Summer 2016

Development of Visualization-Animation Software for Learning Transportation Algorithms

Ivan P. Makohon
Old Dominion University

Recommended Citation
Makohon, Ivan P.. "Development of Visualization-Animation Software for Learning Transportation Algorithms" (2016). Master of Science (MS), thesis, Modeling Simul & Visual Engineering, Old Dominion University, DOI: 10.25777/n91a-1h95
https://digitalcommons.odu.edu/msve_etds/2

DEVELOPMENT OF VISUALIZATION-ANIMATION SOFTWARE FOR LEARNING TRANSPORTATION ALGORITHMS

by

Ivan P. Makohon
B.S. April 2001, Christopher Newport University

A Thesis Submitted to the Faculty of
Old Dominion University in Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

MODELING AND SIMULATION

OLD DOMINION UNIVERSITY
August 2016

ABSTRACT

DEVELOPMENT OF VISUALIZATION-ANIMATION SOFTWARE FOR LEARNING TRANSPORTATION ALGORITHMS

Ivan P. Makohon
Old Dominion University, 2016
Co-Directors: Dr. Duc T. Nguyen
Dr. ManWo Ng

Recognizing the steady 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 significant budget resources to STEM education research [1-2] in the past years, with the ultimate goals being to improve both the quality and number of highly trained US educators, student workforce in STEM topics, in today's highly competitive global markets. With the explosion of the 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 educational research communities, such as video lectures (YouTube), "flipped" class lectures (where students are encouraged to read the lecture materials on their own time, and problem solving and/or question/answer sessions are conducted in the usual classroom environments), STEM summer camps, game-based-learning (GBL) [3-5], virtual laboratories [6] and concept inventory [7].

The goal of this study is to develop useful, user friendly Java computer animation for "teaching" these basic/important STEM algorithms that will not only help both the students and their instructors to master this technical subject, but also provide a valuable tool for obtaining

the solutions for homework assignments, class examinations, and self-assessment tools. Java software tools were developed for this research which include the Unloading and Pre-Marshalling algorithms for Terminal Yard Operations, the Hungarian algorithm for worker to job optimum assignment, the Dijkstra algorithm for solving the shortest-path of a transportation network, and the Cholesky Decomposition algorithm for solving simultaneous linear equations. This "educational version" of the Java-based application were implemented with several desirable features, such as:

- A detailed, precise and clear step-by-step algorithm will be displayed in text and human voice during the animation of the algorithm.
- Options to hear animated voice in several major languages (English, Chinese and Spanish).
- Options to input/output data (CVS file), or manually edit the data using an editor, or "randomly generating" data.
- Output of the "final/optimal" results can be exported to text so that the users/learners can check/verify their "hand-calculated" results, which is an important part of the learning process.

ACKNOWLEDGMENTS

First, I would like to thank Old Dominion University (ODU) for giving me the outmost opportunity to attend graduate school to further my education. I would also like to thank the Department of Modeling, Simulation & Visualization Engineering (MSVE) for accepting me into the Master's program.

I would like to express my gratitude and deepest appreciation to my committee members, Dr. Duc T. Nguyen, Dr. ManWo Ng, Dr. Yuzhong Shen, and Dr. Mecit Cetin for their assistance and the opportunity to write/publish conference/journal papers. In addition, I am extremely grateful for Dr. Nguyen's advice and encouragement in pushing me to write conference/journal papers early on during my graduate studies and for allowing me to work on several projects.

I would like to thank my parents, Nestor and Yong Makohon and all my friends for their help, support, and words of encouragement throughout these stressful couple of years. I would like to give a special thanks to an ODU Alumni and colleague, Justin Pearl for introducing me to ODU's MSVE department. Without his word of mouth, I would have not known about the MSVE department and not have pursued a Master's in Modeling and Simulations.

The Terminal Container Yard chapter was in part funded by Mid-Atlantic Transportation Sustainability University Transportation Center (MATS UTC). The Hungarian chapter was in part funded by the TranLive Tier I University Transportation Center. The partial support provided by the NSF grant # ACI-1440673 (ODU-RF Project # 100507-010) to Duc T. Nguyen is gratefully acknowledged.

TABLE OF CONTENTS

Page

LIST OF FIGURES viii

Chapter

1. INTRODUCTION 1
   1.1 Background of Study 1
   1.2 Literature Review 2
   1.3 Goals, Objectives, and Scopes 3

2. UNLOADING AND PRE-MARSHALLING ALGORITHM 6
   2.1 Summary of the Unloading and Pre-Marshalling Algorithm 9
   2.2 Visualization and Animation using Java 18
   2.3 Numerical Examples/Results 21
   2.4 Conclusions 23

3. HUNGARIAN ALGORITHM 25
   3.1 Summary of the Hungarian Algorithm 26
   3.2 Visualization and Animation using Java 31
   3.3 Numerical Examples/Results 39
   3.4 Conclusions 42

4. DIJKSTRA ALGORITHM 43
   4.1 Summary of the Dijkstra Algorithm 44
   4.2 Visualization and Animation using Java 52
   4.3 Numerical Examples/Results 57
   4.4 Conclusions 62

5. CHOLESKY DECOMPOSITION ALGORITHM 64
   5.1 Summary of the Cholesky Decomposition Algorithm 66
   5.2 Visualization and Animation using Java 69
   5.3 Numerical Examples/Results 70
   5.4 Conclusions 72

6. CONCLUSIONS AND RECOMMENDATIONS 74
   6.1 Potential Benefits of Visualization-Animation 75
   6.2 Recommendations 75

REFERENCES 77

APPENDICES 82
APPENDIX A: UNLOADING AND PRE-MARSHALLING INPUT/OUTPUT 82
APPENDIX B: HUNGARIAN INPUT/OUTPUT 83
APPENDIX C: DIJKSTRA INPUT/OUTPUT 84

Page

APPENDIX D: CHOLESKY INPUT/OUTPUT 85

VITA 86

LIST OF FIGURES

Figure Page

1. Container Yard (Storage Yard). 6
2. Container Terminal. 7
3. The initial input parameters and layout (given). 9
4. Unloading Algorithm (Movement Iterations). 12
5. Pre-Marshalling (Reshuffling) Algorithm (Movement Iterations). 15
6. Terminal (Yard) Algorithm Educational Application. 19
7. Terminal Editor GUI. 20
8. Unloading and Pre-Marshalling (Reshuffling) Algorithm Results. 22
9. Main Graphical User Interface (GUI). 32
10. Assignment Problem Selection. 33
11. Subtracting each Row with its Corresponding Minimum Number. 34
12. Subtract each Column with its Corresponding Minimum Number. 35
13. Cover ALL Zeros with the “Minimum Number of Lines (MNOL)” 36
14. Compute the “Minimum Uncovered Number (MUN)”. 37
15. Optimum Assignment. 38
16. Tutorial Step-By-Step Window. 39
17. A 4x4 Maximization Assignment Problem Example. 40
18. A 3x3 Maximization Assignment Problem Example. 40
19. A 4x4 Minimization Assignment Problem Example. 41
20. A 5x5 Minimization Assignment Problem Example. 41
21. A 4x5 Minimization Assignment Problem Example. 42

Figure Page

22. Matrix and Graph View of Nodes and Links. 44

23. A 6x6 Matrix and Graph View. 45

24. Data Structures (Iteration k = 0 upon initialization). 46

25. Data Structures (Iteration k = 1; Source Node = 1) 47

26. Data Structures (Iteration k = 2; Source Node = 3). 48

27. Data Structures (Iteration k = 3; Source Node = 4). 49

28. Data Structures (Iteration k = 4; Source Node = 2). 50

29. Data Structures (During Iteration k = 5; Source Node = 5). 51

30. Data Structures (Vector [d] and [pred], and Set {s} iteration updates). 52

31. Dijkstra Graphical User Interface. 52

32. Matrix Editor (Matrix [A] Data). 53

33. Dijkstra GUI During Step 0: Initialization. 54

34. Dijkstra GUI During Step 1: Outgoing Edges & Update Bookkeeping. 55

35. Dijkstra GUI During Step 2. 56

36. Dijkstra GUI During Step 3. 56

37. Dijkstra 8x8 Matrix and Network Graph Example. 57

38. Dijkstra 8x8 Matrix and Network Graph Results. 58

39. Dijkstra 9x9 Matrix and Network Graph Example. 58

40. Dijkstra 9x9 Matrix and Network Graph Results. 59

41. Dijkstra 8x8 Matrix and Network Graph Example. 59

42. Dijkstra 8x8 Matrix and Network Graph Results. 60

43. Dijkstra 6x6 Matrix and Network Graph Example. 61

Figure Page

44. Dijkstra 6x6 Matrix and Network Graph Results. 61

45. Cholesky Decomposition Main Window... 69

46. Student Percentage Scores Fall 2013 (a) and Fall 2014 (b)....................................... 71

47. Unloading & Pre-Marshalling Input & Output Example
(A) CSV File; (B) Java Software. 82

48. Hungarian Input & Output Example (A) CSV File; (B) Java Software. 83

49. Dijkstra Input & Output Example (A) CSV File; (B) Java Software. 84

50. Cholesky Input & Output Example (A) CSV File; (B) Java Software. 85

CHAPTER 1

INTRODUCTION

1.1 Background of Study

Many young people in the United States are reluctant to enter into fields that require a background in Science, Technology, Engineering, and/or Mathematics (STEM). STEM is a curriculum based on the ultimate goals to improve both the quality and number of highly trained US educators and develop a strong student workforce in STEM topics, in today's highly competitive global markets. The challenge of STEM is to construct a strong and coordinated STEM education system to ensure coherence in STEM learning and adequate supply of well-prepared and highly effective STEM teachers in place to get students back on top in the international arena[1].

Similarly, one of the educational challenges of teaching in colleges and universities today is the quality of teaching, learning, and assessment in furthering the student's education and skills. Another challenge is to understand how to improve the methodology of learning with the use of improved technologies [8]. Technology today tends to change the quality of teaching with access to computers and with the use of the internet. One such change is flipped classrooms which inverts traditional teaching methods; that is, lectures occur after normal hours (usually at home or work) when the students watch lectures at their own pace and convenience.

One means to improve teaching is to capture and attract the attention of students by the use of visualization-animation for teaching STEM topics (with transportation engineering applications). Students learn best and most when they enjoy what they are doing. Using

[1] IEEE Transactions and Journals style is used in this thesis for formatting figures, tables, and references.

visualization-animation as a learning tool to encourage and develop the student’s learning is not only fun, but it would be an effective approach in teaching the transportation algorithm.

1.2 Literature Review

Effective teaching, learning, and assessment require the use of appropriate pedagogies and methodologies to meet STEM demands for the next generation of young people in the United States. Young people today need to be educated to think deeply and to expand their knowledge to become the next innovators, educators, researchers, and leaders who can solve the challenges facing the world.

For decades, traditional classroom lectures have been the dominant teaching methodology with classrooms full of students all focused on one professor. Teaching methods vary between professors; some demonstrate or discuss, some focus on principles or applications, some emphasize on memory or understanding. Overall, traditional classroom lectures depend on the student’s native ability and prior preparation to learn and understand the materials being covered [9].

Active learning in classrooms became popular in the 1990’s as a means to teach students. Research has shown that students must be actively engaged with the materials that are being taught. This active engagement helps promote the student to a much deeper level of processing and learning which allows her/him to create a stronger connection [10]. Active learning basically shifts the focus from the professor’s delivery of the course content to the students and their active engagement towards the course materials. This would promote their levels of thinking and facilitate memory storage and retrieval than traditional classroom lectures [11].

With the increasing technology innovations of the digital age today, students are equipped with the latest technology such as smart phones, tablets and laptops as learning tools.

There’s been a study that showed that the use of laptops in classrooms has benefitted students’ performance by increasing motivation and collaboration, strengthening connections between disciplines, improving problem solving skills, and promoting academic achievements [12].

In 2006, flipped (inverted) classrooms were introduced by high school chemistry teachers from Colorado. Flipped classrooms involve the use of live video recordings and screen casting software to record classroom lectures. These recorded classroom lectures were posted on YouTube for students to view [13]. This allowed students to access the video lectures anytime and anywhere as long as internet connectivity was accessible. Flipped classrooms were convenient to those students who missed classroom lectures due to circumstances (i.e. school absence or even traveling to other schools for competition). Flipped classrooms have become a very attractive and successful approach to teaching. Course materials and videos are provided to the students prior to coming to class, which allows the students to engage more in active learning in using case studies, labs, simulations, or experiments during the classroom lecture [14].

1.3 Goals, Objectives, and Scopes

To improve both the quality and number of highly trained US educators, innovations or improvements need to be applied towards the appropriate pedagogies and methodologies for teaching and learning. One such innovation is to allow the students or lecturers to use visualization-animation software applications for teaching transportation algorithms.

Animation is a graphical, artistic way of expression while visualization is the process of representing data as an image that can aid in the understanding of data. Visualization-animation can be dated back to the performance of Chinese Shadow Puppetry during the Han Dynasty. Shadow Puppetry is a unique form of visualization-animation where shadows casted by a sculpture are essential for artistic effect [15]. Using shadows to animate and visualize objects was an ideal way to win the hearts and minds of an audience by its exquisite sculptures, music sounds, and colorful and lively performance to tell a story and capture the audience's attention.

In the classroom, lectures with static illustrations, text, and videos can sometimes be difficult to conceptualize for students. Visualization-animation software can help illuminate the concept easily for the students and teachers [16]. A recent survey of students stated that they enjoyed lectures that included a simple short animation. It was mentioned that the students were more engaged with the teachings using the animations within a PowerPoint presentation or video [17]. It is also mentioned in the literature that students should adapt to interactive learning systems to better improve their learning by capturing their attention and to help with memorizing [18].

The goals, objectives, and scope of this work are to effectively use visualization-animation to develop useful, user friendly, and Java computer animation for "teaching" transportation algorithms (Unloading/Pre-Marshalling, Hungarian, Dijkstra, and Cholesky) that will help both the students and their instructor to not only master this technical subject, but also provide a valuable tool for obtaining the solutions for homework assignments, class examinations, and self-assessment tools. This "educational version" of visualization-animation Java-based application has several desirable features, such as:

- A detailed, precise and clear step-by-step algorithm will be displayed in text and human voice during the animation of the algorithm.
- Options to hear animated voice in several major languages (English, Chinese and Spanish).
- Options to input/output data (CVS file), or manually edit the data using an editor, or "randomly generating" data.

- Output of the “final/optimal” results can be exported to text, so that the users/learners can check/verify their “hand-calculated” results, which is an important part of the learning process.

Java was the programming language of choice for implementing these visualization-animation software applications for teaching those listed transportation algorithms: Unloading/Pre-Marshalling, Hungarian, Dijkstra, and Cholesky. Java is a general-purpose and object-oriented computer programming language [19]. Additional supporting Java libraries were used to implement the management of the matrix data [20], the Google translate text-to-speech [21], and the G Java 2D graphics library [22]. These transportation algorithm software applications are covered and discussed in greater detail in the remaining chapters.

CHAPTER 2

UNLOADING AND PRE-MARSHALLING ALGORITHM

Container Terminal Yard operations [see Figures 1-2] reveal an increasing traffic of cargo container shipment around world ports [23], which makes planning decisions critical with great importance to optimizing the terminal container yard's unloading and pre-marshalling (or reshuffling) of cargo containers within the yard. Container Terminal Yards consist of container bays, where each bay contains a set of container stacks, and each container stack contains stackable cargo containers.

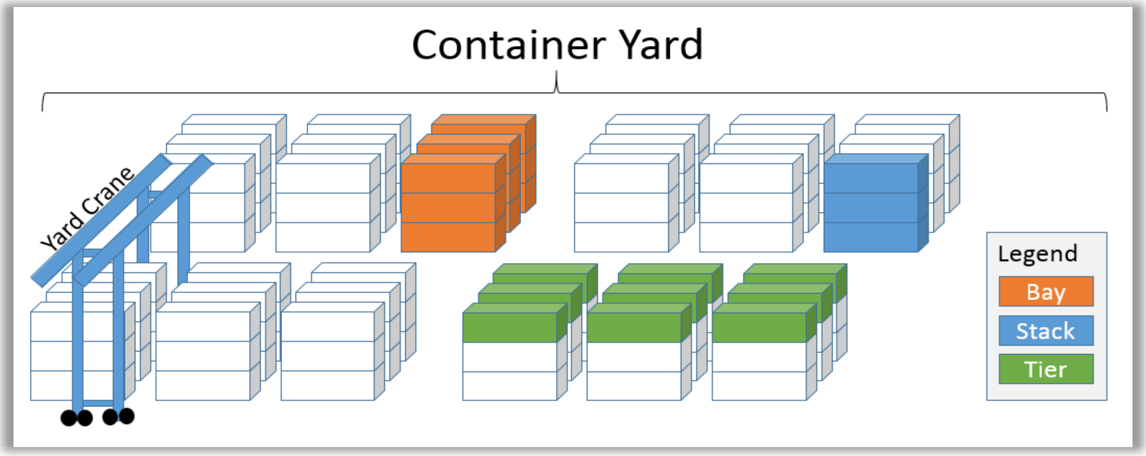


Fig. 1. Container Yard (Storage Yard).

It's an important storage point because it links the seaside with the landside. More precisely, it's a temporary storage point for loading and unloading cargo containers between vessels and vehicles (trucks or trains) for further distribution. The yard operations department is responsible for allocating space and equipment needed to maintain terminal efficiency.

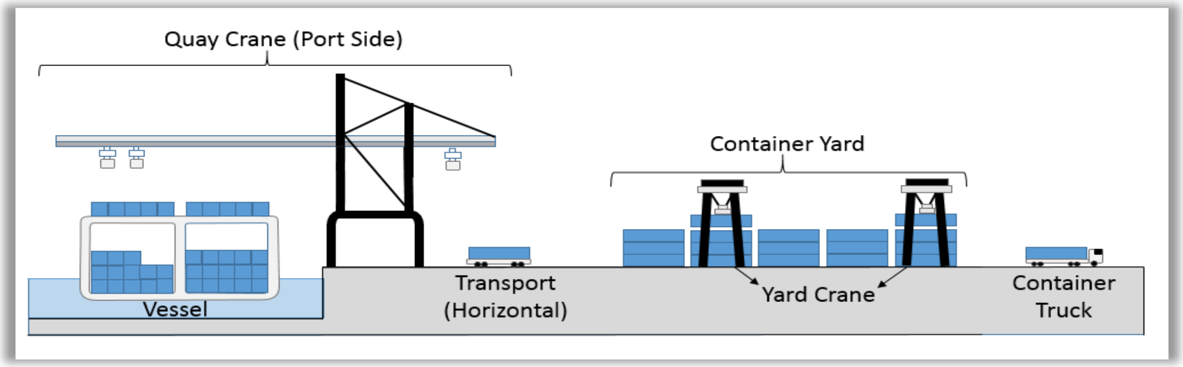


Fig. 2. Container Terminal.

The unloading and pre-marshalling algorithms for container terminal yard operations are revisited with the primary goals of developing simple heuristic “unloading” and “pre-marshalling” algorithms, for port operations. For safety reasons, moving containers from one bay to another bay by crane is usually prohibited. The proper movement involves putting the container on a truck first, moving the truck from one bay to another and then moving the container into the desired bay [24]. Therefore, these algorithms perform efficient unloading and pre-marshalling tasks in container terminals.

This container yard stacking problem is very similar to the Blocks World planning domain problem, which consists of a finite number of blocks stacked into columns. The goal is to turn the initial state into a goal state by moving one block at a time from the top of one column onto another column. The Block World planning problem is to get to the goal state in a minimal number of moves [25].

Since efficient operations during the “unloading” and “pre-marshalling” phases will save time in port operations, leading to significant economic gains, a large amount of the existing literature has been devoted to these needed algorithms [23-26, 27, 28-33]. Researchers have also surveyed and synthesized various available methods and algorithms for unloading and pre-marshaling [29].

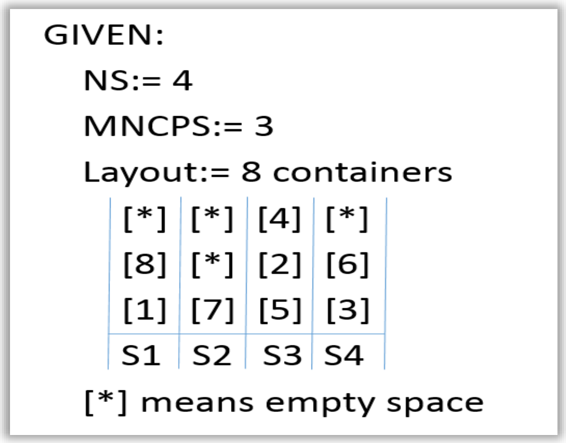
Various models have been proposed in the literature including a simulation model for stacking containers in a container terminal through developing and applying a genetic algorithm (GA) for container location assignment that minimizes total lifting time and therefore, increases service efficiency of the container terminals [30].

Researchers have presented an algorithm selection benchmark based on optimal search algorithms for solving the Container Pre-Marshalling Problem (CPMP), an NP-hard problem from the field of container terminal optimization [27-28]. The CPMP deals with the sorting of containers in a set of stacks (called a bay) of intermodal containers based on their exit times from the stacks such that containers that must leave the stacks first are placed on top of containers that must leave later. A recent approach for solving the CPMP to optimality [28] presents two state-of-the-art approaches based on A* and IDA*. The researchers then use parameterized versions of these approaches to form a benchmark for algorithm selection. An example (with 3 stacks, maximum height per stack is 3, and having a total of 6 containers) is highlighted by the authors' CPMP for algorithm selection. The researchers have presented an exact algorithm based on branch and bound algorithms, and it is shown to be NP-hard [30].

Two heuristic algorithms have been proposed to solve for the pre-marshalling problems [31]. While the existing literature extensively discussed (including some numerical examples) unloading and pre-marshalling (or re-shuffling) operations at the port terminals, step-by-step numerical algorithms have either not been given or have been described without sufficient details. Thus, it is not an easy task to implement and compare the performance of various proposed algorithms. In this work, some simple heuristic algorithms for both "unloading" and "pre-marshalling" operations are proposed. Detailed step-by-step algorithms are explained and presented so that readers can reproduce the presented results. Several numerical examples are used to validate the proposed heuristic algorithms.

2.1 Summary of the Unloading and Pre-Marshalling Algorithm

External vehicles (trucks or trains) are responsible for transporting cargo containers in and out of the container yard whereas the internal vehicles are responsible for transporting containers within the terminal from the storage yard to the quayside. The storage yard is where containers are stored temporarily until a vehicle arrives to further distribute them to their next location [29].



GIVEN:
NS:= 4
MNCPS:= 3
Layout:= 8 containers

[*]	[*]	[4]	[*]
[8]	[*]	[2]	[6]
[1]	[7]	[5]	[3]
S1	S2	S3	S4

[*] means empty space

Fig. 3. The initial input parameters and layout (given).

Two types of algorithms (unloading and pre-marshalling) are discussed and proposed along with a discussion of a visualization-animation software tool for teaching, learning, and improving the algorithms. The unloading algorithm approach is based upon having the stack reshuffled to get to the priority container when a vehicle arrives. The pre-marshalling algorithm approach is based on having the stack ordered before the first vehicle arrives so that the priority container is on top.

To facilitate the discussion in this section, the given input variables and layout are defined [see Figure 3] as follows.

- N = NS = the number of container stacks (columns).
- M = MNCPS = the maximum number of containers per stack (rows).
- NAES = the number of available empty spaces (total number of empty spaces in each stack).
- NESR = the number of empty spaces required.
- NCC = the number of cargo containers.
- TNS = the total number of spaces in the container yard

We also initialize and define some variables for collecting results:

- NM = the number of moves (container relocations).
- NR = the number of removes (containers unloaded).

