

Algoritmi di Ottimizzazione

Agent Scheduling Problem

Solved with an
Integer Quadratic Programming
Model

Francesco Ghinelli
Matr: 150117

1 Agent Scheduling Problem

This isn't the classic Traveling Salesman Problem with Time Windows because there are more constraints to respect. Those constraints are:

- 30 minutes for lunch between 12:00 AM and 2:00 PM;
- Limited working time (8 hours);
- Limited waiting and traveling time;
- Minimum time to spent in office (1 hour);

Furthermore, the problem requires to:

- Maximize production activities,
- Prioritize meetings,
- Minimize travel,
- Encourage large blocks of equal work (minimize changes between office work and meetings).

1.1 Approach to the problem

To solve this problem in an easier way, I've calculated a distance matrix, representing distance between all different positions.

To solve the problem of multiple admissible routes to the office from the same customer, I used variables that contain double the positions actually needed. In fact, the positions will start with that relating to the office, followed by those relating to the various customers, then there is a position that would be empty, which has been used as a fictitious starting and ending point, finally there are a quantity of positions equal to number of customers used to represent a journey from the customer to the office.

This approach is needed to keep a simpler model creating circular and continuous agent paths.

Data

for this model is:

- $clients \rightarrow$ Clients count.
- $distance_{i,j}$, for i, j in $clients + 1$. \rightarrow Distance matrix.
- $work_start \rightarrow$ Agents working day start.
- $work_end \rightarrow$ Agents working day end.
- $lunch_start \rightarrow$ Agents lunch time window start.
- $lunch_end \rightarrow$ Agents lunch time window end.
- $window_start_i$, for i in $clients + 1$. \rightarrow Office/clients service window start.
- $window_end_i$, for i in $clients + 1$. \rightarrow Office/clients service window end.

Variables

for this model are:

- $x_{i,j,a} \in \{0,1\}$, $i \in ALL$, $j \in ALL$, $a \in A \rightarrow$ Representing agent a doing path between i and j .

- $s_{i,a} \in \mathbb{N}, i \in ALL, a \in A \rightarrow$ Representing service time of agent a to client i .
- $c_{i,a} \in \mathbb{N}, i \in ALL, a \in A \rightarrow$ Representing agent a service lasting to client i .
- $w_{i,a} \in \mathbb{N}, i \in ALL, a \in A \rightarrow$ Representing agent a wait time before client i .
- $l_{i,j,a} \in \{0,1\}, i \in ALL, j \in ALL, a \in A \rightarrow$ Representing agent a doing lunch between i and j .
- $t_a \in \mathbb{N}, a \in A \rightarrow$ Representing max service time of agent a .

Objective function

Objective function for this model want to minimize agents traveling time, wait time and service time. First two are obvious, last one is needed to keep wait time near to the client that require them.

Objective function use two multiplicative coefficients applied to traveling times and to wait times to give more importance to one or the other.

Constraints

for this model are:

1. Serve all clients.
2. All duplicated Offices must be visited from at most an Agent.
3. All Agents start their trip from fictitious location.
4. Agents can't start their trip from fictitious location and go to only destination Office.
5. All Agents end their trip in fictitious location.
6. Each served location has a served location before
7. Agent can't do loop between same Client.
8. Sum of time spent into a location in a specific trip must be equal to Service time of that location.
9. Sum of time spent into Office in a specific trip must be equal to Office service time.
10. Agent can go to a location only to spend time.
11. Agent must go office at least once or start from it [0].
12. Agent can't go office from office.
13. That is an only start Office position (Not fictitious).
14. Agent can go to (fictitious) Office only if he visited the associated Client.
15. Agent can go to (fictitious) Office just after he visit the associated Client.
16. Agent serve Client after his time window start.
17. Agent serve Client before his time window end.
18. If Agent has lunch it must be after lunch time start.
19. If Agent has lunch it must be before lunch time end.
20. If Agent have lunch between Clients I and J, agent have to make trip between I and J.
21. Getting maximum Agent service time.
22. If Agent working time is greater than lunch time than Agent must have lunch.

23. If Agent working time is lesser than lunch time than Agent must not have lunch.
24. Service time of Client J is equal to sum of: wait time of client J, service time of client I before J, trip between I and J, time of service of client I, lunch time if present.
25. Sum of minutes spent in: travels, servicing clients, waiting, eating at lunch.

2 Model

$$\min \text{travel_cost} * \sum_{i \in \text{ALL}} \sum_{j \in \text{ALL}} \sum_{a \in A} x_{i,j,a} * \text{distance}_{i,j} + \text{wait_cost} * \sum_{i \in C} \sum_{a \in A} w_{i,a} + \sum_{i \in \text{RBLE}} \sum_{a \in A} s_{i,a}$$

1. $\sum_{i \in \text{ALL}} \sum_{a \in A} x_{i,j,a} = 1, \quad j \in C$
 2. $\sum_{i \in \text{ALL}} \sum_{a \in A} x_{i,j,a} \leq 1, \quad j \in \text{FO}$
 3. $\sum_{j \in \text{REAL}} x_{\text{start},j,a} = 1, \quad a \in A$
 4. $x_{\text{start},j,a} = 0, \quad j \in \text{FO}, a \in A$
 5. $\sum_{i \in \text{RBLE}} x_{i,s,a} = 1, \quad a \in A$
 6. $\sum_{i \in \text{ALL}} x_{i,h,a} - \sum_{j \in \text{ALL}} x_{h,j,a} = 0, \quad h \in \text{ALL}, a \in A$
 7. $x_{i,i,a} = 0, \quad i \in \text{ALL}, a \in A$
 8. $c_{j,a} = \text{service}_j * \sum_{i \in \text{ALL}} x_{i,j,a}, \quad i \in C, a \in A$
 9. $\sum_{i \in \text{ALL}} \sum_{j \in o+\text{FO}} c_{j,a} * x_{i,j,a} \geq \text{service}_o, \quad a \in A$
 10. $c_{j,a} \geq x_{i,j,a}, \quad i \in \text{ALL}, j \in \text{RBLE}, a \in A$
 11. $\sum_{i \in \text{ALL}} \sum_{i \in o+\text{FO}} x_{i,j,a} \geq 1, \quad a \in A$
 12. $x_{i,j,a} = 0, \quad i \in o + \text{FO}, j \in o + \text{FO}, a \in A$
 13. $x_{i,o,a} = 0, \quad i \in C, a \in A$
 14. $x_{j,j+\text{clients},a} \leq \sum_{i \in \text{ALL}} x_{i,j,a}, \quad j \in C, a \in A$
 15. $\{x_{i,j,a} = 0 \text{ if } i \neq j - \text{clients}\}, \quad i \in \text{REAL}, j \in \text{FO}, a \in A$
 16. $\text{window_start}_i * \sum_{h \in \text{ALL}} x_{h,i,a} \leq s_{i,a}, \quad i \in C, a \in A$
 17. $\text{window_end}_i * \sum_{h \in \text{ALL}} x_{h,i,a} \geq s_{i,a}, \quad i \in C, a \in A$
 18. $x_{i,j,a} * (s_{i,a} + c_{i,a} + \text{distance}_{i,j} + w_{j,a}) \geq l_{i,j,a} * \text{lunch_start}, i \in \text{RBLE}, j \in \text{RBLE}, a \in A$
 19. $l_{i,j,a} * (s_{i,a} + c_{i,a} + \text{distance}_{i,j} + w_{j,a}) \leq x_{i,j,a} * \text{lunch_end}, i \in \text{RBLE}, j \in \text{RBLE}, a \in A$
 20. $l_{i,j,a} \leq x_{i,j,a} * \text{lunch_end}, \quad i \in \text{RBLE}, j \in \text{RBLE}, a \in A$
 21. $t_a = \max(s_{i,a}, i \in \text{RBLE}), \quad a \in A$
 22. $\text{work_end} * \sum_{i \in \text{RBLE}} \sum_{j \in \text{RBLE}} l_{i,j,a} \geq t_a - \text{lunch_start}, \quad a \in A$
 23. $\text{work_end} * \sum_{i \in \text{RBLE}} \sum_{j \in \text{RBLE}} (1 - l_{i,j,a}) \geq \text{lunch_start} - t_a, \quad a \in A$
 24. $s_{j,a} = \sum_{i \in \text{RBLE}, i \neq j} x_{i,j,a} * (s_{i,a} + c_{i,a} + \text{distance}_{i,j} + l_{i,j,a} * \text{lunch_len}) + w_{j,a}, \quad j \in \text{RBLE}, a \in A$
 25. $\sum_{i \in \text{RBLE}} \sum_{j \in \text{RBLE}} x_{i,j,a} * \text{distance}_{i,j} + \sum_{i \in \text{ALL}} \sum_{j \in \text{RBLE}} x_{i,j,a} * \text{service}_j + \sum_{i \in \text{RBLE}} w_{i,a} + \sum_{i \in \text{RBLE}} \sum_{j \in \text{RBLE}} l_{i,j,a} * \text{lunch_len} + \text{work_start} \leq \text{work_end}, \quad a \in A$
- $x_{i,j,a} \in \{0,1\}, i \in \text{ALL}, j \in \text{ALL}, a \in A$
 $s_{i,a} \in \mathbb{N}, i \in \text{ALL}, a \in A$
 $c_{i,a} \in \mathbb{N}, i \in \text{ALL}, a \in A$
 $w_{i,a} \in \mathbb{N}, i \in \text{ALL}, a \in A$
 $l_{i,j,a} \in \{0,1\}, i \in \text{ALL}, j \in \text{ALL}, a \in A$

