International Journal of Statistics and Applied Mathematics 2025; 10(9): 29-32

International Journal of Statistics and Applied Mathematics

ISSN: 2456-1452
NAAS Rating (2025): 4.49
Maths 2025; 10(9): 29-32

https://www.mathsjournal.com
Received: 13-08-2025
Accepted: 15-09-2025

Dr. Pradeep Jha
Visiting Faculty, Department of Mathematics, LJ Institute of Engineering and Technology, Ahmedabad, Gujarat, India

Shivansh N Jha
Student, Department of Computer Science and Engineering, Semester 7, LJ Institute of Engineering and Technology, Ahmedabad, Gujarat, India

Corresponding Author:
Dr. Pradeep Jha
Visiting Faculty, Department of Mathematics, LJ Institute of Engineering and Technology, Ahmedabad, Gujarat, India

Assignment problem (maximization): A new approach

Pradeep Jha and Shivansh N Jha

DOI: https://www.doi.org/10.22271/maths.2025.v10.i10a.2176

Abstract
This paper introduces a novel and faster approach to finding the optimal feasible solution of assignment problem in maximization scenarios. The proposed method, termed 'Jha's Method' offers a more practical alternative over the long-known Hungarian method. It eliminates the need for the calculation of the opportunity loss matrix while maintaining computational efficiency. The Jha's method, it's optimality test, and its computer code using Python language are unique till now.

Keywords: Assignment problem, jha's method, optimization, maximization, python

1. Introduction
In this note we attempt to present the salient features of a new assignment technique designed to reach optimal feasible solution. Keeping the prime objective of efficiency in mind, we aim towards sharing this new approach that focuses on real-life situational problems.
The assignment problem (minimization type) is fundamentally an optimization problem in operations research, where the objective is to assign M agents to M tasks such that the total cost of the assignment is minimized (or profit is maximized) and ensuring that each agent is assigned exactly one of the M tasks.

1.1 Traditional Algorithm - The Hungarian Method
In 1955, the Hungarian method, refined by Harold Kuhn, was developed to find the optimal solution for assignment problems, primarily dealing with minimization cases. However, by applying the opportunity loss (or regret) technique, it can solve maximization problems on converting all cell entries of the given maximization assignment problem to opportunity loss matrix and applying the standard minimization algorithm to the resulting matrix.
In this framework, both minimization and maximization assignments fall under the same general structure of the Hungarian Method, though this conversion to opportunity loss matrix adds computational overhead.

1.2 Mathematical Foundation
The Hungarian Method solves an assignment problem represented by an $M \times M$ matrix (where $M \in \mathbb{N}$) with M^2 entries - each entry denotes the cost (or profit) of assigning a task to an agent. In the assignment problem, a linear programming model, has M row and M column constraints. With one dependent constraint, this leaves $2M - 1$ independent equality constraints. Therefore, a basic feasible solution (B.F.S) has exactly $2M - 1$ basic variables. In the B.F.S, exactly M cells are assigned, often resulting in a highly degenerate solution.
A positive feature of the traditional method is its ability to identify multiple optimal solutions when they exist.

2. The New Approach - Jha's Method
In this section, we describe the complete procedure for finding a basic feasible solution to the assignment problem (maximization type)

2.1 Initial Basic Feasible Solution
We describe the procedure to obtain the Basic Feasible Solution (B.F.S.). Given an $M \times M$ assignment matrix with rows denoted as $R_1, R_2, \ldots, R_M$ and columns as $C_1, C_2, \ldots, C_M$, the steps are as follows:

1. Identify the maximum entry in the matrix, denoted by x_{ij}. Mark this entry and eliminate the corresponding i^{th} row and j^{th} column
2. The elimination results in a reduced square matrix of order $(M-1)$. Determine the maximum entry in this reduced matrix and eliminate its corresponding row and column.
3. Repeat this procedure iteratively until all rows and columns are eliminated.

