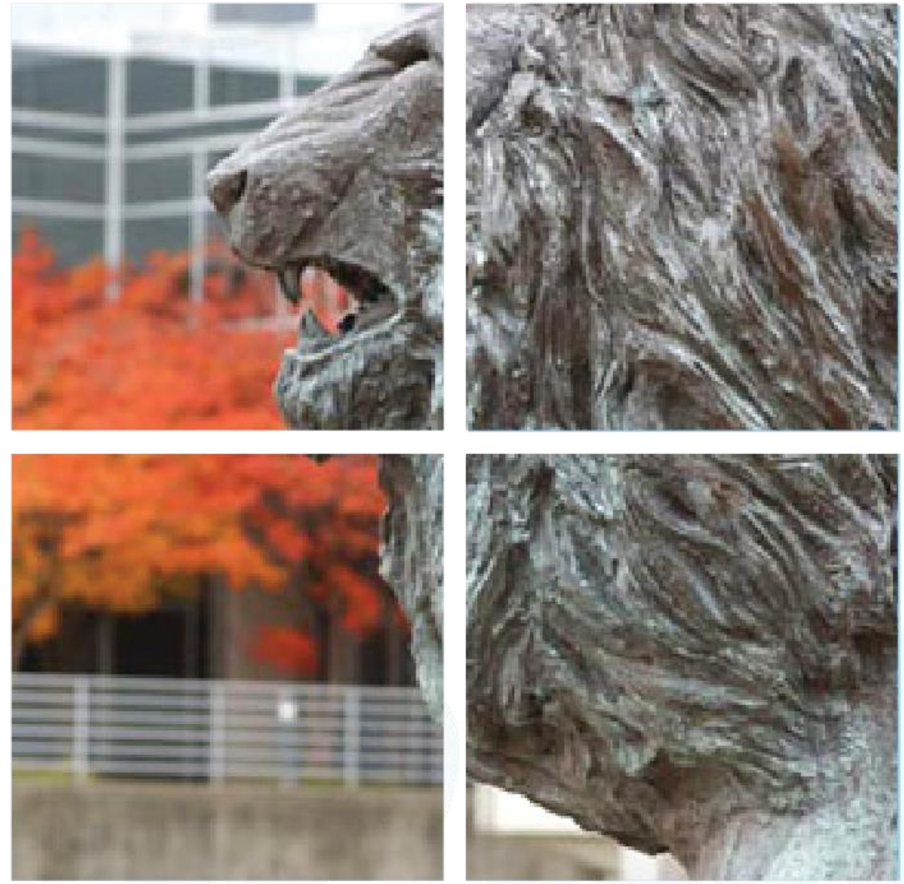
OLD DOMINION UNIVERSITY

IDEA FUSION



Hungarian Optimum Assignment Algorithm with Java Computer Animation

1 April 2016

Ivan Makohon
Graduate Student
Modeling, Simulation & Visualization Engineering (MSVE)
Old Dominion University
Norfolk, Virginia 23529, USA
imako001@odu.edu

Duc T. Nguyen
Professor
Civil & Environmental Engineering (CEE) and MSVE Departments
Old Dominion University
Norfolk, Virginia 23529, USA

Mecit Cetin
Associate Professor
CEE Department
Old Dominion University
Norfolk, Virginia 23529, USA
mcetin@odu.edu

Manwo Ng
Assistant Professor
Department of Information Technology and Decision
Old Dominion University
Norfolk, Virginia 23529, USA
mng@odu.edu

Overview

- Java Computer Animations to provide a precise and clear detailed Step-by-Step teachings of the Hungarian Algorithm:
  - Provides Text-To-Speech (TTS) to animate the voice in several languages (i.e. English, Spanish)
  - Provides importing/exporting of the Matrix [A] data
  - Provides a views of the Matrix [A] data being modified
  - Provides a Step-by-Step logging results (Export Capable)
  - Provides voice narrative filtering (i.e. None, Terse, Verbose)

Overview

- Solves the Minimum Optimum Assignment (By Default)

$$[A] = \begin{bmatrix} \boxed{1.0} & 2.0 & 3.0 \\ 3.0 & \boxed{3.0} & 3.0 \\ 3.0 & 3.0 & \boxed{2.0} \end{bmatrix}$$