First, we begin with a given initial storage layout [see Figure 3], the given N = NS = (4 stacks), M = MNCPS = (3 Tiers). From this we can compute the TNS (12 total spaces), NAES (4 available spaces based on the given initial layout) and NESR (3 Tiers) based on the equations:

$$TNS = (MxN)$$
$$NAES = TNS - NCC$$
$$NESR = MNCPS.$$

Figure 3 provides a 2D initial layout that will be discussed throughout this paper. The number in each container represents the priority order: the order sequence in which the containers need to be unloaded onto a vehicle.

Once the initial parameters and layout are given, we begin each algorithm performing Step 0. Step 0's purpose is to determine whether we have enough spaces to perform the algorithms. The verifications for both algorithms are defined as:

1. Unloading Algorithm

- If the NAES is greater than or equal to NESR-1, the NESR-1 spaces should allow us to get to the priority cargo container for removal.

- If the NS is greater than 1 container stacks, we need at least 2 or more container stacks to be able to do the unloading. NAES should provide enough empty spaces initially to get to the priority cargo container if it's on the bottom.

2. Pre-Marshalling (or Reshuffling) Algorithm

   - If the NAES is greater than or equal to NESR, we need NAES spaces in order to do reshuffling without removal.

   - If the NS is greater than 2 container stacks, we need at least 3 or more container stacks to be able to do reshuffling. Note: some layouts can be solved without these validations for reshuffling, though these verifications avoid deadlock cases.

The remainder of the section provides an overview of each algorithm in pseudo code and walks through the proposed step-by-step numerical algorithms.

2.1.1 Unloading Algorithm

In the Unloading Algorithm, cargo containers remain where they reside and are unloaded when a vehicle arrives. If the prioritized cargo container is not on top of the stack, then those cargo containers above it must be relocated. This approach frees up a space before the next vehicle arrives; therefore, the NAES is increased by 1 after every unloaded container.

Before the algorithm begins, the given initial input data is required (Figure 3). From this initial data, the algorithm determines if there's enough available empty spaces in the container stacks to solve this problem (Step 0). For this example, there's enough available empty space (NAES-1 >= NESR).

Next, the algorithm determines if Cargo Containers can be unloaded from the Container Stacks (Step 1). This step loops until the Container Stacks are empty (all Cargo Containers unloaded). Before unloading Cargo Containers, this step determines the Cargo Container with the highest priority and then performs the Cargo Container move (Step 2).

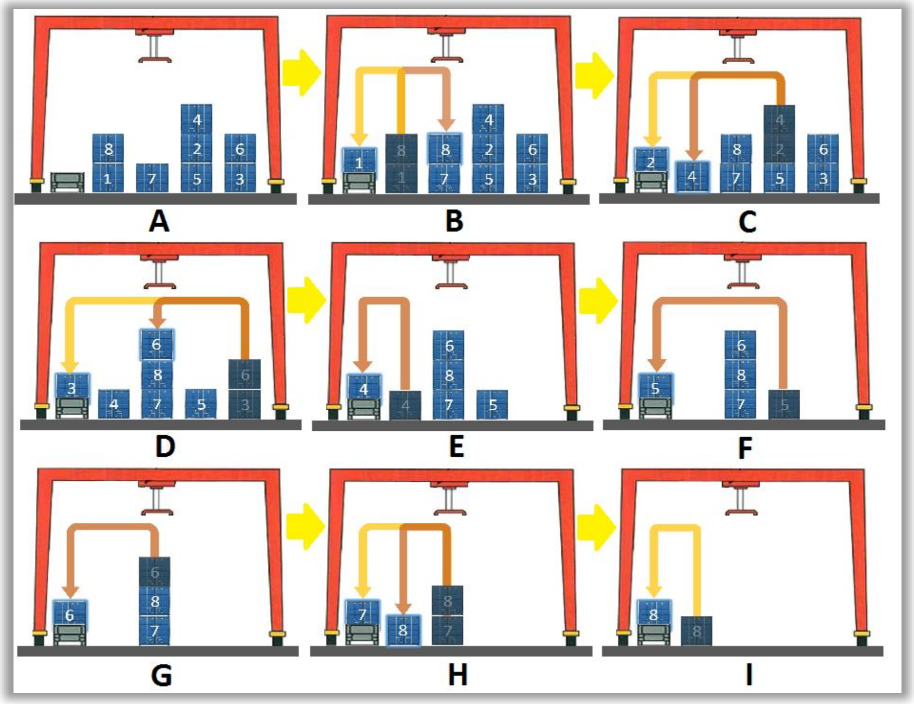


Fig. 4. Unloading Algorithm (Movement Iterations).

Step 2 determines whether the given next highest priority Cargo Container is on top of the Container Stack. If it is, it is removed (unloaded onto a truck); otherwise, the algorithm finds the Container Stack with a highest priority (sorted) but lesser priority than the one that needs to be moved. Once the move occurs, it then loops back and repeats until all the Cargo Containers are unloaded.

For example, Figure 4 shows the iterations of the movements and unloads using the given initial data against the unloading algorithm. Figure 4B shows that Cargo Container 1 is the next highest priority to unload, though in order to unload it those Cargo Container(s) above it must be relocated. Therefore, Cargo Container 8 is relocated to Container Stack 2. Note that there's no Cargo Container that is the highest priority container lesser than the one that needs to be relocated, so we randomly pick the closest Container Stack with an available space. We can improve this to pick the closest stack, pick the least number of containers in the stack, or pick the least priority container to research what kind of impact it would have. Once Cargo Container 8 is relocated (number of moves = 1), Cargo Container 1 can be unloaded onto a vehicle (a truck in this example).

Figure 4C shows that Cargo Container 2 is the next highest priority to unload, but in order to unload it those Cargo Container(s) above it must be relocated; therefore, Cargo Container 4 is relocated to Container Stack 1. Once Cargo Container 4 is relocated (number of moves = 2), Cargo Container 2 can be unloaded onto a vehicle.

Figure 4D shows that Cargo Container 3 is the next highest priority to unload, but in order to unload it those Cargo Container(s) above it must be relocated; therefore, Cargo Container 6 is relocated to Container Stack 2. Once Cargo Container 6 is relocated (number of moves = 3), Cargo Container 3 can be unloaded onto a vehicle.

Figure 4E shows that Cargo Container 4 is the next highest priority to unload. Cargo Container 4 is on top of Container Stack 1; therefore, no movement is required, so Cargo Container 2 can be unloaded onto a vehicle.

Figure 4F shows that Cargo Container 5 is the next highest priority to unload. Cargo Container 5 is on top of Container Stack 3; therefore, no movement is required, so Cargo Container 5 can be unloaded onto a vehicle.

Figure 4G shows that Cargo Container 6 is the next highest priority to unload. Cargo Container 6 is on top of Container Stack 2; therefore, no movement is required, so Cargo Container 6 can be unloaded onto a vehicle.

Figure 4H shows that Cargo Container 7 is the next highest priority to unload, but in order to unload it those Cargo Container(s) above it must be relocated; therefore, Cargo

Container 8 is relocated to Container Stack 1. Once Cargo Container 8 is relocated (number of moves = 4), Cargo Container 7 can be unloaded onto a vehicle.

Figure 4I shows that Cargo Container 8 is the next highest priority to unload. Cargo Container 8 is on top of Container Stack 1; therefore, no movement is required, so Cargo Container 8 can be unloaded onto a vehicle. For the initial example given above, the final results show that there was a total of 4 container relocations (number of moves = 4), and all 8 Cargo Containers were unloaded onto a vehicle.

2.1.2 Pre-Marshalling (Reshuffling) Algorithm

In the Pre-Marshalling Algorithm, the container yard is reshuffled and prioritized before the first vehicle arrives. The goal is to have the container stacks in order with the highest priority containers on top and the lower priority on the bottom. This algorithm is identical to the Block World planning domain problem and is more complex than the unloading algorithm since it's limited to the initial number of spaces available whereas the unloading algorithm frees up a space when a priority container is unloaded onto a vehicle.

The algorithm begins by initializing the given input data required (Figure 3) along with additional variables for bookkeeping purposes. From this initial data, the algorithm determines if there's enough available empty spaces and number of stacks available in order to perform reshuffling of cargo containers with the number of container stacks (Step 0). For this example, there's enough available empty space (NAES >= NESR and NS > 2).

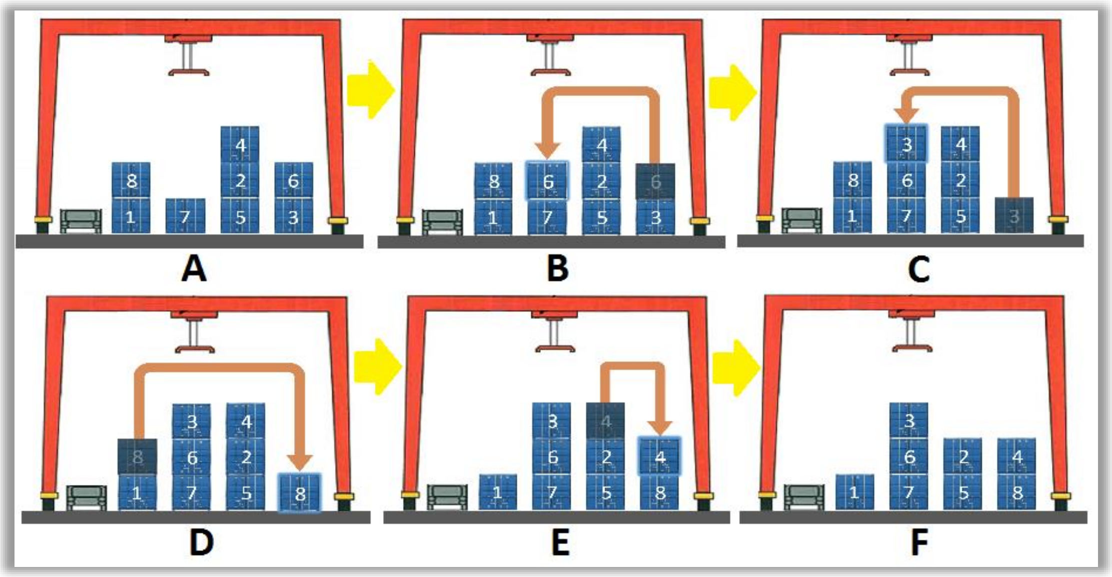


Fig. 5. Pre-Marshalling (Reshuffling) Algorithm (Movement Iterations).

After the initial given data is verified for reshuffling, Step 1 is performed. This step begins the main loop which loops through all the Container Stacks and checks if all the Container Stacks are in order. If they're not, then the goal is to reshuffle all the highest priority Cargo Containers to the top and the lowest to the bottom while each Container Stack is sorted from highest to the lowest in priority. In doing so, the algorithm must determine the source and destination of the Container Stacks.

Step 1 first begins to look at any container stack with a single cargo container. If there is one that exists, the algorithm determines if it's the source or destination Container Stack. In Figure 5A, the initial container layout example shows that Cargo Container 7 is the only Cargo Container in Container Stack 2, so it meets this condition. The same step determines if this Cargo Container can be placed on top of a sorted Container Stack with that selected Container Stack still prioritized or if another higher priority (lower number) Cargo Container can be placed on top of Cargo Container 7 in Container Stack 2. Step 1 determines that Container Stack 2 is selected as the destination Container Stack since there's not another sorted Container Stack on which Cargo Container 7 can be placed. Figure 5B shows that Container Stack 4 is determined to be the source Container Stack; therefore, Cargo Container 6 is placed on top of Cargo Container 7 in Container Stack 2. After this move (relocation), the algorithm is then restarted back at the beginning of the loop.

Step 1 is again repeated; once again there's a condition where one Cargo Container exists within a Container Stack shown in Figure 5B. The algorithm again needs to determine if this is the source or destination Container Stack. In this case, Container Stack 2 is a sorted stack that will allow Cargo Container 3 to move on top of it; therefore, Container Stack becomes the source Container Stack while Container Stack 2 becomes the destination. Figure 5C shows that Cargo Container 3 is moved (relocated) from Container Stack 4 onto Container Stack 2. After this move (relocation), the algorithm is then restarted at the beginning of the loop.

Note that the first two moves are not placed in bookkeeping for containers to move back towards their original Container Stack. The original stack being looked at is Container Stack 2; there have been no Cargo Containers moved from this Container Stack.

Once again, Step 1 is repeated; during this iteration there are no Container Stacks with one Cargo Container, so that check doesn't meet that condition. From this point, the algorithm looks for an unsorted Container Stack with the least amount of Cargo Containers that is not empty. In this case, Container Stack 1 is selected; refer to Figure 5D. From this point, the algorithm enters Step 2 which is to reshuffle (relocate) the top Cargo Container and temporarily put it onto a different Container Stack.

Step 2 is now entered with Container Stack 1 as the selected source Container Stack. The goal of this step is to reshuffle this selected Container Stack and temporarily put the Cargo Containers into a different Container Stack. The step keeps track of the least priority Cargo

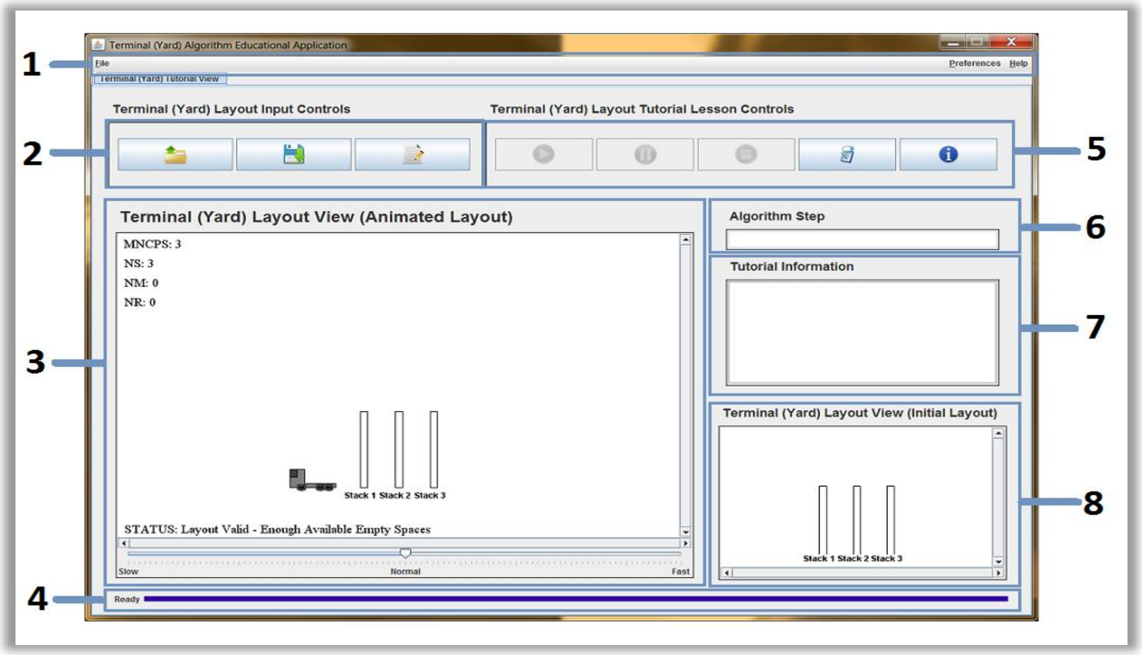
Container being moved (relocated) so that it can always be placed on the bottom of the Container Stack when moved back. In Figure 5D, the algorithm takes the top Cargo Container in Container Stack 1, which is Cargo Container 8 and picks the Container Stack with the least amount of Cargo Containers. Container Stack 4 is empty; therefore, it is selected as the destination Container Stack. Cargo Container 8 is then moved from Container Stack 1 to Container Stack 4. In this case, since Container Stack 4 is empty and Cargo Container 8 is the Cargo Container with the least priority in Container Stack 1, the algorithm does not place Cargo Container 8 in bookkeeping for containers to move back towards their original Container Stack. After this move, the algorithm loops back and looks at the next Cargo Container on Container Stack 1, so Step 2 is repeated.

In this next iteration in Step 2, the algorithm continues to try to temp spot the next top Cargo Container in the selected Container Stack 1. Before the algorithm checks for a destination Container Stack for Cargo Container 1, it looks for the highest priority Cargo Containers in the other unsorted Container Stacks. Container Stack 3 in Figure 5E shows that Cargo Container 4 is the top Cargo Container for all unsorted Container Stacks. Since this Cargo Container isn't in the Cargo Containers to move back bookkeeping, it's moved (relocated) to Container Stack 4 since it's a higher priority than Cargo Container 8. Since Cargo Container 6 isn't from the original source Container Stack, it isn't placed within the Cargo Containers to move back to bookkeeping. After this move, the algorithm loops back and looks at the next Cargo Container on Container Stack 1, so Step 2 is repeated. At the very beginning of this loop, there's a check to see if all the Container Stacks are in order. Figure 5F shows that all Container Stacks are sorted from highest to lowest priority after Cargo Container 4 was moved to Container Stack 4; therefore, the algorithm breaks out of its loop and returns back to Step 1 and continues after it called Step 2's procedures. After this procedure there's a check to see if all the Container Stacks are in order so that it too can trigger the restart of Step 2's main loop. At the very top of the main loop, if all the Container Stacks are in order the algorithm can proceed to Step 3. Figure 5F shows the final state of the reshuffled sorted Container Stacks.

In Step 3, all the Container Stacks are sorted from highest priority Cargo Containers on top with the lowest priority on the bottom. With all the highest priority Cargo Containers on top, unloading Cargo Containers on to vehicles doesn't require any more reshuffling. For the initial example given above, the final results show that there was a total of 4 container relocations (number of moves = 4) and all 8 Cargo Containers were unloaded onto a vehicle.

2.2 Visualization and Animation using Java

The software is custom written in Java and is meant to create 2D animations and provide voice explanations for both the unloading and pre-marshalling algorithms for the Terminal Container Yard operations. The Java software is 100% written in Java and developed from scratch using several open-source software options. The software uses various third-party libraries, such as the 2D graphics library (Geosoft's G 2D) for animations, the text-to-speech library (Google API Translate) for voice, and the matrix library (Efficient Java Matrix) for data storage. Applying open-source libraries to this software allowed lots of Graphical User Interface (GUI) functionality and many features to be developed.


Terminal (Yard) Algorithm Educational Application
File
Preferences
Help
Terminal (Yard) Tutorial View
Terminal (Yard) Layout Input Controls
Terminal (Yard) Layout Tutorial Lesson Controls
Terminal (Yard) Layout View (Animated Layout)
Algorithm Step
Tutorial Information
Terminal (Yard) Layout View (Initial Layout)
Stack 1 Stack 2 Stack 3
Slow
Normal
Fast
Ready
1
2
3
4
5
6
7
8

MNCPS: 3
NS: 3
NM: 0
NR: 0

STATUS: Layout Valid - Enough Available Empty Spaces

Fig. 6. Terminal (Yard) Algorithm Educational Application.

The software GUI is developed with JFC/Swing, included within the Java Development Kit (JDK). The Main GUI consists of eight main components as shown in Figure 6.

1. <u>Menu Buttons (File, Preferences, and Help)</u>
   Provide basic options, such as an option to open/save Container Layouts, 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. <u>Input Control Buttons</u>
   Buttons on the main window to open, save or edit a container yard layout.

3. <u>Terminal (Yard) Layout Animation View</u>
   Provides an animated view for Cargo Container movement and attributes (NS, MNCPS, NM, and NR fields) updates.

4. <u>Status View</u>
   Provides a meaningful status: ready, play, or paused.

5. <u>Lesson Control Buttons</u>
   Buttons on the main window to play, pause, or stop a lesson (an algorithm). The buttons are enabled when an initial layout is given. The information button provides a user guide for the Main GUI.

6. Algorithm Step
   Textbox view of the current algorithm step (during play session).

7. Tutorial Information View
   Textbox view of instructions or steps being done within the algorithm (during play session).

8. Terminal (Yard) Layout Initial View
   Provides a given initial layout view of the terminal yard.

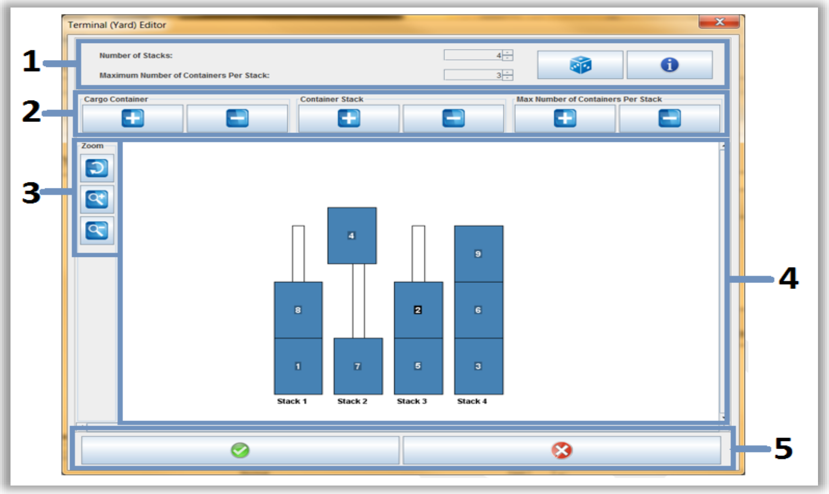


Fig. 7. Terminal Editor GUI.

The Terminal Yard Editor GUI in Figure 7 provides a means to edit terminal layouts and consists of 5 main components.

1. Display and Information View
   This view displays the NS and MNCPS for this layout. It also provides a random button that will randomly generate a layout for the given NCC, and the information button provides a user guide for this GUI.

2. Editor Buttons
   The editor buttons allow the user to add/remove a container from the container stack and to increase/decrease the NS and MNCPS.

3. Zoom Buttons
   The zoom buttons zoom in/out or reset the view back to the original state.

4. Visualization Editor View

The visualization editor allows the user to drag-and-drop Cargo Containers from one Container Stack to another (left-click hold to drag) and to select a Cargo Container to remove (right-click).

5. Bottom Buttons
These buttons either accept the user's edited layout or cancel it.

As mentioned above, the software has special functionality and features, such as animating the Cargo Container movements based on a selected algorithm applied towards a given Terminal Yard Layout. It supports step-by-step animation with voice for a given terminal layout and a selected algorithm. This mode aims to help students understand the working mechanism of the algorithm and allows developers to visually fine tune it. To achieve this goal, the software provides steps within the algorithm with voice and animates the movement of containers from one stack to another. In addition to the visible aspects, a computer generated (text-to-speech) voice accompanies the animation.

2.3 Numerical Examples/Results

Several terminal container yard layouts were referenced from the literature, and some were referenced from classroom lecture [26]. These layouts were used within the Java computer animation software and executed for both the unloading and pre-marshalling (or reshuffling) algorithms. The algorithm results are recorded below within the figure. Results in Figure 8 that show "N/A" for the pre-marshalling algorithm did not provide any results since the layout did not meet the validation checks described earlier in Section 2.1. The main reason for no results for the pre-marshalling algorithm was either that the NAES was not greater than or equal to NESR and/or the NS was not greater than 2 container stacks.