$$t_a \in \mathbb{N}, a \in A$$

$$start = clients + 1$$

$$o = 0$$

$$C = \{1, \dots, clients\}$$

$$ALL = \{0, \dots, 2clients + 1\}$$

$$REAL = \{0, \dots, clients\}$$

$$RBLE = ALL - \{s\}$$

$$FO = \{P + 2, \dots, 2clients + 1\}$$

3 Model implementation using Gurobi

3.0.1 Importing libraries

```
[1]: from gurobipy import*

import math
import matplotlib.pyplot as plt
import networkx as nx
import numpy as np
import pandas as pd
import pickle
```

3.0.2 Define file with clients data

```
[2]: FILE = "./TEST_SETS/test_2.txt"
```

3.0.3 Define constant parameters

```
[3]: # Default params
SUPPORTED_FORMAT = ['NUM', 'X', 'Y', 'DEMAND', 'READYTIME', 'DUEDATE', 'SERVICE']
# Macros for time values conversions
MINUTES = 60
HOURS = 3600
OFFSET_TIMES = 8*HOURS

COLUMNS_OPS = {'NUM': lambda x: float(x),
                'X': lambda x: float(x),
                'Y': lambda x: float(x),
                'DEMAND': lambda x: 1,
                'READYTIME': lambda x: float(x),
                'DUEDATE': lambda x: float(x),
                'SERVICE': lambda x: float(x)
               }

# Agents count
AGENTS = 3

# Agent Working day start and end
WORKING_TIME_RANGE = (0, 8*HOURS)

# Agent Lunch break time range, lasting
LUNCH_BREAK_RANGE = (12*HOURS-OFFSET_TIMES, 13.5*HOURS-OFFSET_TIMES)
LUNCH_BREAK_TIME = 30*MINUTES

# Agents office parameters
OFFICE_NUM = 0
OFFICE_X = .0
OFFICE_Y = .0
OFFICE_READYTIME = WORKING_TIME_RANGE[0]
OFFICE_DUEDATE = WORKING_TIME_RANGE[1]
```

```
OFFICE_SERVICE = 1*HOURS
```

3.0.4 Read clients data

```
[4]: def read_input_tsptw(filename):  
    """ Function used to convert input file to usable data.  
        :params filename: File to convert,  
        :return: A dict with nodes parameters,  
                A distance matrix between nodes,  
                Nodes coordinates.  
    """  
  
    # Dict sed for locations parameters  
    data_dict = dict()  
  
    # List of node positions for plots  
    nodes_x = list()  
    nodes_y = list()  
  
    # Add office to data  
    data_dict.update({OFFICE_NUM: {'X': OFFICE_X,  
                                   'Y': OFFICE_Y,  
                                   'DEMAND': AGENTS,  
                                   'READYTIME': OFFICE_READYTIME,  
                                   'DUE DATE': OFFICE_DUE DATE,  
                                   'SERVICE': OFFICE_SERVICE,}})  
  
    # Add office to nodes  
    nodes_x.append(OFFICE_X)  
    nodes_y.append(OFFICE_Y)  
  
    # Open file and read lines  
    with open(filename, "r") as file:  
        # Initialize columns in empty dict  
        columns = file.readline().replace("#", "").split()  
        if columns != SUPPORTED_FORMAT:  
            print("ERROR! Format not supported.")  
            return  
  
        # For each data line  
        for line in file.readlines():  
            node_dict = {k: COLUMNS_OPS[k](val) for k, val in zip(columns, line.split())}  
            # Get id  
            node_id = node_dict.pop('NUM')  
            # Insert new node in data dict  
            data_dict.update({int(node_id): node_dict})  
            # Get nodes positions  
            nodes_x.append(float(line.split()[columns.index('X')]))  
            nodes_y.append(float(line.split()[columns.index('Y')]))  
  
    # Get distance matrix  
    distance_matrix = compute_distance_matrix(nodes_x, nodes_y)
```



```

return (data_dict, distance_matrix, dict(enumerate(zip(nodes_x, nodes_y))))

def compute_distance_matrix(nodes_x, nodes_y):
    """ Function used to compute the euclidean distance matrix.
        :param nodes_x: List of nodes x coordinates,
        :param nodes_y: List of nodes y coordinates,
        :return: Distance matrix between nodes. """
    # Get clients count and initialize distance matrix
    clients = len(nodes_x)
    distance_matrix = [[None for i in range(clients)] for j in range(clients)]
    for i in range(clients):
        # Set cost of trip between same agent and himself as null
        distance_matrix[i][i] = 0
        for j in range(clients):
            # Compute distance matrix calculating euclidean distance between each node
            dist = compute_dist(nodes_x[i], nodes_x[j], nodes_y[i], nodes_y[j])
            distance_matrix[i][j] = dist
            distance_matrix[j][i] = dist
    return distance_matrix

def compute_dist(xi, xj, yi, yj):
    """ Function used to compute euclidean distance.
        :param xi: x coordinate of first node,
        :param xj: x coordinate of second node,
        :param yi: y coordinate of first node,
        :param yj: y coordinate of second node,
        :return: Euclidean distance between nodes.
        """
    exact_dist = math.sqrt(math.pow(xi - xj, 2) + math.pow(yi - yj, 2))
    return int(math.floor(exact_dist + 0.5))

```

```

[5]: # Getting locations parameters
data_dict, distance_matrix, positions = read_input_tsptw(FILE)