2.2 Key Observation
On completion of the above operations, we observe that exactly one entry is marked in each row and each column. This procedure yields a basic feasible solution (each agent has been assigned a marked entry in its respective column), which serves as the starting point for applying Jha's optimality test. the above procedure ends up int the following I.B.F.S

Row	Entry
R_1	x_{1j}
R_2	x_{2q}
$\vdots$	$\vdots$
R_M	x_{Mz}

As shown above, the result leads to an initial basic feasible solution (I.B.F.S).

3. Jha's Optimality Test
Upon finding the I.B.F.S as mentioned, we describe the procedure that leads to optimal basic feasible solution. The optimality test examines each marked entry and determines whether improvements are possible.

3.1 Test Procedure
Beginning with the first marked entry in the 1st row and j^{th} column; that is x_{1j}.

1. Identify potential improvements
Search for all values numerically higher than x_{1j} in the 1st row. If no higher values exist, then x_{1j}, as of now, remains optimal for the 1st row.

2. Calculate improvement parameters
For the first higher value x_{1k} found in column k (where $k \neq j$), calculate:

$$A_1 = |x_{1k} - x_{1j}|$$

3. Examining column impact
For this column k, identify the current marked entry x_{tk} in column k (row t) and calculate:

$$B_1 = |x_{tk} - x_{tj}|$$

4. Compute net benefit
For each potential improvement, calculate the net benefit as:

$$Z_1 = A_1 - B_1$$

5. Calculate Z values
In the same way continue calculating $Z_1, Z_2, Z_3, \ldots$ for other numerically higher values than x_{1j}

6. Optimality condition

- If all $Z_k \le 0$ ($k = 1,2,3 \ldots$) Keep x_{1j} unchanged (when $Z_k = 0$, multiple optimal solutions exist)
- If $Z_k > 0$ For the column k and row t yielding maximum Z_k, replace x_{1j} with x_{1k} and replace x_{tk} with x_{tj}

3.2 Iteration Process
we repeat the above steps for the marked entries of the remaining rows $R_2, R_3, \ldots, R_M$ During this process if at least one swap occurs then we have to begin the procedure described above in Section 3.1 (Test Procedure).

4. Convergence to Optimal Solution
The above mentioned Jha's optimality test is terminated if there is no swap executed (checking on from R_1 to R_M) After completing Jha's optimality test for each entry in the initial basic feasible solution, the algorithm converges to an optimal basic feasible solution. The test ensures that there are no further beneficial exchanges and yields optimality. It has been concluded that the final marked enteries for each row constitutes the optimal basic feasible Solution. We sound this logic by an illustration

5 Illustration
Consider the given square matrix *M* of order 5:

$$M = \begin{array}{c|ccccc} & \boldsymbol{C_1} & \boldsymbol{C_2} & \boldsymbol{C_3} & \boldsymbol{C_4} & \boldsymbol{C_5} \\ \boldsymbol{R_1} & 10 & 18 & 17 & 20 & 24 \\ \boldsymbol{R_2} & 24 & 25 & 15 & 22 & 19 \\ \boldsymbol{R_3} & 17 & 20 & 19 & 24 & 21 \\ \boldsymbol{R_4} & 19 & 22 & 16 & 17 & 21 \\ \boldsymbol{R_5} & 14 & 18 & 20 & 17 & 16 \end{array}$$

Step 1
Following the prescribed method in 2.1, we systematically identify the maximum entries. Maximum entry = 25 at cell position (2,2). Mark and eliminate Row 2 and Column 2.

$$\begin{bmatrix} 10 & \times & 17 & 20 & 24 \\ \times & \boxed{25} & \times & \times & \times \\ 17 & \times & 19 & 24 & 21 \\ 19 & \times & 16 & 17 & 21 \\ 14 & \times & 20 & 17 & 16 \end{bmatrix}$$

Step 2
In the remaining matrix, maximum entry = 24 at cell position (1,5). Mark and eliminate Row 1 and Column 5.