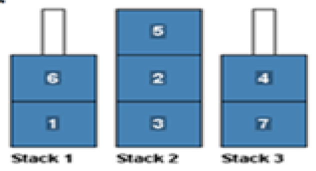
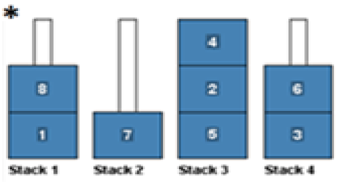
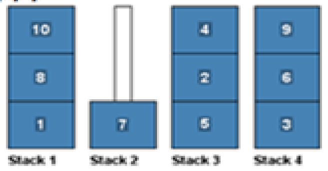
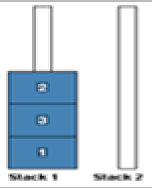
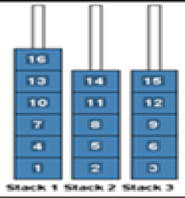
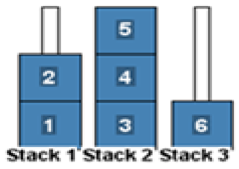
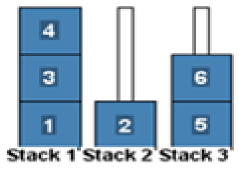
Container Yard Layout	Algorithm	NS	MNCPS	TNS	NCC	NAES	NESR	NM	NR
* 	Unloading	3	3	9	7	2	3	3	7
	Reshuffling	3	3	9	7	2	3	N/A	N/A
* 	Unloading	4	3	12	8	4	3	4	8
	Reshuffling	4	3	12	8	4	3	4	8
*** 	Unloading	4	3	12	10	2	3	9	10
	Reshuffling	4	3	12	10	2	3	N/A	N/A
*** 	Unloading	2	5	10	3	7	5	3	3
	Reshuffling	2	5	10	3	7	5	N/A	N/A
*** 	Unloading	3	8	24	16	8	8	39	16
	Reshuffling	3	8	24	16	8	8	51	16
** 	Unloading	3	3	9	6	3	3	3	6
	Reshuffling	3	3	9	6	3	3	3	6
* 	Unloading	3	3	9	6	3	3	4	6
	Reshuffling	3	3	9	6	3	3	6	6

Fig. 8. Unloading and Pre-Marshalling (Reshuffling) Algorithm Results.

(* denotes from reference [29], ** denotes from reference [27], and *** denotes from reference [26]).

2.4 Conclusions

The terminal container yard unloading and pre-marshalling (or reshuffling) algorithms have been proposed, explained and verified through several numerical examples. Detailed descriptions of the proposed step-by-step algorithms are also provided so that readers can reproduce the presented results. A secondary goal for this paper is to develop the Java computer animated software (associated with the proposed unloading and pre-marshalling algorithms) to be used as an additional tool for teaching and learning these unloading and pre-marshalling algorithms. The software tool is custom developed in Java and integrated with open-source libraries. It provides desirable teaching/learning features for these transportation algorithms for unloading/pre-marshalling, such as:

- Software tool with a user friendly and easy to use GUI.
- 2D Graphical visualization-animation for displaying container movement (relocation).
- 2D Graphical visualization to edit a new or existing terminal container yard layout.
- Ability to allow the user/learner to load/store a terminal container yard layout.
- Ability to allow the user/learner to output/save the step-by-step results.
- Provides a clear and attractive computer animated voice that provides step-by-step instructions of the algorithms.
- Animated voice can be configurable to translate text-to-speech into another language, such as English, Spanish and Chinese.
- Provides results of the algorithms, such as NM, NR, NS, and MNCPS.

The software tool itself can be used as a visualization-animation framework for including other implementations of the unloading/pre-marshalling algorithms. Applying the JBoss Rules (Drools) engine which uses the Rete Algorithm could potentially be considered as further work. The Rete Algorithm is a pattern matching algorithm for implementing production rule systems.

It’s known to be helpful in planning rule-based systems [25]. By using this approach, we can experiment with JBoss rules and use the Java animation teaching tool to refine and optimize both the unloading and pre-marshalling algorithms for Terminal Container Yard operations.

CHAPTER 3

HUNGARIAN ALGORITHM

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 community. In this chapter, the basic Hungarian algorithm is revisited, with the ultimate goal of developing a useful, user friendly, attractive Java computer animation for “effectively teaching” this basic/important optimum assignment algorithm.

The major objective for this work is to develop an “educational tool”, which will help students fully understand undergraduate STEM subjects (such as “numerical methods”), to help alleviate the time consuming tasks of designing homework problems and preparing the associated solutions. For this purpose, the “Hungarian” algorithm (for finding the optimum assignment problems) is selected for use in this work since this topic does not require any specific engineering discipline’s backgrounds as the prerequisite and due to its general application. 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”, or “rectangular” input cost matrices can be treated.
- A detailed, precise, and clear step-by-step explanation of the Hungarian algorithm is provided in text and via a Java computer animated voice (for listening purposes).
- The ability to change the voice (female or male) that is providing the step-by-step instructions.
- The ability to change and hear the computer animated voice in 3 major languages (English, Chinese, and Spanish).

- The ability to either read input data from a text file, manually input data into the GUI, or randomly generate data with a press of a button.
- A pausing capability to temporarily pause the Hungarian process.
- The ability to print the step-by-step instructions and "final/optimum assignment" results. This allows users/learners the ability to check/verify their "hand calculated" results, which is an important part of their learning process.

It should be emphasized here that existing/documented literatures on the Hungarian algorithm is quite extensive and includes videos and YouTube lectures [32-33]. However, to the best of the author's knowledge, none of the existing tools have offered all of the above desirable features. The remaining section of this chapter is organized as follows. A simple (small-scale) 5x5 matrix example is introduced in Section 3.1 to facilitate the explanation of a modified form of the step-by-step Hungarian algorithm (such as how to handle a "maximization" assignment problem, how to handle cases where the input data is the "rectangular/tall", or "rectangular/fat" matrix). Section 3.2 covers the Java computer animation features 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 highlighted. In Section 3.3 several small-scale matrix examples are used within the Java computer animation and the final results are provided and discussed. Finally, conclusions and future research works are summarized in Section 3.4.

3.1 Summary of the 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).

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

In a "transportation" example, 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 is 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.

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

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

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

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

D. Step 3 – Cover ALL zeros in the current matrix [A] with the "Minimum 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 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 that is not covered by an existing vertical or horizontal line. If this condition is met, the column index for this column is stored, and the 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 or 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 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 met, the row index for this row is stored and the number of zeros is incremented. After looping through all the rows for a column and if it is determined that 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.

$$\begin{bmatrix} 3 & 0 & 2 & 0 & 3 \\ 4 & 1 & 0 & 2 & 1 \\ 0 & 2 & 4 & 5 & 4 \\ 5 & 3 & 0 & 1 & 0 \\ 0 & 6 & 1 & 3 & 2 \end{bmatrix}$$

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 or 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 "Minimum 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 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 or 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 are 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

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

3.2 Visualization and Animation using Java

The software is custom written in Java and is meant to create 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 options. 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 [20-21]. 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.

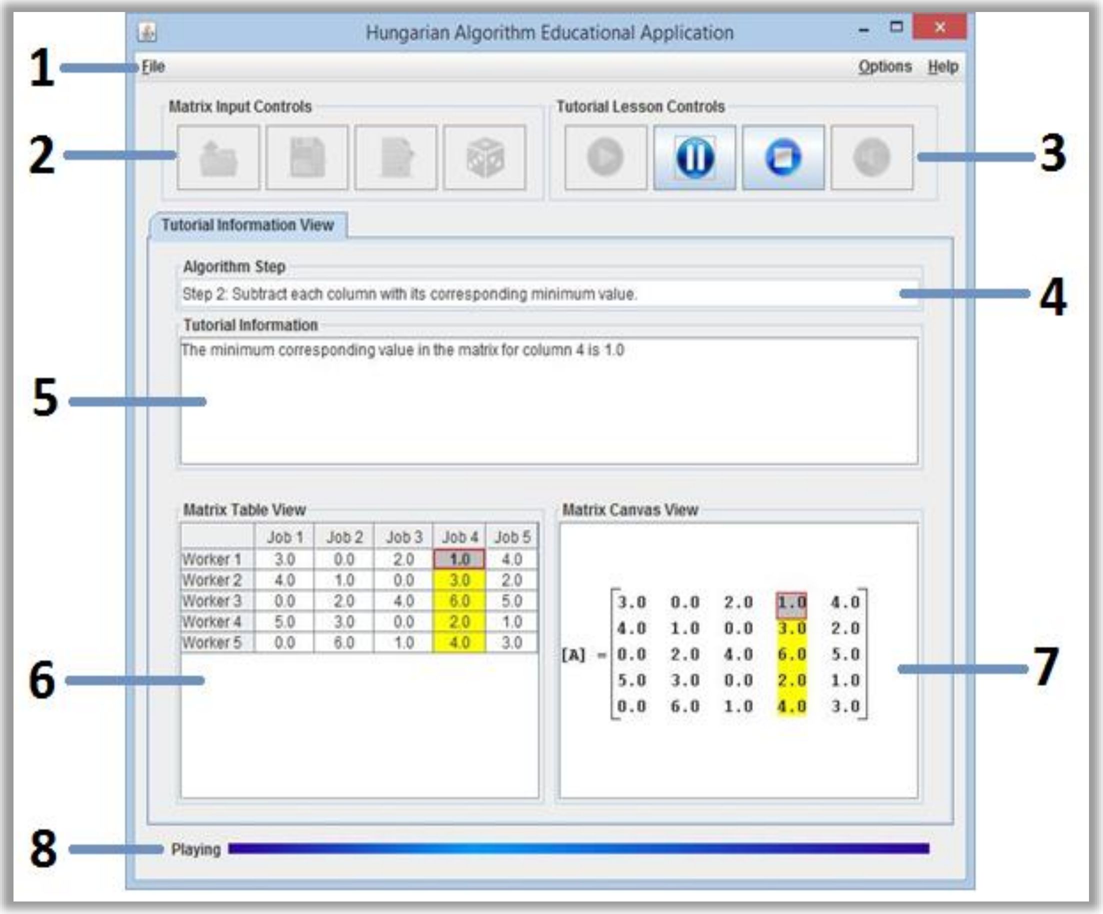
4. Algorithm Step
   Textbox view of the current algorithm step (during play session).

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.



Step 2: Subtract each column with its corresponding minimum value.

The minimum corresponding value in the matrix for column 4 is 1.0

	Job 1	Job 2	Job 3	Job 4	Job 5
Worker 1	3.0	0.0	2.0	1.0	4.0
Worker 2	4.0	1.0	0.0	3.0	2.0
Worker 3	0.0	2.0	4.0	6.0	5.0
Worker 4	5.0	3.0	0.0	2.0	1.0
Worker 5	0.0	6.0	1.0	4.0	3.0

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

Fig. 9. Main Graphical User Interface (GUI).

The given small-scale numerical 5x5 matrix [A] data given in Section 3.1 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.

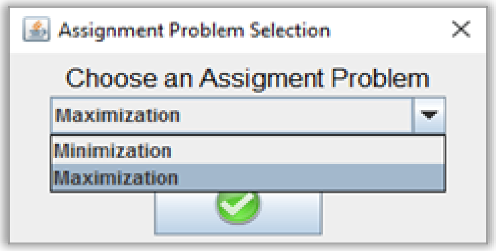


Fig. 10. Assignment Problem Selection.

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 [see Figure 10]. 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.

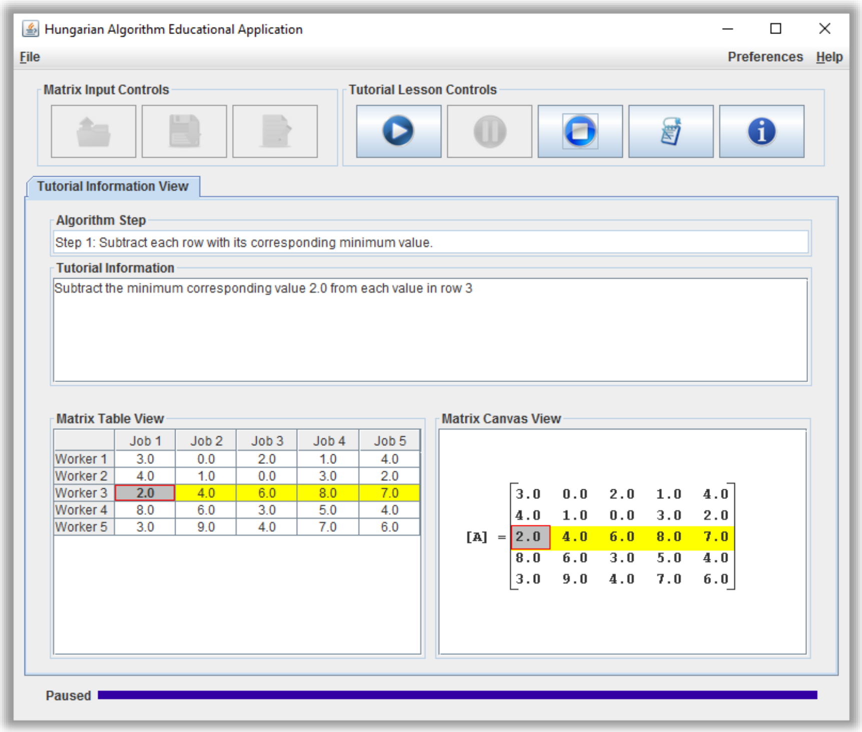
A. Step 0 – “Dummy” Rows (or Columns)

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 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 11 displays the minimum value 2.0 selected for Row 3. The value 2.0 is then subtracted from each value in row 3.

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



Step 1: Subtract each row with its corresponding minimum value.

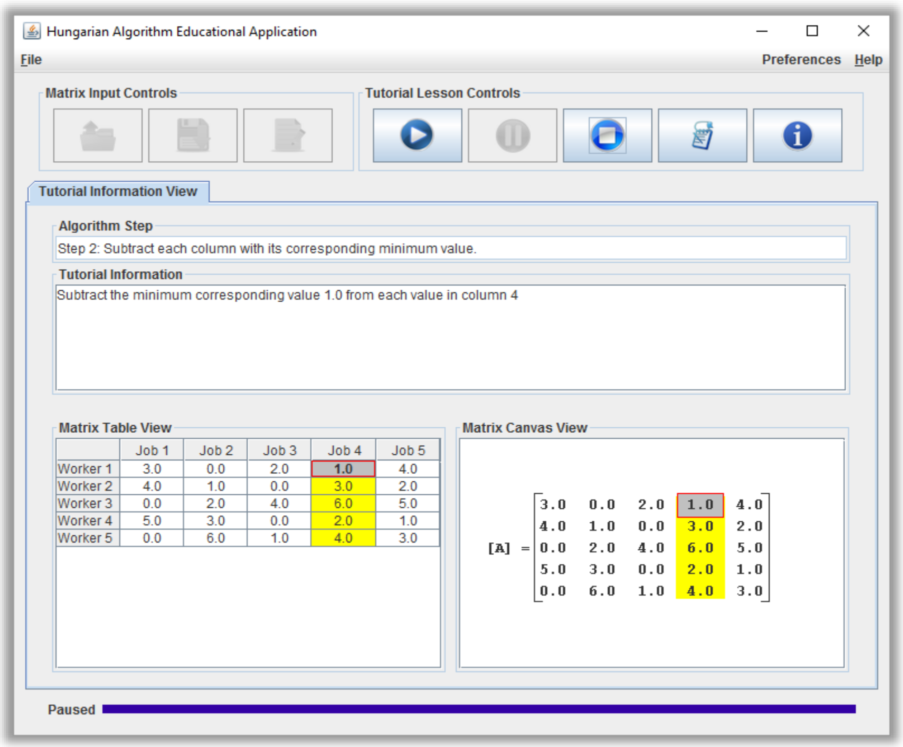
Subtract the minimum corresponding value 2.0 from each value in row 3

	Job 1	Job 2	Job 3	Job 4	Job 5
Worker 1	3.0	0.0	2.0	1.0	4.0
Worker 2	4.0	1.0	0.0	3.0	2.0
Worker 3	2.0	4.0	6.0	8.0	7.0
Worker 4	8.0	6.0	3.0	5.0	4.0
Worker 5	3.0	9.0	4.0	7.0	6.0

$$[A] = \begin{bmatrix} 3.0 & 0.0 & 2.0 & 1.0 & 4.0 \\ 4.0 & 1.0 & 0.0 & 3.0 & 2.0 \\ 2.0 & 4.0 & 6.0 & 8.0 & 7.0 \\ 8.0 & 6.0 & 3.0 & 5.0 & 4.0 \\ 3.0 & 9.0 & 4.0 & 7.0 & 6.0 \end{bmatrix}$$

Fig. 11. Subtracting each Row with its Corresponding Minimum Number.

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 highlighted 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 12 displays the minimum value 1.0 selected for column 4. The value 1.0 is then subtracted from each value in column 4.



Algorithm Step

Step 2: Subtract each column with its corresponding minimum value.

Tutorial Information

Subtract the minimum corresponding value 1.0 from each value in column 4

Matrix Table View

	Job 1	Job 2	Job 3	Job 4	Job 5
Worker 1	3.0	0.0	2.0	1.0	4.0
Worker 2	4.0	1.0	0.0	3.0	2.0
Worker 3	0.0	2.0	4.0	6.0	5.0
Worker 4	5.0	3.0	0.0	2.0	1.0
Worker 5	0.0	6.0	1.0	4.0	3.0

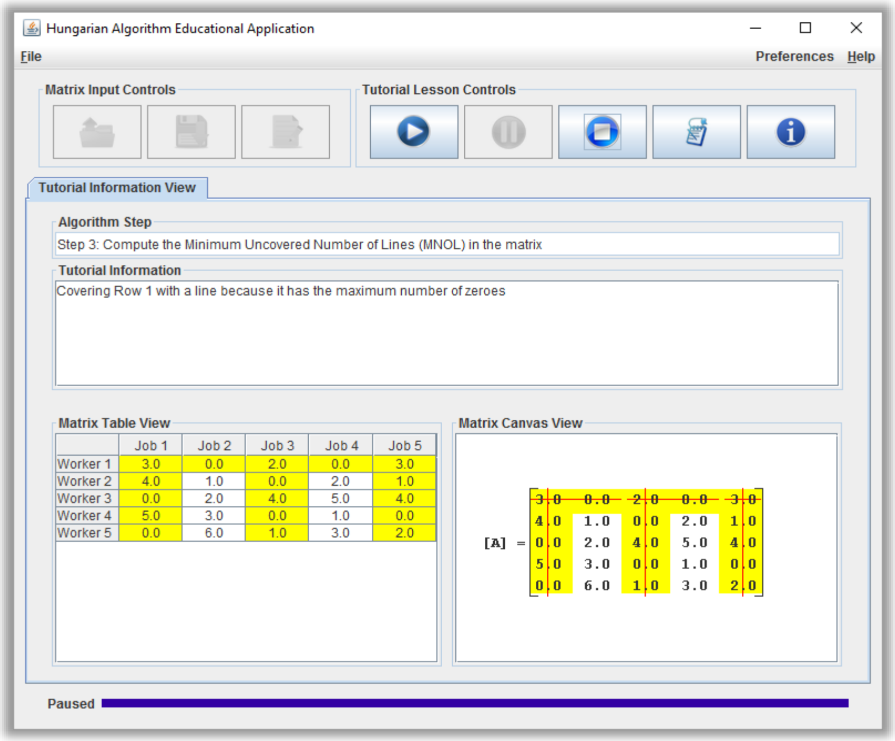
Matrix Canvas View

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

Fig. 12. Subtract each Column with its Corresponding Minimum Number.

D. Step 3 – Cover ALL zeros in the current matrix [A] with the “Minimum 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 3.1.



Algorithm Step

Step 3: Compute the Minimum Uncovered Number of Lines (MNOL) in the matrix

Tutorial Information

Covering Row 1 with a line because it has the maximum number of zeroes

Matrix Table View

	Job 1	Job 2	Job 3	Job 4	Job 5
Worker 1	3.0	0.0	2.0	0.0	3.0
Worker 2	4.0	1.0	0.0	2.0	1.0
Worker 3	0.0	2.0	4.0	5.0	4.0
Worker 4	5.0	3.0	0.0	1.0	0.0
Worker 5	0.0	6.0	1.0	3.0	2.0

Matrix Canvas View

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

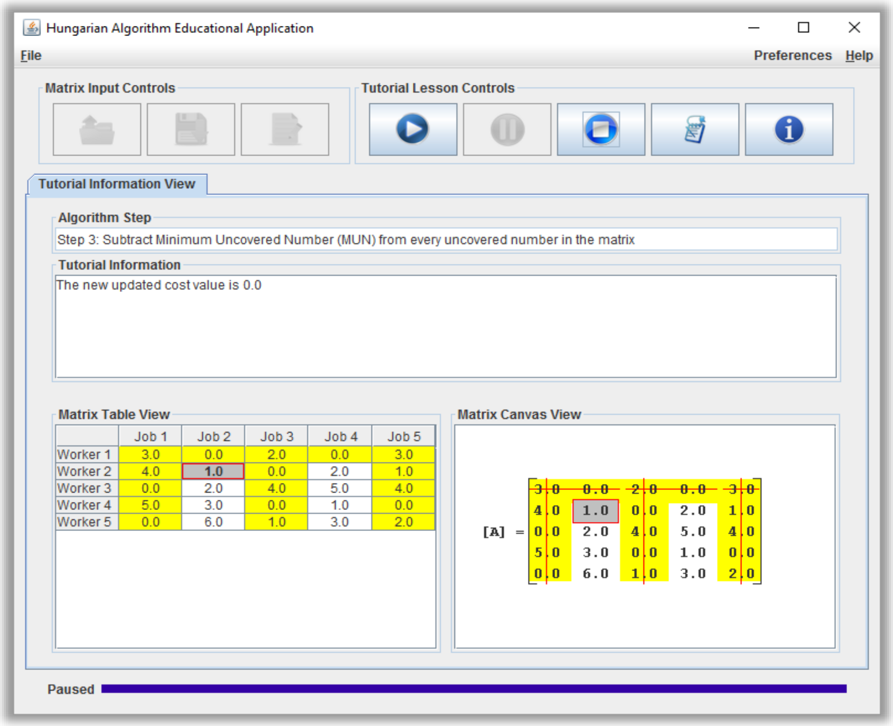
Fig. 13. Cover ALL Zeros with the "Minimum Number of Lines (MNOL)".

First, loop through each row and then each column in matrix [A] to determine which column has the most zeros. In Figure 13, 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 zeros. In Figure 13, row 1 is covered by a horizontal line. In Figure 13, all the zeros are covered 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 "<u>M</u>inimum <u>U</u>ncovered <u>N</u>umber (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 adding the MUN to every number covered with two lines (intersecting lines or yellow highlights) in matrix [A].



Algorithm Step

Step 3: Subtract Minimum Uncovered Number (MUN) from every uncovered number in the matrix

Tutorial Information

The new updated cost value is 0.0

Matrix Table View

	Job 1	Job 2	Job 3	Job 4	Job 5
Worker 1	3.0	0.0	2.0	0.0	3.0
Worker 2	4.0	1.0	0.0	2.0	1.0
Worker 3	0.0	2.0	4.0	5.0	4.0
Worker 4	5.0	3.0	0.0	1.0	0.0
Worker 5	0.0	6.0	1.0	3.0	2.0

Matrix Canvas View

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

Fig. 14. Compute the “Minimum Uncovered Number (MUN)”.

In Figure 14, 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 matrix [A] and added to every number covered by intersecting lines.

