KONERU LAKSHMAIAH UNIVERSITY

COMPUTER SCIENCE AND ENGINEERING DEPARTMENT

PROJECT REPORT

On

Transportation problem with row minima & Tabu Search

SUBMITTED BY

2100032288 GUNDA KARTHIK

UNDER THE ESTEEMED GUIDANCE OF

Dr. Akhilesh Kumar Dubey

CSE

KL UNIVERSITY

Green fields,

Vaddeswaram – 522 502

Guntur Dt., AP, India.

Transportation problem with Row minima Method

- ⇨ The transportation problem is a well-known linear programming problem that deals with optimizing the transportation of goods from a set of sources to a set of destinations at a minimum cost. The transportation problem has numerous applications in various fields, such as supply chain management, logistics, and operations research.

- ⇨ One of the most popular methods to solve the transportation problem is the Row minima method, which is also known as the North-West Corner Method. The Row minima method is a simple and straightforward algorithm that provides an initial feasible solution to the transportation problem.

- ⇨ The Row minima method starts by selecting the cell in the top-left corner of the transportation matrix, which corresponds to the first source and the first destination. Then, the method determines the maximum amount of goods that can be transported from the selected source to the selected destination, based on the availability of goods at the source and the demand at the destination. The method fills the selected cell with this amount and moves to the next cell in the same row.

- ⇨ If the total demand at the current destination has been satisfied, the method moves to the next destination in the same column. If the total supply at the current source has been exhausted, the method moves to the next source in the same row. This process continues until all the cells in the transportation matrix have been filled.

- ⇨ The Row minima method has some limitations, such as the possibility of obtaining a suboptimal solution. However, it is widely used as an initial feasible solution for more complex optimization algorithms, such as the Vogel's Approximation Method or the Modified Distribution Method.

OBJECTIVE: -

- The objective of the transportation problem with Row minima Method is to optimize the transportation of goods from a set of sources to a set of destinations at a minimum cost. The Row minima Method is used to obtain an initial feasible solution to the transportation problem, which satisfies the supply and demand constraints.

- The objective of the transportation problem with Row minima Method is to optimize the transportation of goods from a set of sources to a set of destinations at a minimum cost. The Row minima Method is used to obtain an initial feasible solution to the transportation problem, which satisfies the supply and demand constraints.

ADVANTAGES: -

There are several advantages of using the Row minima Method to solve the transportation problem:

1. **Simplicity**: The Row minima Method is a simple and easy-to-understand algorithm that does not require any complex mathematical knowledge or software.

2. **Speed**: The Row minima Method is a fast algorithm that can quickly provide an initial feasible solution to the transportation problem. This allows for a quicker decision-making process in transportation planning.

3. **Feasibility**: The Row minima Method always provides a feasible solution to the transportation problem, which satisfies the supply and demand constraints.

4. **Flexibility**: The Row minima Method can be easily modified to handle situations where the transportation matrix is not balanced or where there are multiple optimal solutions.

5. **Cost-effectiveness**: The Row minima Method is a cost-effective algorithm that does not require any specialized software or equipment.

6. **Ease of implementation**: The Row minima Method can be implemented using a spreadsheet program, such as Microsoft Excel, making it accessible to a wider range of users.

7. **Good starting point**: The Row minima Method provides a good starting point for more complex optimization algorithms to find the optimal solution.

DISADVANTAGES: -

Although the Row minima Method has several advantages, there are also some disadvantages to using this method to solve the transportation problem:

1. **Suboptimal solutions**: The Row minima Method can result in a suboptimal solution to the transportation problem, which may not be the most cost-effective solution.

2. **Lack of optimality test**: The Row minima Method does not provide a mechanism to test whether the initial feasible solution obtained is optimal or not.

3. **Dependency on initial values**: The Row minima Method depends on the initial values of the transportation matrix and may result in different solutions depending on the starting point.

4. **Limited application**: The Row minima Method is suitable only for transportation problems that have balanced supply and demand, and do not involve restrictions on the transportation capacity or other constraints.