```

3.1 Gurobi Model

3.1.1 Setup model parameters

```
[6]: # DEBUG RESTRICTIONS
CLIENTS = len(data_dict)
data_dict = {k: v for k,v in data_dict.items() if k < CLIENTS}
distance_matrix = [dm[:CLIENTS] for dm in distance_matrix[:CLIENTS]]

# ADD FICTITIOUS LOCATION
# This location is used to have a complete loop in Agent trips without interfering
# with trips costs. Having a complete loop simplify the job of creating a trip.
# To not interfere with costs it's distance to all other locations is 0.
distance_matrix = [dm + [0,] for dm in distance_matrix]
distance_matrix = distance_matrix + [[0]*(CLIENTS+1)]
# Add location data
data_dict.update({CLIENTS: {'X': 0,
                              'Y': 0,
                              'DEMAND': AGENTS,
                              'READYTIME': WORKING_TIME_RANGE[0],
                              'DUE DATE': WORKING_TIME_RANGE[1],
                              'SERVICE': 0,}})

# POSITIONS SETS FOR CLEANER MODEL
agent_list = list(range(AGENTS))
all_pos = list(range(CLIENTS*2))
start_pos = CLIENTS
client_pos = list(range(1,CLIENTS))
only_start_office_pos = 0
no_duplicates_pos = list(range(0,CLIENTS))
destination_office_pos = list(range(CLIENTS+1, CLIENTS*2))
office_pos = [only_start_office_pos]+destination_office_pos
reachable_pos = [p for p in range(CLIENTS*2) if p not in [start_pos,]]
```

3.1.2 Add Multipliers to minimize wait time over travel time

```
[7]: # Multiplier for distance and wait costs
TRAVEL_COST_MULTIPLIER = 1
WAIT_COST_MULTIPLIER = 2
```

3.1.3 Create the model

```
[8]: # Create model
mod = Model("TSPTW")
```

Warning: your license will expire in 11 days

Academic license - for non-commercial use only - expires 2021-08-11
Using license file /opt/gurobi/gurobi.lic

3.1.4 Variables

IMPORTANT! Office IS ONE AND ONLY ONE but because i wasn't unable to find a better solution for multiple Agent visiting it I've repeated Office (originally in position 0) in all position greater than number_of_clients + 1.

WARNING! Position from number_of_clients + 2 forward are office positions reachable only from client_pos = office_pos - number_of_clients.

WARNING! Position number_of_clients + 1 is used as fictitious start and end trip loop location, position 0 is office too, Agents can only start from office 0 they can't go there.

```
[9]: # Agent trip
x = mod.addVars({(i,j,a): 0 for i in all_pos
                    for j in all_pos
                    for a in agent_list},
                name="x",
                vtype=GRB.BINARY)

# Serve time
s = mod.addVars({(i,a): 0 for i in all_pos
                    for a in agent_list},
                name="s",
                vtype=GRB.INTEGER)

# Serve Client/Office lasting
c = mod.addVars({(i,a): 0 for i in all_pos
                    for a in agent_list},
                name="c",
                vtype=GRB.INTEGER)

# Wait time
w = mod.addVars({(i,a): 0 for i in all_pos
                    for a in agent_list},
                name="w",
                vtype=GRB.INTEGER,
                lb=0)

# Lunch done between customers
l = mod.addVars({(i,j,a): 0 for i in all_pos
                    for j in all_pos
                    for a in agent_list},
                name="l",
                vtype=GRB.BINARY)

# Max working time
t = mod.addVars({(a): 0 for a in agent_list},
                name="t",
                vtype=GRB.INTEGER)
```

3.1.5 Constraints

Trip constraints

```
[10]: # 1 - All client must be visited from an Agent
      _= mod.addConstrs((quicksum(x[i,j,a]
                                for i in all_pos
                                for a in agent_list) == 1
                                for j in client_pos),
                        name="ServeAll")

      # 2 - All duplicated Offices must be visited from at most an Agent
      _= mod.addConstrs((quicksum(x[i,j,a]
                                for i in all_pos
                                for a in agent_list) <= 1
                                for j in destination_office_pos),
                        name="ServeDuplicatesOffice")

      # 3 - All Agents start their trip from fictitious location
      _= mod.addConstrs((quicksum(x[start_pos,j,a]
                                for j in no_duplicates_pos) == 1
                                for a in agent_list),
                        name="StartFromFitticious")

      # 4 - Agents can't start their trip from fictitious location and go to only destination_
      ↪Office
      _= mod.addConstrs((x[start_pos,j,a] == 0
                                for j in destination_office_pos
                                for a in agent_list),
                        name="NotStartFromFitticiousToOffice")

      # 5 - All Agents end their trip in fictitious location
      _= mod.addConstrs((quicksum(x[i,start_pos,a]
                                for i in reachable_pos) == 1
                                for a in agent_list),
                        name="EndInFitticious")

      # 6 - Each served location has a served location before
      _= mod.addConstrs((quicksum(x[i,h,a] for i in all_pos) -
                        quicksum(x[h,j,a] for j in all_pos) == 0
                        for h in all_pos
                        for a in agent_list),
                        name="ContinuousLoops")

      # 7 - Agent can't do loop between same Client
      _= mod.addConstrs((x[i,i,a] == 0
                        for a in agent_list
                        for i in all_pos),
                        name="NoSelfLoops")
```

Service time constraints

```
[11]: # 8 - Sum of time spent into a location in a specific trip must be equal to Service
      #      time of that location
      _ = mod.addConstrs((c[j,a] == quicksum(x[i,j,a] for i in all_pos) *
      ↪data_dict[j]['SERVICE']
          for j in client_pos
          for a in agent_list),
          name="ServingTime")

      # 9 - Sum of time spent into Office in a specific trip must be equal to Office service
      ↪time
      _ = mod.addConstrs((quicksum(c[j,a]*x[i,j,a]
          for i in all_pos
          for j in office_pos) >= data_dict[0]['SERVICE']
          for a in agent_list),
          name="ServingTimeOffice")

      # 10 -Agent can go to a location only to spend time
      _ = mod.addConstrs((c[j,a] >= x[i,j,a]
          for i in all_pos
          for j in reachable_pos
          for a in agent_list),
          name="GoOnlyIfNeeded")
```

Office constraints

```
[12]: # 11 - Agent must go office at least once or start from it [0]
      _ = mod.addConstrs((quicksum(x[i,j,a]
          for i in all_pos
          for j in office_pos) >= 1
          for a in agent_list),
          name="ServeOffice")

      # 12 - Agent can't go office from office
      _ = mod.addConstrs((x[i,j,a] == 0
          for i in office_pos
          for j in office_pos
          for a in agent_list),
          name="NoOfficeFromOfficeB")

      # 13 - That is an only start Office position (Not Fitticious)
      _ = mod.addConstrs((x[i,0,a] == 0
          for i in client_pos
          for a in agent_list),
          name="NoOfficeFromOfficeC")

      # 14 - Agent can go to (Fitticious) Office only if he visited the associated Client
      _ = mod.addConstrs((x[j,j+CLIENTS,a] <= quicksum(x[i,j,a] for i in all_pos)
          for j in client_pos
          for a in agent_list),
```

```
name="GoOfficeOnlyIfVisitedItsClient")
```

```
# 15 - Agent can go to (Fitticious) Office just after he visit the associated Client
```

```
_ = mod.addConstrs((x[i,j,a] == 0
    for i in no_duplicates_pos
    for j in destination_office_pos
    for a in agent_list
    if i != j-CLIENTS),
    name="GoOfficeAfterItsClient")
```