$$\begin{bmatrix} \times & \times & \times & \times & \boxed{24} \\ \times & \boxed{25} & \times & \times & \times \\ 17 & \times & 19 & 24 & \times \\ 19 & \times & 16 & 17 & \times \\ 14 & \times & 20 & 17 & \times \end{bmatrix}$$

Step 3
Similarly, we get the final matrix with all maximum marked entries in order:

$$\begin{bmatrix} \times & \times & \times & \times & \boxed{24} \\ \times & \boxed{25} & \times & \times & \times \\ \times & \times & \times & \boxed{24} & \times \\ \boxed{19} & \times & \times & \times & \times \\ \times & \times & \boxed{20} & \times & \times \end{bmatrix}$$

Initial Basic Feasible Solution depicted as follow:

Agent	Task	Profit
1	5	24
2	2	25
3	4	24
4	1	19
5	3	20

Total Cost = 112... (a)

Initial Basic Feasible Solution: 112

Step 4: Optimality test
On the initial basic feasible solution denoted above (a), we apply optimality test as follows
Examining 1st row values, the marked value $x_{15} = 24$, There is no numerically higher entry than 24 in this row. Then Move to next row.

- **Step 5:** Examining 2nd row values, the marked value $x_{22} = 25$, There is no numerically higher entry than 24 in this row. Move to next row.
- **Step 6:** Examining 3rd row values, the marked value $x_{34} = 24$, There is no numerically higher entry than 24 in this row. Move to next row.
- **Step 7:** Examining 4th row values, marked value is $x_{41} = 19$. Entries numerically higher than 19 exist, which are 22, and 21

We now calculate new profit for each of them
i) Current $x_{41} = 19$, potential candidate $x_{42} = 22$

$A_1 = |x_{42} - x_{41}| = 3$

Impact on Column 2: currently assigned maximum entry $x_{22} = 25$, candidate if swap occurs $x_{21} = 24$
$B_1 = |x_{22} - x_{21}| = 1$
Z_1 (Net gain) = $X - Y = 3 - 1 = 2$

ii) Current $x_{41} = 19$, potential candidate $x_{45} = 21$
$A_2 = |x_{45} - x_{41}| = 2$

Impact on Column 5: currently assigned maximum entry $x_{15} = 24$, candidate if a swap occurs $x_{11} = 10$
$B_2 = |x_{15} - x_{11}| = 14$
Z_2 (Net gain) = $A - B = 2 - 14 = -12$

The maximum net gain possible is 2, Z_1 swap is comparatively beneficial.
Make the swap in marked entries:

- x_{42} becomes the new marked entry for Row 4
- x_{21} becomes the new marked entry for Row 2

$$\begin{bmatrix} \times & \times & \times & \times & \boxed{24} \\ \boxed{\mathbf{24}} & \times & \times & \times & \times \\ \times & \times & \times & \boxed{24} & \times \\ \times & \boxed{\mathbf{22}} & \times & \times & \times \\ \times & \times & \boxed{20} & \times & \times \end{bmatrix}$$

Step 8: Since a swap occurred before completion of one full iteration, we check again from Row 1.
Examining 1st row values, the marked value $x_{15} = 24$, There is no numerically higher entry than 24 in this row. Then Move to next row.

Step 9: Examining 2nd row values, marked value is $x_{21} = 24$. Entry numerically greater than 24 exists, which is 25
i) Current $x_{21} = 24$, potential candidate $x_{22} = 25$
$A_1 = |x_{22} - x_{21}| = 1$

Impact on Column 2: currently assigned $x_{42} = 22$, candidate if swap occurs $x_{41} = 19$
$B_1 = |x_{42} - x_{41}| = 3$
Z_1 (Net gain) = $A - B = 1 - 3 = -2$

The maximum net gain is -2, but since it's negative, there are no beneficial swaps for row 2. We keep the current assignment x_{21}.

Step 10: Examining 3rd row values, marked value is $x_{34} = 24$, no entries numerically greater than 24 exist.We Move to next row.