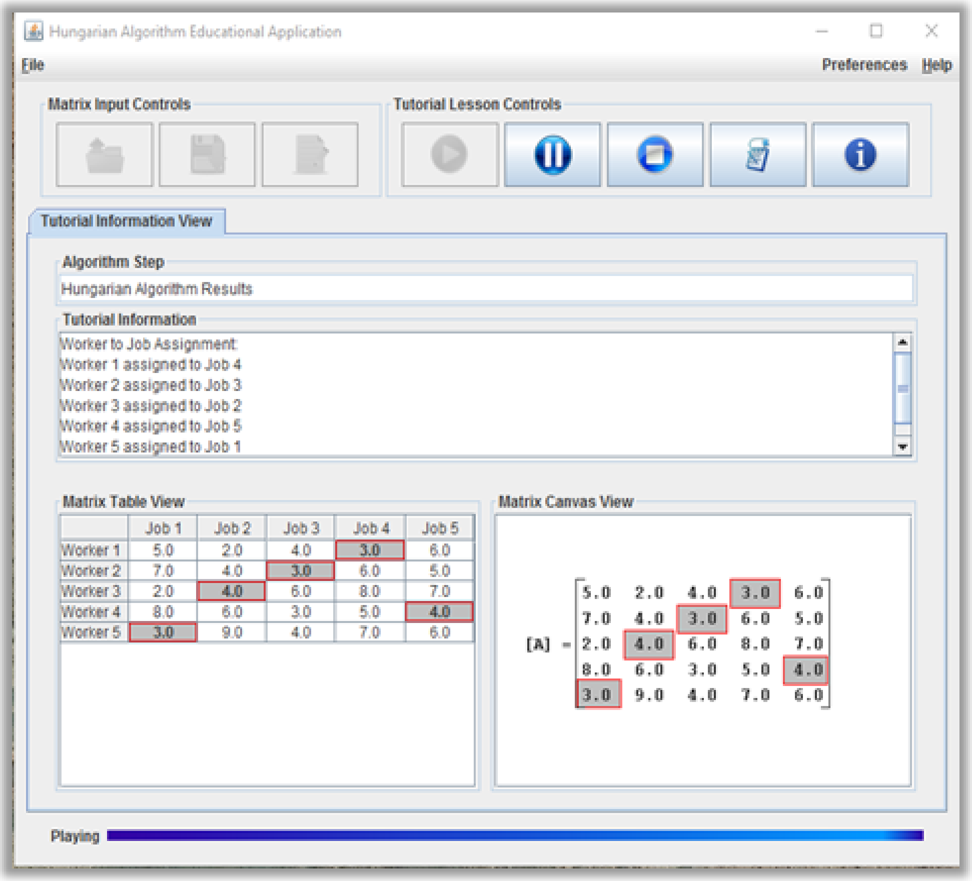
After the MUN has been computed, the horizontal and vertical lines are removed and Step 3 – Cover ALL zeros in the current matrix [A] with the MNOL is repeated. This cycle continues until matrix [A] is converged. Once matrix [A] is converged, we proceed to Step 4.

F. Step 4 – Optimum Assignment

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

In Figure 15, 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



Algorithm Step

Hungarian Algorithm Results

Tutorial Information

Worker to Job Assignment:
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

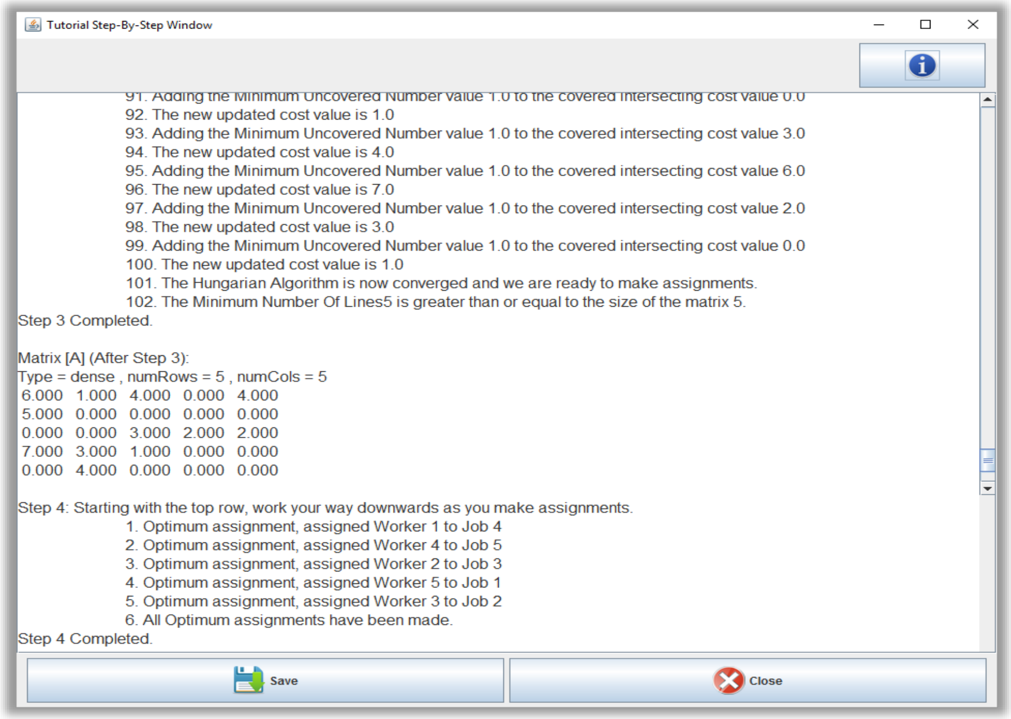
	Job 1	Job 2	Job 3	Job 4	Job 5
Worker 1	5.0	2.0	4.0	3.0	6.0
Worker 2	7.0	4.0	3.0	6.0	5.0
Worker 3	2.0	4.0	6.0	8.0	7.0
Worker 4	8.0	6.0	3.0	5.0	4.0
Worker 5	3.0	9.0	4.0	7.0	6.0

$$[A] = \begin{bmatrix} 5.0 & 2.0 & 4.0 & \boxed{3.0} & 6.0 \\ 7.0 & 4.0 & \boxed{3.0} & 6.0 & 5.0 \\ 2.0 & \boxed{4.0} & 6.0 & 8.0 & 7.0 \\ 8.0 & 6.0 & 3.0 & 5.0 & \boxed{4.0} \\ \boxed{3.0} & 9.0 & 4.0 & 7.0 & 6.0 \end{bmatrix}$$

Fig. 15. 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 16). This window allows these instructions to be exported to an output file at the user's convenience.



```
                 91. Adding the Minimum Uncovered Number value 1.0 to the covered intersecting cost value 0.0
                 92. The new updated cost value is 1.0
                 93. Adding the Minimum Uncovered Number value 1.0 to the covered intersecting cost value 3.0
                 94. The new updated cost value is 4.0
                 95. Adding the Minimum Uncovered Number value 1.0 to the covered intersecting cost value 6.0
                 96. The new updated cost value is 7.0
                 97. Adding the Minimum Uncovered Number value 1.0 to the covered intersecting cost value 2.0
                 98. The new updated cost value is 3.0
                 99. Adding the Minimum Uncovered Number value 1.0 to the covered intersecting cost value 0.0
                 100. The new updated cost value is 1.0
                 101. The Hungarian Algorithm is now converged and we are ready to make assignments.
                 102. The Minimum Number Of Lines5 is greater than or equal to the size of the matrix 5.
Step 3 Completed.

Matrix [A] (After Step 3):
Type = dense , numRows = 5 , numCols = 5
 6.000   1.000   4.000   0.000   4.000
 5.000   0.000   0.000   0.000   0.000
 0.000   0.000   3.000   2.000   2.000
 7.000   3.000   1.000   0.000   0.000
 0.000   4.000   0.000   0.000   0.000

Step 4: Starting with the top row, work your way downwards as you make assignments.
                 1. Optimum assignment, assigned Worker 1 to Job 4
                 2. Optimum assignment, assigned Worker 4 to Job 5
                 3. Optimum assignment, assigned Worker 2 to Job 3
                 4. Optimum assignment, assigned Worker 5 to Job 1
                 5. Optimum assignment, assigned Worker 3 to Job 2
                 6. All Optimum assignments have been made.
Step 4 Completed.
```

Fig. 16. Tutorial Step-By-Step Window.

3.3 Numerical Examples/Results

Several Matrix [A] input data for the Hungarian Algorithm where referenced from the literature [34-37]. These input data were used within the Java computer animation software and executed against the Java computer animation's Hungarian algorithm. The optimum assignment scores from both the literature and from the Java computer animation are discussed and shown in this section.

Figure 17a shows the example input data for the 4x4 Matrix to be solved for the maximum optimum assignment. Figure 17b shows the literature's final maximum optimum assignment score (8 + 3 + 7 + 9 = 27). The final optimum assignment score from the literature was the same when using the Java computer animation (Figure 17c).

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

(a) (b) (c)

Fig. 17. A 4x4 Maximization Assignment Problem Example.

Figure 18a shows the example input data for the 3x3 Matrix to be solved for the maximum optimum assignment. Figure 18b shows the literature's final maximum optimum assignment score (9 + 8 + 3 = 20). The final optimum assignment score from the literature was the same when using the Java computer animation (Figure 18c).

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

(a) (b) (c)

Fig. 18. A 3x3 Maximization Assignment Problem Example.

Figure 19a shows the example input data for the 4x4 Matrix to be solved for the minimum optimum assignment. Figure 19b shows the literature's final minimum optimum assignment score (2 + 7 + 3 + 1 = 13). The final optimum assignment score from the literature was the same when using the Java computer animation (Figure 19c).

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

(a) (b) (c)

Fig. 19. A 4x4 Minimization Assignment Problem Example.

Figure 20a shows the example input data for the 5x5 Matrix to be solved for the minimum optimum assignment. Figure 20b shows the literature's final minimum optimum assignment score (0 + 5 + 1 + 2 + 1 = 9). The final optimum assignment score from the literature was the same when using the Java computer animation (Figure 20c).

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

(a) (b) (c)

Fig. 20. A 5x5 Minimization Assignment Problem Example.

Figure 21a shows the example input data for the 4x5 Matrix to be solved for the minimum optimum assignment. Figure 21b shows the literature's final minimum optimum assignment score (10 + 17 + 7 + 14 = 48). The final optimum assignment score from the literature was the same when using the Java computer animation (Figure 21c).

$$[A] = \begin{bmatrix} 10 & 19 & 8 & 15 \\ 10 & 18 & 7 & 17 \\ 13 & 16 & 9 & 14 \\ 12 & 19 & 8 & 18 \\ 14 & 17 & 10 & 19 \end{bmatrix} \quad [A] = \begin{bmatrix} 10 & 19 & 8 & 15 \\ 10 & 18 & 7 & 17 \\ 13 & 16 & 9 & 14 \\ 12 & 19 & 8 & 18 \\ 14 & 17 & 10 & 19 \end{bmatrix} \quad [A] = \begin{bmatrix} 10 & 19 & 8 & 15 \\ 10 & 18 & 7 & 17 \\ 13 & 16 & 9 & 14 \\ 12 & 19 & 8 & 18 \\ 14 & 17 & 10 & 19 \end{bmatrix}$$

(a) (b) (c)

Fig. 21. A 4x5 Minimization Assignment Problem Example.

The optimum score was the same for all 5 examples given from the literature when executed against the Java computer animation. Some of the literature examples were difficult to understand in writing; however, executing the examples with the Java computer animation gave precise and clear step-by-step instructions for solving for either the minimization or maximization optimum score.

3.4 Conclusions

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 “transportation professionals”, and to help course instructors to avoid the time consuming tasks of designing homework problems and preparing the associated solutions.

Although the “Hungarian” algorithm (for finding the optimum assignment problems) is selected for demonstration purposes (since this topic does not require any specific engineering discipline’s backgrounds as the pre-requisite and due to its general application), other technical topics in a “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 many “unique/desirable” features, that have not yet been seen in the existing literature.

CHAPTER 4

DIJKSTRA ALGORITHM

This chapter covers the Dijkstra Algorithm for finding the shortest time (ST), or the shortest distance (SD) and its corresponding shortest path (SP) to travel from any i-th “source” node to any j-th “destination (or target)” node of a given transportation network is an important, fundamental problem in transportation modelling. Efficient SP algorithms, such as the Label Correction Algorithm (LCA) and its improved version of Polynomial (or partitioned) LCA, forward Dijkstra, backward Dijkstra, Bi-directional Dijkstra, A* algorithms have been developed, tested and well documented in the literatures [38-40]. Teaching the SP algorithms (such as the Dijkstra algorithm), however, can be a difficult/challenging task!

While some teaching information, including lectures, tools, and animation, for Dijkstra algorithms has existed in the literature [41-44], none seems to be suitable/appropriate for our students’ learning environments, due to the lack of one (or more) of the following desirable features/capabilities:

1. The developed software tool should be user friendly (easy to use).
2. Graphical/colorful animation should be extensively used to display equations, and/or intermediate/final output results.
3. Clear/attractive computer animated instructor’s voice should be incorporated in the software tool.
4. The instructor’s voice for teaching materials should be in different/major languages (English, Chinese, and Spanish).
5. User’s input data can be provided in either interactive mode or in edited input data file mode, or by graphical mode.
6. Options for partial (or intermediate) results and/or complete (final results) are available for the user to select.

7. Options for displaying all detailed intermediate results in the first 1-2 iterations, and/or directly show the final answers are available for users.

8. Users/learners can provide their own data and compare hand-calculated results with the computer software's generated results (in each step of the algorithm) for enhancing/improving learning.

The remaining sections of this chapter are organized as follows. In Section 4.1, the basic forward Dijkstra algorithm is summarized (for the readers' convenience). Java is adopted in this work due to its powerful graphical and animated features. Special and useful features of the developed Java software for teaching Dijkstra algorithm are highlighted and demonstrated in (including some computer screen captures) Section 4.2, and the numerical examples and results are shown in Section 4.3. Conclusions are summarized/suggested in Section 4.4.

4.1 Summary of the Dijkstra Algorithm

The basic forward Dijkstra algorithm is a graph search algorithm that solves for the shortest path, time, or distance from any given source node to a destination node. The graph is represented/stored within a 2-Dimentional NxN matrix where the rows and columns of the matrix headers are represented as the nodes and the values within the matrix at a location (A_{ij}) are the link's value (path, distance, or time value) [see Figure 22]. In Figure 22 the simple 3 x 3 matrix shows where Node 1 → Node 2 has a link value of 6, Node 1 → Node 3 have a link value of 4, and Node 2 → Node 3 has a link value of 2.

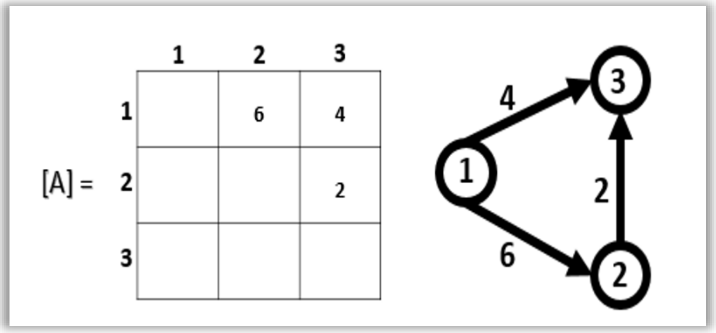
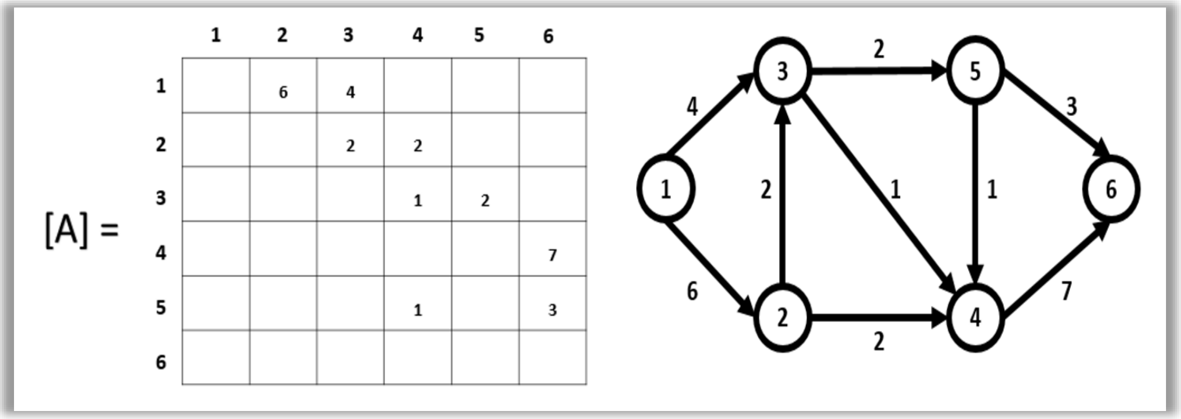


Fig. 22. Matrix and Graph View of Nodes and Links.

With any given NxN size matrix, the shortest path, time, or distance can be solved using the basic forward Dijkstra shortest path algorithm. For example, we demonstrate and solve for a sample 6 x 6 matrix and graph [see Figure 23] to find the shortest path, time, or distance from Node 1 (source node) to Node 6 (destination node).



$$[A] = \begin{array}{c|cccccc} & 1 & 2 & 3 & 4 & 5 & 6 \\ \hline 1 & & 6 & 4 & & & \\ 2 & & & 2 & 2 & & \\ 3 & & & & 1 & 2 & \\ 4 & & & & & & 7 \\ 5 & & & & 1 & & 3 \\ 6 & & & & & & \end{array}$$

$$\begin{tikzcd} & 3 \arrow[r, "2"] \arrow[dr, "1"] & 5 \arrow[d, "1"] \arrow[dr, "3"] & \\ 1 \arrow[ur, "4"] \arrow[dr, "6"'] & & & 6 \\ & 2 \arrow[uu, "2"] \arrow[r, "2"'] & 4 \arrow[ur, "7"'] & \end{tikzcd}$$

Fig. 23. A 6x6 Matrix and Graph View.

We demonstrate and use a simple (easy to use) bookkeeping [see Figure 24] method while iterating through the forward Dijkstra algorithm. Figure 24 shows the initial values for the Set s and Set s prime. Set s is empty during initialization and is used to bookkeep nodes already visited. Set s prime contains all the nodes during initialization and nodes are removed as they are visited. The vector {d} bookkeeps the shortest path, time, or distance value and the vector {pred} bookkeeps the predecessor nodes after each iteration within the Dijkstra algorithm.

During the first iteration, we set the source node to Node 1 and update Set s = {1} and Set s prime = {2, 3, 4, 5, 6}. Note that Node 1 is removed from Set s prime and added to Set s.

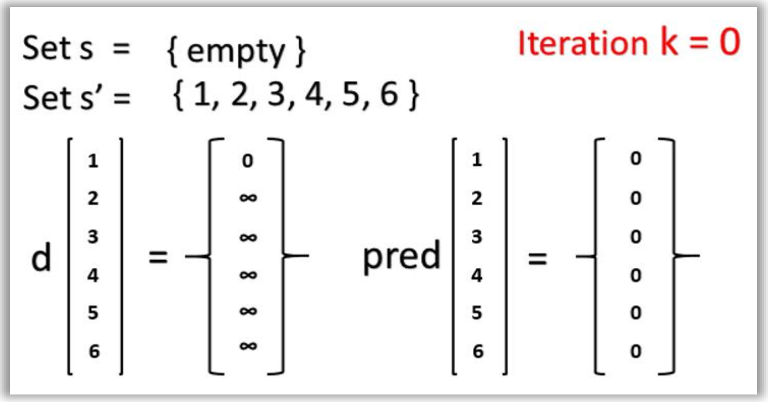
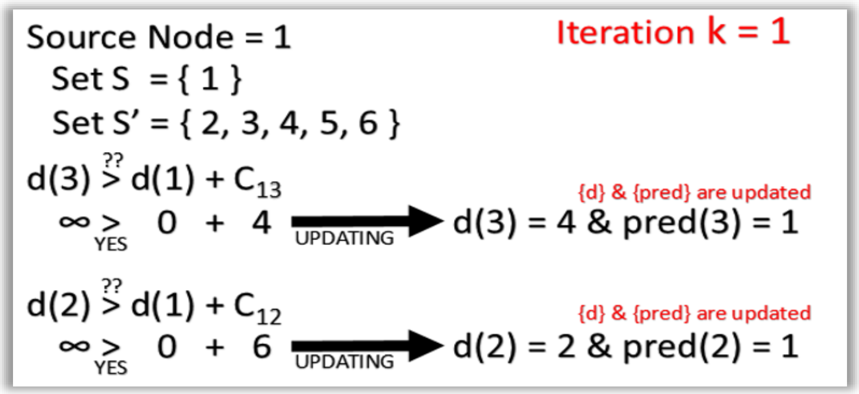


Fig. 24. Data Structures (Iteration k = 0 upon initialization).

From Node 1, there are 2 outgoing links, Node 2 and Node 3 (Figure 23). We check the outgoing Nodes (2 and 3) from Node 1 (Source Node) to determine if Vector [d] and [pred] need to be updated with the shortest path, time or distance. This can be done referencing the values stored in Vector d[2] and d[3]. Compare the stored Vector d[2] value to see if it's greater than the computed value, which is the stored Vector d[1] plus the value at C_{13} (link value between Node 1 and 3). Compare the stored Vector d[3] value to see if it's greater than the computed value, which is the stored Vector d[1] plus C_{12} (link value between Node 1 and 2). If the stored values are greater than the computed values, then update Vector d with the computed value. If not, then no update is needed and the stored value remains the same.

In both of these cases, the stored values for d[2] and d[3] are infinity (∞) so the values in Vector [d] and [pred] will need to be updated with the computed values [see Figure 25]. In Figure 25, the simple (easy to use) bookkeeping method and updates (if needed) are shown. Once this calculation and update has taken place, we determine the smallest value among the nodes in Set {s prime} and then we move the node to the Set {s}. Because Vector d[3] has the smallest value among the nodes in Set {s prime}, we move Node 3 to Set {s} = { 1, 3 } and remove it from Set {s prime} = { 2, 4, 5, 6 }. Now Node 3 becomes the next source node.



Source Node = 1

Set S = { 1 }

Set S’ = { 2, 3, 4, 5, 6 }

Iteration k = 1

$d(3) \overset{??}{>} d(1) + C_{13}$

$\infty \underset{\text{YES}}{>} 0 + 4 \xrightarrow[\text{UPDATING}]{} d(3) = 4 \ \&\ pred(3) = 1$

{d} & {pred} are updated

$d(2) \overset{??}{>} d(1) + C_{12}$

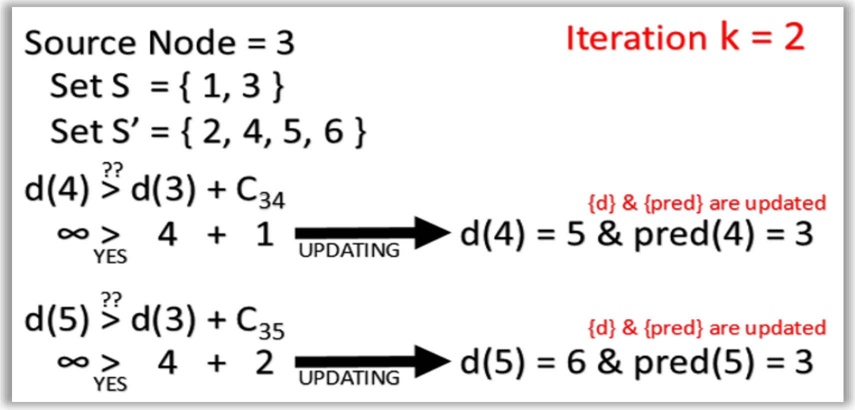
$\infty \underset{\text{YES}}{>} 0 + 6 \xrightarrow[\text{UPDATING}]{} d(2) = 2 \ \&\ pred(2) = 1$

{d} & {pred} are updated

Fig. 25. Data Structures (Iteration k = 1; Source Node = 1)

The Next iteration (k = 2), begins by checking the outgoing nodes from Node 3 (source node) [see Figure 23]. We check the outgoing Nodes (4 and 5) from Node 3 (Source Node) to determine if Vector [d] and [pred] need to be updated with the shortest path, time or distance. This can be done by referencing the values stored in Vector d[4] and d[5]. Compare the stored Vector d[4] value to see if it’s greater than the computed value, which is the stored Vector d[3] plus the value at C_{34} (link value between Node 3 and 4). Compare the stored Vector d[5] value to see if it’s greater than the computed value, which is the stored Vector d[3] plus C_{35} (link value between Node 3 and 5). If the stored values are greater than the computed values, then update the Vector d with the computed value. If not, then no update is needed and the stored value remains the same.

