.98714 | 153.5752 |
| 2 | [[0, 17, 9, 19, 0, 3, 19, 3, 4, | **1844** | 65.77477764 | 2346 | 2627 | 2455 | 81.1665 | **11.24238** | 2287 | 343.413633 | 2229 | 2616 | 2363.1 | 121.2055 | 151.4988 |
| 3 | [[0, 7, 3, 13, 13, 4, 7, 0, 18, | **1436** | 151.513077 | 1988 | 2117 | 2065.6 | 45.4024 | **11.10672** | 1828 | 346.3244733 | 1950 | 2104 | 2033.3 | 47.20887 | 151.8805 |
| 4 | [[0, 6, 8, 6, 15, 15, 14, 5, 2, | **1810** | 152.6558536 | 2407 | 2569 | 2497.2 | 61.52109 | **11.1866** | 2335 | 343.9190943 | 2372 | 2648 | 2485 | 90.70832 | 153.9645 |
| 5 | [[0, 11, 13, 14, 1, 10, 6, 11, | **1518** | 151.5319507 | 2185 | 2381 | 2288.7 | 63.97925 | **11.23335** | 2118 | 343.3807498 | 2179 | 2398 | 2249.8 | 71.25354 | 154.7751 |
| 6 | [[0, 17, 6, 19, 9, 0, 3, 10, 3, | **1866** | 151.5089474 | 2513 | 2732 | 2606.3 | 74.80798 | **11.0537** | 2377 | 344.119521 | 2381 | 2690 | 2520 | 88.56009 | 155.5073 |
| 7 | [[0, 4, 16, 4, 6, 11, 3, 3, 2, | **1815** | 151.9208296 | 2281 | 2498 | 2414.5 | 64.83698 | **11.22015** | 2320 | 342.1507165 | 2298 | 2542 | 2415.3 | 72.93688 | 157.0348 |
| 8 | [[0, 3, 8, 9, 13, 10, 11, 13, | **1635** | 151.570418 | 2104 | 2369 | 2220.6 | 90.33788 | **11.2178** | 2097 | 343.0403545 | 2101 | 2410 | 2255.5 | 106.569 | 155.9096 |
| 9 | [[0, 4, 4, 7, 2, 7, 7, 19, 6, 0, | **1972** | 151.9373676 | 2533 | 2716 | 2640.1 | 57.54892 | **11.08803** | 2517 | 345.0125803 | 2501 | 2681 | 2597.1 | 58.39606 | 151.7655 |
| 10 | [[0, 18, 8, 4, 8, 9, 5, 12, 1, | **1962** | 151.5614681 | 2475 | 2657 | 2560.3 | 70.79242 | **11.13924** | 2412 | 342.3906829 | 2396 | 2659 | 2515.5 | 86.74644 | 151.6127 |

| N = 16 | | Exact Algorithms | | Heuristic Algorithms | | | | |
|---|---|---|---|---|---|---|---|---|
| | | OR-Tools (time_limit = 150) | | Local Search | | | | |
| No. | Input (distance_matrix) | Cost | Time | Cost_min | Cost_max | Cost_avg | SD | Time_avg |
| 1 | [[0, 16, 2, 19, 10, 0, 10, 11, | **2578** | 152.288767 | 3230 | 3536 | 3362.9 | 107.1743 | **26.34445** |
| 2 | [[0, 11, 12, 5, 17, 15, 15, 1 | **2372** | 152.77251 | 3040 | 3268 | 3130.8 | 75.54807 | **26.338** |
| 3 | [[0, 16, 3, 10, 9, 5, 1, 4, 14, | **2482** | 152.439697 | 3259 | 3449 | 3346.6 | 57.92562 | **25.5058** |
| 4 | [[0, 17, 14, 17, 2, 6, 14, 4, | **2731** | 152.320037 | 3365 | 3701 | 3471.4 | 112.1202 | **25.49031** |
| 5 | [[0, 17, 12, 14, 6, 1, 2, 7, 1 | **2270** | 153.297634 | 2814 | 3003 | 2913.7 | 65.54227 | **25.51691** |
| 6 | [[0, 11, 6, 12, 7, 12, 2, 0, 1 | **2277** | 152.292395 | 2778 | 3185 | 2997.5 | 124.8699 | **25.50795** |
| 7 | [[0, 13, 8, 7, 2, 3, 11, 4, 9, | **2458** | 152.322567 | 3139 | 3355 | 3264.8 | 70.39697 | **25.51071** |
| 8 | [[0, 11, 15, 0, 14, 17, 13, 1 | **2657** | 153.44855 | 3188 | 3533 | 3322.8 | 105.1188 | **25.50811** |
| 9 | [[0, 7, 4, 15, 4, 12, 0, 7, 12, | **2716** | 152.254155 | 3227 | 3432 | 3292.5 | 62.99956 | **25.39263** |
| 10 | [[0, 17, 1, 4, 15, 11, 12, 19, | **2425** | 152.259414 | 3118 | 3416 | 3230 | 85.18216 | **25.49691** |