Step 11: Similarly after examining the remaining rows, we can see that one full full iteration completed without any swaps. Hence, we have reached optimality.

Final Optimal Solution

$x_{15} = 24$
$x_{21} = 24$
$x_{34} = 24$
$x_{42} = 22$
$x_{53} = 20$

Optimal solution = 114

Conclusion
The method we have found and exemplified above has many salient features:

- It is simpler than the original method
- It quickly approaches to optimality through our optimality test
- It is a general method that can be applied to any $M \times M$ matrix (Maximization type)
- It is more practical and amenable to computer programming (shown in Appendix)
- It is open to further research

References

1. Dr. Pradeep J Jha. Operations Research. McGraw Hill Education (India) Private Limited. 2014. ISBN: 978-1-25-902673-7.
2. Afrooz HD, Dr. Hossen MA. New Proposed Method for Solving Assignment Problem and Comparative Study with the Existing Methods. IOSR Journal of Mathematics (IOSR-JM). 2017;13(2):84-88. e-ISSN: 2278-5728, p-ISSN: 2319-765X.
3. Dr. Sharma VK, Sachdeva P. Comparative Study of Various Approaches for Solving the Assignment Problem, IJFANS International Journal of Food and Nutritional Sciences, UGC CARE Listed (Group -I) Journal. 2022;11:10.

4. Gothi MM, Patel RG, Patel BS. Optimal Solution to The Assignment Problem, Annals of Mathematics and Computer Science. 2023;16:112-122. ISSN: 2789-7206.
5. Supian S, Wahyuni S, Nahar J, Subiyanto. "Optimization of Personnel Assignment Problem Based on Traveling Time by Using Hungarian Methods: Case Study on the Central Post Office Bandung", 4th International Conference on Operational Research (Interi OR), IOP Conf. Series: Materials Science and Engineering. 2018;300:012005. Doi:10.1088/1757-899X/300/1/012005.
6. Prof. Prajapati R, Dr. Haque A, Dr. Jain J, Prof. Singh S. A study on solving Assignment Problem, VIVA-Tech International Journal for Research and Innovation ISSN(Online): 2581-7280, 2021, 1(4).
7. Kadhim HJ, Shiker MAK, Al-Dallal HAH. A New Technique for Finding the Optimal Solution to Assignment Problems with Maximization Objective Function, Journal of Physics: Conference Series. 2021;1963:012104.

Appendix: Python Implementation

```
def Jhas_optimality_criterion(matrix):
M = len(matrix)
marked_entries = []
used_rows = set()
used_cols = set()
for i in range(M):
max_val = -999999
max_row = -1
max_col = -1
for r in range(M):
if r in used_rows:
continue
for c in range(M):
if c in used_cols:
continue
if matrix[r][c] > max_val:
max_val = matrix[r][c]
max_row = r
max_col = c
marked_entries.append((max_row, max_col, max_val))
used_rows.add(max_row)
used_cols.add(max_col)
while True:
row_to_col = {}
col_to_row = {}
for row, col, val in marked_entries:
row_to_col[row] = col
col_to_row[col] = row
swap_happened = False
for current_row in range(M):
current_col = row_to_col[current_row]
current_val = matrix[current_row][current_col]
best_profit = 0
best_col = -1
for k in range(M):
if k == current_col:
continue
candidate_val = matrix[current_row][k]
if candidate_val >= current_val:
gain = candidate_val - current_val
displaced_row = col_to_row[k]
loss = (matrix[displaced_row][k] -
matrix[displaced_row][current_col])
profit = gain - loss
if profit > best_profit:
best_profit = profit
best_col = k
if best_profit > 0:
displaced_row = col_to_row[best_col]
row_to_col[current_row] = best_col
row_to_col[displaced_row] = current_col
col_to_row[current_col] = displaced_row
col_to_row[best_col] = current_row
swap_happened = True
if not swap_happened:
break
marked_entries = []
for row in range(M):
col = row_to_col[row]
val = matrix[row][col]
marked_entries.append((row, col, val))
return marked_entries
```