In both of these cases, the stored values for d[4] and d[5] are infinity (∞) so the values in Vector [d] and [pred] will need to be updated with the computed values (Figure 26). In Figure 26, the simple (easy to use) bookkeeping method and updates (if needed) are shown. Once this calculation and update has taken place, we determine the smallest value among the nodes in Set {s prime} and then we move the node to the Set {s}. Because Vector d[4] has the smallest value among the nodes in Set {s prime}, we move Node 4 to Set {s} = { 1, 3, 4 } and remove it from Set {s prime} = { 2, 5, 6 }. Now Node 4 becomes the next source node.



Source Node = 3

Iteration k = 2

Set S = { 1, 3 }

Set S' = { 2, 4, 5, 6 }

$d(4) \overset{??}{>} d(3) + C_{34}$

{d} & {pred} are updated

$\infty \underset{\text{YES}}{>} 4 + 1 \xrightarrow[\text{UPDATING}]{} d(4) = 5 \;\&\; pred(4) = 3$

$d(5) \overset{??}{>} d(3) + C_{35}$

{d} & {pred} are updated

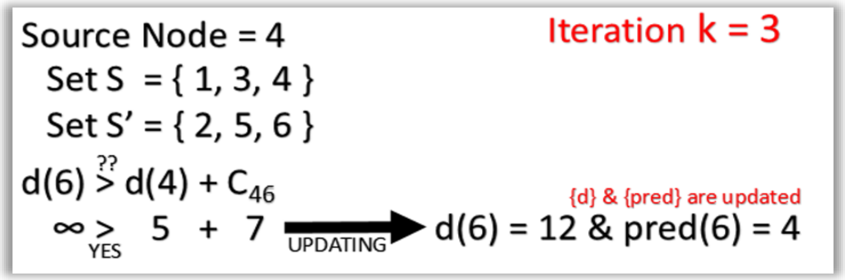
$\infty \underset{\text{YES}}{>} 4 + 2 \xrightarrow[\text{UPDATING}]{} d(5) = 6 \;\&\; pred(5) = 3$

Fig. 26. Data Structures (Iteration k = 2; Source Node = 3).

The Next iteration (k = 3), begins by checking the outgoing nodes from Node 4 (source node) [see Figure 23]. We check the outgoing Node 6 from Node 4 (Source Node) to determine if Vector [d] and [pred] need to be updated with the shortest path, time or distance. This can be done by referencing the values stored in Vector d[6]. Compare the stored Vector d[4] value to see if it's greater than the computed value, which is the stored Vector d[4] plus the value at C_{46} (link value between Node 4 and 6). If the stored value is greater than the computed value, then update the Vector d with the computed value. If not, then no update is needed and the stored value remains the same.

In this case, the stored value for Vector d[6] is infinity (∞) so the value in Vector [d] and [pred] will need to be updated with the computed value (Figure 27). In Figure 27, the simple (easy to use) bookkeeping method and updates (if needed) are shown. Once this calculation and update has taken place, we determine the smallest value among the nodes in Set {s prime} and then we move the node to the Set {s}. Because Vector d[2] and d[5] has the smallest value, 6 among the nodes in Set {s prime}, we arbitrarily select and move Node 2 to Set {s} = { 1, 3, 4, 2

} and remove it from Set {s prime} = { 5, 6 }. Now Node 2 becomes the next source node. Note: we could do further research here to pick the better of the 2 choices instead or arbitrarily selecting one.



Source Node = 4

Set S = { 1, 3, 4 }

Set S' = { 2, 5, 6 }

Iteration k = 3

$d(6) \overset{??}{>} d(4) + C_{46}$

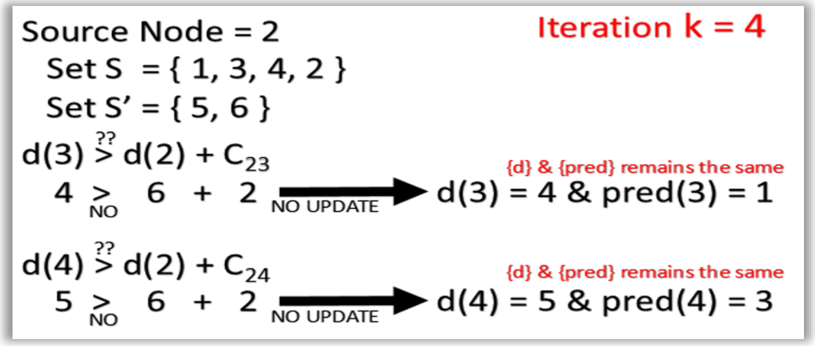
$\infty \underset{\text{YES}}{>} 5 + 7 \xrightarrow[\text{UPDATING}]{} d(6) = 12$ & $pred(6) = 4$

{d} & {pred} are updated

Fig. 27. Data Structures (Iteration k = 3; Source Node = 4).

The Next iteration (k = 4), begins by checking the outgoing nodes from Node 2 (source node) [see Figure 23]. We check the outgoing Nodes (3 and 4) from Node 2 (Source Node) to determine if Vector [d] and [pred] needs to be updated with the shortest path, time or distance. This can be done by referencing the values stored in Vector d[3] and d[4]. Compare the stored Vector d[3] value to see if it's greater than the computed value, which is the stored Vector d[2] plus the value at C_{23} (link value between Node 2 and 3). Compare the stored Vector d[4] value to see if it's greater than the computed value, which is the stored Vector d[2] plus C_{24} (link value between Node 2 and 4). If the stored values are greater than the computed values, then update the Vector d with the computed value. If not, then no update is needed and the stored value remains the same.

In both of these cases, the stored values for Vector d[3] and d[4] are 4 and 5 respectively so the values are not greater than the computed values so no update is needed (Figure 28). In Figure 28, the simple (easy to use) bookkeeping method and updates (if needed) are shown. In this case, no update is needed since the stored values are not greater than the computed values. Once this calculation and update has taken place, we determine the smallest value among the nodes in Set {s prime} and then we move the node to the Set {s}. Because Vector d[5] has the smallest value among the nodes in Set {s prime}, we move Node 5 to Set {s} = { 1, 3, 4, 2, 5 } and remove it from Set {s prime} = { 6 }. Now Node 5 becomes the next source node. Note, when determining the smallest value was a tie for Nodes 2 and 5 (see previous iteration), we were better off selecting and moving Node 5 instead of arbitrarily selecting Node 2.


Source Node = 2

Iteration k = 4

Set S = { 1, 3, 4, 2 }

Set S' = { 5, 6 }

$d(3) \overset{??}{>} d(2) + C_{23}$

$4 \underset{NO}{>} 6 + 2 \xrightarrow[\text{NO UPDATE}]{} d(3) = 4$ & pred(3) = 1

{d} & {pred} remains the same

$d(4) \overset{??}{>} d(2) + C_{24}$

$5 \underset{NO}{>} 6 + 2 \xrightarrow[\text{NO UPDATE}]{} d(4) = 5$ & pred(4) = 3

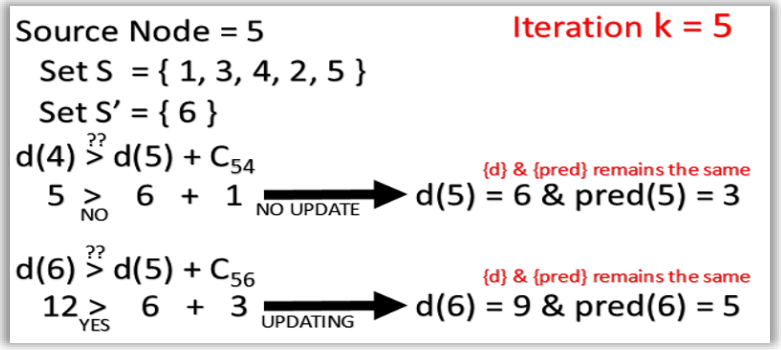
{d} & {pred} remains the same

Fig. 28. Data Structures (Iteration k = 4; Source Node = 2).

The Next iteration (k = 5), begins by checking the outgoing nodes from Node 5 (source node) [see Figure 23]. We check the outgoing Nodes (4 and 6) from Node 5 (Source Node) to determine if Vector [d] and [pred] need to be updated with the shortest path, time or distance. This can be done by referencing the values stored in Vector d[4] and d[6]. Compare the stored Vector d[4] value to see if it's greater than the computed value, which is the stored Vector d[5] plus the value at C_{54} (link value between Node 5 and 4). Compare the stored Vector d[6] value to see if it's greater than the computed value, which is the stored Vector d[5] plus C_{56} (link value between Node 5 and 6). If the stored values are greater than the computed values, then update the Vector d with the computed value. If not, then no update is needed, and the stored value remains the same.

In both of these cases, the stored values for d[4] and d[6] are 5 and 12 respectively, so the value for d[4] is not greater than the computed values so no update is needed (Figure 29).

However, the value for Vector d[6] is greater than the computed values in Vector [d] and [pred] will need to be updated with the computed values (Figure 29). In Figure 29, the simple (easy to use) bookkeeping method and updates (if needed) are shown. In this case, one outgoing node update is needed since the stored values are greater than the computed values. Once this calculation and update has taken place, we determine the smallest value among the nodes in Set {s prime} and then we move the node to the Set {s}. Because Vector d[6] is the only remaining node and is the destination node, we move Node 6 to Set {s} = { 1, 3, 4, 2, 5, 6 } and remove it from Set {s prime} = { empty }. Now Node 6 becomes the next source node and is our destination node. This completes the shortest path, time, or distance for solving for the basic forward Dijkstra algorithm.



Source Node = 5

Iteration k = 5

Set S = { 1, 3, 4, 2, 5 }

Set S' = { 6 }

$d(4) \overset{??}{>} d(5) + C_{54}$

{d} & {pred} remains the same

$5 \underset{\text{NO}}{>} 6 + 1 \xrightarrow[\text{NO UPDATE}]{} d(5) = 6 \ \& \ pred(5) = 3$

$d(6) \overset{??}{>} d(5) + C_{56}$

{d} & {pred} remains the same

$12 \underset{\text{YES}}{>} 6 + 3 \xrightarrow[\text{UPDATING}]{} d(6) = 9 \ \& \ pred(6) = 5$

Fig. 29. Data Structures (During Iteration k = 5; Source Node = 5).

The updates for each iteration for Vector [d], Vector [pred], and Set {s} are displayed in Figure 30. For each iteration, the highlighted circle (light blue circle) shows the corresponding values being updated during that iteration.

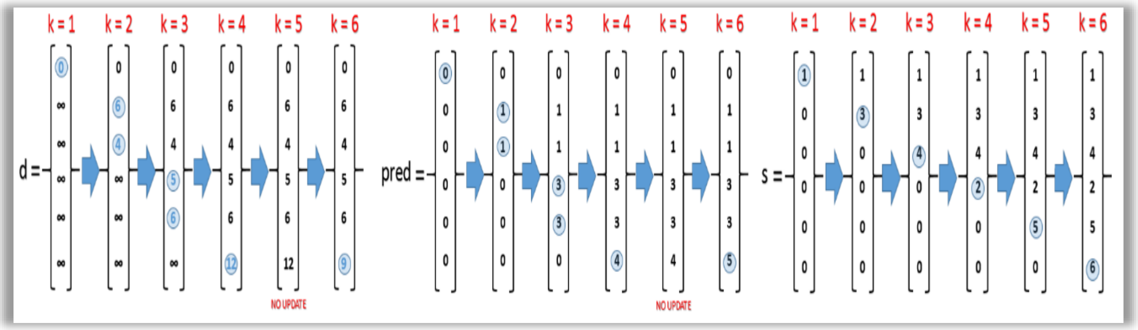


Fig. 30. Data Structures (Vector [d] and [pred], and Set {s} iteration updates).

4.2 Visualization and Animation using Java

The previous section's small-scale 6 x 6 Matrix [A] graph data will be used for the Java [19-21] computer animated software tool for teaching the Dijkstra algorithm. The user/learner will initialize and run the Java software. Once initialized, the application's "Main Control" Graphical User Interface (GUI) is displayed (Figure 31). This GUI controls the flow of the basic forward Dijkstra algorithm teaching steps from start to finish and will provide detailed algorithm steps while solving for the shortest path, time, or distance.

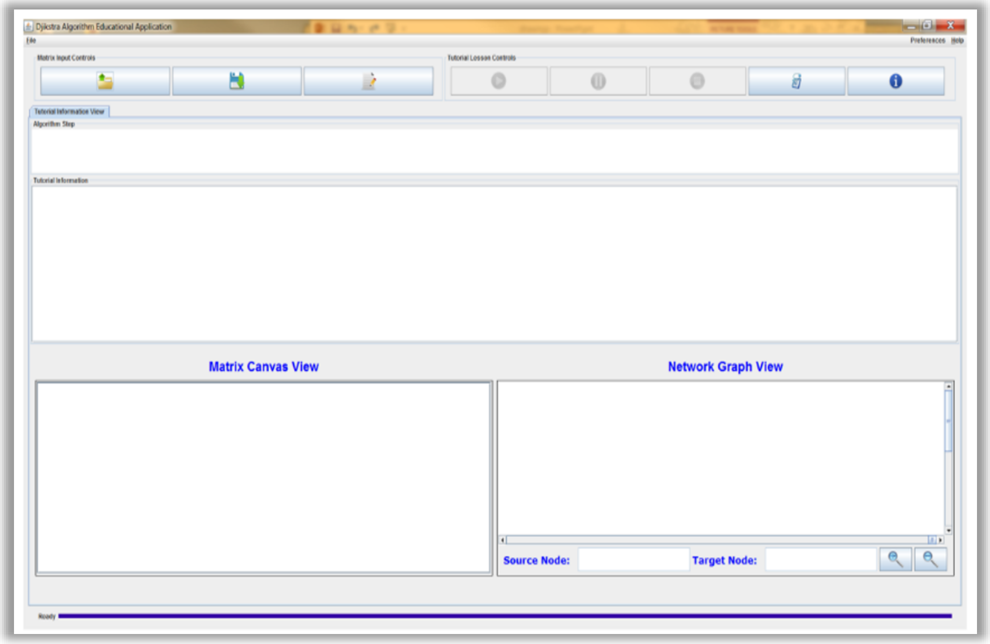
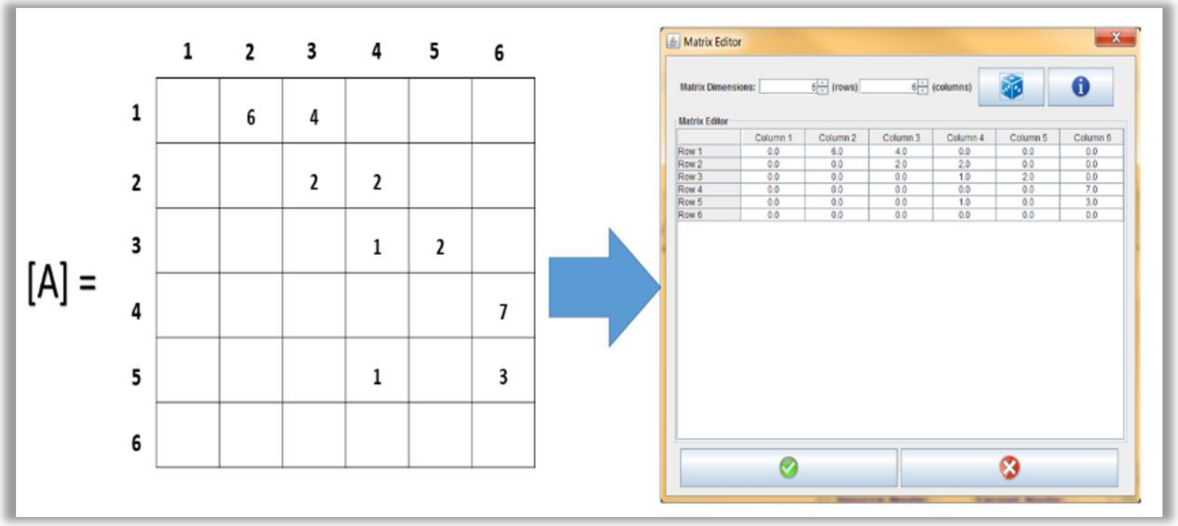


Fig. 31. Dijkstra Graphical User Interface.

The user/learner will be able to input the matrix [A] data from several input options (input file, manual input or randomly generated values) from the “Main Control” GUI. When an input option is selected the user/learner will be able to modify the data within the “Matrix Editor” GUI. The “Matrix Editor” GUI (Figure 32) allows the user/learner to change the size of the matrix [A] dimensions and modify the values before solving for the shortest path, time, or distance.

After the Matrix data [A] is entered into the "Matrix Editor" and "Ok" is selected, the focus is returned to the "Main Control" GUI. The matrix data [A] is shown in to viewable formats, the “Matrix Canvas”, "Data", and "Network Graph" views. Both views will be updated accordingly throughout the steps in the basic forward Dijkstra Algorithm. Once the matrix data [A] is input, the "Play" button is enabled.



$$[A] = \begin{array}{c|cccccc} & 1 & 2 & 3 & 4 & 5 & 6 \\ \hline 1 & & 6 & 4 & & & \\ 2 & & & 2 & 2 & & \\ 3 & & & & 1 & 2 & \\ 4 & & & & & & 7 \\ 5 & & & & 1 & & 3 \\ 6 & & & & & & \end{array}$$

Matrix Editor

Matrix Dimensions: 6 (rows) 6 (columns)

Matrix Editor

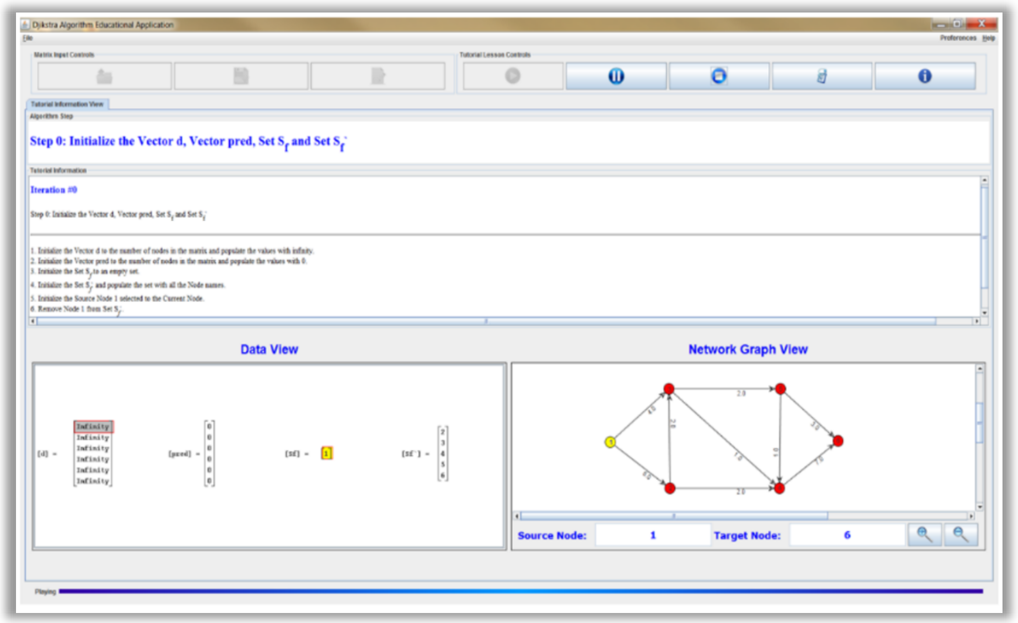
	Column 1	Column 2	Column 3	Column 4	Column 5	Column 6
Row 1	0.0	6.0	4.0	0.0	0.0	0.0
Row 2	0.0	0.0	2.0	2.0	0.0	0.0
Row 3	0.0	0.0	0.0	1.0	2.0	0.0
Row 4	0.0	0.0	0.0	0.0	0.0	7.0
Row 5	0.0	0.0	0.0	1.0	0.0	3.0
Row 6	0.0	0.0	0.0	0.0	0.0	0.0

Fig. 32. Matrix Editor (Matrix [A] Data).

Once the user/learner selects the "Play" button, the teaching of the basic forward Dijkstra Algorithm steps begins. The basic forward Dijkstra Algorithm performs the algorithm steps against the matrix data [A], the matrix data [A] is updated accordingly and displayed within the Views, and the algorithm steps are conveyed to the user/learner by a computer animated voice.

A. Step 0 – Initialize the Vector d, Vector pred, Set S_f and $S_{f\ prime}$.

The Java Computer Animation for Teaching the Forward Dijkstra Algorithm will handle the matrix data if provided as a rectangular or tall matrix. The algorithm will add "dummy" rows (or columns) with the maximum corresponding value within the matrix. The Java Computer Animation for Teaching the Forward Dijkstra Algorithm will solve for shortest path, time, or distance. The GUI will prompt an input dialog for the user/learner to select the source and destination nodes. The Java Computer Animation for Teaching the Forward Dijkstra Algorithm will initialize the data vectors and sets and set the source node (Figure 33).



Step 0: Initialize the Vector d, Vector pred, Set S_f and Set $S_f^{'}$

Iteration #0

Step 0: Initialize the Vector d, Vector pred, Set S_f and Set $S_f^{'}$

1. Initialize the Vector d to the number of nodes in the matrix and populate the values with infinity.
2. Initialize the Vector pred to the number of nodes in the matrix and populate the values with 0.
3. Initialize the Set S_f to an empty set.
4. Initialize the Set $S_f^{'}$ and populate the set with all the Node names.
5. Initialize the Source Node 1 selected to the Current Node.
6. Remove Node 1 from Set $S_f^{'}$.

