Hungarian Optimum Assignment Algorithm with Java Computer Animation

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

Duc T. Nguyen
Professor
CEE and MSVE Departments
Old Dominion University
Norfolk, Virginia 23529, USA
dnguyen@odu.edu

Mecit Cetin
Associate Professor
Civil & Environmental Engineering (CEE) Department
Old Dominion University
Norfolk, Virginia 23529, USA
mcetin@odu.edu

ManWo Ng
Assistant Professor
Strome College of Business
Old Dominion University
Norfolk, Virginia 23529, USA
mng@odu.edu

***Abstract*—The classical, popular Hungarian algorithm for solving the "optimum assignment" problems (with its broad engineering/science applications) has been well-documented in the literature. Other (more efficient) variations of the Hungarian algorithm have also been extensively studied by the research communities. In this paper, the basic Hungarian algorithm is re-visited, with the ultimate goal of developing a useful, user friendly, attractive Java computer animation for "effectively teaching" this basic/important optimum assignment algorithm. The final product from this work will help both the students and their instructor to not only mastering this technical subject, but also provide valuable tool for obtaining the solutions for homework assignments, class examinations, self-assessment tools, etc. A demo video of the Hungarian Algorithm's animation and result can be viewed online from any web browser using the website provided in reference [9].**

Keywords—Hungarian Algorithm; Optimum Assignment; Java; Animation

I. INTRODUCTION

Recognizing the steadily decline in US Science Technology Engineering Mathematics (STEM) interests and enrollments, the National Science Foundation (NSF) and the White House have developed national strategies and provided the significant budget to STEM education research [1-2] in the past years, with the ultimate goals to improve both the quality and number of highly trained US educators, student's workforce in STEM topics, in today highly competitive global markets. With the explosions of internet's capability and availability, it is even more critical to effectively train this future USA-STEM work-force and/or to develop effective STEM related teaching tools to reach a maximum possible number of "distance learners/audiences".

Various teaching philosophies have been proposed, tested and documented by the educational research communities, such as YouTube lectures, "flipped" class lectures (where students are encouraged to read the lecture materials at their own time at homes, and problem solving and/or questions/answers sessions are conducted in the usual classroom environments), STEM summer camps, game-based-learning (GBL) [3-5], virtual laboratories [6], concept inventory [7] etc, to name just a few.

The major objective for this work is to develop an "educational tool", which will help students to fully understand undergraduate STEM subjects (such as "numerical methods"), to help the course instructors to alleviate the time consuming tasks of designing the home-work problems and preparing the associated solutions. For this purpose, the "Hungarian" algorithm (for finding the optimum assignment problems) is selected to use in this work, since this topic does NOT require any specific engineering discipline's backgrounds as the pre-requisite, and due to its general applications. The final developed "educational product" from this work will be a Java-based Hungarian algorithm/animation with the following desirable features:

- Both "minimum" (or "maximum") objective function can be handled.
- Both "square", "rectangular/tall" matrix, or "rectangular" input cost matrix can be treated.
- Each detailed step-by-step of the Hungarian algorithm will be precisely and clearly explained through carefully text editing on the computer screen (for visualization purpose) as well as through Java computer animated voice (for hearing purpose).
- Options to hear female, or male voice are provided.
- Options to hear computer animated voice in 3 major languages (English, Chinese, and Spanish) are available.

978-1-5090-2246-5/16/$31.00 ©2016 IEEE

- Options to provide the input data by reading from a "text file" mode, or by providing input data in "interactive" mode, or by providing input data through "randomly generated" mode is available.
- Fully exploit Java language to take advantage of "computer animation", such as high-lighted (with different colors, and/or flashing lights, etc.) certain intermediate step/result during different stages of the Hungarian algorithm.
- The user can push the "pause" button (to temporarily stop the Hungarian process), or to push the "continued" button to continue the process.
- Not only printing out the "final/optimum assignment" results, but intermediate results can also be printed, so that the users/learners can check/verify with his "hand-calculated" results, which is an important part of his/her learning process.

It should be emphasized here that existing/documented literatures on the Hungarian algorithm are quite extensive, including video recorded, YouTube lectures [8-9]. However, to the best of the authors' knowledge, none of the existing tools have offered "all" the above desirable features.