Time windows constraints

[13]: *# 16 - Agent serve Client after his time window start*

```
_ = mod.addConstrs((data_dict[i]['READYTIME'] *
    quicksum(x[h,i,a]
        for h in all_pos) <= s[i,a]
    for i in client_pos
    for a in agent_list),
    name="ServeAfterTWStart")
```

```
# 17 - Agent serve Client before his time window end
```

```
_ = mod.addConstrs((data_dict[i]['DUEDATE'] *
    quicksum(x[h,i,a]
        for h in all_pos) >= s[i,a]
    for i in client_pos
    for a in agent_list),
    name="ServeBeforeTWEnd")
```

Lunch break constraints

[14]: *# 18 - If Agent has lunch it must be after lunch time start*

```
_ = mod.addConstrs((x[i,j,a]*(s[i,a] +
    c[i,a] +
    distance_matrix[i if i in no_duplicates_pos else 0][j if j in
    no_duplicates_pos else 0] +
    w[j,a]) >= l[i,j,a]*LUNCH_BREAK_RANGE[0]
    for i in reachable_pos
    for j in reachable_pos
    for a in agent_list),
    name="LunchTimeStart")
```

```
# 19 - If Agent has lunch it must be before lunch time end
```

```
_ = mod.addConstrs((l[i,j,a]*(s[i,a] +
    c[i,a] +
    distance_matrix[i if i in no_duplicates_pos else 0][j if j in
    no_duplicates_pos else 0] +
    w[j,a]) <= x[i,j,a]*LUNCH_BREAK_RANGE[1]
    for i in reachable_pos
    for j in reachable_pos
    for a in agent_list),
    name="LunchTimeEnd")
```

```

# 20 - If Agent have lunch between Clients I and J, agent have to make trip between I
↳ and J
_= mod.addConstrs((l[i,j,a] <= x[i,j,a]
                for i in reachable_pos
                for j in reachable_pos
                for a in agent_list),
                name="LunchTime")

# 21 - Getting maximum Agent service time
_= mod.addConstrs((t[a] == max_(s[i,a] for i in reachable_pos)
                for a in agent_list),
                name="MaxServiceTime")

# 22 - If Agent working time is greater than lunch time than Agent must have lunch
_= mod.addConstrs((WORKING_TIME_RANGE[1] * (quicksum(l[i,j,a]
                for i in reachable_pos
                for j in reachable_pos)) >= t[a] -
↳ LUNCH_BREAK_RANGE[0]
                for a in agent_list),
                name="AgentNeedLunchI")

# 23 - If Agent working time is lesser than lunch time than Agent must not have lunch
_= mod.addConstrs((WORKING_TIME_RANGE[1] * (1 - quicksum(l[i,j,a]
                for i in reachable_pos
                for j in reachable_pos)) >=
↳ LUNCH_BREAK_RANGE[0] - t[a]
                for a in agent_list),
                name="AgentNeedLunchII")

```


Service time constraint

```
[15]: # 24 - Service time of Client J is equal to sum of:
#      wait_time_of_client_J, service_time_of_client_I_before_of_J,
#      trip_between_I_and_J,
#      time_of_service_of_client_I, lunch_time_if_present
_= mod.addConstrs((s[j,a] == quicksum(x[i,j,a]*
                                     (s[i,a] +
                                     c[i,a] +
                                     distance_matrix[i if i in no_duplicates_pos else 0][j if j in no_duplicates_pos else 0] +
                                     l[i,j,a]*LUNCH_BREAK_TIME)
                                     for i in reachable_pos if i!=j) +
                                     w[j,a]
                                     for a in agent_list
                                     for j in reachable_pos),
name="RouteInTime")
```

Working day constraint

```
[16]: # 25 - Sum of minutes spent in: travels, servicing clients, waiting, eating at lunch
_= mod.addConstrs((quicksum(x[i,j,a] *
                             distance_matrix[i if i in no_duplicates_pos else 0][j if j in no_duplicates_pos else 0]
                             for i in reachable_pos
                             for j in reachable_pos) +
quicksum(x[i,j,a] *
         data_dict[j if j in no_duplicates_pos else 0]['SERVICE']
         for i in all_pos
         for j in reachable_pos) +
quicksum(w[i,a]
         for i in reachable_pos) +
quicksum(l[i,j,a]
         for i in reachable_pos
         for j in reachable_pos) *
LUNCH_BREAK_TIME +
WORKING_TIME_RANGE[0] <= WORKING_TIME_RANGE[1]
for a in agent_list),
name="MaxHours")
```

3.1.6 Objective Function

This objective function want to minimize time spent traveling between clients and awaiting for clients time windows.

```
[17]: mod.setObjective(quicksum(x[i,j,a] *  
                                distance_matrix[i if i in no_duplicates_pos else 0][j if j in_  
←no_duplicates_pos else 0]  
                                for i in all_pos  
                                for j in all_pos  
                                for a in agent_list) * TRAVEL_COST_MULTIPLIER +  
quicksum(w[i,a]  
                                for i in reachable_pos  
                                for a in agent_list) * WAIT_COST_MULTIPLIER +  
quicksum(s[i,a]  
                                for i in reachable_pos  
                                for a in agent_list),  
GRB.MINIMIZE)
```

3.2 Model optimization

Optimize problem using simplex method.

```
[18]: mod.params.Method = 0
      mod.params.TimeLimit=3000
```

```
Changed value of parameter Method to 0
  Prev: -1  Min: -1  Max: 5  Default: -1
Changed value of parameter TimeLimit to 3000.0
  Prev: inf  Min: 0.0  Max: inf  Default: inf
```

```
[19]: mod.optimize()
```

```
Gurobi Optimizer version 9.1.2 build v9.1.2rc0 (linux64)
Thread count: 6 physical cores, 12 logical processors, using up to 12 threads
Optimize a model with 13402 rows, 10965 columns and 72009 nonzeros
Model fingerprint: 0xf9d39e64
Model has 10212 quadratic constraints
Model has 3 general constraints
Variable types: 0 continuous, 10965 integer (10584 binary)
Coefficient statistics:
  Matrix range      [1e+00, 3e+04]
  QMatrix range     [1e+00, 2e+03]
  QLMatrix range    [1e+00, 2e+04]
  Objective range   [1e+00, 1e+03]
  Bounds range      [1e+00, 1e+00]
  RHS range         [1e+00, 3e+04]
  QRHS range        [4e+03, 4e+03]
Presolve removed 10518 rows and 5370 columns
Presolve time: 0.32s
Presolved: 41245 rows, 27423 columns, 135609 nonzeros
Presolved model has 5289 SOS constraint(s)
Variable types: 0 continuous, 27423 integer (10023 binary)

Root relaxation: objective 2.237344e+05, 3051 iterations, 0.17 seconds
```

Nodes		Current Node			Objective Bounds			Work	
Expl	Unexpl	Obj	Depth	IntInf	Incumbent	BestBd	Gap	It/Node	Time
0	0	223734.410	0	591	-	223734.410	-	-	2s
0	0	223966.911	0	668	-	223966.911	-	-	3s
0	0	224634.710	0	435	-	224634.710	-	-	3s
0	0	224664.326	0	625	-	224664.326	-	-	3s
0	0	224664.326	0	628	-	224664.326	-	-	4s
0	0	225601.972	0	723	-	225601.972	-	-	4s
0	0	225682.500	0	647	-	225682.500	-	-	4s
0	0	225686.526	0	635	-	225686.526	-	-	4s
0	0	225890.724	0	753	-	225890.724	-	-	5s
0	0	225992.140	0	712	-	225992.140	-	-	5s
0	0	226010.909	0	727	-	226010.909	-	-	5s
0	0	226114.390	0	717	-	226114.390	-	-	5s