  - Minimum Optimum Assignment = (1.0 + 3.0 + 2.0) = 6.0

- Solves the Maximum Optimum Assignment

$$[A] = \begin{bmatrix} 1.0 & 2.0 & \boxed{3.0} \\ 3.0 & \boxed{3.0} & 3.0 \\ \boxed{3.0} & 3.0 & 2.0 \end{bmatrix}$$

  - Maximum Optimum Assignment = (3.0 + 3.0 + 3.0) = 9.0

Overview

- Solves for ANY size Matrix (i.e. 3x2, 3x3, 2x3, etc)

$$[A] = \begin{bmatrix} 2.0 & 3.0 & 0.0 \\ 3.0 & 3.0 & 0.0 \\ 3.0 & 2.0 & 0.0 \end{bmatrix} \qquad [B] = \begin{bmatrix} 1.0 & 2.0 & 3.0 \\ 3.0 & 3.0 & 3.0 \\ 3.0 & 3.0 & 2.0 \end{bmatrix}$$

$$[C] = \begin{bmatrix} 1.0 & 2.0 & 3.0 \\ 3.0 & 3.0 & 3.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix}$$

*yellow-highlight denotes added “dummy” rows/columns

Hungarian Algorithm Steps

- Initialization:

N = Matrix [A] Dimension

MNOL = Minimum Number of Lines

MUN = Minimum Uncovered Number

$$\text{Matrix [A]} = \begin{bmatrix} 1.0 & 2.0 & 3.0 \\ 3.0 & 3.0 & 3.0 \\ 3.0 & 3.0 & 2.0 \end{bmatrix}$$

Hungarian Algorithm Steps

- Step 0A – Determine if the Matrix [A] is squared.
  - The user's input of the matrix [A] is assumed to be a SQUARE matrix, and the problem is assumed to be a Minimization Problem. If matrix [A] is a Rectangular matrix, then either dummy row(s), or column(s) need to be added in order to make it become a square matrix.
  - Examples:

$$[A] = \begin{bmatrix} 1.0 & 2.0 & 0.0 \\ 3.0 & 3.0 & 0.0 \\ 3.0 & 3.0 & 0.0 \end{bmatrix} \qquad [A] = \begin{bmatrix} 1.0 & 2.0 & 3.0 \\ 3.0 & 3.0 & 3.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix}$$

*yellow-highlight denotes added "dummy" rows/columns

Hungarian Algorithm Steps

- Step 0B – Determine if the Matrix [A] is being solved as Minimization (By Default) or Maximization problem (User Input).
  - If the problem is a Maximization problem, then it can be transformed/converted to a Minimization problem, as follows:

Replace each cell entry A_{ij} with negative A_{ij} + *C*,
where C equals the maximum cell value (C = 3)

$$[A] = \begin{bmatrix} 1.0 & 2.0 & 3.0 \\ 3.0 & 3.0 & 3.0 \\ 3.0 & 3.0 & 2.0 \end{bmatrix} \Rightarrow [A] = \begin{bmatrix} 2.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 1.0 \end{bmatrix}$$

Maximization matrix   Minimization matrix

Hungarian Algorithm Steps

- Step 1 – Subtract each row with its corresponding minimum value in Matrix [A].

$$[A] = \begin{bmatrix} \boxed{1.0} & 2.0 & 3.0 \\ \boxed{3.0} & 3.0 & 3.0 \\ 3.0 & 3.0 & \boxed{2.0} \end{bmatrix} \Rightarrow [A] = \begin{bmatrix} 0.0 & 1.0 & 2.0 \\ 0.0 & 0.0 & 0.0 \\ 1.0 & 1.0 & 0.0 \end{bmatrix}$$

Hungarian Algorithm Steps

- Step 2 – Subtract each <u>column</u> with its corresponding minimum value in Matrix [A].

$$[A] = \begin{bmatrix} \boxed{0.0} & 1.0 & 2.0 \\ 0.0 & \boxed{0.0} & \boxed{0.0} \\ 1.0 & 1.0 & 0.0 \end{bmatrix} \Rightarrow [A] = \begin{bmatrix} 0.0 & 1.0 & 2.0 \\ 0.0 & 0.0 & 0.0 \\ 1.0 & 1.0 & 0.0 \end{bmatrix}$$

Hungarian Algorithm Steps

- Step 3 - Cover **ALL** zeros with the MNOL drawn through columns and rows.