5. **Ignores transportation cost changes**: The Row minima Method assumes that the transportation costs remain constant, which may not be the case in real-world transportation problems.

6. **Ignores intermediate locations**: The Row minima Method assumes that the goods are transported directly from the sources to the destinations, without considering intermediate locations where the goods may need to be stored or transferred.

ALGORITHM / PROCEDURE: -

The algorithm for the Transportation problem with Row minima Method can be summarized as follows:

Step 1: Set up the transportation matrix, which represents the costs of transporting goods from each source to each destination.

Step 2: Identify the minimum value in each row of the transportation matrix, which represents the minimum cost of transporting goods from each source to any destination.

Step 3: Allocate as much of the available supply from each source to the destination with the minimum transportation cost, until either the demand at the destination or the supply at the source is exhausted.

Step 4: If the demand at a destination is met, cross out the corresponding column and move to the next column.

Step 5: If the supply at a source is exhausted, cross out the corresponding row and move to the next row.

Step 6: Repeat steps 3-5 until all the cells in the transportation matrix have been allocated.

Step 7: Check if the total supply equals the total demand. If not, add a dummy row or column to balance the matrix.

Step 8: Calculate the total cost of transportation, which is the sum of the products of the allocated quantities and their corresponding transportation costs.

Step 9: If necessary, use a more complex optimization algorithm to improve upon the initial feasible solution obtained using the Row minima Method.

EXAMPLE: -

The Row Minima Method

Step 1 : The smallest cost in the first row of the transportation table is determined. Let it be C_{ij}. Allocate the maximum feasible x_{ij}=min (a_1,b_j) in the cell (1,j).

Step 2: If $x_{1j}=a_1$, cross off the 1st row of the transportation table and move down to the second row. If $x_{1j}=b_j$, cross off the jth column of the transportation table and reconsider the first row with the remaining availability.

If $x_{1j}=a_1=b_j$, cross off the 1st column and make the second allocation $x_{1k}=0$ in the cell (1,k) with C_{1k} being the new minimum cost in the first row. Cross of the first row and move down to the second row.

Step 3: Repeat steps1 and 2 for the resulting reduced transportation table until all the rim requirements are satisfied.

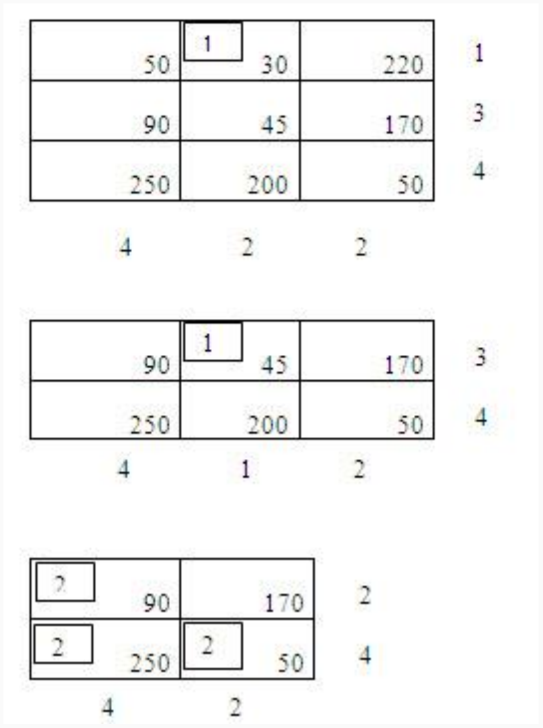
Problem:

Determine an initial basic feasible solution to the following transportation problem following row minima method.