The remaining of this paper is organized as following. A simple (small-scale) example is introduced in Section 2 to facilitate the explanation of a modified form of the step-by-step Hungarian algorithm. In Section 3, extensions for the basic Hungarian algorithm (such as how to handle "maximization (instead of minimization)" assignment problem, how to handle cases where the input data is the "rectangular/tall", or "rectangular/fat" matrix), and Java computer animation features are added/inserted into the basic Hungarian algorithm to make the learning process easier, and more attractive. Users', learners', and/or students' interactions with the developed "Java Hungarian Animation" and the potential benefits for both students and their instructor from the developed "teaching tool" are described and high-lighted in Section 4. Finally, conclusions and future research works are summarized in Section 5.

II. Step-by-Step Hungarian Algorithm

To facilitate this step-by-step discussion, the following small-scale numerical 5x5 matrix [A] data is used as a "generic" case for the assignment problem (5 workers/rows will be assigned to 5 jobs/columns, in such a way to minimize the total cost).

$$[A] = \begin{bmatrix} 5 & 2 & 4 & 3 & 6 \\ 7 & 4 & 3 & 6 & 5 \\ 2 & 4 & 6 & 8 & 7 \\ 8 & 6 & 3 & 5 & 4 \\ 3 & 9 & 4 & 7 & 6 \end{bmatrix} \quad (1)$$

In an example "transportation" perspective, the above "generic" 5x5 matrix [A] with its elements [Aij] can be interpreted as the cost to transport the i-th bus (say, located at the Grey Hound bus station in Norfolk) to the j-th terminal (so that the bus drivers will carry passengers from the j-th terminal to certain destinations). Thus, one needs to have "optimum assignments" (which bus to be assigned to which terminal?) for minimizing the total cost, and making sure that every bus, and every terminal can be utilized.

A. *Step 0 – "Dummy" Rows (or Columns)*

If the original matrix is rectangular, then "dummy" rows (or columns) can be added to make the matrix become a square matrix. The maximum value in the original matrix [A] can be used in these "dummy" rows (or columns).

B. *Step 1 – Subtract each Row with its Corresponding Minimum Value*

Each row in matrix [A] is subtracted by its minimum corresponding value.

$$[A] = \begin{bmatrix} 5 & \mathbf{2} & 4 & 3 & 6 \\ 7 & 4 & \mathbf{3} & 6 & 5 \\ \mathbf{2} & 4 & 6 & 8 & 7 \\ 8 & 6 & \mathbf{3} & 5 & 4 \\ \mathbf{3} & 9 & 4 & 7 & 6 \end{bmatrix} \rightarrow [A] = \begin{bmatrix} 3 & \mathbf{0} & 2 & 1 & 4 \\ 4 & 1 & \mathbf{0} & 3 & 2 \\ \mathbf{0} & 2 & 4 & 6 & 5 \\ 5 & 3 & \mathbf{0} & 2 & 1 \\ \mathbf{0} & 6 & 1 & 4 & 3 \end{bmatrix}$$

C. *Step 2 – Subtract each Column with its Corresponding Minimum Value*

Each column in matrix [A] is subtracted by its minimum corresponding value.

$$[A] = \begin{bmatrix} 3 & \mathbf{0} & 2 & \mathbf{1} & 4 \\ 4 & 1 & \mathbf{0} & 3 & 2 \\ \mathbf{0} & 2 & 4 & 6 & 5 \\ 5 & 3 & \mathbf{0} & 2 & \mathbf{1} \\ \mathbf{0} & 6 & 1 & 4 & 3 \end{bmatrix} \rightarrow [A] = \begin{bmatrix} 3 & \mathbf{0} & 2 & \mathbf{0} & 3 \\ 4 & 1 & \mathbf{0} & 2 & 1 \\ \mathbf{0} & 2 & 4 & 5 & 4 \\ 5 & 3 & \mathbf{0} & 1 & \mathbf{0} \\ \mathbf{0} & 6 & 1 & 3 & 2 \end{bmatrix}$$

D. *Step 3 – Cover ALL zeros in the current matrix [A] with the "Mininum Number of Lines (MNOL)"*

Compare each column and row in matrix [A] to determine which has the most zeros to be covered first by a horizontal or vertical line.