$$[A] = \begin{bmatrix} 0.0 & 1.0 & 2.0 \\ 0.0 & 0.0 & 0.0 \\ 1.0 & 1.0 & 0.0 \end{bmatrix}$$

Cover Column 1 with a line
Cover Column 3 with a line
Cover Row 2 with a line

Hungarian Algorithm Steps

- Step 3.1 – Compute the MUN in the matrix.
  - Determine the smallest entry not covered by any line (MUN = 1.0)
  - Subtract the MUN from each uncovered value
  - Add the MUN to each value covered by the two intersecting lines

$$[A] = \begin{bmatrix} 0.0 & 1.0 & 2.0 \\ 0.0 & 0.0 & 0.0 \\ 1.0 & 1.0 & 0.0 \end{bmatrix} \Rightarrow [A] = \begin{bmatrix} 0.0 & 0.0 & 2.0 \\ \boxed{1.0} & 0.0 & \boxed{1.0} \\ 1.0 & 0.0 & 0.0 \end{bmatrix}$$

MUN = 1.0

Rectangles denotes added entries

Ovals denotes subtracted entries

Hungarian Algorithm Steps

- Step 3.2 – Compute the MNOL in the Matrix.
  - If MNOL greater-than or equal-to the Matrix [A] Dimension then the Matrix is converged; continue to Step 4.
  - If MNOL is less-than N; continue to Step 3.

$$[A] = \begin{bmatrix} 0.0 & 0.0 & 2.0 \\ 1.0 & 0.0 & 1.0 \\ 1.0 & 0.0 & 0.0 \end{bmatrix}$$

MNOL = 3 is greater-than equal-to N = 3

Matrix [A] is Converged

Continue to Step 4

Hungarian Algorithm Steps

- Step 4 – Optimum Assignment (Minimum)
  - Starting with the top row and work your way downwards as you make assignments.
  - An assignment can be uniquely made when there is exactly one zero in a row.

$$[A] = \begin{bmatrix} \boxed{0.0} & 0.0 & 2.0 \\ 1.0 & \boxed{0.0} & 1.0 \\ 1.0 & 0.0 & \boxed{0.0} \end{bmatrix} \Rightarrow [A] = \begin{bmatrix} \boxed{1.0} & 2.0 & 3.0 \\ 3.0 & \boxed{3.0} & 3.0 \\ 3.0 & 3.0 & \boxed{2.0} \end{bmatrix}$$

Assignment	Score
Worker #1 assigned to Job #1	1.0
Worker #2 assigned to Job #2	3.0
Worker #3 assigned to Job #3	2.0