	0	0	226114.390	0	696	-	226114.390	-	-	5s
	0	0	226259.373	0	736	-	226259.373	-	-	5s
	0	0	226279.348	0	674	-	226279.348	-	-	5s
	0	0	226354.044	0	763	-	226354.044	-	-	5s
	0	0	226407.768	0	677	-	226407.768	-	-	6s
H	4	0			276992.00000	226817.496	18.1%	0.0		9s
H	4	0			273973.00000	227674.285	16.9%	0.0		9s
	4	0	228157.905	0	436	273973.000	228157.905	16.7%	0.0	10s
H	4	0			273283.00000	228188.201	16.5%	0.0		10s
H	42	41			273220.00000	228256.325	16.5%	247		10s
H	70	71			273160.00000	228256.325	16.4%	166		11s
H	82	82			272980.00000	228256.325	16.4%	154		11s
H	85	82			272674.00000	228256.325	16.3%	152		11s
H	287	170			270220.00000	228296.374	15.5%	76.6		12s
	967	433	235537.392	23	324	270220.000	228923.919	15.3%	59.1	15s
*	1094	441		37		268621.00000	229144.784	14.7%	56.8	15s
*	1931	665		37		267208.00000	229847.641	14.0%	51.3	16s
	3910	1103	252960.078	27	902	267208.000	230821.713	13.6%	43.9	20s
H	3919	1053				266842.00000	230821.713	13.5%	43.8	21s
	3941	1073	230821.713	26	395	266842.000	230821.713	13.5%	45.2	25s
	4176	1105	infeasible	37		266842.000	230821.713	13.5%	46.0	30s
	4397	1072	245802.823	38	209	266842.000	230821.713	13.5%	45.6	35s
	5099	982	cutoff	33		266842.000	230821.713	13.5%	45.1	40s
	5819	820	237684.588	44	69	266842.000	230934.000	13.5%	44.0	45s
	6501	659	231909.938	36	95	266842.000	231909.938	13.1%	42.8	51s
	7849	540	236500.000	36	3	266842.000	233868.832	12.4%	40.8	56s
	9239	582	237035.829	39	389	266842.000	236473.150	11.4%	39.7	61s
	10018	628	257314.763	35	104	266842.000	237430.183	11.0%	39.4	65s
	12918	488	cutoff	48		266842.000	248800.867	6.76%	35.6	71s

Cutting planes:

Learned: 3
 Cover: 12
 Implied bound: 252
 Projected implied bound: 15
 Clique: 11
 MIR: 35
 StrongCG: 1
 Flow cover: 33
 Inf proof: 1
 RLT: 22
 Relax-and-lift: 54

Explored 17271 nodes (539415 simplex iterations) in 74.42 seconds
 Thread count was 12 (of 12 available processors)

Solution count 10: 266842 267208 268621 ... 273973

Optimal solution found (tolerance 1.00e-04)

Best objective 2.668420000000e+05, best bound 2.668420000000e+05, gap 0.0000%

3.3 Evaluating results

First I'll extract agents trips with relative data and order those trips in a domino like form.

```
[20]: def domino(trip):
    """ Function to calculate a domino sorted trip from a random one.
        :param trip: List of trips between nodes,
        :return: Domino list of visitated nodes,
                Ordered list of visitated nodes.
    """
    sorted_trip = list()
    sorted_visit = list()
    # Set dest as fitticious start position
    dest = start_pos
    # If trip not empty
    while len(trip):
        # For each position couple ([start, end]) in trip
        for t in trip:
            # If start position is dest
            if t[0] == dest:
                # If positions in couple aren't fitticious start position
                # append couple replacing fitticious offices with real one
                if t[0] != start_pos and t[1] != start_pos:
                    sorted_trip.append((t[0] if t[0] < CLIENTS else 0,
                                         t[1] if t[1] < CLIENTS else 0))
                    # Update destination
                    dest = t[1]
                    break
            # If fitticious end is reached stop loop oterwhise append latest
            # position reached
            if dest == start_pos:
                break
            else:
                sorted_visit.append(dest if dest < CLIENTS else 0)
    return sorted_trip, sorted_visit

# Initialize main lists
domino_list = list()
agent_visit_list = list()

# For each agent
for a in agent_list:
    agent_trip = list()
    # For each position
    for i in all_pos:
        # For each other position
        for j in all_pos:
            # If agent do trip between those store trip
            if x[i,j,a].X:
                agent_trip.append((i,j))
    # Sort trip locations and make a domino list
```

```

domino_agent, agent_visit = domino(agent_trip)

# update main lists
domino_list.append(domino_agent)
agent_visit_list.append(agent_visit)

```

3.3.1 Print sorted agents trips

```

[21]: # For each agent trip
for a, avl in enumerate(agent_visit_list):
    print(f"Agent {a+1} trip: {avl}")

```

```

Agent 1 trip: [0, 8, 11, 2, 20, 10, 15, 4, 1]
Agent 2 trip: [0, 13, 12, 19, 5, 0, 16]
Agent 3 trip: [14, 7, 6, 17, 3, 9, 18, 0]

```

3.3.2 Print trips stats

Print all stats regarding agents trips in an ordered way.

```

[22]: # Enlarge printable size
pd.set_option('display.max_columns', 10)
pd.set_option('display.width', 1000)

stats = dict()
# For each agent visit list
for a, avl in enumerate(agent_visit_list):
    agent_trip_desc = list()
    # For each visited position
    for i, v in enumerate(avl):
        prev = avl[i-1] if i else start_pos
        # If previous is a fictitious office
        if prev == 0 and i-2 > 0:
            prev = avl[i-2]+CLIENTS

        # If fictitious office
        v_mod = prev+CLIENTS if not v and i-1 >= 0 else v

    # Get stats
    pos_desc = {"POSITION": v if v else 'OFFICE',
                "COST TO REACH": distance_matrix[avl[i-1]][v] if i else 0.,
                "REACHED AT": s[v_mod,a].X,
                "SERVICE TIME": data_dict[v]['SERVICE'],
                "REAL S. TIME": c[v_mod, a].X,
                "LEAVING TIME": s[v_mod, a].X+c[v_mod, a].X,
                "LUNCH_BEFORE": l[prev, v_mod, a].X,
                "WAIT_BEFORE": w[v_mod, a].X,
                "TW_START": data_dict[v]['READYTIME'],
                "TW_END": data_dict[v]['DUE DATE']}

```

```

agent_trip_desc.append(pos_desc)

# Store path stats
stats.update({a+1: pd.DataFrame(agent_trip_desc)})

# Print agent stats
print(f"\nAgent {a+1}")
print(pd.DataFrame(agent_trip_desc))

```

Agent 1

	POSITION	COST TO REACH	REACHED AT	SERVICE TIME	REAL S. TIME	LEAVING TIME
	LUNCH_BEFORE	WAIT_BEFORE	TW_START	TW_END		
0	OFFICE	0.0	-0.0	3600.0	3600.0	3600.0
0.0		-0.0	0.0	28800.0		
1	8	364.0	3964.0	2400.0	2400.0	6364.0
0.0		-0.0	3600.0	4600.0		
2	11	82.0	6446.0	2400.0	2400.0	8846.0
0.0		-0.0	6000.0	7000.0		
3	2	269.0	9115.0	2400.0	2400.0	11515.0
0.0		-0.0	8400.0	9400.0		
4	20	914.0	12429.0	2400.0	2400.0	14829.0
0.0		-0.0	12300.0	13300.0		
5	10	228.0	15057.0	2400.0	2400.0	17457.0
0.0		-0.0	15000.0	16000.0		
6	15	652.0	19909.0	2400.0	2400.0	22309.0
1.0		-0.0	19800.0	20800.0		
7	4	41.0	22350.0	2400.0	2400.0	24750.0
0.0		-0.0	22000.0	23000.0		
8	1	144.0	24894.0	2400.0	2400.0	27294.0
0.0		-0.0	24800.0	25800.0		

