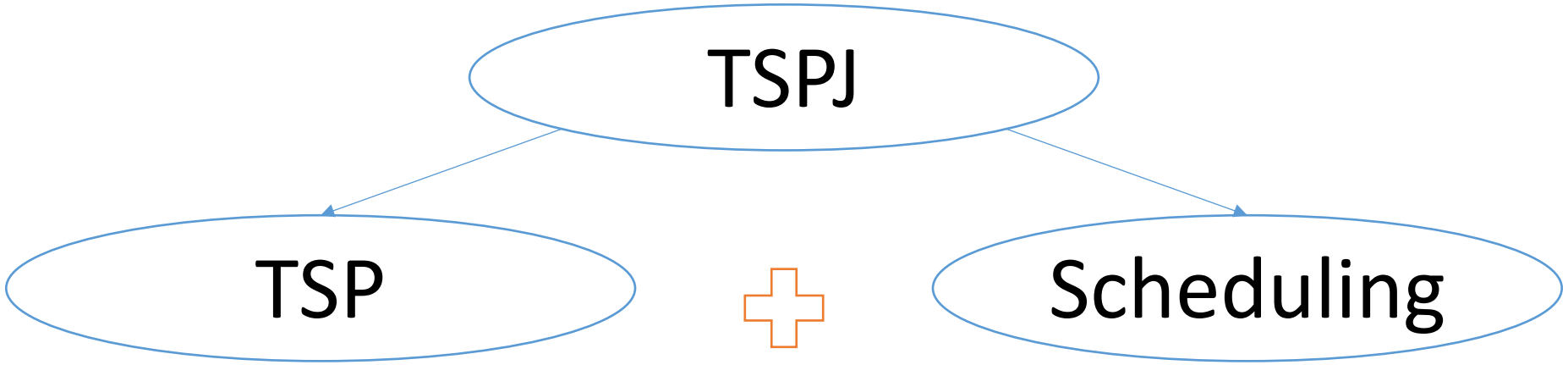
# The Traveling Salesman Problem with Job-times

Project Presentation

Zihan Ma 20238181

2024/06/14

# Traveling Salesman Problem with Job-times (TSPJ)




- A traveler moves through n nodes, visits all locations and each location exactly once to assign and initiate one of n jobs, and then returns to the first location.

- After the initiation of a job, the traveler moves to the next location immediately and the job continues autonomously.

# Traveling Salesman Problem with Job-times (TSPJ)

$$
\begin{aligned}
\min \quad & C_{\max} \\
\text{s.t.} \quad & C_{\max} \geq TS_i + \sum_{k=1}^{n} Z_{ik} t_{ik} \quad \forall i = 1, \ldots, n \\
& C_{\max} \geq TS_i + X_{\mathrm{i0}} tt_{i0} \quad \forall i = 1, \ldots, n \\
& \sum_{i=1}^{n} Z_{ik} = 1 \quad \forall k = 1, \ldots, n \\
& \sum_{k=1}^{n} Z_{ik} = 1 \quad \forall i = 1, \ldots, n \\
& \sum_{j=0}^{n} X_{ij} = 1 \quad \forall i = 0, \ldots, n \quad i \neq j \\
& \sum_{i=0}^{n} X_{ij} = 1 \quad \forall j = 0, \ldots, n \quad i \neq j \\
& \sum_{j=0}^{n} Y_{ij} - \sum_{j=0}^{n} Y_{ji} = 1 \quad \forall i = 1, \ldots, n \quad i \neq j \\
& Y_{ij} \leq n X_{ij} \quad \forall i = 1, \ldots, n \quad \forall j = 0, \ldots, n \quad i \neq j \\
& TS_i + tt_{ij} - (1 - X_{ij})\,\mathbf{M} \leq TS_j \quad \forall i = 0, \ldots, n \quad \forall j = 1, \ldots, n \quad i \neq j \\
& X_{ij}, Z_{ik} \in \{0, 1\} \quad TS_i, Y_{ij} \geq 0 \quad \forall i \forall j
\end{aligned}
$$

Minimize the maximum completion time and the time to return to the depot

# Heuristics for Traveling Salesman Problem with Job-times (TSPJ)

- Algorithm 1: TSPJ-NN(Nearest Neighbor)
- Algorithm 2: TSPJ-2-OPT
- Algorithm 3: Reverse assignments
- Algorithm 4: Local Search Improvement Heuristic