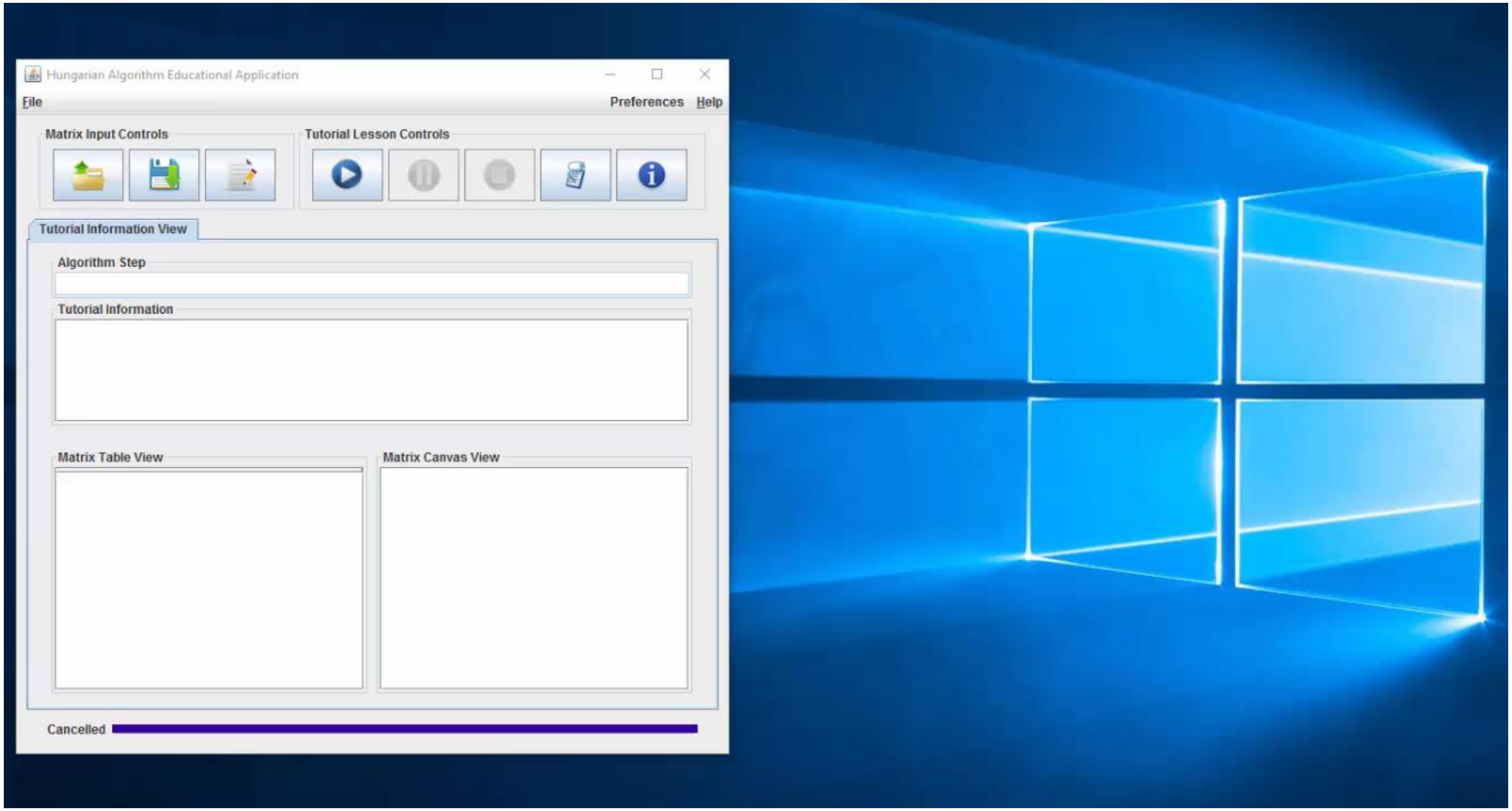
Optimum Minimum Score:	6.0

Demo – Minimization Optimum Assignment Problem

- Demo video will show the opening of an existing stored Matrix [A] (below) and then solve for the Minimization Optimum Assignment.

$$[A] = \begin{bmatrix} 1.0 & 2.0 & 3.0 \\ 3.0 & 3.0 & 3.0 \\ 3.0 & 3.0 & 2.0 \end{bmatrix}$$

Demo Video



Hungarian Algorithm Educational Application
File
Preferences
Help
Matrix Input Controls
Tutorial Lesson Controls
Tutorial Information View
Algorithm Step
Tutorial Information
Matrix Table View
Matrix Canvas View
Cancelled

Questions/Comments

Additional Demos Online:
http://www.lions.odu.edu/~imako001/

Appendix A: 4x3 Maximization Problem Step-by-Step

- Starting with a user input “tall” (4x3) Matrix [A], and the original problem is to “maximize” the “profit”.

$$[A] = \begin{bmatrix} 1.0 & 2.0 & 3.0 \\ 2.0 & 5.0 & 4.0 \\ 1.0 & 6.0 & 8.0 \\ 3.0 & 7.0 & 2.0 \end{bmatrix}$$

Hungarian Algorithm Steps

- Step 0A – Determine if the Matrix [A] is squared.
  - The Matrix [A] is NOT squared since it's a 4x3
  - Add a "Dummy" column with 0.0 values so that Matrix [A] becomes a 4x4 matrix

$$[A] = \begin{bmatrix} 1.0 & 2.0 & 3.0 \\ 2.0 & 5.0 & 4.0 \\ 1.0 & 6.0 & 8.0 \\ 3.0 & 7.0 & 2.0 \end{bmatrix} \Rightarrow [A] = \begin{bmatrix} 1.0 & 2.0 & 3.0 & 0.0 \\ 2.0 & 5.0 & 4.0 & 0.0 \\ 1.0 & 6.0 & 8.0 & 0.0 \\ 3.0 & 7.0 & 2.0 & 0.0 \end{bmatrix}$$