Agent 2

	POSITION	COST TO REACH	REACHED AT	SERVICE TIME	REAL S. TIME	LEAVING TIME
	LUNCH_BEFORE	WAIT_BEFORE	TW_START	TW_END		
0	OFFICE	0.0	-0.0	3600.0	2383.0	2383.0
0.0		-0.0	0.0	28800.0		
1	13	731.0	3114.0	2400.0	2400.0	5514.0
0.0		-0.0	2400.0	4300.0		
2	12	186.0	5700.0	2400.0	2400.0	8100.0
0.0		-0.0	5700.0	6700.0		
3	19	447.0	8547.0	2400.0	2400.0	10947.0
0.0		-0.0	8500.0	9500.0		
4	5	361.0	11308.0	2400.0	2400.0	13708.0
0.0		-0.0	10900.0	11900.0		
5	OFFICE	1338.0	15046.0	3600.0	2003.0	17049.0
0.0		-0.0	0.0	28800.0		
6	16	551.0	19400.0	2400.0	2400.0	21800.0
1.0		-0.0	19400.0	20400.0		

Agent 3

	POSITION	COST TO REACH	REACHED AT	SERVICE TIME	REAL S. TIME	LEAVING TIME
	LUNCH_BEFORE	WAIT_BEFORE	TW_START	TW_END		
0	14	0.0	-0.0	2400.0	2400.0	2400.0
0.0	-0.0	0.0	1000.0			
1	7	700.0	3100.0	2400.0	2400.0	5500.0
0.0	-0.0	3100.0	4100.0			
2	6	171.0	5671.0	2400.0	2400.0	8071.0
0.0	-0.0	5500.0	6500.0			
3	17	324.0	8395.0	2400.0	2400.0	10795.0
0.0	-0.0	8000.0	9000.0			
4	3	258.0	11053.0	2400.0	2400.0	13453.0
0.0	-0.0	10600.0	11600.0			
5	9	94.0	13547.0	2400.0	2400.0	15947.0
0.0	-0.0	13000.0	14000.0			
6	18	144.0	16091.0	2400.0	2400.0	18491.0
0.0	-0.0	15300.0	16300.0			
7	OFFICE	984.0	21275.0	3600.0	3600.0	24875.0
1.0	-0.0	0.0	28800.0			

```
[23]: # Dump result dict on file
with open(f"./TEST_SETS/{FILE.split('/')[0]}_solution_iqp", "wb") as file:
    pickle.dump({"stats": stats, "domino": domino_list, "positions": positions}, file)
```


4 Compare results from different test sets

4.1 Test set 1

Filename: test_1.txt

Minimum agent count: 2

Data:

NUM	X	Y	DEMAND	READYTIME	DUE DATE	SERVICE
OFFICE	0.00	0.00	0.00	0.00	28800.00	3600.00
1	340.00	360.00	0.00	24800.00	25800.00	2400.00
2	220.00	180.00	0.00	8400.00	9400.00	2400.00
3	320.00	860.00	0.00	10600.00	11600.00	2400.00
4	260.00	480.00	0.00	22400.00	23400.00	2400.00
5	990.00	900.00	0.00	10900.00	11900.00	2400.00
6	740.00	470.00	0.00	6100.00	7100.00	2400.00
7	900.00	530.00	0.00	3700.00	4700.00	2400.00
8	100.00	350.00	0.00	3600.00	4600.00	2400.00
9	240.00	810.00	0.00	13000.00	14000.00	2400.00
10	630.00	970.00	0.00	15800.00	16800.00	2400.00

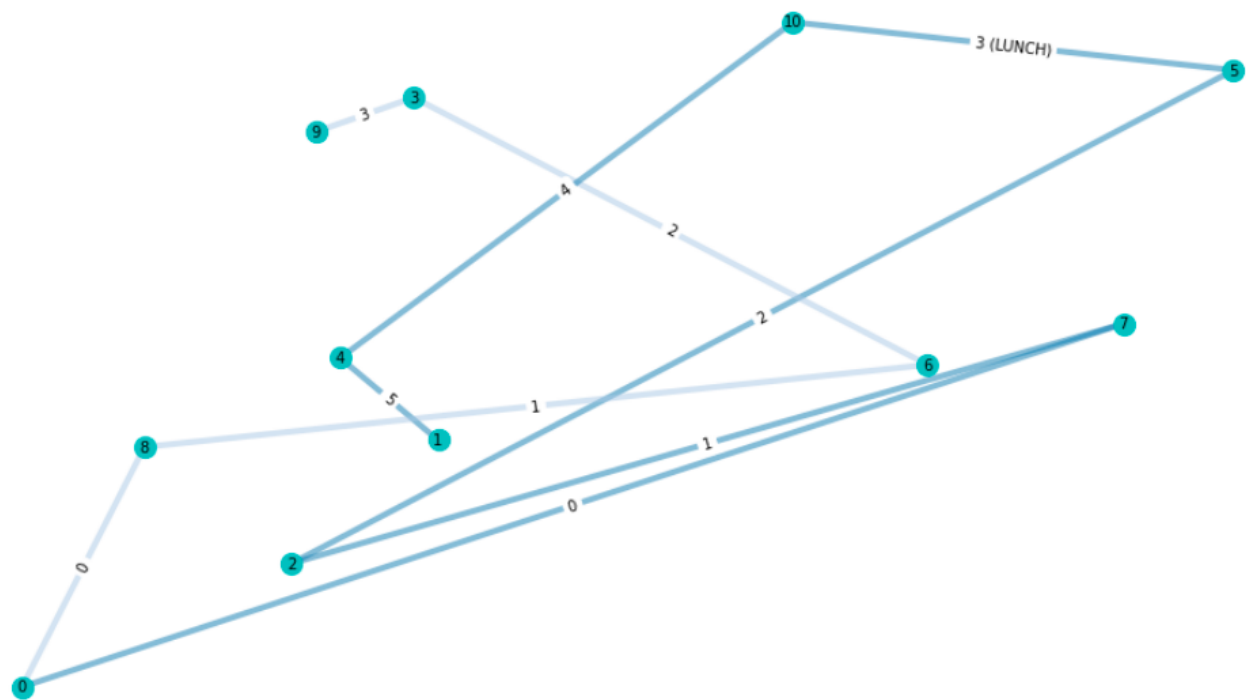
Model time: 0.64 sec

Model cost (travel time + wait time): 9782 time units

Results:

POSITION	TRAVEL TIME	REACHED	SERVICE	LEAVED	LUNCH BEFORE	WAIT
AGENT 1						
OFFICE	0.0	-0.0	3656.0	3656.0	0.0	-0.0
7	1044.0	4700.0	2400.0	7100.0	0.0	-0.0
2	765.0	8400.0	2400.0	10800.0	0.0	535.0
5	1054.0	11854.0	2400.0	14254.0	0.0	-0.0
10	367.0	16421.0	2400.0	18821.0	1.0	-0.0
4	614.0	22400.0	2400.0	24800.0	0.0	2965.0
1	144.0	24944.0	2400.0	27344.0	0.0	-0.0
AGENT 2						
OFFICE	0.0	-0.0	3600.0	3600.0	0.0	-0.0
8	364.0	3964.0	2400.0	6364.0	0.0	-0.0
6	651.0	7015.0	2400.0	9415.0	0.0	-0.0
3	573.0	10600.0	2400.0	13000.0	0.0	612.0
9	94.0	13094.0	2400.0	15494.0	0.0	-0.0