# TSPJ-NN(Nearest Neighbor)

**Algorithm 1 TSPJ-NN**

Input: Read Travel Time table (TT) as matrix $t_{ij}$
Input: Read Job-time table (TJ) as matrix $t_{ik}$
Output: Best_sequence
Define $C$: List of completion time of all jobs in the sequence
$C[0] \leftarrow 0$
**for** $s$ in range(1, Len(TT)) **do**
  node_pool $\leftarrow TT$
  job_pool $\leftarrow I$
  sequence[0] $= s$
  **for** $i$ in range(1, Len(TT)) **do**
    sequence[$i$] $\leftarrow \arg\min(tt_{i-1,j} \quad \forall j = 1, \ldots, n)$
    Remove selected node from node_pool
  **end for**
  **for** $i$ in range(Len(TT), 1, -1) **do**
    job[$i$] $\leftarrow \arg\min(t_{ik} \quad \forall k = 1, \ldots, K)$
    Remove selected job from job_pool
  **end for**
  new-sequence $\leftarrow$ sequence + job
  Calculate $C_{\text{new-sequence}}$
  **if** $C_{\max} > C_{\text{new-sequence}}$ **then**
    Best_sequence $\leftarrow$ new-sequence
  **end if**
**end for**
**return** Best_sequence

The **TSPJ-NN Algorithm** begins by selecting any node as the starting point and constructing a tour by visiting the nearest neighboring nodes until a complete tour is formed. Once the tour is completed, the algorithm resumes from the last node along the tour and assigns the job with the minimum processing time at that node to that node.

# TSPJ-2-OPT

**Algorithm 2 TSPJ-2-opt**

```
Input: Read Travel Time table (TT) as matrix t_ij
Input: Read Job-time table (TJ) as matrix t_ik
Input: Read sequence
Output: Best_sequence
Define C: List of completion time of all jobs in the sequence
Best_sequence ← sequence
Start_again
while there is improvement do
    for i in range(1, Len(TT) - 1) do
        for j in range(i + 1, Len(TT) + 1) do
            if swapping is profitable then
                sequence ← Swap nodes
                job_pool ← T
                for i in range(Len(TT), 1, -1) do
                    job[i] ← arg min(t_ik   ∀k = 1, ..., K)
                    Remove selected job from job-pool
                end for
                Update jobs in sequence
                Best_sequence ← sequence
                Goto Start_again
            end if
        end for
    end for
end while
Return Best_sequence
```

The **TSPJ-2-opt Algorithm** starts from the first node of the existing tour and evaluates the maximum tour completion time resulting from swapping node pairs. If an improvement is possible, the swap is accepted, and the algorithm starts from the last node along this tour and assigns the job with minimum processing time at the node to it.

# Reverse Assignments

**Algorithm 3 Reverse Assignments**

Input: Read sequence as list $X$
Input: Read Job-time table $(T)$ as matrix $t_{xk}$
Output: Reads job list as List $K$
$n \leftarrow \text{Len}(X)$
$X' \leftarrow X$
$K \leftarrow \text{Zeros}(n)$
**while** $X'$ **do**
  $K[-1] \leftarrow \text{argmin}\{t_{xk}, k \in \tau(x)\}$
  **for** $j$ in range (1,n) **do**
    $t_{jk[-1]} \leftarrow \infty$
  **end for**
  $X' \leftarrow X'[:-1]$
**end while**
**return** $K$

The **Reverse Assignments Algorithm** moves backward through the existing tour and assigns the next un-assigned job with minimum processing time to the node till all jobs are assigned.

# Local Search Improvement (LSI)

**Algorithm 4** Local Search Improvement heuristic

```
Input: Read Travel Time table (TT) as matrix t_ij
Input: Read Job-time table (TJ) as matrix t_ik
Input: Read sequence
Output: Best_sequence
Start_again
while there is improvement do
  if job-time of C_max is not minimum job-time for node then
    Assign minimum job-time to C_max-node
    Reassign other jobs
    calculate C_new_sequence
    if C_max > C_new_sequence then
      Update Best_sequence
      Goto Start_again
    end if
  end if
  for S in range (C_max-node + 1, Len(TT)) do
    if swapping jobs conditions are satisfied then
      Swap job between C_max-node and S
      Update Best_sequence
      Goto Start_again
    end if
  end for
  C[0] = 0
  for S in range (C_max-node −1, 1, −1) do
    if reverse swapping conditions are satisfied then
      Swap C_max-node and S
      Update Best_sequence
      Goto Start_again
    end if
  end for
end while
Return Best_sequence
```

The **Local Search Improvement (LSI) Algorithm** improves TSPJ solutions by first considering jobs swapping while keeping the node sequences fixed and evaluating node swapping for nodes that precede the node that results in the maximum completion time.