*yellow-highlight denotes the added "dummy" columns with zeros

Hungarian Algorithm Steps

- Step 0B – Determine if the Matrix [A] is being solved as Minimization (By Default) or Maximization problem (User Input).
  - If the problem is a Maximization problem, then it can be transformed/converted to a Minimization problem, as follows:

Replace each cell entry A_{ij} with negative A_{ij} + *C*, *where C equals the maximum cell value (C = 8)*

$$[A] = \begin{bmatrix} 1.0 & 2.0 & 3.0 & 0.0 \\ 2.0 & 5.0 & 4.0 & 0.0 \\ 1.0 & 6.0 & 8.0 & 0.0 \\ 3.0 & 7.0 & 2.0 & 0.0 \end{bmatrix} \Rightarrow [A] = \begin{bmatrix} 7.0 & 6.0 & 5.0 & 8.0 \\ 6.0 & 3.0 & 4.0 & 8.0 \\ 7.0 & 2.0 & 0.0 & 8.0 \\ 5.0 & 1.0 & 6.0 & 8.0 \end{bmatrix}$$

Maximization matrix

Minimization matrix

Hungarian Algorithm Steps

- Step 1 – Subtract each <u>row</u> with its corresponding minimum value in Matrix [A].

$$[A] = \begin{bmatrix} 7.0 & 6.0 & \boxed{5.0} & 8.0 \\ 6.0 & \boxed{3.0} & 4.0 & 8.0 \\ 7.0 & 2.0 & \boxed{0.0} & 8.0 \\ 5.0 & \boxed{1.0} & 6.0 & 8.0 \end{bmatrix} \Rightarrow [A] = \begin{bmatrix} 2.0 & 1.0 & 0.0 & 3.0 \\ 3.0 & 0.0 & 1.0 & 5.0 \\ 7.0 & 2.0 & 0.0 & 8.0 \\ 4.0 & 0.0 & 5.0 & 7.0 \end{bmatrix}$$

Subtract 5.0 from each value in Row 1
Subtract 3.0 from each value in Row 2
Subtract 0.0 from each value in Row 3
Subtract 1.0 from each value in Row 4

Hungarian Algorithm Steps

- Step 2 – Subtract each column with its corresponding minimum value in Matrix [A].

$$[A] = \begin{bmatrix} \boxed{2.0} & 1.0 & \boxed{0.0} & \boxed{3.0} \\ 3.0 & \boxed{0.0} & 1.0 & 5.0 \\ 7.0 & 2.0 & 0.0 & 8.0 \\ 4.0 & 0.0 & 5.0 & 7.0 \end{bmatrix} \Rightarrow [A] = \begin{bmatrix} 0.0 & 1.0 & 0.0 & 0.0 \\ 1.0 & 0.0 & 1.0 & 2.0 \\ 5.0 & 2.0 & 0.0 & 5.0 \\ 2.0 & 0.0 & 5.0 & 4.0 \end{bmatrix}$$

Subtract 2.0 from each value in Column 1
Subtract 0.0 from each value in Column 2
Subtract 0.0 from each value in Column 3
Subtract 3.0 from each value in Column 4

Hungarian Algorithm Steps

- Step 3 - Cover <u>ALL</u> zeros with the MNOL drawn through columns and rows.

$$[A] = \begin{bmatrix} 0.0 & 1.0 & 0.0 & 0.0 \\ 1.0 & 0.0 & 1.0 & 2.0 \\ 5.0 & 2.0 & 0.0 & 5.0 \\ 2.0 & 0.0 & 5.0 & 4.0 \end{bmatrix}$$

Cover Column 2 with a line
Cover Column 3 with a line
Cover Row 1 with a line

Hungarian Algorithm Steps

- Step 3.1 – Compute the MUN in the matrix.
  - Determine the smallest entry not covered by any line (MUN = 1.0)
  - Subtract the MUN from each uncovered value
  - Add the MUN to each value covered by the two intersecting lines

$$[A] = \begin{bmatrix} 0.0 & 1.0 & 0.0 & 0.0 \\ 1.0 & 0.0 & 1.0 & 2.0 \\ 5.0 & 2.0 & 0.0 & 5.0 \\ 2.0 & 0.0 & 5.0 & 4.0 \end{bmatrix} \Rightarrow [A] = \begin{bmatrix} 0.0 & \boxed{2.0} & \boxed{1.0} & 0.0 \\ 0.0 & 0.0 & 1.0 & 1.0 \\ 4.0 & 2.0 & 0.0 & 4.0 \\ 1.0 & 0.0 & 5.0 & 3.0 \end{bmatrix}$$