Sources	Destimation			Supply, Units
	A	B	C	
I	50	30	220	1

II	90	45	170	3
III	250	200	50	4
Requirements	4	2	2	

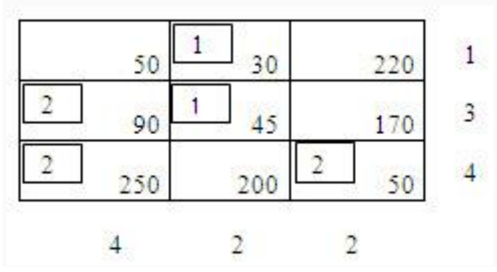
Solution:



50	$\boxed{1}$ 30	220	1
90	45	170	3
250	200	50	4
4	2	2	

90	$\boxed{1}$ 45	170	3
250	200	50	4
4	1	2	

$\boxed{2}$ 90	170	2
$\boxed{2}$ 250	$\boxed{2}$ 50	4
4	2	

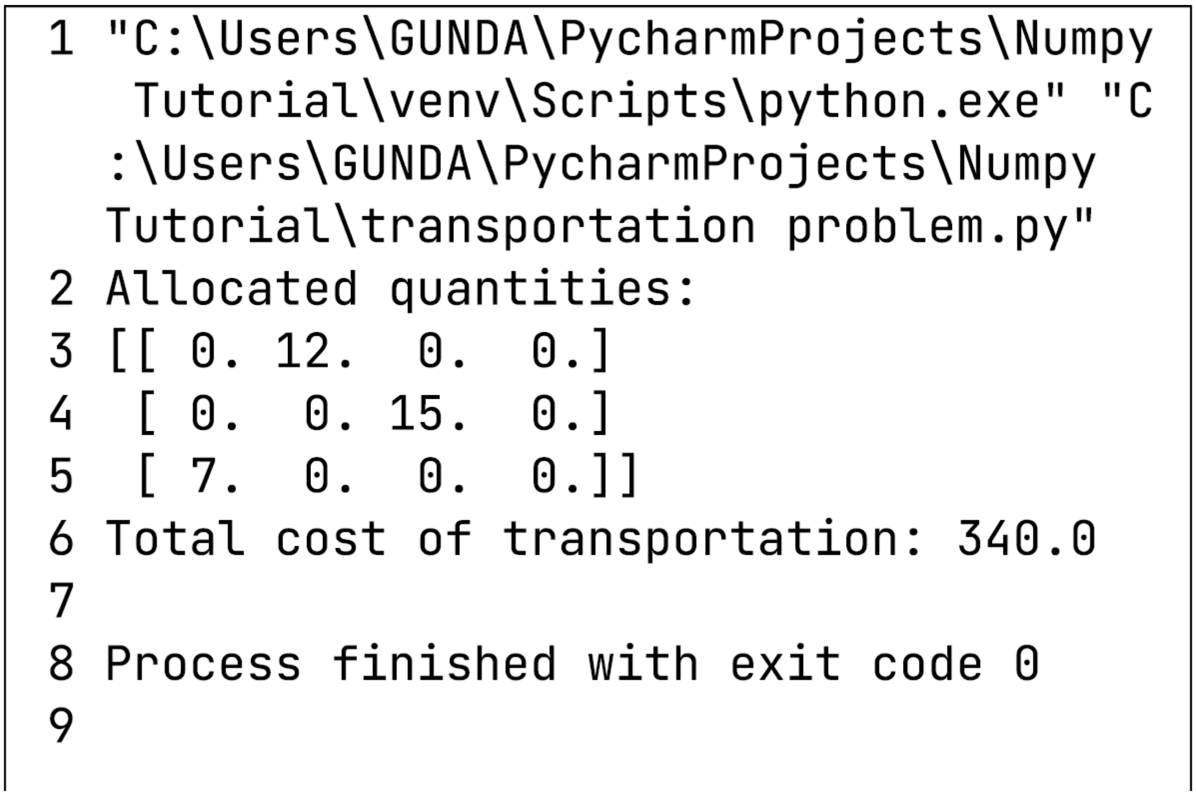


50	$\boxed{1}$ 30	220	1
$\boxed{2}$ 90	$\boxed{1}$ 45	170	3
$\boxed{2}$ 250	200	$\boxed{2}$ 50	4
4	2	2	

