JAVA BASED VISUALIZATION AND ANIMATION FOR TEACHING THE DIJKSTRA SHORTEST PATH ALGORITHM IN TRANSPORTATION NETWORKS

Ivan Makohon[1], Duc T. Nguyen[2], Masha Sosonkina[3], Yuzhong Shen[4] and Manwo Ng[5]

[1]Graduate Student, Department of Modeling, Simulation & Visualization Engineering (MSVE), Old Dominion University (ODU), Norfolk, Virginia

[2]Professor, Civil & Environmental Engineering (CEE) Department, ODU, Norfolk, Virginia

[3]Professor, Department of MSVE,ODU, Norfolk, Virginia

[4]Associate Professor, Department of MSVE, ODU, Norfolk, Virginia

[5]Assistant Professor, Department of Information Technology and Decision, ODU, Norfolk, Virginia

ABSTRACT

Shortest path (SP) algorithms, such as the popular Dijkstra algorithm has been considered as the "basic building blocks" for many advanced transportation network models. Dijkstra algorithm will find the shortest time (ST) and the corresponding SP to travel from a source node to a destination node. Applications of SP algorithms include real-time GPS and the Frank-Wolfe network equilibrium.

For transportation engineering students, the Dijkstra algorithm is not easily understood. This paper discusses the design and development of a software that will help the students to fully understand the key components involved in the Dijkstra SP algorithm. The software presents an intuitive interface for generating transportation network nodes/links, and how the SP can be updated in each iteration. The software provides multiple visual representations of colour mapping and tabular display. The software can be executed in each single step or in continuous run, making it easy for students to understand the Dijkstra algorithm. Voice narratives in different languages (English, Chinese and Spanish) are available.A demo video of the Dijkstra Algorithm's animation and result can be viewed online from any web browser using the website: http://www.lions.odu.edu/~imako001/dijkstra/demo/index.html.

KEYWORDS

Transportation, Dijkstra Algorithm, Java, Visualization, Animation

1. INTRODUCTION

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 [1-3]. Teaching the SP algorithms (such as the Dijkstra algorithm), however, can be a difficult/challenging task!

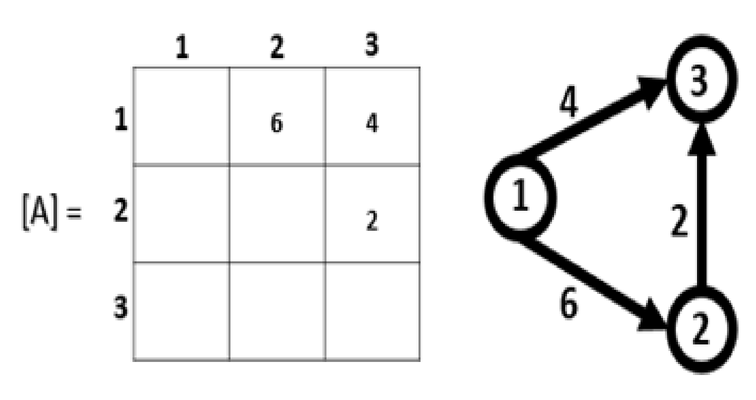
While some teaching information/lecture/tool/animation for Dijkstra algorithms have existed/appeared in the literatures [4-6], none seems to be suitable/appropriate for our students’ learning environments, due to the lack of one (or more) of the following desirable features/capabilities:

1. The developed software tool should be user friendly (easy to use).
2. Graphical/colourful 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 can 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 his/her own data, and compare his/her hand-calculated results with the computer software’s generated results (in each step of the algorithm) for enhancing/improving his/her learning abilities.

The remaining sections of this article is organized as follow. In section II, the basic forward Dijkstra algorithm is summarized (for the readers’ convenience). Java language 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 high-lighted and demonstrated in (including some computer screen captures) Section III. Conclusions are summarized/suggested in Section IV.

2. SUMMARY OF THE BASIC FORWARD DIJKSTRA SHORTEST PATH 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 (Aij) are the link’s value (path, distance, or time value), refer to Figure 1. Figure 1 shows a simple 3 x 3 matrix 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