# Four Procedures for the TSPJ Heuristic

- Procedure 1: (Considering Job Assignment) + TSPJ-NN + TSPJ-2-OPT
- Procedure 2: (Not Considering Job Assignment) + NN + 2-OPT + Reverse assignments + Local Search Improvement
- Procedure 3: TSPJ-NN + Local Search Improvement Heuristic + TSPJ-2-OPT
- Procedure 4: (Greatest Distance) + TSPJ-NN + Local Search Improvement Heuristic + TSPJ-2-OPT

# Implementation

## Structure

```
data/
|--Small_problems/
|--TSPLIB-J/
src/
|--__init__.py
|--cal_time.py // To calculate the maximum completion time
|--LSI.py // Local Search Improvements (Algorithm 4)
|--NN.py // TSPJ-NN (Algorithm 1)
|--re_assign.py // Reverse Assignments (Algorithm 3)
|--two_opt.py // TSPJ-2-opt (Algorithm 2)
|--main.py // To load the dataset and execute each procedure
```

## Data

# Experiments - Solution Performance

My results compared with GAMS results reported in the paper and gaps.

| | GAMS | Procedure 1 | | Procedure 2 | | Procedure 3 | | Procedure 4 | |
|---|---|---|---|---|---|---|---|---|---|
| | $C_{max}$ | $C_{max}$ | % | $C_{max}$ | % | $C_{max}$ | % | $C_{max}$ | % |
| gr48-J | 7288.0 | 7560.0 | 3.6 | 8395.0 | 13.2 | 7560.0 | 3.6 | 7589.0 | 4.0 |
| berlin52-J | 11087.0 | 11251.2 | 1.5 | 12935.6 | 14.3 | 11251.2 | 1.5 | 11790.5 | 6.0 |
| gr24-J | 1806.0 | 1858.0 | 2.8 | 2022.0 | 10.7 | 1858.0 | 2.8 | 1858.0 | 2.8 |
| eli51-J | 630.0 | 648.7 | 2.9 | 677.6 | 7.0 | 648.7 | 2.9 | 634.3 | 0.7 |
| gr21-J | 7788.0 | 7959.0 | 2.1 | 10102.0 | 22.9 | 7959.0 | 2.1 | 7959.0 | 2.1 |
| bays29-J | 2937.0 | 2963.0 | 0.9 | 3487.0 | 15.8 | 2963.0 | 0.9 | 2980.0 | 1.4 |
| eli76-J | 802.0 | 821.1 | 2.3 | 1000.6 | 19.8 | 821.1 | 2.3 | 826.4 | 3.0 |
| gr17-J | 2760.0 | 2759.0 | (0.0) | 3278.0 | 15.8 | 2759.0 | (0.0) | 2759.0 | (0.0) |
| eil101-J | 947.4 | 982.8 | 3.6 | 1037.0 | 8.6 | 982.8 | 3.6 | 975.3 | 2.9 |
| fri26-J | 1283.0 | 1309.0 | 2.0 | 1338.0 | 4.1 | 1309.0 | 2.0 | 1300.0 | 1.3 |

Table 1: Computational results: the heuristics procedures and the GAMS solutions

# Experiments - Solution Performance

Table 6: Solution Performance Gap(%)

| | Procedure 1 | Procedure 2 | Procedure 3 | Procedure 4 |
|---|---|---|---|---|
| Paper | 2.61 | 4.68 | 2.48 | 2.64 |
| Mine | 2.16 | 13.23 | 2.16 | 2.41 |

- The solution performance gaps for Procedures 1, 3, and 4 are similar, with Procedure 2 exhibiting a larger gap than the others.
  - The initial tour is generated without considering job assignments in Procedure 2.
  - Although the job assignments are subsequently adjusted and local search improvements are applied in Procedure 2, the overall performance of this procedure is inferior to the others.