PSEUDOCODE: -

```
import numpy as np
def transportation_problem_row_minima_method(c, s, d):
    n, m = c.shape
    max_val = np.max(c)
    c[c == np.inf] = max_val + 1
    x = np.zeros((n, m))
    while np.sum(x) < np.sum(s):
        row_min = np.min(c, axis=1)
        row_min_idx = np.argmin(c, axis=1)

        for i in range(n):
            if s[i] == 0:
                continue
            j = row_min_idx[i]
            x[i, j] = min(s[i], d[j])
            s[i] -= x[i, j]
            d[j] -= x[i, j]
            if s[i] == 0:
                c[i, :] = max_val + 1
            if d[j] == 0:
                c[:, j] = max_val + 1

    cost = np.sum(x * c)
    return x, cost
def main():
    c = np.array([[6, 3, 5, 4],
                  [5, 9, 2, 7],
                  [5, 7, 8, 6]])
    s = np.array([22, 15, 8])
    d = np.array([7, 12, 17, 9])
    x, cost =
transportation_problem_row_minima_method(c, s, d)
    print("Allocated quantities:")
    print(x)
    print("Total cost of transportation:", cost)


if __name__ == "__main__":
    main()
```

OUTPUT: -



```
"C:\Users\GUNDA\PycharmProjects\Numpy Tutorial\venv\Scripts\python.exe" "C:\Users\GUNDA\PycharmProjects\Numpy Tutorial\transportation problem.py"
Allocated quantities:
[[ 0. 12.  0.  0.]
 [ 0.  0. 15.  0.]
 [ 7.  0.  0.  0.]]
Total cost of transportation: 340.0

Process finished with exit code 0

```

TOPIC – 2: -

TABU SEARCH: -

- ⇨ Tabu Search is a metaheuristic optimization algorithm that is used to solve combinatorial optimization problems. These types of problems involve finding the best possible solution from a finite set of possible solutions, where the objective is to minimize or maximize an objective function subject to a set of constraints.
- ⇨ The Tabu Search algorithm is a local search method that iteratively makes small moves from the current solution to neighbouring solutions. The algorithm explores the search space to find the optimal solution by keeping track of the best solution found so far. The algorithm maintains a tabu list of recently visited solutions, which prevents the search from revisiting the same solution and getting stuck in local optima.

- ⇨ Tabu Search is an iterative algorithm that starts with an initial solution and searches for better solutions by exploring the search space using a neighbourhood function. The neighbourhood function generates a set of neighbouring solutions by making small moves from the current solution. The algorithm evaluates each neighbouring solution and selects the best neighbour that is not on the tabu list to move to. The tabu list is updated by adding the current move and removing the oldest move.
- ⇨ Tabu Search is a powerful optimization algorithm that can handle large search spaces efficiently and can find high-quality solutions. However, it requires setting many parameters, such as the size of the tabu list and the number of iterations, which can affect the performance of the algorithm.

OBJECTIVE: -

- ⇨ The objective of Tabu Search is to find the best possible solution to a given combinatorial optimization problem within a feasible set of solutions. The algorithm aims to minimize or maximize an objective function subject to a set of constraints.
- ⇨ The objective of Tabu Search is to find the best solution possible within a reasonable amount of time. The algorithm can be used for a wide range of optimization problems, including scheduling, routing, allocation, and design problems. The algorithm is flexible and can be tailored to different optimization problems by modifying the objective function and the constraints.
- ⇨ Overall, the objective of Tabu Search is to provide a powerful optimization tool that can handle complex optimization problems efficiently and find high-quality solutions.

ADVANTAGES: -

The advantages of using Tabu Search for optimization problems are:

1**. Efficiently handles large search spaces**: Tabu Search is an efficient optimization algorithm that can handle large search spaces with many possible solutions. The algorithm can explore the search space efficiently and find high-quality solutions within a reasonable amount of time.

2**. Escapes local optima**: Tabu Search prevents the search from getting stuck in local optima by keeping track of the recently visited solutions in a tabu list. The algorithm can escape local optima and continue exploring the search space to find better solutions.