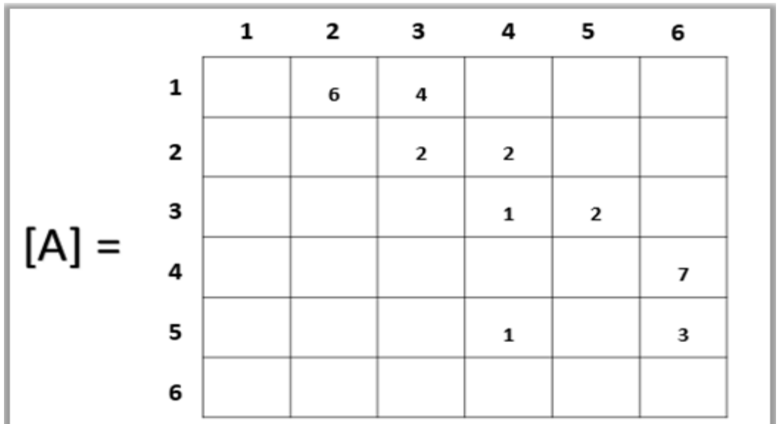


$$[A] = \begin{array}{c|ccc} & 1 & 2 & 3 \\ \hline 1 & & 6 & 4 \\ 2 & & & 2 \\ 3 & & & \end{array}$$

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

Figure 1: Matrix and Graph View of Nodes and Links

With any given N x N 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 (Figure 2) and graph (Figure 3) to find the shortest path, time, or distance from Node 1 (source node) to Node 6 (destination node).



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

Figure 2: 6 x 6 Matrix

We demonstrate and use a simple (easy to use) bookkeeping (Figure 4) method while iterating through the forward Dijkstra algorithm. Figure 4 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.

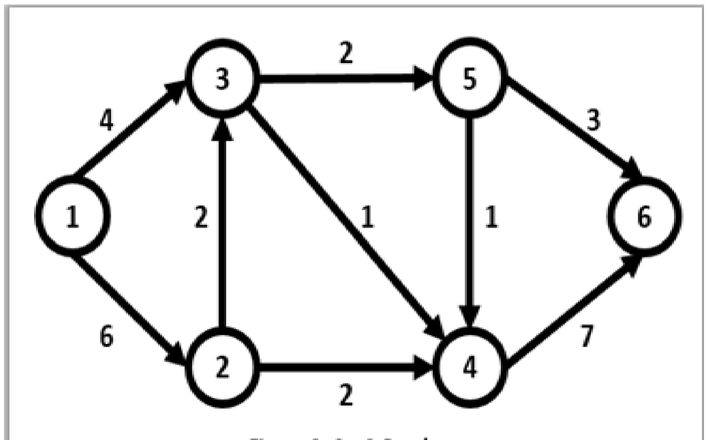
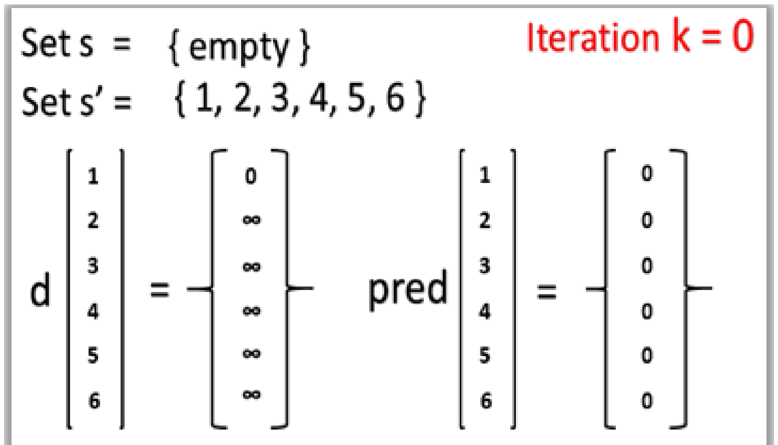


Figure 3: 6 x 6 Graph

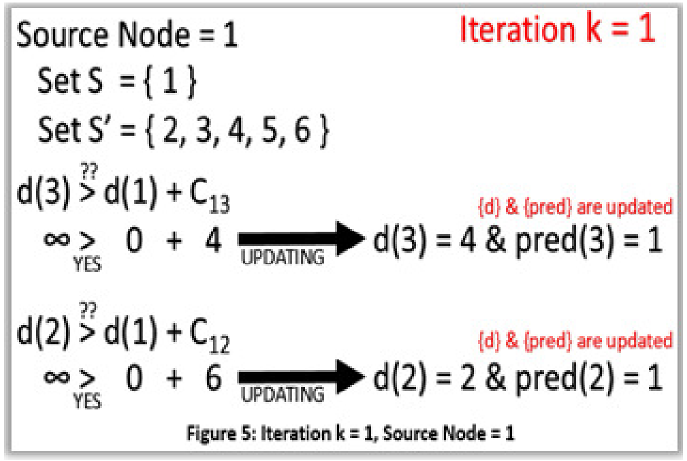
During the first iteration, we set the source node to Node 1 and update Set s = {1} and Set s prime = {2, 3, 4, 5, 6}. Note that Node 1 is removed from Set s prime and added to the Set s.