First, loop through each row and then each column in matrix [A] to determine which column has the most number of zeros. For each row, the number of zeros (number_of_zeros = 0) and column index (column_index = -1) is reset. Each column is looped and checked for when a zero is found and if it is not covered by an existing vertical or horizontal line. If this condition is met, the column index for this column is stored and number of zeros is incremented. After looping through all the columns for a row and if it is determined if the number of zeros is less-than equal-to the number of zero limit (number_of_zero_limit = 1) and is greater-than zero, then the column that was marked by the column_index is covered since it has the most zeros.

So, in Row 1 Column 2, the zero value is noted and there is no existing vertical or horizontal line, so this column index is stored and the number of zeros is incremented (column_index = 2, number_of_zeros = 1). The remaining columns are checked and another zero is found in Row 1 Column 4, so this column index is now stored and the number of zeros is incremented (column_index = 4, number_of_zeros = 2). After all the columns are checked, the number of zeros is not less-than or equal-to the number of zero limit; therefore, no column is marked and the next row is checked. In Row 2 Column 3, the zero value is noted and there is no existing vertical or horizontal line, so this column index is stored and the number of zeros is incremented

(column_index = 3, number_of_zeros = 1). The remaining columns are checked for this row and there are no remaining zeros; therefore, the number of zeros is less-than the number of zero limit so Column 3 is marked. This is continued until all the rows are looped.

After all the rows have been looped, the matrix [A] marked lines are as follows where Column 3 is marked first, then Column 1 and then finally Column 5.

$$[A] = \begin{bmatrix} 3 & 0 & 2 & 0 & 3 \\ 4 & 1 & 0 & 2 & 1 \\ 0 & 2 & 4 & 5 & 4 \\ 5 & 3 & 0 & 1 & 0 \\ 0 & 6 & 1 & 3 & 2 \end{bmatrix}$$

Next, loop through each column and then each row in matrix [A] to determine which row has the most number of zeros. For each column, the number of zeros (number_of_zeros = 0) and row index (row_index = -1) is reset. Each row is looped and checked for when a zero is found and if it is not covered by an existing vertical or horizontal line. If this condition is meet, the row index for this row is stored and number of zeros is incremented. After looping through all the rows for a column and if it is determined if the number of zeros is less-than equal-to the number of zero limit (number_of_zero_limit = 1) and is greater-than zero, then this row that was marked by the row_index is covered since it has the most zeros.

After all the columns have been looped, the matrix [A] marked lines are as follows where Row 1 is only marked.

$$[A] = \begin{bmatrix} 3 & 0 & 2 & 0 & 3 \\ 4 & 1 & 0 & 2 & 1 \\ 0 & 2 & 4 & 5 & 4 \\ 5 & 3 & 0 & 1 & 0 \\ 0 & 6 & 1 & 3 & 2 \end{bmatrix}$$

At this point, all the zeros are covered with a line (yellow highlight) with the minimum number of lines (MNOL = 4). If MNOL is greater-than of equal-to the matrix [A] dimension (N = 5) proceed to Step 4 – Optimum Assignment; otherwise, proceed to Step 3.1 – Compute the "Minimum Uncovered Number (MUN)" in matrix [A].

E. Step 3.1 – Compute the "Mininum Uncovered Number (MUN)" in matrix [A]

Since the MNOL is less-than matrix [A] dimension (N = 5), the MUN can be computed. This is done by subtracting the MUN from every uncovered number matrix [A]. And by adding the MUN to every number covered with two lines (intersecting lines or yellow highlights) matrix [A].

The MUN for matrix [A] is 1. Every uncovered number (blue font text) is subtracted from MUN while every intersecting lines (red font text) is added by MUN. The resulting matrix [A] is as follows:

$$[A] = \begin{bmatrix} 3 & 0 & 2 & 0 & 3 \\ 4 & 1 & 0 & 2 & 1 \\ 0 & 2 & 4 & 5 & 4 \\ 5 & 3 & 0 & 1 & 0 \\ 0 & 6 & 1 & 3 & 2 \end{bmatrix} \rightarrow [A] = \begin{bmatrix} 4 & 0 & 3 & 0 & 4 \\ 4 & 0 & 0 & 1 & 1 \\ 0 & 1 & 4 & 4 & 4 \\ 5 & 2 & 0 & 0 & 0 \\ 0 & 5 & 1 & 2 & 2 \end{bmatrix}$$

After the MUN has been computed, the horizontal and vertical lines are removed and Step 3 – Cover ALL zeros in the current matrix [A] with the MNOL is repeated.