MUN = 1.0

Rectangles denotes added entries

Ovals denotes subtracted entries

Hungarian Algorithm Steps

- Step 3.2 – Compute the MNOL in the Matrix.
  - If MNOL greater-than or equal-to the Matrix [A] Dimension then the Matrix is converged; continue to Step 4.
  - If MNOL is less-than N; continue to Step 3.

$$[A] = \begin{bmatrix} 0.0 & 2.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 1.0 & 1.0 \\ 4.0 & 2.0 & 0.0 & 4.0 \\ 1.0 & 0.0 & 5.0 & 3.0 \end{bmatrix}$$

MNOL = 3 is **NOT** greater-than equal-to N = 4

Matrix [A] is NOT Converged

Clear ALL lines & Repeat Step 3

Hungarian Algorithm Steps

- Step 3 – Cover <u>**ALL**</u> zeros with the MNOL drawn through columns and rows (Iteration #2).

$$[A] = \begin{bmatrix} 0.0 & 2.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 1.0 & 1.0 \\ 4.0 & 2.0 & 0.0 & 4.0 \\ 1.0 & 0.0 & 5.0 & 3.0 \end{bmatrix}$$

Cover Column 3 with a line
Cover Column 2 with a line
Cover Row 1 with a line
Cover Column 1 with a line

Hungarian Algorithm Steps

- Step 3.1 – Compute the MUN in the matrix.
  - Determine the smallest entry not covered by any line (MUN = 1.0)
  - Subtract the MUN from each uncovered value
  - Add the MUN to each value covered by the two intersecting lines

$$[A] = \begin{bmatrix} 0.0 & 2.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 1.0 & 1.0 \\ 4.0 & 2.0 & 0.0 & 4.0 \\ 1.0 & 0.0 & 5.0 & 3.0 \end{bmatrix} \qquad [A] = \begin{bmatrix} \boxed{1.0} & \boxed{3.0} & \boxed{2.0} & 0.0 \\ 0.0 & 0.0 & 1.0 & 0.0 \\ 4.0 & 2.0 & 0.0 & 3.0 \\ 1.0 & 0.0 & 5.0 & 2.0 \end{bmatrix}$$

MUN = 1.0

Rectangles denotes added entries

Ovals denotes subtracted entries

Hungarian Algorithm Steps

- Step 3.2 – Compute the MNOL in the Matrix.
  - If MNOL greater-than or equal-to the Matrix [A] Dimension then the Matrix is converged; continue to Step 4.
  - If MNOL is less-than N; continue to Step 3.

$$[A] = \begin{bmatrix} 1.0 & 3.0 & 2.0 & 0.0 \\ 0.0 & 0.0 & 1.0 & 0.0 \\ 4.0 & 2.0 & 0.0 & 3.0 \\ 1.0 & 0.0 & 5.0 & 2.0 \end{bmatrix}$$

MNOL = 4 is greater-than equal-to N = 4

Matrix [A] is Converged

Continue to Step 4

Hungarian Algorithm Steps

- Step 4 – Optimum Assignment (Maximum)
  - Starting with the top row and work your way downwards as you make assignments.
  - An assignment can be uniquely made when there is exactly one zero in a row.

$$[A] = \begin{bmatrix} 1.0 & 3.0 & 2.0 & \boxed{0.0} \\ \boxed{0.0} & 0.0 & 1.0 & 0.0 \\ 4.0 & 2.0 & \boxed{0.0} & 3.0 \\ 1.0 & \boxed{0.0} & 5.0 & 2.0 \end{bmatrix} \Rightarrow [A] = \begin{bmatrix} 1.0 & 2.0 & 3.0 & \boxed{0.0} \\ \boxed{2.0} & 5.0 & 4.0 & 0.0 \\ 1.0 & 6.0 & \boxed{8.0} & 0.0 \\ 3.0 & \boxed{7.0} & 2.0 & 0.0 \end{bmatrix}$$

<u>Assignment</u>	<u>Score</u>
Worker #1 assigned to Job #4	0.0
Worker #2 assigned to Job #1	2.0
Worker #3 assigned to Job #3	8.0
Worker #4 assigned to Job #2	7.0

Optimum <u>Maximum</u> Score:	17.0