From Node 1, there are 2 outgoing links, Node 2 and Node 3 (Figure 3). 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 the Vector d with the computed value. If not, then no update is needed and the stored value remains the same.



$$d \begin{bmatrix}1\\2\\3\\4\\5\\6\end{bmatrix} = \begin{bmatrix}0\\\infty\\\infty\\\infty\\\infty\\\infty\end{bmatrix} \quad pred \begin{bmatrix}1\\2\\3\\4\\5\\6\end{bmatrix} = \begin{bmatrix}0\\0\\0\\0\\0\\0\end{bmatrix}$$

Figure 4: Iteration k = 0, Initialization



Source Node = 1 Iteration k = 1

Set S = { 1 }

Set S' = { 2, 3, 4, 5, 6 }

$d(3) \overset{??}{>} d(1) + C_{13}$ {d} & {pred} are updated

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

$d(2) \overset{??}{>} d(1) + C_{12}$ {d} & {pred} are updated

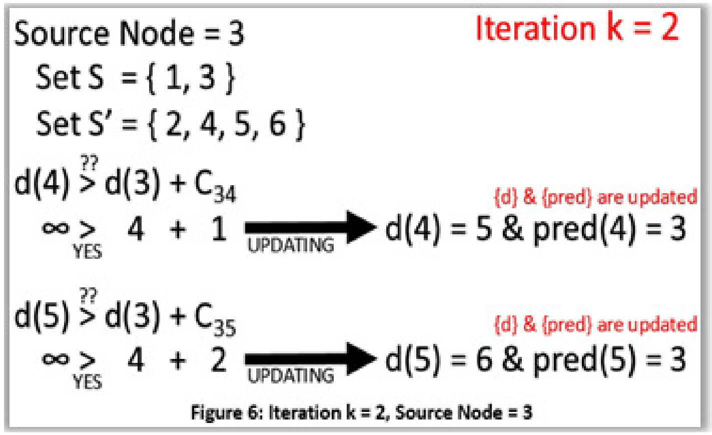
$\infty \underset{YES}{>} 0 + 6 \xrightarrow[\text{UPDATING}]{} d(2) = 2 \;\&\; pred(2) = 1$

Figure 5: Iteration k = 1, Source Node = 1

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 (Figure 5). Figure 5 shows the simple (easy to use) bookkeeping method and updates (if needed).

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.

The Next iteration (k = 2), begins by checking the outgoing nodes from Node 3 (source node). 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 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.



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

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

$d(5) \overset{??}{>} d(3) + C_{35}$ {d} & {pred} are updated

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

Figure 6: Iteration k = 2, Source Node = 3

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 6). Figure 6 shows the simple (easy to use) bookkeeping method and updates (if needed).

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.

The Next iteration (k = 3), begins by checking the outgoing nodes from Node 4 (source node). 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 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.

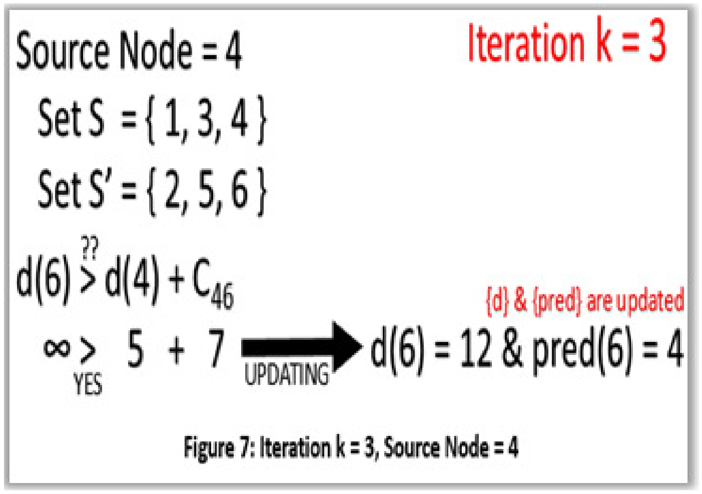


Figure 7: Iteration k = 3, Source Node = 4

In this case, the stored value for d[6] is infinity (∞) so the value in Vector [d] and [pred] will need to be updated with the computed value (Figure 7). Figure 7 shows the simple (easy to use) bookkeeping method and updates (if needed).

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.

The Next iteration (k = 4), begins by checking the outgoing nodes from Node 2 (source node). We check the outgoing Nodes (3 and 4) from Node 2 (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[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.