F. Step 4 – Optimum Assignment

When the MNOL is greater-than of equal-to the matrix [A] dimension (N = 5), the matrix [A] is converged and is ready for assignment. Starting with the top row, work your way downwards as assignments is made. An assignment can be (uniquely) made when there is exactly one zero in a row.

The optimum assignment portion of the algorithm is done in a similar fashion as in Step 3 for covering all zeros with the MNOL. The final optimum assignment (optimum minimum score = 3 + 3 + 4 + 4 + 3 = 17) for this matrix [A] is as follows:

- Worker #1 assigned to Job #4
- Worker #2 assigned to Job #3
- Worker #3 assigned to Job #2
- Worker #4 assigned to Job #5
- Worker #5 assigned to Job #1

$$[A] = \begin{bmatrix} 6 & 1 & 4 & 0 & 4 \\ 5 & 0 & 0 & 0 & 0 \\ 0 & 0 & 3 & 2 & 2 \\ 7 & 3 & 1 & 0 & 0 \\ 0 & 4 & 0 & 0 & 0 \end{bmatrix} \rightarrow [A] = \begin{bmatrix} 5 & 2 & 4 & 3 & 6 \\ 7 & 4 & 3 & 6 & 5 \\ 2 & 4 & 6 & 8 & 7 \\ 8 & 6 & 3 & 5 & 4 \\ 3 & 9 & 4 & 7 & 6 \end{bmatrix}$$

III. Step-by-Step Hungarian Algorithm with the Java Computer Animation

The software is custom written in Java and is meant to create

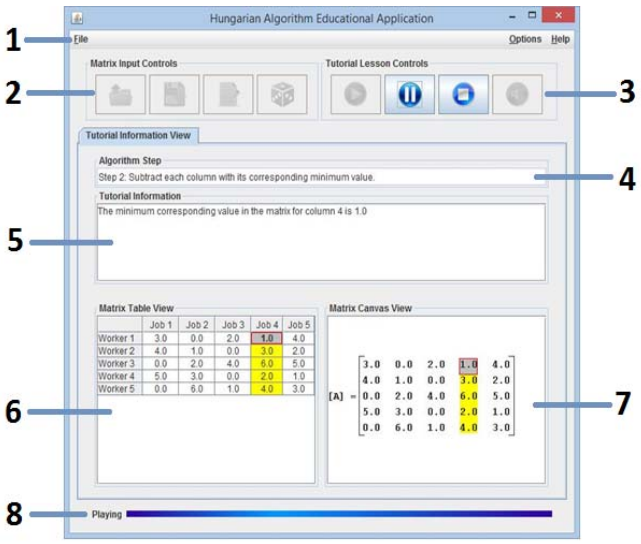

Fig. 1 Main Graphical User Interface (GUI)

2D animations and provide voice explanations for solving for

the "final/optimum assignment" for the Hungarian Algorithm. The Java software is 100% written in Java and developed from scratch using several open-source software. The software uses various third-party libraries, such as the 2D graphics library (Swing) for animations, the text-to-speech library (Google API Translate) for voice, and the matrix library (Efficient Java Matrix) for data storage [10-11]. Applying open-source libraries to this software allowed lots of Graphical User Interface (GUI) functionality and features to be developed. The software GUI is developed with JFC/Swing which is included within the Java Development Kit (JDK). The Main GUI consists of eight main components, as shown in Figure 1.

1. Menu Buttons (File, Preferences, and Help) – Provides basic options, such as an option to open/save the matrix, an option to provide terse, verbose, or no voice step-by-step instructions, an option to change the voice language to English, Spanish, and Chinese, and an option to display a user manual.
2. Input Control Buttons – Buttons to open, save or edit a matrix.
3. Lesson Control Buttons – Buttons to play, pause, or stop the step-by-step lesson. The buttons are enabled when an initial matrix is given. The step-by-step window button displays a scrollable window with the step-by-step instructions. The information button provides a user guide for the Main GUI.
4. Algorithm Step – Textbox view of the current algorithm step (during play session).
5. Tutorial Information View – Textbox view of instructions or steps being done within the algorithm (during play session).
6. Animation View (Matrix Table View) – Provides an animated table view of the matrix during each step within the algorithm.
7. Animation View (Matrix Canvas View) – Provides an animated canvas view of the matrix during each step within the algorithm.
8. Status View – Provides a status view which displays meaningful status: ready, playing, or paused.