| N = 18 | | Exact Algorithms | | Heuristic Algorithms | | | | |
|---|---|---|---|---|---|---|---|---|
| | | OR-Tools (time_limit = 150) | | Local Search | | | | |
| No. | Input (distance_matrix) | Cost | Time | Cost_min | Cost_max | Cost_avg | SD | Time_avg |
| 1 | [[0, 1, 19, 5, 12, 18, 0, 14, | **3345** | 154.4567192 | 3917 | 4190 | 4069.8 | 99.68261 | **87.82674** |
| 2 | [[0, 0, 7, 5, 5, 8, 10, 1, 1, 1 | **3522** | 153.8324763 | 3900 | 4203 | 4084.6 | 95.81951 | **87.72683** |
| 3 | [[0, 18, 6, 6, 18, 15, 0, 7, 1 | **3682** | 153.312624 | 4238 | 4618 | 4424.8 | 125.7933 | **87.47296** |
| 4 | [[0, 4, 7, 17, 13, 16, 19, 13, | **3908** | 153.2661388 | 4320 | 4732 | 4529.5 | 140.4685 | **87.82188** |
| 5 | [[0, 6, 11, 15, 10, 8, 11, 6, | **3174** | 154.4719492 | 3820 | 4053 | 3930 | 68.75237 | **87.27406** |
| 6 | [[0, 14, 0, 7, 13, 1, 19, 19, | **3525** | 154.4091918 | 4020 | 4277 | 4149 | 83.36799 | **87.25857** |
| 7 | [[0, 9, 1, 10, 3, 15, 16, 1, 1 | **3302** | 153.2680487 | 4019 | 4382 | 4188.3 | 119.0267 | **87.62574** |
| 8 | [[0, 5, 19, 18, 14, 2, 18, 3, | **3776** | 153.3741745 | 4448 | 4681 | 4587 | 80.65427 | **87.26774** |
| 9 | [[0, 15, 8, 6, 19, 14, 6, 16, | **3386** | 154.3056025 | 3899 | 4173 | 4014.4 | 96.7611 | **83.39593** |
| 10 | [[0, 11, 2, 8, 10, 4, 17, 6, 1 | **3393** | 154.3556985 | 3782 | 4155 | 3962 | 119.4534 | **83.23291** |

Because the solving time for backtracking, iterative local search, and gibb sampling wasn't feasible, so accordingly, we only tested for N <= 4; N <=14; N<= 14.

As shown, for N = 4, OR-Tools and backtracking do always give the optimal solution, while beam-search occasionally does. For N >= 6, no algorithm of these could find the optimal solution.

In terms of cost, for N = 6, OR-Tools outperforms the other algorithms on most of the tests, with some exceptions from beam-search finding a better solution. As N increases, solutions from the heuristic algorithms, compared to OR-Tools feasible solution with a time_limit (as shown), are mostly bad, and worse over time.

In terms of solving time, local search did best, but its cost is not acceptable compared to OR-Tools' or Iterative local search's ones. However, as we could see, the time it takes for our local search algorithm to solve with different data are comparatively the same, so we think our get_neighbors function has some flaws, and therefore could be improved in the future.

So, based on the collected data, we recommend using OR-Tools for getting a feasible solution to this problem.

## Future works

Improve the performance of backtracking and beam search by optimizing the get_sub_node function.
Improve our heuristic algorithms by improving our get_neighbors function.
Implement Integer programming solver of OR-Tools.

## List of tasks

Group 20 – Topic 19

1. **Trần Quốc Đệ 20210179** de.tq210179@sis.hust.edu.vn
    - Code (50%)
    - Presentation (70%)
    - Report (15%)
2. **Trần Văn Toàn 20214932** toan.tv214932@sis.hust.edu.vn
    - Code (50%)
    - Presentation (30%)
    - Report (15%)
3. **Vũ Nhật Minh 20214919** minh.vn214919@sis.hust.edu.vn
    - Report (45%)
    - Collection and compilation of data (50%)
    - Slide (50%)
4. **Đỗ Mạnh Cường 20214948** cuong.dm214948@sis.hust.edu.vn
    - Report (25%)
    - Collection and compilation of data (50%)
    - Slide (50%)