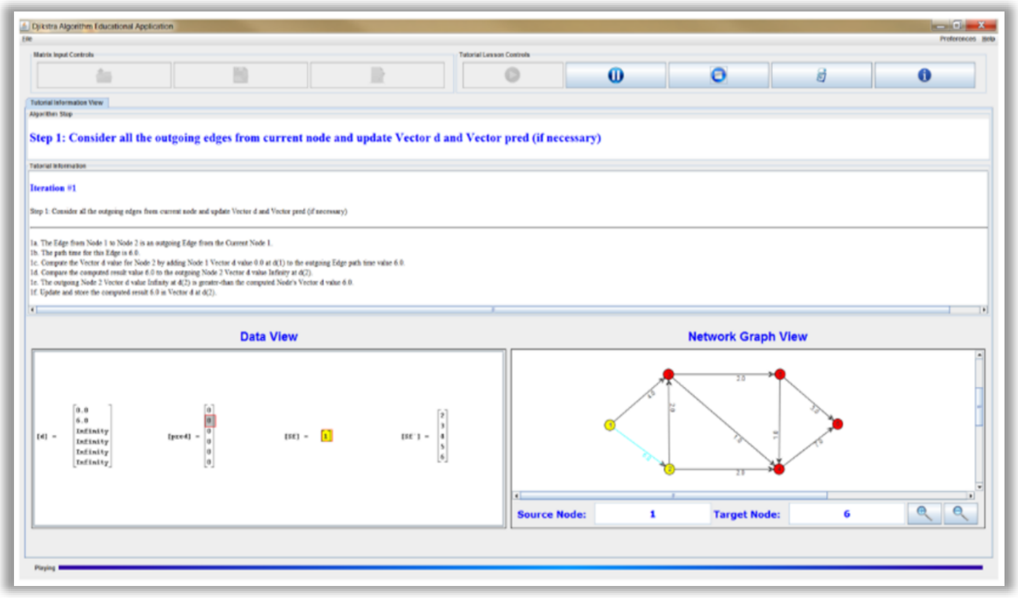
$[d] = \begin{bmatrix} \text{Infinity} \\ \text{Infinity} \\ \text{Infinity} \\ \text{Infinity} \\ \text{Infinity} \\ \text{Infinity} \end{bmatrix}$ $[pred] = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$ $[Sf] = [1]$ $[Sf^{'}] = \begin{bmatrix} 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{bmatrix}$

Source Node: 1 Target Node: 6

Fig. 33. Dijkstra GUI During Step 0: Initialization.

B. Step 1 – Consider ALL Outgoing Edges from the Current Node & Update Vector d and Vector pred (if necessary).

The Java Computer Animation for Teaching the Forward Dijkstra Algorithm will consider all outgoing links from the current node. The Java Computer Animation for Teaching the Forward Dijkstra Algorithm will determine if the stored Vector [d] values for the outgoing nodes are greater than the computed values of the previous node plus the link (edge) values. If the stored Vector [d] value is greater than the computed value then the Vector [d] value is updated accordingly; otherwise, the value remains the same. The Java Computer Animation for Teaching the Forward Dijkstra Algorithm will have animated each step and provided detailed animated voice and text for each step (Figure 34).



Step 1: Consider all the outgoing edges from current node and update Vector d and Vector pred (if necessary)

Iteration #1

Step 1: Consider all the outgoing edges from current node and update Vector d and Vector pred (if necessary)

1a. The Edge from Node 1 to Node 2 is an outgoing Edge from the Current Node 1.
1b. The path time for this Edge is 6.0.
1c. Compute the Vector d value for Node 2 by adding Node 1 Vector d value 0.0 at d(1) to the outgoing Edge path time value 6.0.
1d. Compare the computed result value 6.0 to the outgoing Node 2 Vector d value Infinity at d(2).
1e. The outgoing Node 2 Vector d value Infinity at d(2) is greater-than the computed Node's Vector d value 6.0.
1f. Update and store the computed result 6.0 in Vector d at d(2).

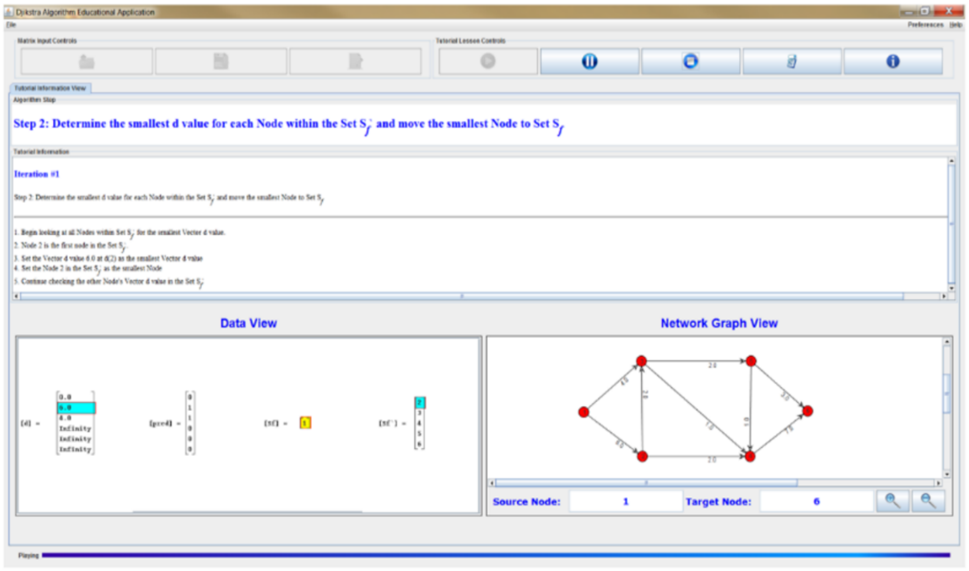
$$[d] = \begin{bmatrix} 0.0 \\ 6.0 \\ \text{Infinity} \\ \text{Infinity} \\ \text{Infinity} \\ \text{Infinity} \end{bmatrix} \quad [pred] = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad [Sf] = \begin{bmatrix} 1 \end{bmatrix} \quad [Sf'] = \begin{bmatrix} 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{bmatrix}$$

Source Node: 1 Target Node: 6

Fig. 34. Dijkstra GUI During Step 1: Outgoing Edges & Update Bookkeeping.

C. Step 2 – Determine the Smallest d Value for Each Node Within the Set $S_{f\ prime}$ and Move the Smallest Node to Set S_f.

The Java Computer Animation for Teaching the Forward Dijkstra Algorithm will determine the smallest value from the Set {s prime} and remove the smallest from Set {s prime} and add it to Set {s}. The Java Computer Animation for Teaching the Forward Dijkstra Algorithm will update the current node as the source node for the next iteration (Figure 35).



Step 2: Determine the smallest d value for each Node within the Set S_f' and move the smallest Node to Set S_f

Iteration #1

Step 2: Determine the smallest d value for each Node within the Set S_f' and move the smallest Node to Set S_f

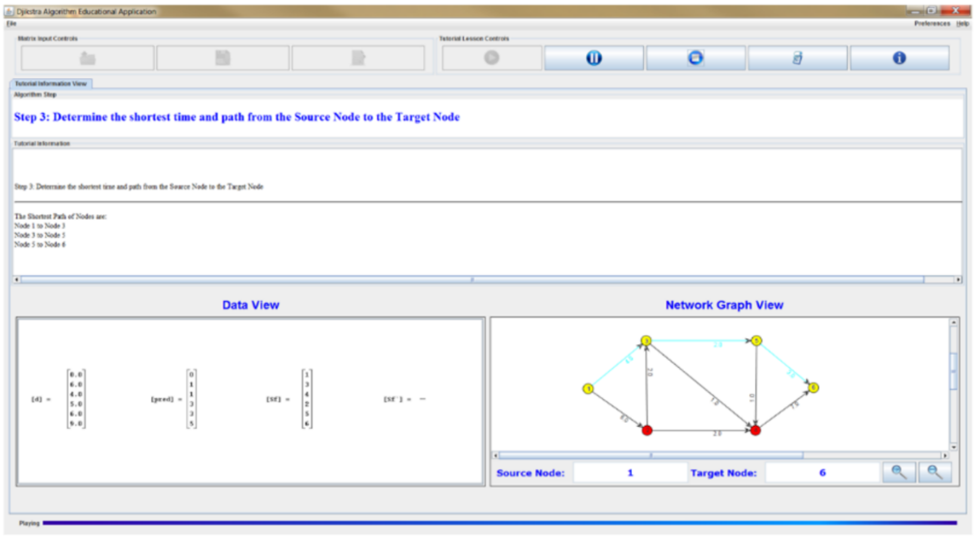
1. Begin looking at all Nodes within Set S_f' for the smallest Vector d value.
2. Node 2 is the first node in the Set S_f'.
3. Set the Vector d value 6.0 at d(2) as the smallest Vector d value
4. Set the Node 2 in the Set S_f' as the smallest Node
5. Continue checking the other Node's Vector d value in the Set S_f'

$$[d] = \begin{bmatrix} 0.0 \\ 6.0 \\ 4.0 \\ \text{Infinity} \\ \text{Infinity} \\ \text{Infinity} \end{bmatrix} \quad [pred] = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad [Sf] = \begin{bmatrix} 1 \end{bmatrix} \quad [Sf'] = \begin{bmatrix} 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{bmatrix}$$

Fig. 35. Dijkstra GUI During Step 2.

D. Step 3 – Determine the Shortest Time and Path from the Source Node to the Target Node.

The Java Computer Animation for Teaching the Forward Dijkstra Algorithm will determine the shortest path, time, or distance based on the outcome of the Dijkstra algorithms using the bookkeeping of Vector [d], Vector [pred], Set {s}, Set {s prime}. The Java Computer Animation for Teaching the Forward Dijkstra Algorithm will highlight the shortest path (Figure 36).



Step 3: Determine the shortest time and path from the Source Node to the Target Node

Step 3: Determine the shortest time and path from the Source Node to the Target Node

The Shortest Path of Nodes are:
Node 1 to Node 3
Node 3 to Node 5
Node 5 to Node 6

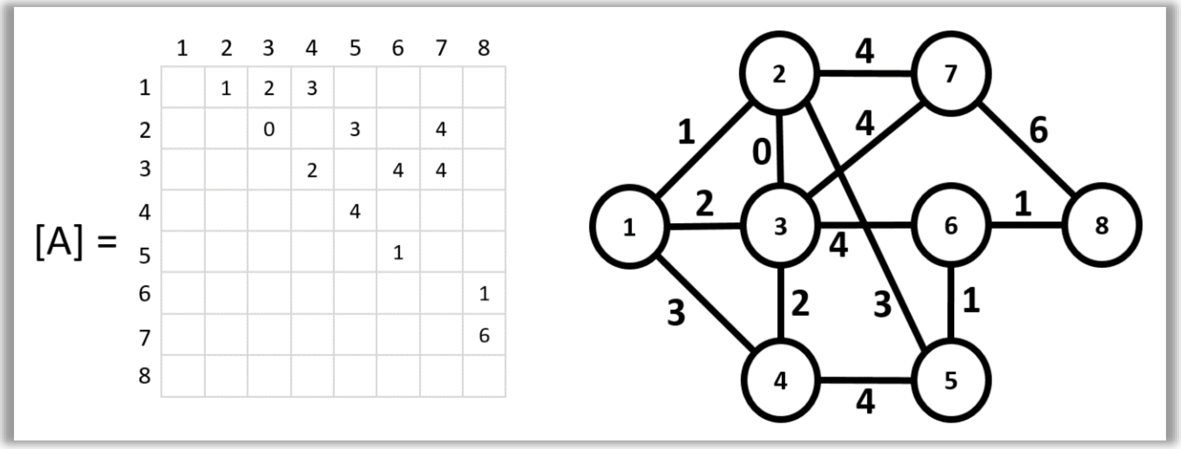
$$[d] = \begin{bmatrix} 0.0 \\ 6.0 \\ 4.0 \\ 5.0 \\ 6.0 \\ 9.0 \end{bmatrix} \quad [pred] = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 3 \\ 3 \\ 5 \end{bmatrix} \quad [Sf] = \begin{bmatrix} 1 \\ 3 \\ 4 \\ 2 \\ 5 \\ 6 \end{bmatrix} \quad [Sf'] = -$$

Fig. 36. Dijkstra GUI During Step 3.

The Java Computer Animation for Teaching the Forward Dijkstra Algorithm will display the final results of the shortest path, time, or distance by highlighting the nodes and links from the source to destination node.

4.3 Numerical Examples/Results

Several Matrix [A] input data (graph data) for the Dijkstra Algorithm were referenced from the literature [45-48]. These input data were used within the Java computer animation software and executed against the Java computer animation's Dijkstra algorithm. The algorithm's shortest path from both the literature and from the Java computer animation are discussed and shown in this section.



$$[A] = \begin{array}{c|cccccccc}
 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\
\hline
1 & & 1 & 2 & 3 & & & & \\
2 & & & 0 & & 3 & & 4 & \\
3 & & & & 2 & & 4 & 4 & \\
4 & & & & & 4 & & & \\
5 & & & & & & 1 & & \\
6 & & & & & & & & 1 \\
7 & & & & & & & & 6 \\
8 & & & & & & & &
\end{array}$$

Fig. 37. Dijkstra 8x8 Matrix and Network Graph Example.

Figure 37 shows an example input data for the 8x8 Matrix and Network Graph to be solved for the shortest path using the Forward Dijkstra algorithm. This is one example taken from the literature to be compared with the results of the Java computer animation. The literature solves this shortest path problem with Node 1 as the given source node and Node 8 as the target node.

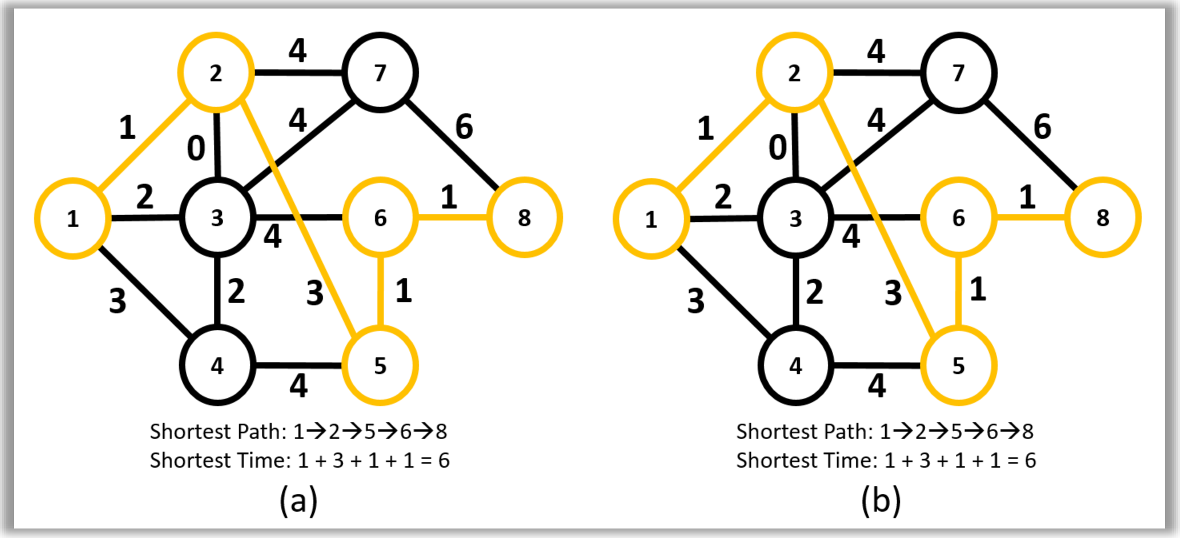

Shortest Path: 1→2→5→6→8
Shortest Time: 1 + 3 + 1 + 1 = 6
(a)
Shortest Path: 1→2→5→6→8
Shortest Time: 1 + 3 + 1 + 1 = 6
(b)

Fig. 38. Dijkstra 8x8 Matrix and Network Graph Results.

The shortest path results from the literature showed that the shortest time was 6 with these nodes selected (Node 1→2→5→6→8) as the shortest path [see Figure 38a]. The same input data was used with the Java computer animation, and the final results were the same as the literature [see Figure 38b].

$$[A] = \begin{array}{c|ccccccccc} & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ \hline 1 & & 2 & & 9 & 5 & & & & \\ 2 & & & 1 & & 3 & & & & \\ 3 & & & & & 1 & & 6 & & \\ 4 & & & & & & & & 4 & \\ 5 & & & & 2 & & 9 & & 7 & \\ 6 & & & & & & & 5 & & 1 \\ 7 & & & & & & & & & 5 \\ 8 & & & & & & 1 & & & 5 \\ 9 & & & & & & & & & \end{array}$$

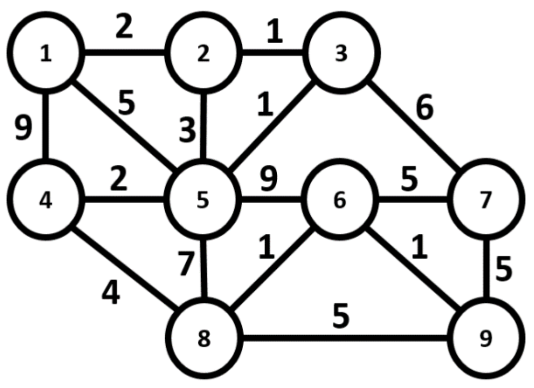


Fig. 39. Dijkstra 9x9 Matrix and Network Graph Example.

Figure 39 shows another example input data for the 9x9 Matrix and Network Graph to be solved for the shortest path using the Forward Dijkstra algorithm. This is another example taken from the literature to be compared with the results of the Java computer animation. The literature solves this shortest path problem with Node 1 as the given source node and Node 9 as the target node.

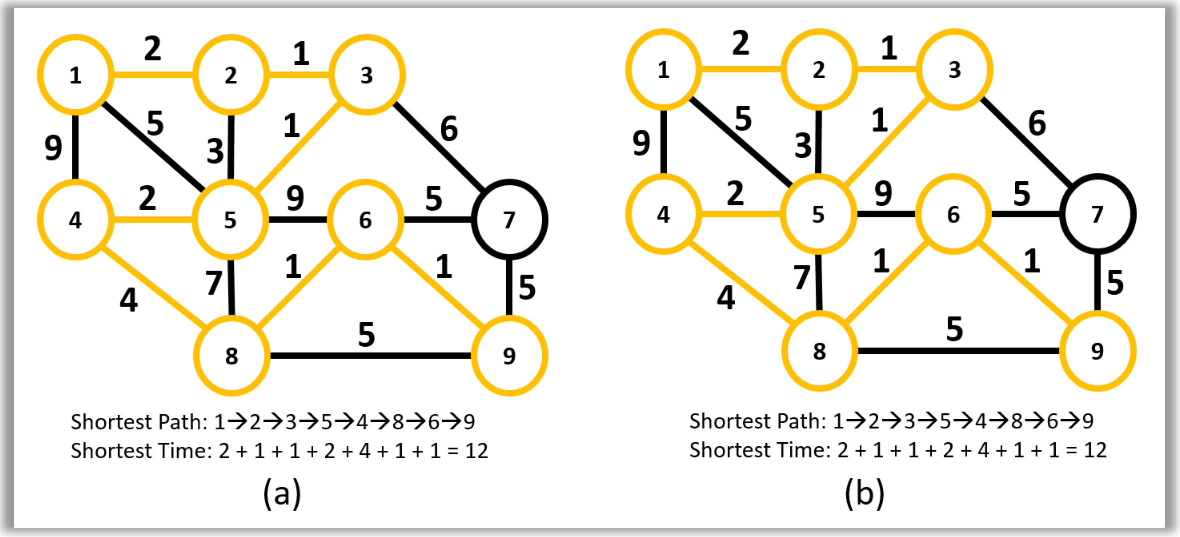

Shortest Path: 1→2→3→5→4→8→6→9
Shortest Time: 2 + 1 + 1 + 2 + 4 + 1 + 1 = 12
(a)
Shortest Path: 1→2→3→5→4→8→6→9
Shortest Time: 2 + 1 + 1 + 2 + 4 + 1 + 1 = 12
(b)

Fig. 40. Dijkstra 9x9 Matrix and Network Graph Results.

The shortest path results from the literature showed that the shortest time was 12 with these nodes selected (Node 1→2→3→5→4→8→6→9) as the shortest path [see Figure 40a]. The same input data was used with the Java computer animation and the final results were the same as the literature [see Figure 40b].

$$[A] = \begin{array}{c|cccccccc} & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ \hline 1 & & 1 & 2 & & & & & \\ 2 & 1 & & 1 & 5 & 2 & & & \\ 3 & 2 & 1 & & 2 & 1 & 4 & & \\ 4 & & 5 & 2 & & 3 & 6 & 8 & \\ 5 & & 2 & 1 & 3 & & & 7 & \\ 6 & & & 4 & 6 & & & 5 & 2 \\ 7 & & & & 8 & 7 & 5 & & 6 \\ 8 & & & & & & 2 & 6 & \end{array}$$

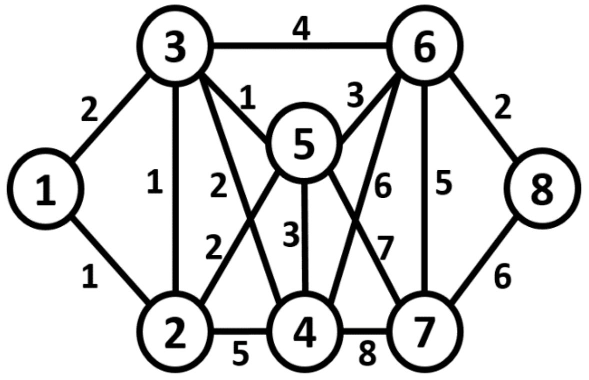


Fig. 41. Dijkstra 8x8 Matrix and Network Graph Example.

Figure 41 shows another example of input data for the 8x8 Matrix and Network Graph to be solved for the shortest path using the Forward Dijkstra algorithm. This is another example taken from another literature to be compared with the results of the Java computer animation. The literature solves this shortest path problem with Node 1 as the given source node and Node 8 as the target node.

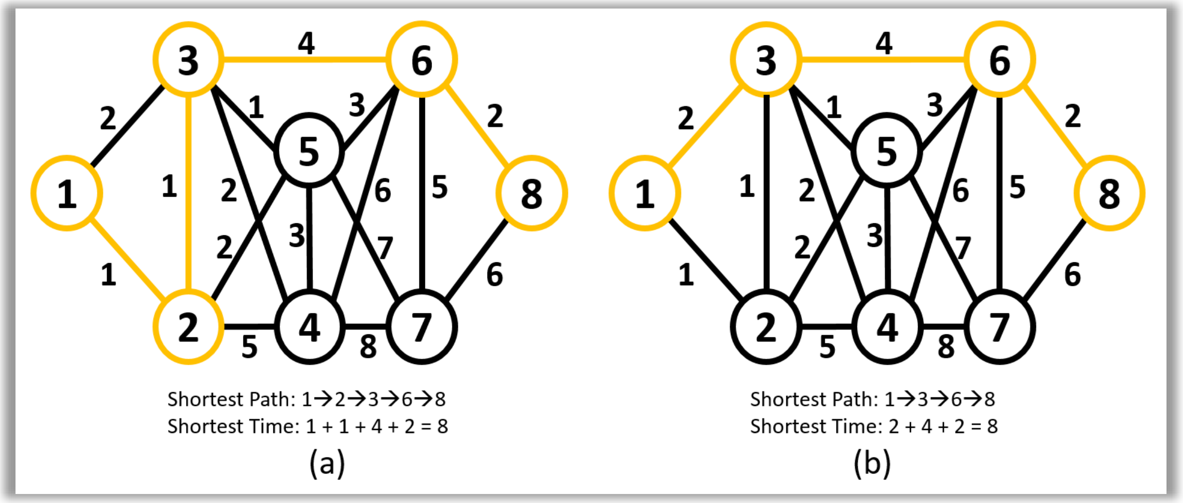
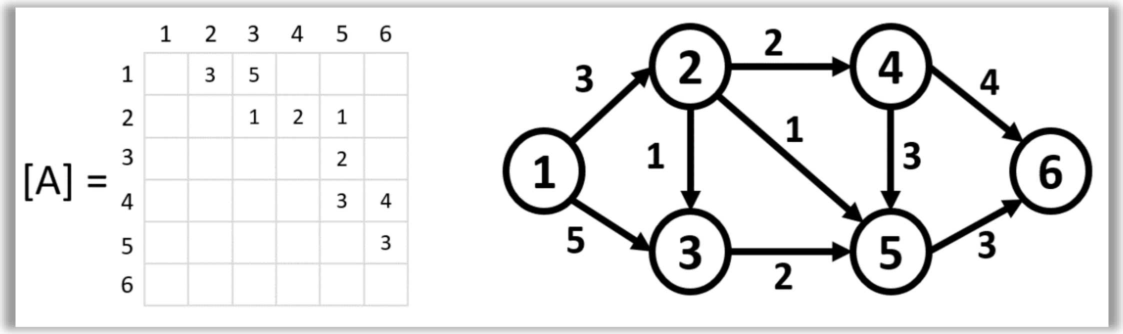

Shortest Path: 1→2→3→6→8
Shortest Time: 1 + 1 + 4 + 2 = 8
(a)
Shortest Path: 1→3→6→8
Shortest Time: 2 + 4 + 2 = 8
(b)

Fig. 42. Dijkstra 8x8 Matrix and Network Graph Results.