The given small-scale numerical 5x5 matrix [A] data given in Section 2 is used again as a "generic" case for the assignment problem (5 workers/rows will be assigned to 5 jobs/columns, in such a way to minimize the total cost) and is inputted into the Java computer animation tool. The tool will perform the same step-by-step instructions along with voice, visualization and animation.

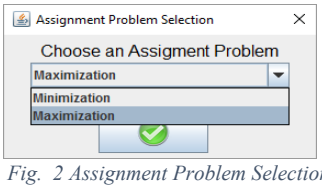

Fig. 2 Assignment Problem Selection

Prior to starting the step-by-step instructions for a given matrix, the Java tool will prompt the user to either solve for the "minimization" or "maximization" optimum assignment. If "maximization" is selected, an additional step prior to Step 0 will occur to convert the assignment problem into a "maximization" assignment problem. This is done by locating the maximum cost value in the matrix. Once that is determined, every cost value in the matrix is multiplied by -1 and then added to the maximum cost value. This will convert the given matrix for the solving for the "maximization" assignment problem. In this example, we solve for the "minimization" assignment problem.

A. Step 0 – "Dummy" Rows (or Columns)

During Step 0, the Java tool determines if the original matrix is non-squared ("rectangular" or "tall"). If so, "dummy" rows (or columns) with the values set to zero are added to make the matrix to become squared. In our given example, the matrix is squared so no addition rows (or columns) are added.

B. Step 1 – Subtract each Row with its Corresponding Minimum Value

Again for Step 1, each row in matrix [A] is subtracted by its minimum corresponding value. The minimum corresponding value for each row is initially highlight in the Java tool and then is used to subtract each value in that row. While each value in that row is being subtracted it is then highlighted while the subtraction occurs. In the tutorial information textbox, a brief description of what's occurring is stated here. Figure 3 displays the minimum value 2.0 selected for Row 3. The value 2.0 is then subtracted from each value in row 3.

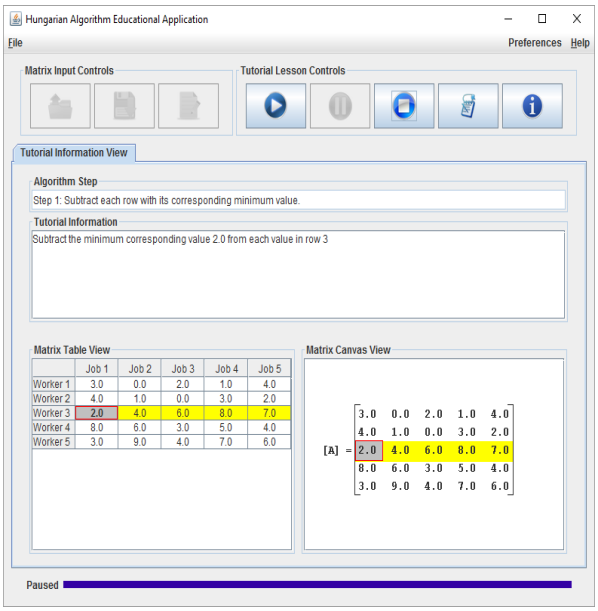

Fig. 3 Step 1 – Subtract each Row with its Corresponding Minimum Value

C. Step 2 – Subtract each Column with its Corresponding Minimum Value

Again for Step 2, each column in matrix [A] is subtracted by its minimum corresponding value. The minimum corresponding value for each column is initially highlight in the Java tool and then is used to subtract each value in that column. While each value in that column is being subtracted it is then highlighted while the subtraction occurs. In the tutorial information textbox, a brief description of what's occurring is stated here. Figure 4

displays the minimum value 1.0 selected for column 4. The value 1.0 is then subtracted from each value in column 4.