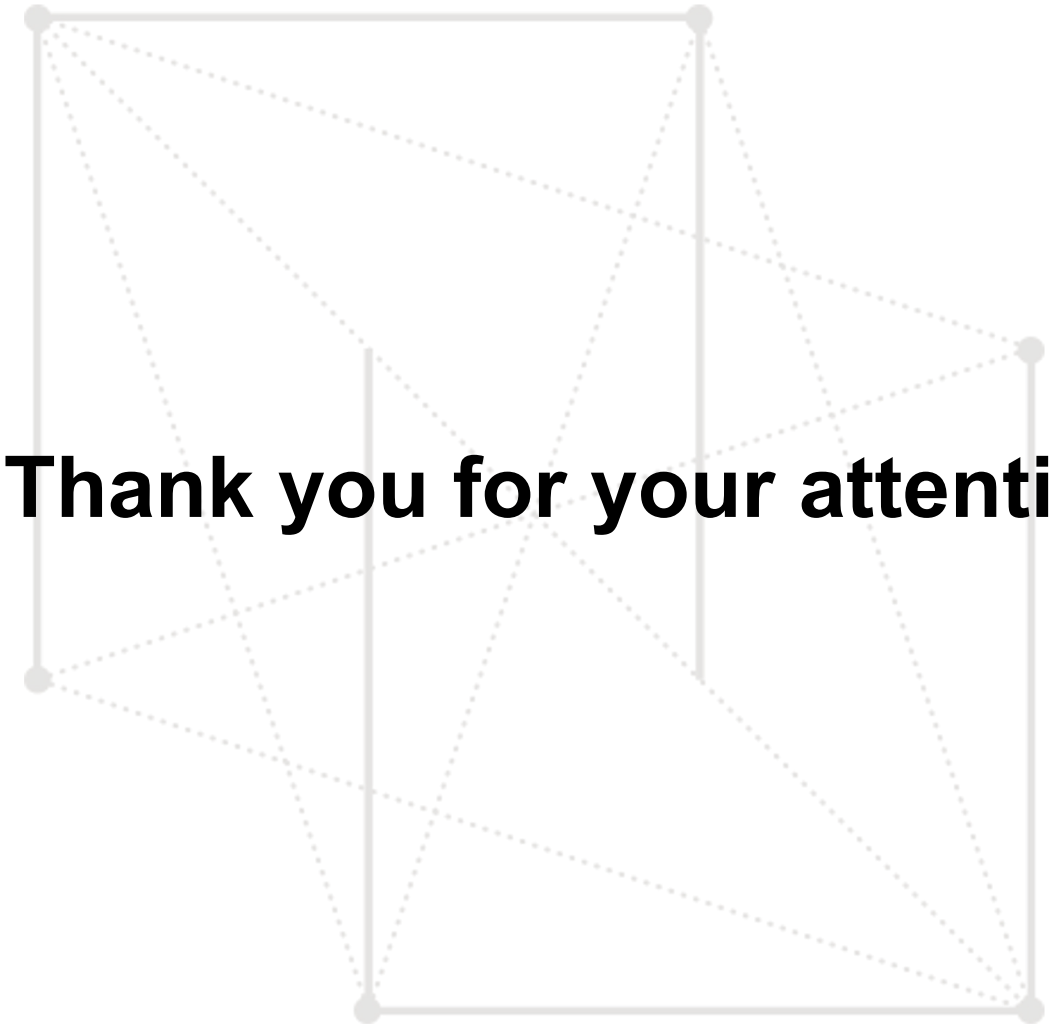
# Experiments - Time Performance

Table 7: Mean Processing Time(s) and Mean Ratio

| | **Procedure 1** | **Procedure 2** | **Procedure 3** | **Procedure 4** |
|---|---|---|---|---|
| Mean Processing Time(s) | 3.359787 | 1.883384 | 3.273189 | 4.458672 |
| Mean Ratio (J/P) | 0.001829 | 0.001971 | 0.001898 | 0.001621 |

- Despite Procedure 2 resulting in a larger solution performance gap, it saves significant processing time compared to the other procedures.

- Julia requires considerably less processing time than Python.



# Thank you for your attention!