3. **Flexibility**: Tabu Search can be adapted to different optimization problems by modifying the objective function and constraints. The algorithm can be customized to fit specific problem requirements and constraints.

4. **Good balance between exploration and exploitation**: Tabu Search balances the exploration and exploitation of the search space by iteratively making small moves from the current solution to neighbouring solutions. The algorithm evaluates each neighbouring solution and selects the best one that is not on the tabu list.

5. **Global optimization**: Tabu Search is a global optimization algorithm that can find the global optimum, which is the best possible solution in the search space.

DISADVANTAGES: -

The disadvantages of using Tabu Search for optimization problems are:

1. **Parameter tuning**: Tabu Search requires setting many parameters, such as the size of the tabu list, the number of iterations, and the neighbourhood function. The performance of the algorithm can be sensitive to these parameters, and finding the optimal parameter settings can be challenging.

2. **Computationally intensive**: Tabu Search can be computationally intensive, especially for large search spaces. The algorithm can require a significant number of computational resources, such as memory and processing power.

3. **No guarantee of global optimum**: Tabu Search is a stochastic optimization algorithm that can converge to a local optimum. While the algorithm is designed to escape local optima, there is no guarantee that it will find the global optimum.

4. **Complex implementation**: Tabu Search can be challenging to implement, especially for complex optimization problems. The algorithm requires designing a suitable neighbourhood function, defining the tabu list, and managing the stopping criteria.

5. **Sensitivity to problem structure**: Tabu Search can be sensitive to the structure of the optimization problem. The algorithm may perform differently for different problem structures, which can make it difficult to generalize its performance.

ALGORITHM / PROCEDURE: -

The Tabu Search algorithm consists of the following steps:

1. Initialize the current solution as a feasible solution to the optimization problem.

2. Set the size of the tabu list and the number of iterations.

3. While the stopping criteria are not met, repeat the following steps:

    a. Generate a set of neighbouring solutions by applying a neighbourhood function to the current solution.

    b. Evaluate the quality of each neighbouring solution using the objective function.

    c. Select the best neighbouring solution that is not on the tabu list.

d. Update the tabu list by adding the current move and removing the oldest move if the tabu list is full.

e. If the selected neighbouring solution is better than the current solution, update the current solution.

f. If the selected neighbouring solution is worse than the current solution, accept it with a certain probability determined by a temperature parameter. The probability decreases as the algorithm progresses.

g. Update the temperature parameter according to a cooling schedule.

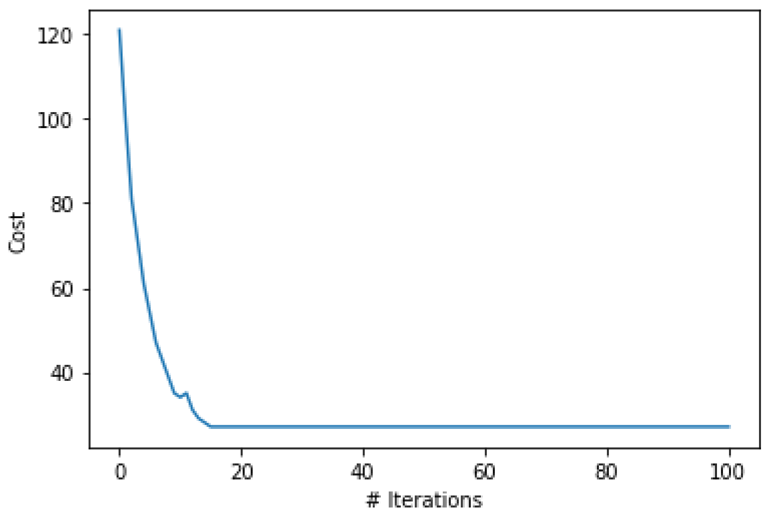
4. Return the best solution found during the search.