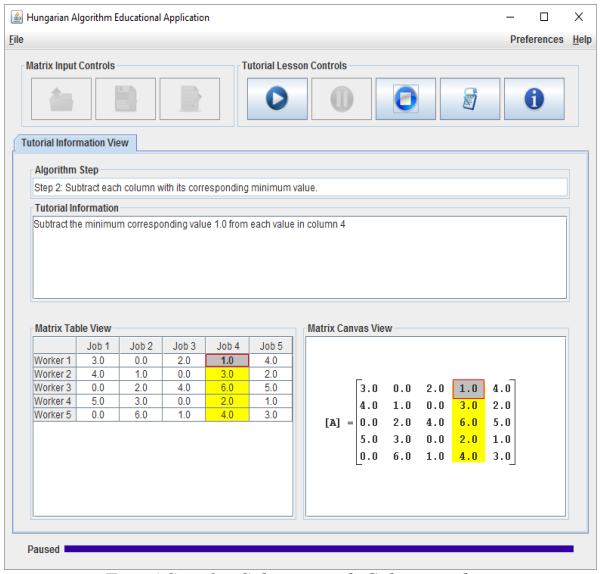

Fig. 4 Step 2 – Subtract each Column with its Corresponding Minimum Value

D. *Step 3 – Cover ALL zeros in the current matrix [A] with the "Mininum Number of Lines (MNOL)"*

Again for Step 3, we compare each column and row in matrix [A] to determine which has the most zeros to be covered first by a horizontal or vertical line. This same step applies to Step 3 discussed in Section 2.

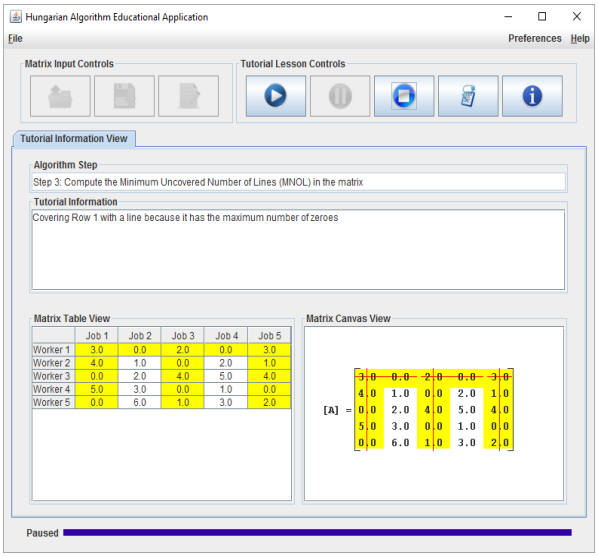

Fig. 5 Step 3 – Cover ALL Zeros in the Current Matrix [A] with the "Minimum Number of Lines (MNOL)"

First, loop through each row and then each column in matrix [A] to determine which column has the most number of zeros. In Figure 5, column 3 is covered first by a vertical line, column 1 is next to be covered followed by column 5.

Next, loop through each column and then each row in matrix [A] to determine which row has the most number of zeros. In Figure 5, row 1 is covered by a horizontal line. In Figure 5, all the zeros are cover with either a horizontal or vertical line, but the matrix [A] is not yet converged; therefore, we proceed to Step 3.1.

E. *Step 3.1 – Compute the "Mininum Uncovered Number (MUN)" in matrix [A]*

Since the MNOL is less-than matrix [A] dimension (N = 5), the MUN can be computed. This is done by subtracting the MUN from every uncovered number matrix [A]. And by adding the MUN to every number covered with two lines (intersecting lines or yellow highlights) matrix [A].

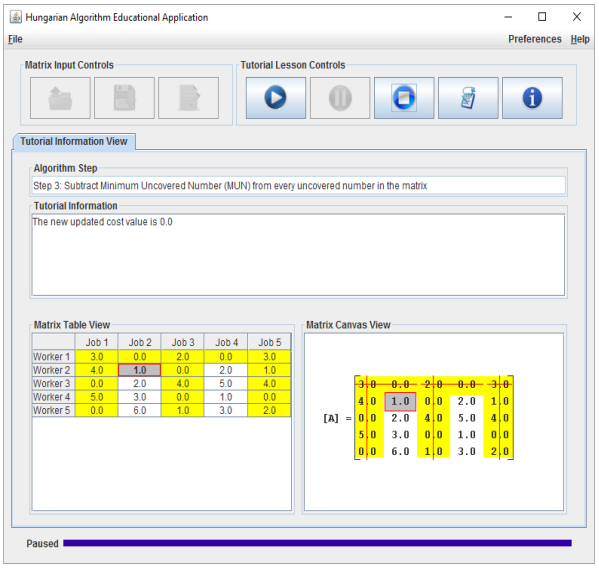

Fig. 6 Step 3.1 – Compute the "Minimum Uncovered Number (MUN)" in Matrix [A]

In Figure 6, the MUN is 1.0 and is highlighted in gray since it is the minimum uncovered number not covered by a horizontal or vertical line. The value 1.0 is to be subtracted by every uncovered number in the matrix [A] and added to every number covered by intersecting lines.