The shortest path results from another source in the literature showed that the shortest time was 8 with these nodes selected (Node 1→2→3→6→8) as the shortest path [see Figure 38a]. The same input data was used with the Java computer animation, and the shortest path time results were the same as the literature, but a different path was selected [see Figure 38b]. The Java computer animation chose these selected nodes (Node 1→3→6→8) as the shortest path. Though both shortest path times were the same value, the java computer animation chose a shortest path with 1 less node visited.



$$[A] = \begin{array}{c|cccccc} & 1 & 2 & 3 & 4 & 5 & 6 \\ \hline 1 & & 3 & 5 & & & \\ 2 & & & 1 & 2 & 1 & \\ 3 & & & & & 2 & \\ 4 & & & & & 3 & 4 \\ 5 & & & & & & 3 \\ 6 & & & & & & \end{array}$$

Fig. 43. Dijkstra 6x6 Matrix and Network Graph Example.

Figure 43 shows another example input data for the 6x6 Matrix and Network Graph to be solved for the shortest path using the Forward Dijkstra algorithm. This is another example taken from another source in the literature to be compared with the results of the Java computer animation. The literature solves this shortest path problem with Node 1 as the given source node and Node 6 as the target node.

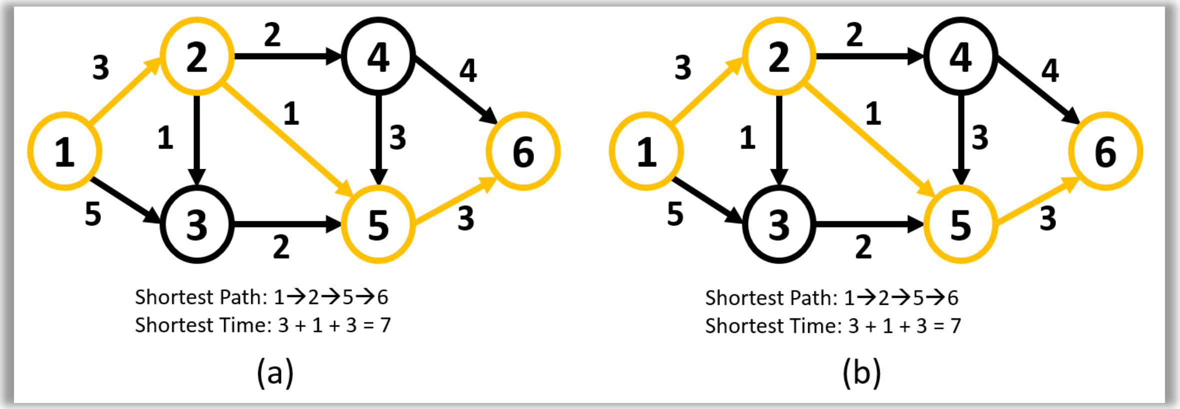


Fig. 44. Dijkstra 6x6 Matrix and Network Graph Results.

The shortest path results from the literature showed that the shortest time was 7 with these nodes selected (Node 1→2→5→6) as the shortest path [see Figure 44a]. The same input data was used with the Java computer animation, and the final results were the same as the literature [see Figure 44b].

4.4 Conclusions

In this chapter, the basic Dijkstra algorithm has been firstly summarized. Then, the Java computer animated software tool that have been developed to enhance the student’s and teacher’s learning has been discussed. The developed Java computer animated software was compared with some example network graphs taken from the literature. The Java computer animated software provided the same results as the literature review and provided precise and clear step-by-step instructions for solving for the shortest path Dijkstra algorithm. The developed Java animated software has all the following desirable features/capabilities:

1. The developed software tool is user friendly (easy to use).
2. Graphical/colorful animation is used extensively to display equations, and/or intermediate/final output results.
3. It provides a clear/attractive computer animated instructor’s voice that is incorporated in the software tool.
4. The instructor’s animated voice for teaching the materials can be spoken in different major languages (English, Chinese, and Spanish).
5. User’s input data can be provided dynamically in either interactive mode, edited input data file mode, or by graphical mode.
6. It includes an option for providing partial (or intermediate) results, and complete (final results) are available for the user to select.
7. It includes an option for displaying all detailed intermediate results in the first 1-2 iterations and/or directly showing the final answers.

8. Users/learners can provide their own data dynamically and compare their hand-calculated results against the computer software results (in each step of the algorithm) for enhancing/improving their learning of the Dijkstra algorithm.

CHAPTER 5

CHOLESKY DECOMPOSITION ALGORITHM

The Cholesky Decomposition (Factorization) Algorithm was presented by Andre Louis Cholesky in an unknown and unpublished 8-page manuscript on December 2, 1910 entitled "On the numerical solutions of systems of linear equations". The method was unknown outside the French circle of topographers until another French officer published a paper explaining the method in 1924. It wasn't until 1950 when the Cholesky Decomposition Method was widely known after John Todd's lectures that several colleagues and students undertook study of the Cholesky method [49]. Today, the Cholesky Decomposition Method is widely known and is used to solve systems of Symmetric Positive Definite (SPD) simultaneous linear equations (SLEs). Cholesky algorithms have very broad engineering and scientific applications, including transportation engineering applications (such as driving with errors/uncertainties, etc.) [50].

The best way to understand the Cholesky method is to see it working in practice by performing and solving small examples by hand and working through the algebra [51]. Various teaching philosophies have been proposed, tested and documented by educational research communities, such as video lectures (YouTube), "flipped" class lectures (where students are encouraged to read the lecture materials on their own time at home and problem solving and/or question/answer sessions conducted in the usual classroom environments), US Science Technology Engineering Mathematics (STEM) summer camps, game-based learning (GBL) [3-5], virtual laboratories [6] and concept inventory [7].

The final product from this work provides experimental results that shows that this developed software/tool helps both the students and their instructor to not only master this technical subject, but also provide a valuable tool for obtaining the solutions for homework assignments, class examinations, self-assessment tools, etc. The developed “educational version” of a Java visualization-animation software tool provides several desirable features, such as:

- A detailed, precise and clear step-by-step algorithm will be displayed in text and human voice during the animation of the algorithm.
- Options to hear animated voice in several major languages (English, Chinese and Spanish).
- Options to input/output terminal container yard layouts (via a CVS file), manually edit the layouts using an editor, or “randomly generate” layouts.
- Output of the “final” results can be exported to text so that the users/learners can check/verify their “hand-calculated” results, which is an important part of the learning process.

In this chapter, a simple Cholesky decomposition example is thoroughly explained along with the derived formulas in Section 5.1. The focus is then shifted toward a Java visualization-animation software tool that provides step-by-step instructions for learning/teaching the Cholesky decomposition method in Section 5.2. This Java visualization-animation software tool was used in an Old Dominion University (ODU) 300-Level Civil and Environmental Engineering (CEE) Computation course (CEE 305) every Fall semester. Lecture materials on how to solve a system of SPD SLEs along with the Java-based software was handed out and briefly discussed with students as a self-study take-home assignment. Students were given an in-class exam on the Cholesky method. The goal was to determine if the Java-based (self-learning) software would improve or worsen the student’s exam performance compared to the traditional face-to-face instructor’s classroom lecture. The experimental results from this case-study are captured and explained in detail in Section 5.3. Conclusions are summarized/suggested in Section 5.4.

5.1 Summary of the Cholesky Decomposition Algorithm

Solving a large (and sparse) system of simultaneous linear equations (SLE) has been (and continues to be) a challenging problem for many real-world engineering/science applications [52-53]. In matrix notation, the SLE can be represented as:

$$[A]\{x\} = \{b\} \tag{1}$$

where

[A] = known coefficient matrix, with dimension NxN
{b} = known right-hand-side (RHS) Nx1 vector
{x} = unknown Nx1 vector.

Symmetrical Positive Definite (SPD) SLE

For many practical SLE, the coefficient matrix [A] (see Eq.1) is SPD. In this case, efficient 3-step Cholesky algorithms can be used.

Step 1: Matrix Factorization Phase

In this step, the coefficient matrix [A] can be decomposed into

$$[A] = [U]^T [U] \tag{2}$$

where [U] is a NxN upper triangular matrix. The following simple example will illustrate how to find the matrix [U]. Various terms of the factorized matrix [U] can be computed/derived as follows (see Eq. 2):

$$\begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{bmatrix} = \begin{bmatrix} u_{11} & 0 & 0 \\ u_{12} & u_{22} & 0 \\ u_{13} & u_{23} & u_{33} \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{bmatrix}. \tag{3}$$

Multiplying 2 matrices on the right-hand-side (RHS) of Eq. (3), then equating each upper-triangular RHS term to the corresponding ones on the upper-triangular left-hand-side (LHS), one gets the following 6 equations for the 6 unknowns in the factorized matrix $[U]$.

$$u_{11} = \sqrt{A_{11}}\, ; u_{12} = \frac{A_{12}}{u_{11}}\, ; u_{13} = \frac{A_{13}}{u_{11}} \tag{4}$$

$$u_{22} = \left(A_{22} - u_{12}^2\right)^{\frac{1}{2}}; u_{23} = \frac{A_{23} - u_{12}u_{13}}{u_{22}}\ ;\ u_{33} = \left(A_{33} - u_{13}^2 - u_{23}^2\right)^{\frac{1}{2}} \tag{5}$$

In general, for a NxN matrix, the diagonal and off-diagonal terms of the factorized matrix $[U]$ can be computed from the following formulas:

$$u_{ii} = \left(A_{ii} - \sum_{k=1}^{i-1} (u_{ki})^2 \right)^{\frac{1}{2}} \tag{6}$$

$$u_{ij} = \frac{A_{ij} - \sum_{k=1}^{i-1} u_{ki}u_{kj}}{u_{ii}}\,. \tag{7}$$

As a quick example, one computes:

$$u_{57} = \frac{A_{57} - u_{15}u_{17} - u_{25}u_{27} - u_{35}u_{37} - u_{45}u_{47}}{u_{55}}\,. \tag{8}$$

Thus, for computing $u(i = 5, j = 7)$, one only needs to use the (already factorized) data in columns # i(=5), and # j(=7) of [U], respectively.

Step 2: Forward Solution phase

Substituting Eq. (2) into Eq. (1), one gets:

$$[U]^T [U]\{x\} = \{b\}\,. \tag{9}$$

Let's define:

$$[U]\{x\} \equiv \{y\}\,. \tag{10}$$

Then, Eq. (9) becomes:

$$[U]^T\ \{y\} = \{b\}\,. \tag{11}$$

Since $[U]^T$ is a lower triangular matrix, Eq. (11) can be efficiently solved for the intermediate unknown vector $\{y\}$, according to the order $\begin{Bmatrix} y_1 \\ y_2 \\ . \\ . \\ y_N \end{Bmatrix}$, hence the name "forward solution".

As a quick example, one has:

$$\begin{bmatrix} u_{11} & 0 & 0 \\ u_{12} & u_{22} & 0 \\ u_{13} & u_{23} & u_{33} \end{bmatrix} \begin{Bmatrix} y_1 \\ y_2 \\ y_3 \end{Bmatrix} = \begin{Bmatrix} b_1 \\ b_2 \\ b_3 \end{Bmatrix} \tag{12}$$

$$u_{11} y_1 = b_1 \rightarrow y_1 = \frac{b_1}{u_{11}} \tag{13}$$

$$u_{12} y_1 + u_{22} y_2 = b_2 \rightarrow y_2 = b_2 - u_{12} y_1 / u_{22} . \tag{14}$$

Similarly,

$$y_3 = \frac{b_3 - u_{13} y_1 - u_{23} y_2}{u_{33}} . \tag{15}$$

In general, one has

$$y_j = \frac{b_j - \sum_{i=1}^{j-1} u_{ij} y_i}{u_{jj}} . \tag{16}$$

Step 3: Backward Solution phase

Since $[U]$ is an upper triangular matrix, Eq. (10) can be efficiently solved for the original unknown vector $\{x\}$, according to the order $\begin{Bmatrix} x_N \\ x_{N-1} \\ x_{N-2} \\ . \\ x_1 \end{Bmatrix}$, hence the name “backward solution”.

As a quick example, one has:

$$\begin{bmatrix} u_{11} & u_{12} & u_{13} & u_{14} \\ 0 & u_{22} & u_{23} & u_{24} \\ 0 & 0 & u_{33} & u_{34} \\ 0 & 0 & 0 & u_{44} \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{Bmatrix} = \begin{Bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{Bmatrix} \tag{17}$$

$u_{44} x_4 = y_4$, hence $x_4 = \dfrac{y_4}{u_{44}}$ (18)

$u_{33} x_3 + u_{34} x_4 = y_3$, hence $x_3 = \dfrac{y_3 - u_{34} x_4}{u_{33}}$. (19)

Similarly: $x_2 = \dfrac{y_2 - u_{23} x_3 - u_{24} x_4}{u_{22}}$ (20)

$$x_1 = \frac{y_1 - u_{12}x_2 - u_{13}x_3 - u_{14}x_4}{u_{11}}. \tag{21}$$

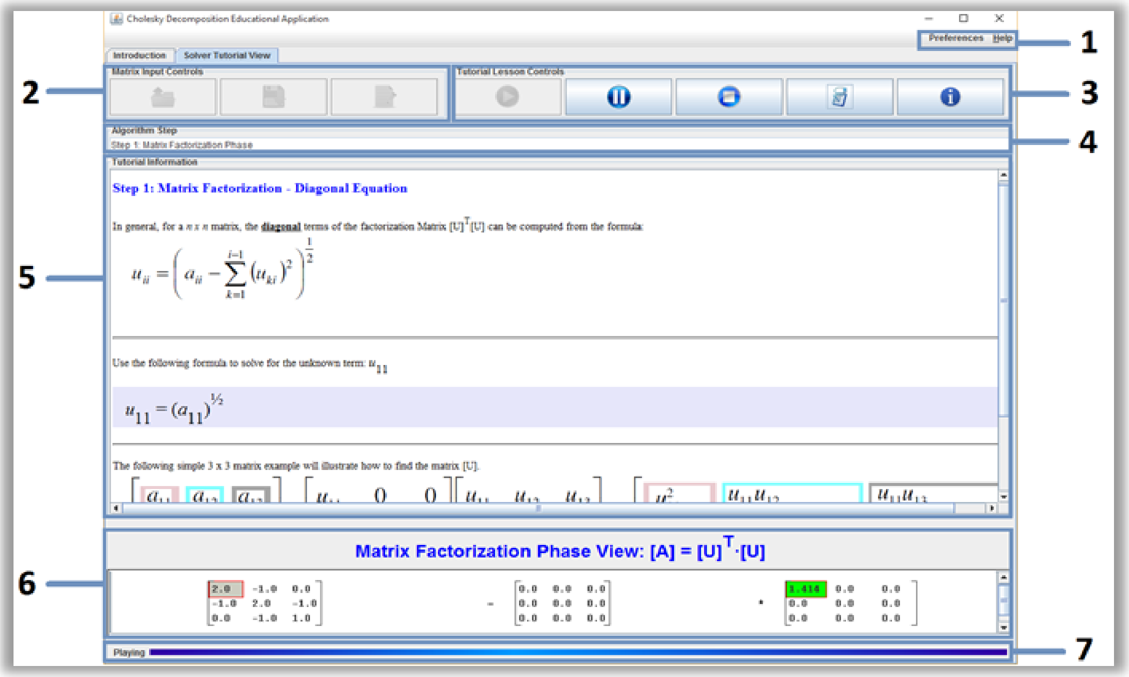
In general, one has:

$$x_j = \frac{y_j - \sum_{i=j+1}^{N} u_{ji}x_i}{u_{jj}}. \tag{22}$$

Note, amongst the above 3-step Cholesky algorithms, the factorization phase in step 1 consumes about 95% of the total SLE solution time.

5.2 Visualization and Animation using Java

The developed software is written in Java [19-21] and is meant to create 2D animations and provide voice step-by-step explanations for the Cholesky Algorithm. The software is developed from scratch using several open-source software options. The software uses various third-party libraries, such as Java Swing library for the animations, the text-to-speech library (Google API Translate) for voice, and the matrix library (Efficient Java Matrix) for data storage.



Step 1: Matrix Factorization - Diagonal Equation

In general, for a $n \times n$ matrix, the **diagonal** terms of the factorization Matrix $[U]^T[U]$ can be computed from the formula:

$$u_{ii} = \left(a_{ii} - \sum_{k=1}^{i-1}(u_{ki})^2\right)^{\frac{1}{2}}$$

Use the following formula to solve for the unknown term: u_{11}

$$u_{11} = (a_{11})^{½}$$

The following simple 3 x 3 matrix example will illustrate how to find the matrix [U].

Matrix Factorization Phase View: $[A] = [U]^T \cdot [U]$

$$\begin{bmatrix} 2.0 & -1.0 & 0.0 \\ -1.0 & 2.0 & -1.0 \\ 0.0 & -1.0 & 1.0 \end{bmatrix} = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \cdot \begin{bmatrix} 1.414 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix}$$

Fig. 45. Cholesky Decomposition Main Window.

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 seven main components as shown in Figure 45.

1. Menu Buttons (Preferences and Help)
   Provide basic options, such as an option to open/save Container Layouts, 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. Matrix Input Control Buttons
   Input control buttons are provided from the main window to open, save or edit the left-hand-side (LHS) Matrix [A] and right-hand-side (RHS) Vector {b} data.

3. Tutorial Lesson Control Buttons
   Lesson control buttons are provided from the main window to play, pause, or stop a lesson (an algorithm). The buttons are enabled when the LHS Matrix [A] and RHS Vector {b} are given. Additional buttons (the step-by-step and information button) are provided. The step-by-step button launches a GUI that provides step-by-step instructions taking place during the play session. These step-by-step instructions can be saved to an output text file. The information buttons provide a user guide for the Main GUI.

4. Algorithm Step
   Textbox view of the current algorithm step during play session.

5. Tutorial Information
   Textbox view of instructions, equations, or steps (scrollable) being done within the algorithm. The equations are dynamically provided based on the input data given.

6. Animation View
   Animation view of the matrices and vectors being updated during the play session.

7. Status View
   Provides a status view which displays meaningful status: ready, play, or paused.

5.3 Numerical Examples/Results

A Civil and Environmental Engineering (CEE) Computation course (CEE 305) has been offered every fall semester. In the fall 2013 semester, the Cholesky method was explained in the traditional (face-to-face) classroom instruction format. However, in the fall 2014 semester, lecture materials on how to solve a system of SPD SLEs along with the Java-based software was handed out to the students (with minimal lecture time) as a "self-study" take-home assignment. In both semesters students were given an in-class exam on the Cholesky method.

The goal was to determine whether the Java-based software would improve or worsen the students' exam performance as compared to the traditional (face-to-face) classroom lecture. The resulting data was collected by the same professor that taught the same course using the normal (face-to-face) classroom instruction and using this new Java software tool. The results are explained below [see Figure 46]. Figure 46a shows the Student Percentage (Exam) Scores for the Fall 2013 semester using the conventional (face-to-face) classroom lectures. Figure 47b shows the Student Percentage (Exam) Scores for the Fall 2014 semester using the Java visualization-animation software tool for teaching the Cholesky Algorithm. Figure 47b clearly shows that a majority of students scored within the 76.67% or above; whereas, in Figure 46a, student scores were scattered with more students on/below the 53.33% mark than in Fall 2014.

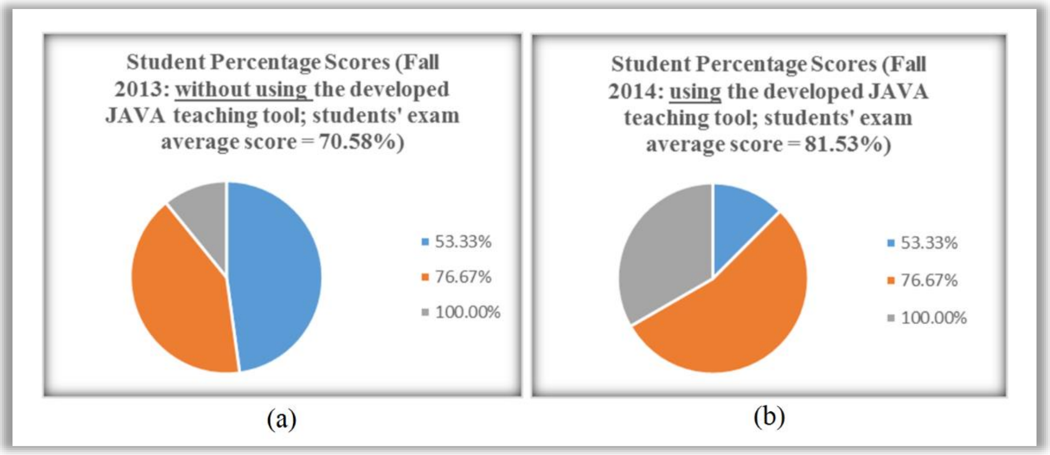


Fig. 46. Student Percentage Scores Fall 2013 (a) and Fall 2014 (b).

Using the Java visualization-animation software tool for instructions and animation in learning the Cholesky Method for solving SLE, and with the user's input problems' data capability for practicing exercises, the student's average performance/exam scores have improved from 70.58% to 81.53% (as shown in Figures 46a and 46b).

5.4 Conclusions

In this paper, the Cholesky Method is once again revisited to show a new and different approach for teaching/learning the Cholesky Decomposition Algorithm. Cholesky algorithms have very broad engineering and science applications, including transportation engineering applications (such as driving with errors/uncertainties, etc.) [50]. The developed Java computer animated software has shown that it can be used as an effective tool for teaching and learning the Cholesky method. The students' average scores from in-class exams in the fall 2013 (traditional face-to-face class lecture instruction mode, without using the developed Java animated teaching tool), and in the fall 2014 (students' self-learning mode [with minimum instructor's lecture time], using the developed Java animated teaching tool) were 70.58 % and 81.53 %, respectively. Thus, an improvement of more than 10% has been observed. The software tool is developed in Java and integrated with open-source libraries. It provides a means to be platform independent and can be run on several operating systems. It provides desirable teaching/learning features for the Cholesky algorithm, such as:

- Software tool with a user friendly GUI.
- 2D Graphical visualization-animation for displaying data update.
- 2D Graphical visualization to edit input data.
- Ability to allow the user/learner to open/save the data for later use.
- Ability to allow the user/learner to output/save the step-by-step results.
- Provide a clear and attractive computer animated voice that provides step-by-step instructions of the algorithms.

- Animated voice can be configurable to translate text-to-speech into another language, such as English, Spanish and Chinese.

- Provides the final and intermediate results of the Cholesky solution process.

CHAPTER 6

CONCLUSIONS AND RECOMMENDATIONS

When it comes to teaching STEM topics, visualization-animation seems like a natural fit in improving both the quality and number of highly trained US educators, student's workforce in STEM topics, in today's highly competitive global markets. Visualization-animation has become very attractive and increasingly popular in teaching biology courses [16-17, 54]. Current research has shown that lecture videos strengthen the student's learning, increase their understanding, and keep them more active and engaged in the course topic and material [55]. Visualization-animation software applications hold powerful step-by-step instructional potential in teaching and have proven to be a very effective, useful, and helpful tool for students to understand, practice, and learn course materials in STEM topics, such as in Mathematics [56] and Computer Science [57-58] courses.