PSEUDOCODE: -

```
# Initialize the graph
import networkx as nx
import random
import matplotlib.pyplot as plt
import math

G = nx.complete_graph(25)
for (u,v) in G.edges():
    G.edges[u,v]['weight'] = random.randint(0,10)

plt.figure(figsize=(25,25))
nx.draw(G, with_labels=True)
plt.show()
# Define a function to calculate the tour cost
def cost_of_tour(G, tour):
    cost = 0
    for u, v in zip(tour, tour[1:]):
        cost += G[u][v]["weight"]
    cost += G[len(tour) - 1][0]["weight"]
    return cost
def get_best_neighbour(G, tour, tabu_history, tabu_limit, aspiration):
    best_neighbour = None
    best_neighbour_cost = math.inf
    # generate a list of all possible neighbours
    # a neighbour is just swapping the position of two nodes within the tour
    for i in range(len(G.nodes)):
        for j in range(len(G.nodes)):
            if i == j:
                continue

            # Swap the ith and jth nodes
            tmp_route = tour.copy()
            tmp = tmp_route[i]
            tmp_route[i] = tmp_route[j]
            tmp_route[j] = tmp
            tmp_cost = cost_of_tour(G, tmp_route)

            # This route is tabu, check aspiration
            if tuple(tmp_route) in tabu_history:
                if tabu_history[tuple(tmp_route)] > 0:
                    if tabu_history[tuple(tmp_route)] > aspiration:
                        continue

            if tmp_cost < best_neighbour_cost:
                best_neighbour_cost = tmp_cost
                best_neighbour = tmp_route
                tabu_history[tuple(best_neighbour)] = tabu_limit

    return best_neighbour
def tabu_search(
    G,
    initial_solution,
    num_iter,
    tabu_history,
    tabu_limit,
    aspiration,
    cost_function,
    neighbour_function,
    use_historical_best=False,
    use_tqdm = False
):
    best_solution = initial_solution
    historical_best = best_solution
    historical_best_cost = cost_function(G,historical_best)
    best_cost = cost_function(G, best_solution)
    states = [best_cost]
    if use_tqdm:
        pbar = tqdm(total=num_iter)
    for _ in range(num_iter):
        # Reduce counter for all tabu
        if use_tqdm: pbar.update()
        for x in tabu_history:
            tabu_history[x] -= 1
        tabu_history = {x: tabu_history[x] for x in tabu_history if tabu_history[x] > 0}


        best_solution = neighbour_function(
            G, best_solution, tabu_history, tabu_limit, aspiration
        )
        best_cost = cost_function(G, best_solution)
        if best_cost <= historical_best_cost:
            historical_best = best_solution
            historical_best_cost = best_cost
        states.append(best_cost)
    return best_solution, best_cost, states
# Initialize some parameters
aspiration = 2
tabu_history = {}
num_iterations = 100
tabu_limit = 5

# Initialize a random solution, and its cost
initial_solution = [*G.nodes()]
random.shuffle(initial_solution)
initial_cost = cost_of_tour(G, initial_solution)
print(f"Initial solution: {initial_solution}")
print(f"Initial cost: {initial_cost}")

best_solution, best_cost, states = tabu_search(
    G,
    initial_solution,
    num_iterations,
    tabu_history,
    tabu_limit,
    aspiration,
    cost_of_tour,
    get_best_neighbour,
)
print(f"Best Solution: {best_solution}")
print(f"Best Cost: {best_cost}")
plt.xlabel("# Iterations")
plt.ylabel("Cost")
plt.plot(states)
plt.show()
```

OUTPUT: -

```
Initial solution: [8, 5, 19, 9, 7, 16, 12, 23, 22, 6, 24, 11, 21, 14, 18, 2, 15, 0, 20, 1, 13, 4, 10, 3, 17]
Initial cost: 121


Best Solution: [14, 11, 15, 7, 9, 17, 1, 23, 24, 12, 18, 10, 5, 19, 22, 2, 8, 0, 20, 6, 13, 4, 21, 3, 16]
Best Cost: 27
```


Cost
# Iterations