After the MUN has been computed, the horizontal and vertical lines are removed and Step 3 – Cover ALL zeros in the current matrix [A] with the MNOL is repeated. This cycle continues until the matrix [A] is converged. Once the matrix [A] is converged, we proceed to Step 4.

F. *Step 4 – Optimum Assignment*

When the MNOL is greater-than of equal-to the matrix [A] dimension (N = 5), the matrix [A] is converged and is ready for assignment. Starting with the top row, work your way downwards as assignments is made. An assignment can be (uniquely) made when there is exactly one zero in a row.

In Figure 7, the final optimum assignment (optimum minimum score = 3 + 3 + 4 + 4 + 3 = 17) for this matrix [A] is as follows:

- Worker #1 assigned to Job #4
- Worker #2 assigned to Job #3
- Worker #3 assigned to Job #2

- Worker #4 assigned to Job #5
- Worker #5 assigned to Job #1

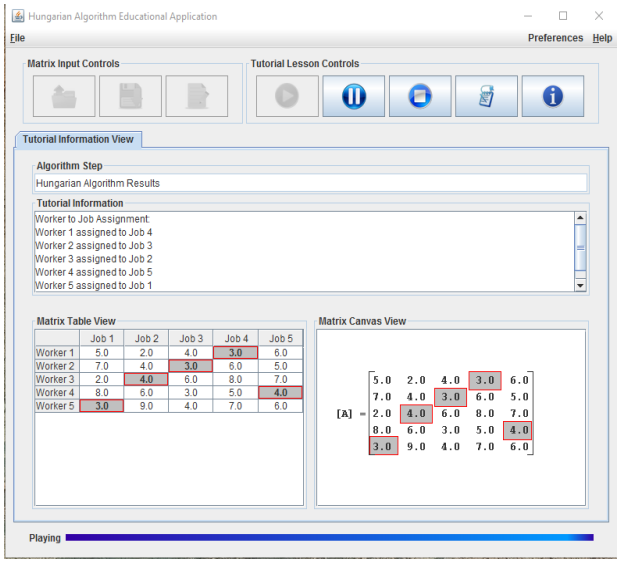

Fig. 7 Step 4 – Optimum Assignment

Another additional feature during the step-by-step instructions is for the user to open the Step-By-Step Window. This window provides to the user a scrollable view of step-by-step instructions in text format while the visualization and animation is occurring from the Main GUI (Figure 1). This window allows these instructions to be exported to an output file at the user's conveniences.

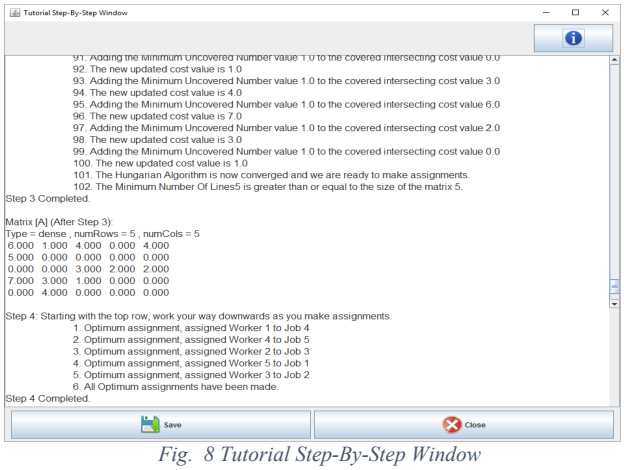

Fig. 8 Tutorial Step-By-Step Window

IV. Potential Benefits for Students and Instructors from the Developed Hungarian Java Computer Animation

With the developed Java based computer animation for teaching/learning the Hungarian (optimum) assignment problems, the following potential benefits can be expected for the students/leaners:

1. Having access to the "instructor" 24/7, where students can "hear" the instructor's animated voice-explanation, "see" the instructor's animated explanation for each step of the Hungarian algorithm, through simple numerical examples.
2. Having the opportunity to "interact" with the "Java animated instructor", by providing his/her own problem data, and to "verify" his/her own (hand-calculated) solution with the Java animated solution. Thus, any mistakes made by the students (during any step of the Hungarian procedures) can be detected. This Java tool, therefore, is quite useful for the students' learning (and self-assessment) process.