Graph:



4.2 Test set 2

Filename: test_2.txt

Minimum agent count: 3

Data:

NUM	X	Y	DEMAND	READYTIME	DUE DATE	SERVICE
OFFICE	0.00	0.00	0.00	0.00	28800.00	3600.00
1	340.00	360.00	0.00	24800.00	25800.00	2400.00
2	220.00	180.00	0.00	8400.00	9400.00	2400.00
3	320.00	860.00	0.00	10600.00	11600.00	2400.00
4	260.00	480.00	0.00	22000.00	23000.00	2400.00
5	990.00	900.00	0.00	10900.00	11900.00	2400.00
6	740.00	470.00	0.00	5500.00	6500.00	2400.00
7	900.00	530.00	0.00	3100.00	4100.00	2400.00
8	100.00	350.00	0.00	3600.00	4600.00	2400.00
9	240.00	810.00	0.00	13000.00	14000.00	2400.00
10	630.00	970.00	0.00	15000.00	16000.00	2400.00
11	120.00	430.00	0.00	6000.00	7000.00	2400.00
12	620.00	190.00	0.00	5700.00	6700.00	2400.00
13	730.00	40.00	0.00	2400.00	4300.00	2400.00
14	400.00	40.00	0.00	0.00	1000.00	2400.00
15	250.00	440.00	0.00	19800.00	20800.00	2400.00
16	30.00	550.00	0.00	19400.00	20400.00	2400.00
17	470.00	650.00	0.00	8000.00	9000.00	2400.00
18	320.00	930.00	0.00	15300.00	16300.00	2400.00
19	870.00	560.00	0.00	8500.00	9500.00	2400.00
20	830.00	860.00	0.00	12300.00	13300.00	2400.00

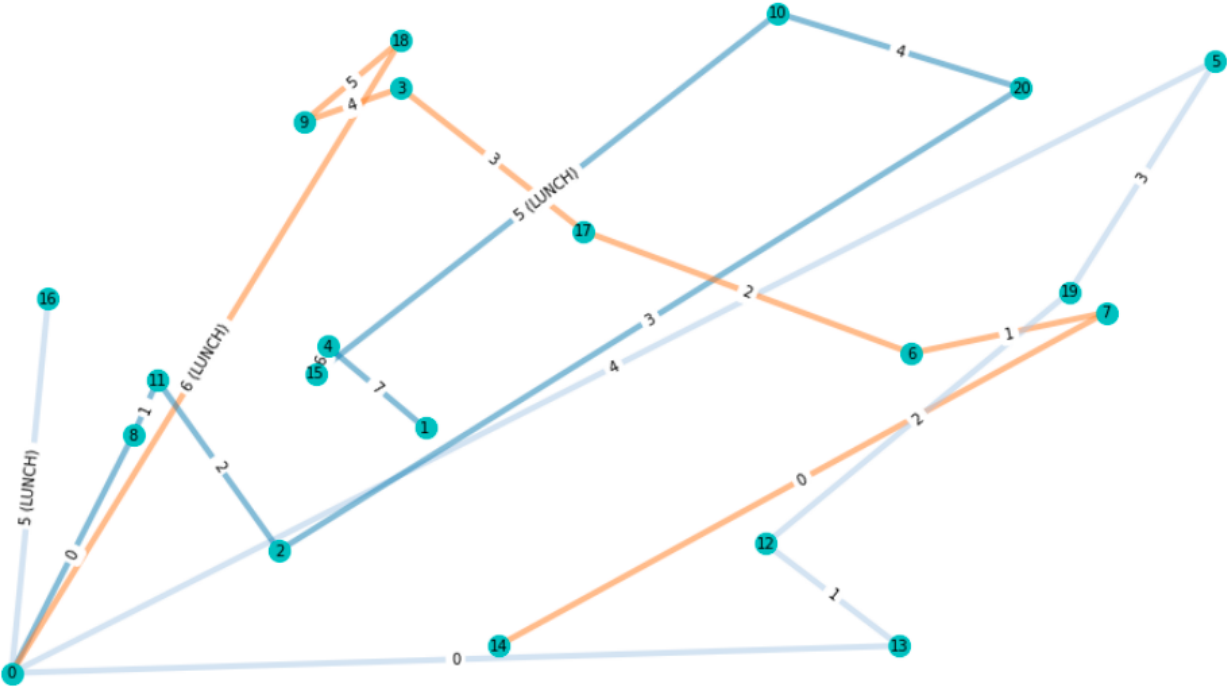
Model time: 65.54 sec

Model cost (travel time + wait time): 8983 time units

Results:

POSITION	TRAVEL TIME	REACHED	SERVICE	LEAVED	LUNCH BEFORE	WAIT
AGENT 1						
OFFICE	0.0	-0.0	3600.0	3600.0	0.0	-0.0
8	364.0	3964.0	2400.0	6364.0	0.0	-0.0
11	82.0	6446.0	2400.0	8846.0	0.0	-0.0
2	269.0	9115.0	2400.0	11515.0	0.0	-0.0
20	914.0	12429.0	2400.0	14829.0	0.0	-0.0
10	228.0	15057.0	2400.0	17457.0	0.0	-0.0
15	652.0	19909.0	2400.0	22309.0	1.0	-0.0
4	41.0	22350.0	2400.0	24750.0	0.0	-0.0
1	144.0	24894.0	2400.0	27294.0	0.0	-0.0
AGENT 2						
OFFICE	0.0	-0.0	2383.0	2383.0	0.0	-0.0
13	731.0	3114.0	2400.0	5514.0	0.0	-0.0
12	186.0	5700.0	2400.0	8100.0	0.0	-0.0
19	447.0	8547.0	2400.0	10947.0	0.0	-0.0
5	361.0	11308.0	2400.0	13708.0	0.0	-0.0
OFFICE	1338.0	15046.0	2003.0	17049.0	0.0	-0.0
16	551.0	19400.0	2400.0	21800.0	1.0	-0.0
AGENT 3						
14	0.0	-0.0	2400.0	2400.0	0.0	-0.0
7	700.0	3100.0	2400.0	5500.0	0.0	-0.0
6	171.0	5671.0	2400.0	8071.0	0.0	-0.0
17	324.0	8395.0	2400.0	10795.0	0.0	-0.0
3	258.0	11053.0	2400.0	13453.0	0.0	-0.0
9	94.0	13547.0	2400.0	15947.0	0.0	-0.0
18	144.0	16091.0	2400.0	18491.0	0.0	-0.0
OFFICE	984.0	21275.0	3600.0	24875.0	1.0	-0.0

Graph:



4.3 Test set 3

Filename: test_3.txt

Minimum agent count: 4

Data:

NUM	X	Y	DEMAND	READYTIME	DUE DATE	SERVICE
OFFICE	0.00	0.00	0.00	0.00	28800.00	3600.00
1	340.00	360.00	0.00	24800.00	25800.00	2400.00
2	220.00	180.00	0.00	8400.00	9400.00	2400.00
3	320.00	860.00	0.00	10600.00	11600.00	2400.00
4	260.00	480.00	0.00	22000.00	23000.00	2400.00
5	990.00	900.00	0.00	10900.00	11900.00	2400.00
6	740.00	470.00	0.00	5500.00	6500.00	2400.00
7	900.00	530.00	0.00	3100.00	4100.00	2400.00
8	100.00	350.00	0.00	3600.00	4600.00	2400.00
9	240.00	810.00	0.00	13000.00	14000.00	2400.00
10	630.00	970.00	0.00	15000.00	16000.00	2400.00
11	120.00	430.00	0.00	6000.00	7000.00	2400.00
12	620.00	190.00	0.00	5700.00	6700.00	2400.00
13	730.00	40.00	0.00	2400.00	4300.00	2400.00
14	400.00	40.00	0.00	0.00	1000.00	2400.00
15	250.00	440.00	0.00	19800.00	20800.00	2400.00
16	30.00	550.00	0.00	19600.00	20600.00	2400.00
17	470.00	650.00	0.00	8000.00	9000.00	2400.00
18	320.00	930.00	0.00	15000.00	16000.00	2400.00
19	870.00	560.00	0.00	8000.00	9000.00	2400.00
20	830.00	860.00	0.00	12300.00	13300.00	2400.00
21	230.00	460.00	0.00	2000.00	3000.00	2400.00
22	130.00	580.00	0.00	13000.00	14000.00	2400.00
23	370.00	830.00	0.00	22600.00	23600.00	2400.00
24	250.00	550.00	0.00	14200.00	15200.00	2400.00
25	750.00	530.00	0.00	24800.00	25800.00	2400.00
26	290.00	310.00	0.00	22500.00	23500.00	2400.00
27	360.00	900.00	0.00	25300.00	26300.00	2400.00

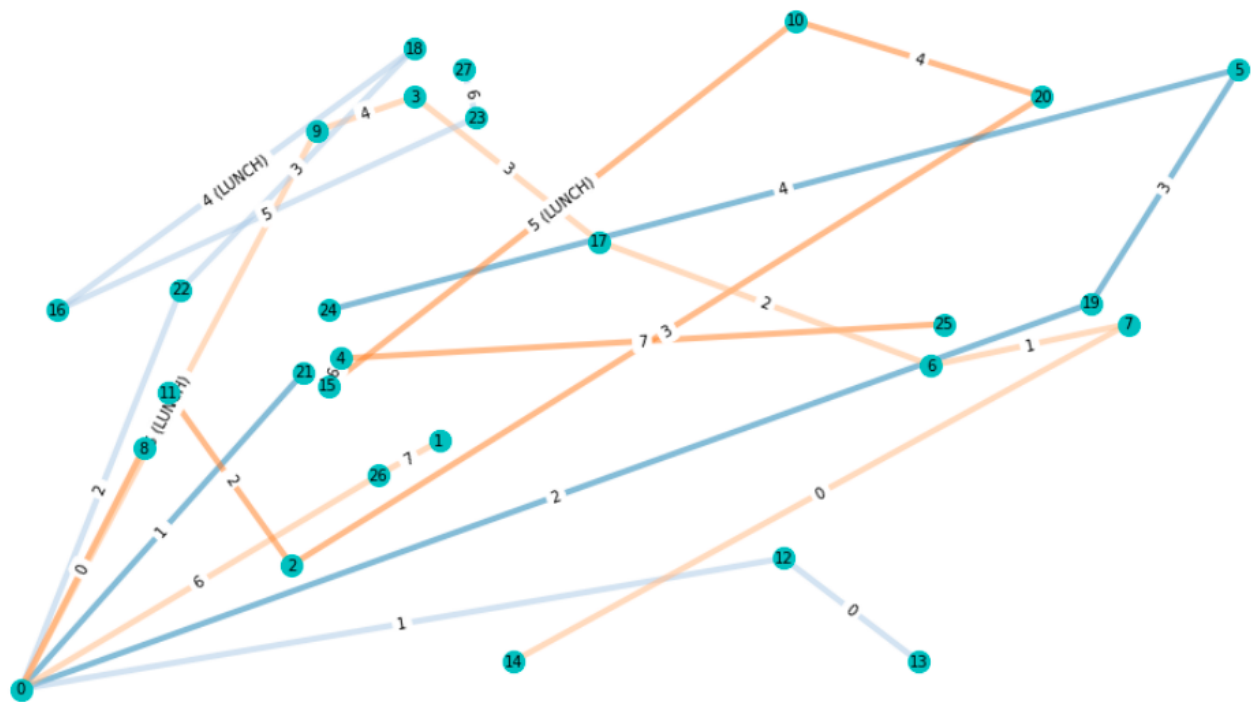
Model time: 2708.05 sec

Model cost (travel time + wait time): 15183 time units

Results:

POSITION	TRAVEL TIME	REACHED	SERVICE	LEAVED	LUNCH BEFORE	WAIT
AGENT 1						
OFFICE	0.0	-0.0	1486.0	1486.0	0.0	-0.0
21	514.0	2000.0	2400.0	4400.0	0.0	-0.0
OFFICE	514.0	4914.0	2190.0	7104.0	0.0	-0.0
19	1035.0	8139.0	2400.0	10539.0	0.0	-0.0
5	361.0	10900.0	2400.0	13300.0	0.0	-0.0
24	819.0	14200.0	2400.0	16600.0	0.0	81.0
AGENT 2						
13	0.0	2400.0	2400.0	4800.0	0.0	2400.0
12	186.0	5700.0	2400.0	8100.0	0.0	714.0
OFFICE	648.0	8748.0	3658.0	12406.0	0.0	-0.0
22	594.0	13000.0	2400.0	15400.0	0.0	-0.0
18	398.0	15798.0	2400.0	18198.0	0.0	-0.0
16	478.0	20476.0	2400.0	22876.0	1.0	-0.0
23	440.0	23316.0	2400.0	25716.0	0.0	-0.0
27	71.0	25787.0	2400.0	28187.0	0.0	-0.0
AGENT 3						
OFFICE	0.0	-0.0	3600.0	3600.0	0.0	-0.0
8	364.0	3964.0	2400.0	6364.0	0.0	-0.0
11	82.0	6446.0	2400.0	8846.0	0.0	-0.0
2	269.0	9115.0	2400.0	11515.0	0.0	-0.0
20	914.0	12429.0	2400.0	14829.0	0.0	-0.0
10	228.0	15057.0	2400.0	17457.0	0.0	-0.0
15	652.0	19909.0	2400.0	22309.0	1.0	-0.0
4	41.0	22350.0	2400.0	24750.0	0.0	-0.0
25	493.0	25243.0	2400.0	27643.0	0.0	-0.0
AGENT 4						
14	0.0	-0.0	2400.0	2400.0	0.0	-0.0
7	700.0	3100.0	2400.0	5500.0	0.0	-0.0
6	171.0	5671.0	2400.0	8071.0	0.0	-0.0
17	324.0	8395.0	2400.0	10795.0	0.0	-0.0
3	258.0	11053.0	2400.0	13453.0	0.0	-0.0
9	94.0	13547.0	2400.0	15947.0	0.0	-0.0
OFFICE	845.0	18592.0	3600.0	22192.0	1.0	-0.0
26	424.0	22616.0	2400.0	25016.0	0.0	-0.0
1	71.0	25087.0	2400.0	27487.0	0.0	-0.0

Graph:



5 Conclusions

This model gives an exact solution but the increase in the number of customers leads to a drastic increase in times, as shown in the following figure.