With flipped (inverted) classrooms being used as a means of teaching over the past several years, students are first exposed to new course materials outside of the classroom either by reading notes or viewing lecture materials. Understanding a complex topic (or a transportation algorithm) can be challenging and frustrating for a student to learn outside of the classroom using static materials which in turn can lead to the student's lack of participation and interaction in classroom learning [59].   Visualization-animation is a complete STEM tool that can be used to animate a topic and provide step-by-step instructions to help students understand, visualize, and interpret the course contents. With the use of visualization-animations software applications, transportation algorithms can be taught and learned in the same manner where students can access the lesson materials off-campus at their own convenience before classroom lectures.

6.1 Potential Benefits of Visualization-Animation

With the developed visualization-animation Java based computer animation for teaching/learning transportation algorithms (Unloading/Pre-Marshalling, Hungarian, Dijkstra, and Cholesky), the following potential benefits can be expected for the students/learners:

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 transportation 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 algorithm's 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 instructors:

1. Having the absolute freedom to design new homework assignments (or new exam questions) for each year without wasting time preparing the homework', and/or exams' solutions.

2. Since the students can learn the transportation algorithms 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 a "flipped classroom teaching mode", rather than the "traditional classroom teaching mode", where the instructor has to spend significant time "face-to-face" lecturing students.

6.2 Recommendations

With rapid technology innovations of the digital age today, students are equipped with the latest technology and are using smart cellular phones, tablets and laptops as their means of a learning tool. Instead of developing stand-alone application-based visualization-animation software for teaching and learning, development should be implemented within a web-based architecture. Having the visualization-animation software as a web-based complete STEM tool, students and learners can easily access the materials with a simple internet browser as long as internet connectivity is available. Thus, students and learners will not need to install the stand-alone application-based visualization-animation software directly on to their laptops/workstations.

Another recommendation would be to provide additional improvements with flipped classroom preparations of the course materials (reading, notes, and videos). These materials need to be carefully tailored for a productive classroom active learning session since students new to the materials can tend to be initially resistant [14]. Instead of using static lecture videos that cover 1 or 2 example problems, a visualization-animation software tool can be used to input dynamic data into the software. By introducing visualization-animation software within the course materials could potentially help alleviate this initial resistance and help students be more comfortable and engaged with the contents by allowing them to understand, visualize, and interpret the materials more clearly and effective.

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] A. Horno López, J. Pertíñez López (2011) The Animation as a Didactic Teaching Method, EDULEARN11 Proceedings, pp. 1123-1126.

[9] Felder, Richard M. (1988). Learning and Teaching Styles in Engineering Education. Journal of Engineering Education, 78(7), pp. 674-681.

[10] McGlynn, A. P. (2005). Teaching millennials, our newest cultural cohort. Educational Digest, 12-16.

[11] Prince, Michael (2004). Does Active Learning Work? A Review of the Research. Journal of Engineering Education, 93(3), 223-231.

[12] Barak, M., Lipson, A., & Lerman, S. (2006). Wireless Laptops as Means for Promoting Active Learning in Large Lecture Halls. Journal of Research on Technology in Education, 38(3), 245-263.

[13] Arnold-Garza, Sara (2014). The Flipped Classroom Teaching Model and its Use for Information Instruction. Communications in Information Literacy, Volume 8, Issue 1.

[14] Herreid, Clyde F. & Schiller, Nancy A. (2013). Case Studies and the Flipped Classroom. National Science Teachers Association (NSTA). Journal of College Science Teaching, Vol. 42, No. 5.

[15] Ewart, Franzeska G. (1998). Let the Shadows Speak: Developing Children's Language Through Shadow Puppetry. Trentham Books.

[16] Windschitl, Mark (1996). Instructional Animations: The In-House Production of Biology Software. Journal of Computing in Higher Education, Volume 7, Number 2, Page 78.

[17] Zanin, Mary K. B. (2016). Creating & Teaching with Simple Animations: Making Biology Instruction Come Alive. National Association of Biology Teachers, Volume 78, Issue 5.

[18] Islam, Md. Baharul, Ahmed, Arif, Islam, Md. Kabirul, & Kalam, Abu (2014). Child Education Through Animation: An Experimental Study. International Journal of Computer Graphics & Animation (IJCGA), Vol. 4, No. 4.

[19] Java Platform, Standard Edition (Java SE). http://www.oracle.com/technetwork/java/javase/overview/index.html.

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

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

[22] G Java 2D Generic Graphics Library. http://geosoft.no/graphics/.

[23] Kap Hwan Kim, Hans-Otto Gunther, "Container Terminals and Cargo Systems: Design, Operations Management, and Logistics Control Issues".

[24] Yusin Lee and Nai-Yun Hsu, 'An optimization model for the container pre-marshalling problem', Computers & Operations Research.

[25] John Slaney, Sylvie Thiebaux, "Blocks World revisited". Artificial Intelligence (14 June 2000).

[26] Nguyen, D.T., Vi Nguyen, Nga Pham, Gelareh Bakhtyar, "Yard Crane Scheduling (YCS), Unloading and Reshuffling Formulations/Algorithms", CEE-775/875:

Computational Transportation Course, Spring'2015 semester (January 14 2015). Demo: http://www.lions.odu.edu/~imako001/.

[25] JBoss Rules (Drools) and the Rete Algorithm. https://docs.jboss.org/drools/release/5.2.0.Final/drools-expert-docs/html/ch03.html.

[26] P. Sriphrabu, K. Sethanan, and B. Amonkijpanich, "A Solution of the Container Stacking Problem by Genetic Algorithm", IACSIT International Journal of Engineering and Technology, Vol. 5, No. 1 (Feb. 2013).

[27] K. Tierney, and Y. Malitsky, "An Algorithm Selection Benchmark of the Container Pre-marshalling Problem".

[28] Tierney, K., Pacino, D., Voß, S.: Solving the pre-marshalling problem to optimality with A* and IDA*. Technical report WP#1401, DS&OR Lab, University of Paderborn (2014).

[29] Lehnfeld, J., Knust, S. "Loading, Unloading and Pre-marshalling of Stacks in Storage Areas: Survey and Classification", Eur. J. Oper. Res. 239(2), 297–312 (2014).

[30] M. van Brink, and R. van der Zwaan, "A Branch and Price Procedure for the Container Pre-marshalling Problem".

[31] S. Huang, and T. Lin, "Heuristic Algorithms for Container Pre-Marshalling Problems", Computers & Industrial Engineering 62, pages 13-20 (2012).

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

[33] 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/.

[34] Kuhn, Harold W. & Yaw, Bryn (1955). The Hungarian Method for the Assignment Problem. Naval Research Logistics Quarterly Vol. 2, pp. 83-97.

[35] Balinski, M. L. & Gomory, R. E. (1964). A Primal Method for the Assignment and Transportation Problems. Management Science, Vol. 10, Issue 3, pp. 578-593.

[36] Ahmad, Hlayel Abdallah (2012). The Best Candidates Method for Solving Optimization Problems. Journal of Computer Science. Volume 8, Issue 5, pp. 711-715.

[37] Liu, Lantao & Shell, Dylan A. (2011). Assessing Optimal Assignment Under Uncertainty: An Interval-Based Algorithm. International Journal of Robotics Research, Vol. 30, No. 7, pp. 936-953.

[38] Sheffi, Y., 1985. Urban Transportation Networks: Equilibrium Analysis with Mathematical Programming Methods. Available free of charge at: http://web.mit.edu/sheffi/www/urbanTransportation.html

[39] Lawson, G., Allen, S., Rose, G., Nguyen, D.T., Ng, M.W. "Parallel Label Correcting Algorithms for Large-Scale Static and Dynamic Transportation Networks on Laptop Personal Computers", Transportation Research Board (TRB) 2013 Annual Meeting (Washington, D.C.; Jan. 13-17, 2013); Session 844 Presentation # 13-2103 (Thursday, Jan. 17-2013; 10:15am-noon); Poster Presentation # P13-6655.

[40] Paul Johnson III, Duc T. Nguyen, and Manwo Ng, "An Efficient Shortest Distance Decomposition Algorithm for Large-Scale Transportation Network Problems", TRB 2014 Annual Meeting (Washington, D.C.; January 2014); Oral, and Poster Presentations.

[41] Dijkstra's Shortest Path Algorithm. http://www.cs.uah.edu/~rcoleman/CS221/Graphs/ShortestPath.html.

[42] Dijkstra Algorithm. http://students.ceid.upatras.gr/~papagel/project/kef5_7_1.htm.

[43] Dijkstra's Algorithm. http://www3.cs.stonybrook.edu/~skiena/combinatorica/animations/dijkstra.html.

[44] Cluster (Parallel) Computing for Large-Scale Engineering & Science Applications, http://www.lions.odu.edu/~skadi002/001-075.pdf. Demo: http://www.lions.odu.edu/~imako001/.

[45] Shu-Xi, Wang (2012). The Improved Dijkstra's Shortest Path Algorithm and its Applications. International Workshop on Information and Electronics Engineering (IWIEE), Volume 29, 2012, pp. 1186-1190.

[46] Rohila, K., Gouthami, P. & Priya M (2014). Dijkstra's Shortest Path Algorithm for Road Network. International Journal of Innovative Research in Computer and Communication Engineering, Vol. 2, Issue 10, October 2014.

[47] Kadry, Seifefine, Abdallah, Ayman & Joumaa, Chilbli (2011). On the Optimization of Dijkstra's Algorithm. Informatics in Control, Automation and Robotics, Volume 133, pp. 393-297.

[48] Singal, Pooja & Chhillar, R. S. (2014). Dijkstra Shortest Path Algorithm using Global Positioning System. International Journal of Computer Applications, Volume 101, No. 6, September 2014.

[49] Brezinski, C. The life and work of Andre Cholesky.

[50] ManWo, Ng and Sathasivan, K. (2014). Probabilistic Modeling of Erroneous Human Response to In-Vehicle Route Guidance Systems: A First Look. Journal of Intelligent

Transportation Systems: Technology, Planning, and Operations, Volume 18, Issue 2, pp. 131-137.

[51] Claxton, Karl and Briggs, Andrew "Decision Modelling for Health Economic Evaluation" (2006).

[52] Nguyen, D.T. (2006). Finite Element Methods: Parallel-Sparse Statics and Eigen-Solutions, Springer Publishers.

[53] Nguyen, D.T. (2002). Parallel-vector Equation Solvers for Finite Element Engineering Applications, Kluwer Academic/Plenum Publishers.

[54] Stith, Bradley J. (2004). Use of Animation in Teaching Cell Biology. Cell Biology Education, A Journal of Life Science Education, Vol. 3, pp. 181-188.

[55] Bell, Randy L., Gess-Newsome, Julie, and Luft Julie (2008). Technology in the Secondary Science Classroom. National Science Teachers Associations (NSTA).

[56] Salim, Kalbin & Tiawa, Dayang Hjh (2015). The Student's Perceptions of Learning Mathematics using Flash Animation Secondary School in Indonesia. Journal of Education and Practice, Vol. 6, No. 34.

[57] Mtebe, Joel S. & Twaakyondo, Hashim M. (2012). Are Animations Effective Tools for Teaching Computer Science Courses in Developing Countries? International Journal of Digital Information and Wireless Communications (IJDIWC), Vol. 2, No. 2, pp. 202-207.

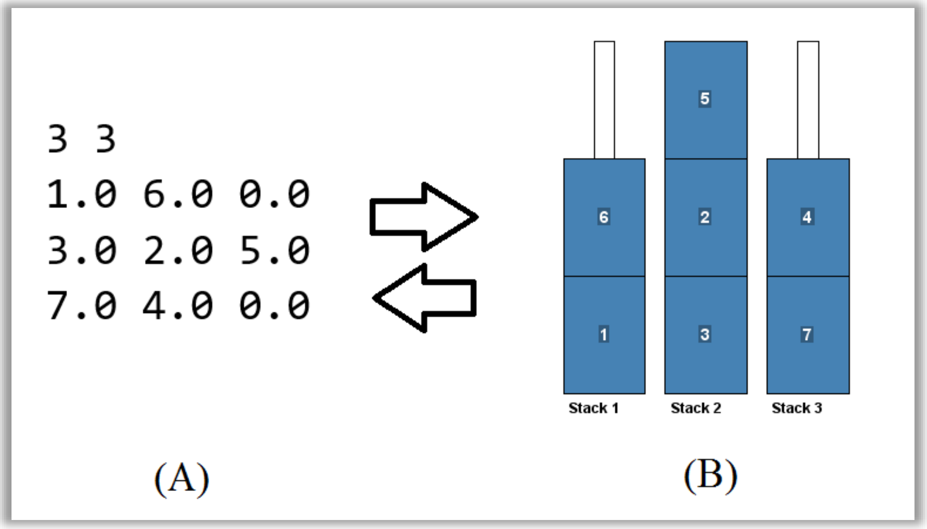
[58] Lin, Chi-Chen & Zhang, Mingrui. The Use of Computer Animation in Teaching Discrete Structure Course. Computer Science Department, Winona State University, 2000.

[59] Nawi, Naafi'ah, Jawawi, Rosmawijah, Matzin, Rohani, Jaidin, Jainatul Halida, Shahrill, Masitah, & Mundia, Lawrence (2015). To Flip or Not to Flip: The Challenges and Benefits of Using Flipped Classroom in Geography Lessons in Brunei Darussalam. Review of European Studies, Vol. 7, No. 12.

APPENDICES

APPENDIX A: UNLOADING AND PRE-MARSHALLING INPUT/OUTPUT

A Column Space Value (CSV) reader and writer is used as the Input/Output mechanism for reading/writing data to/from the visualization-animation Java software that teaches the Unloading and Pre-Marshalling Algorithm.



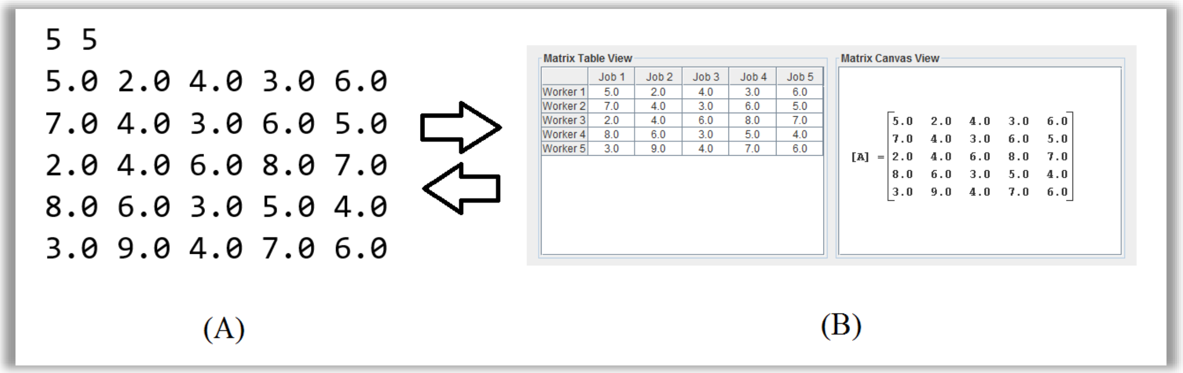
```
3 3
1.0 6.0 0.0
3.0 2.0 5.0
7.0 4.0 0.0
```

Fig. 47. Unloading & Pre-Marshalling Input & Output Example (A) CSV File; (B) Java Software.

The first line in the CSV file represents the matrix size (row and column) where the row is the number of Stacks and the column is the height of the Stack [see Figure 47]. The remainder of the CSV file contains the values of each element in the matrix which represents the priority order and number for that Cargo Container. For Example, Row 1, Column 2 in the matrix input has the value 6.0. This value represents Cargo Container 6 and will be the 6th Cargo Container to be unloaded using either the Unloading or Pre-Marshalling algorithm.

APPENDIX B: HUNGARIAN INPUT/OUTPUT

A Column Space Value (CSV) reader and writer is used as the Input & Output mechanism for reading/writing data to/from the visualization-animation Java software that teaches the Hungarian Algorithm.



```
5 5
5.0 2.0 4.0 3.0 6.0
7.0 4.0 3.0 6.0 5.0
2.0 4.0 6.0 8.0 7.0
8.0 6.0 3.0 5.0 4.0
3.0 9.0 4.0 7.0 6.0
```

	Job 1	Job 2	Job 3	Job 4	Job 5
Worker 1	5.0	2.0	4.0	3.0	6.0
Worker 2	7.0	4.0	3.0	6.0	5.0
Worker 3	2.0	4.0	6.0	8.0	7.0
Worker 4	8.0	6.0	3.0	5.0	4.0
Worker 5	3.0	9.0	4.0	7.0	6.0

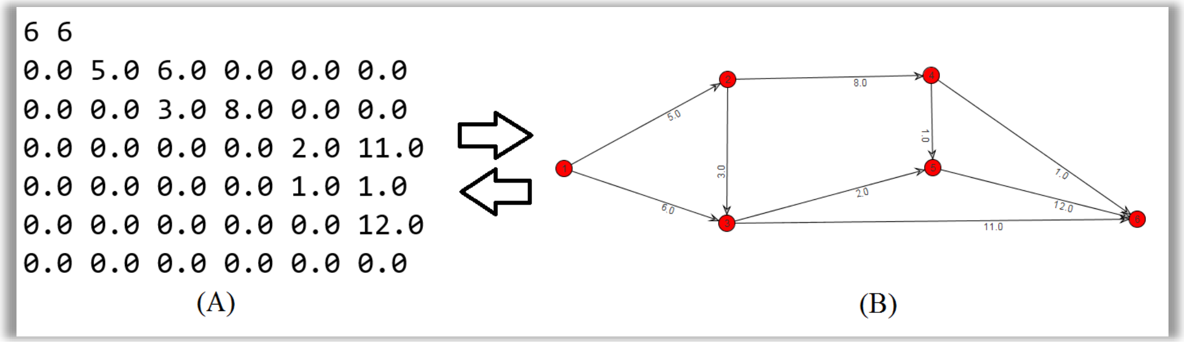
$$[A] = \begin{bmatrix} 5.0 & 2.0 & 4.0 & 3.0 & 6.0 \\ 7.0 & 4.0 & 3.0 & 6.0 & 5.0 \\ 2.0 & 4.0 & 6.0 & 8.0 & 7.0 \\ 8.0 & 6.0 & 3.0 & 5.0 & 4.0 \\ 3.0 & 9.0 & 4.0 & 7.0 & 6.0 \end{bmatrix}$$

Fig. 48. Hungarian Input & Output Example (A) CSV File; (B) Java Software.

The first line in the CSV file represents the matrix size (row and column) where the row is the number of Workers and the column is the number of Jobs [see Figure 48]. The remainder of the CSV file contains the values (Cost Value) of each element in the matrix. For example, Row 1, Column 2 in the matrix input has the value 2.0. Row 1 represents Worker 1 and Column 2 represents Job 2 with a Cost Value of 2.0 within the Java software.

APPENDIX C: DIJKSTRA INPUT/OUTPUT

A Column Space Value (CSV) reader and writer is used as the Input/Output mechanism for reading/writing data to/from the visualization-animation Java software that teaches the Forward Dijkstra Algorithm.



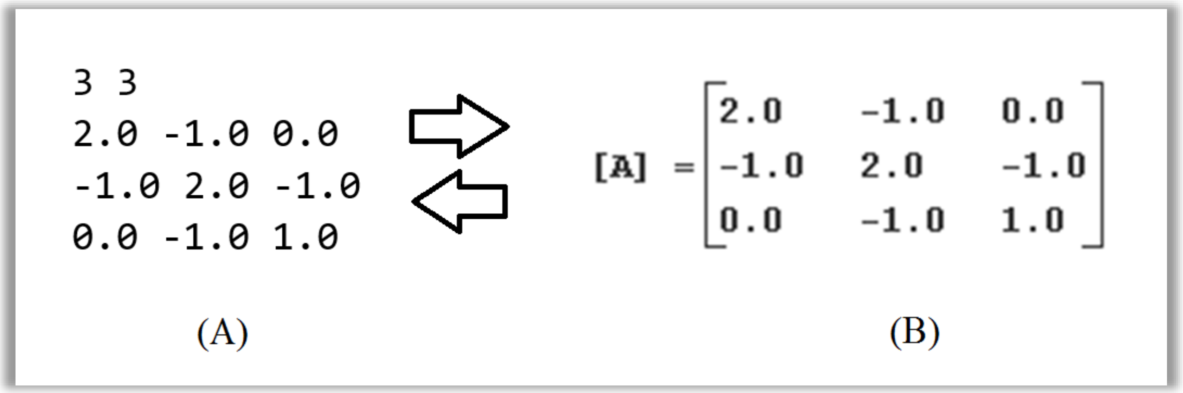
```
6 6
0.0 5.0 6.0 0.0 0.0 0.0
0.0 0.0 3.0 8.0 0.0 0.0
0.0 0.0 0.0 0.0 2.0 11.0
0.0 0.0 0.0 0.0 1.0 1.0
0.0 0.0 0.0 0.0 0.0 12.0
0.0 0.0 0.0 0.0 0.0 0.0
```

Fig. 49. Dijkstra Input & Output Example (A) CSV File; (B) Java Software.

The first line in the CSV file represents the matrix size (row and column) where the row/column is the number of Nodes in the Network [see Figure 49]. The remainder of the CSV file contains the values of each element in the matrix or the Edge Weight. For example, Row 1, Column 2 in the matrix input has the value 5.0. The Row and Columns represent Nodes and the value represents the Edge weight within the Java software. In this example, Row 1 is Node 1 and Column 2 is Node 2 with an Edge weight value of 5.0 between the two Nodes.

APPENDIX D: CHOLESKY INPUT/OUTPUT

A Column Space Value (CSV) reader and writer is used as the Input/Output mechanism for reading/writing data to/from the visualization-animation Java software that teaches the Cholesky Decomposition Algorithm.



```
3 3
2.0 -1.0 0.0
-1.0 2.0 -1.0
0.0 -1.0 1.0
```

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

Fig. 50. Cholesky Input & Output Example (A) CSV File; (B) Java Software.

The first line in the CSV file represents the matrix size (row and column) [see Figure 50]. The remainder of the CSV file contains the values of each element in the known coefficient matrix [A] within the Java software.

VITA

Ivan Makohon is a Scientist at Combat Directions Systems Activity (CDSA), Dam Neck where he’s a Senior Software Engineer/System Administrator for various Department of Defense (DoD) projects. Mr. Makohon holds a Bachelor’s of Science in Computer Science (2001) from Christopher Newport University (CNU) with a minor in Business Administration and is a private pilot.

For over 15 years, Mr. Makohon has acquired knowledge and hands-on expertise within Modeling and Simulations (M&S) through past and present software development projects ranging from Distributed Interactive Simulation (DIS), High-Level Architecture (HLA) Simulations, Discrete Event Simulations, and is now currently getting more involved in Model Based Systems Engineering (MBSE) within the DoD. Mr. Makohon is expanding his knowledge and is currently pursuing his Masters of Science in Modeling and Simulations at Old Dominion University (ODU). His career objectives are to pursue a Masters, and then a PhD, and a Technical Director (TD) or Senior Executive Service (SES) position within DoD. His prior software development projects have shown that M&S plays a hugely cost effective and savings role within DoD training. Building realistic models to train sailors within a land-based training environment has shown to reduce cost and train sailors before they depart to sea.