The following potential benefits can be expected for the Instructors:

1. Having the absolute freedom to design new home-work assignments (or new exams' questions) for each year, without wasting the instructor time to prepare for the home-works', and/or exams' solutions.
2. Since the students can learn the "numerical method" topic (a Hungarian algorithm, in this case) by themselves at home (through the developed Java tool), the instructor can save his/her lecture time for introducing other "new topics" into his/her class.
3. With the developed Java tool, the instructor can conveniently try "flip class-room teaching mode", rather than the "traditional classroom teaching mode", where the instructor has to spend significant amount of time for "face-to-face" lecturing the students.

V. Conclusion and Future Research Work

In this work, an "attractive Java-based computer animated educational tool" has been developed to help students to understand undergraduate STEM subjects (such as "numerical methods"), to help the "transportation professionals", and to help the course instructors to alleviate the time consuming tasks of designing the home-work problems and preparing the associated solutions.

Although the "Hungarian" algorithm (for finding the optimum assignment problems) is selected for demonstrated purposes (since this topic does NOT require any specific engineering discipline's backgrounds as the pre-requisite, and due to its general applications), other technical topics in "numerical methods" course can also be developed, based on the concepts/philosophies developed in this work. The final "educational product" developed from this Java-based Hungarian algorithm/animation has so many "unique/desirable" features, which have not yet been seen in the existing literatures.

Acknowledgment

This research was in part funded by the TranLive Tier I University Transportation Center.

References

[1] National Science Foundation (NSF), National Science Board (NSB), "A National Action Plan for Addressing the Critical Needs of the U.S. Science, Technology, Engineering, and Mathematics (STEM) Education System" (October 30, 2007).

[2] Executive Office of the President of the United States, "Federal Science, Technology, Engineering, and Mathematics (STEM) Education 5-Year

Strategic Plan", a Report from the Committee on STEM Education, National Science and Technology Council (May 2013).

[3] Nguyen, D.T., A.A. Mohammed, S. Kadiam, and Y. Shen "Internet Chess-Like Game and Simultaneous Linear Equations", Global Conference on Learning and Technology Penang (Island), Malaysia; http://www.aace.org/conf/cities/penang; (May 17-20'2010).

[4] Nguyen, D.T., Y. Shen, A.A. Mohammed, and S. Kadiam, "Tossing Coin Game and Linear Programming Big_M Simplex Algorithms", Global Conference on Learning and Technology, Penang (Island), Malaysia; http://www.aace.org/conf/cities/penang; (May 17-20'2010).

[5] Nguyen , D.T. (PI), Autar K. Kaw (Co-PI), Ram Pendyala (Co-PI), and Gwen Lee-Thomas (Co-PI), "Collaborative Research: Development of New Prototype Tools, and Adaptation and Implementation of Current Resources for a Course in Numerical Methods". NSF funded educational grant (Proposal # 0836916; DUE-CCLI Phase 1: Exploratory); (funding period: January 1'2009 – July 30'2011).

[6] Nguyen, D.T. (P.I. = Prof. S. Chaturvedi), "Implementation Grant: Simulation and Visualization Enhanced Engineering Education," National Science Foundation (NSF), funding period: September 2005 – September 2009.

[7] Autar Kaw (P.I.) et al., "Improving and Assessing Student Learning in an Inverted STEM Classroom Setting", NSF (Division of Undergraduate Education) awarded grant # 1322586 (September 2013-September 2016).

[8] Step-by-step procedure and the (statics) video recorded lecture/explanation. http://www.wikihow.com/Use-the-Hungarian-Algorithm

[9] Makohon, I., M. Cetin, M. Ng, and Duc T. Nguyen, "Samples of Java Computer Animation for Teaching the Hungarian Algorithm"; ODU Technical Report No. 07-01-2014, CEE and MSVE Departments, Norfolk, VA 23529 (July 2014). http://www.lions.odu.edu/~imako001/

[10] Efficient Java Matrix Library (EJML). http://code.google.com/p/efficient-java-matrix-library/wiki/EjmlManual

[11] Google Translate Java. http://code.google.com/p/google-api-translate-java/