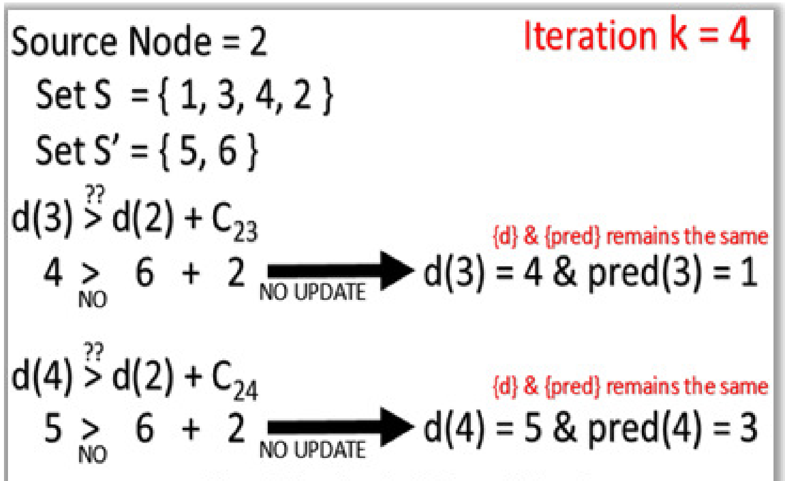

Source Node = 2
Iteration k = 4
Set S = { 1, 3, 4, 2 }
Set S' = { 5, 6 }
$d(3) \overset{??}{>} d(2) + C_{23}$
{d} & {pred} remains the same
$4 \underset{NO}{>} 6 + 2$
NO UPDATE
d(3) = 4 & pred(3) = 1
$d(4) \overset{??}{>} d(2) + C_{24}$
{d} & {pred} remains the same
$5 \underset{NO}{>} 6 + 2$
NO UPDATE
d(4) = 5 & pred(4) = 3

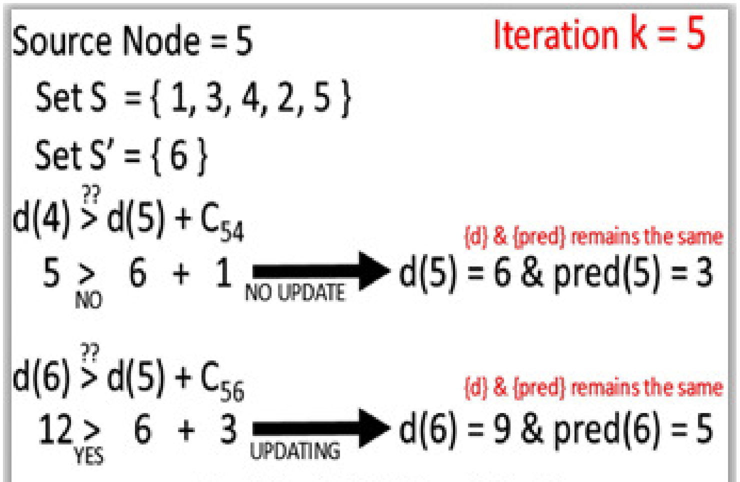
Figure 8: Iteration k = 4, Source Node = 2

In both of these cases, the stored values for 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 8). Figure 8 shows the simple (easy to use) bookkeeping method and updates (if needed). 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. Remark: When determining the smallest value was a tie for Nodes 2 and 5 (see previous iteration), we were better off to select and move Node 5 instead of arbitrarily selecting Node 2.

The Next iteration (k = 5), begins by checking the outgoing nodes from Node 5 (source node). 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 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 9). However, the value for d[6] is greater-than the computed values in Vector [d] and [pred] will need to be updated with the computed values (Figure 9). Figure 9 shows the simple (easy to use) bookkeeping method and updates (if needed). In this case, one outgoing node is updated is needed since the stored values is greater-than the computed values.



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

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

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

{d} & {pred} remains the same

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

Figure 9: Iteration k = 5, Source Node = 5

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.

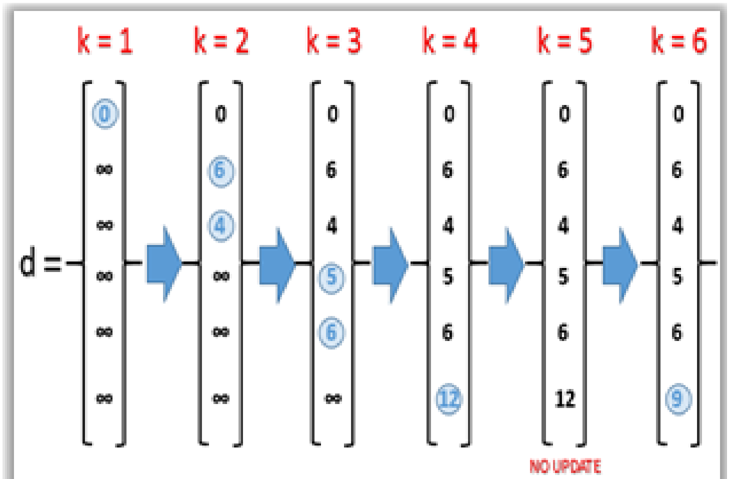


Figure 10: Vector [d] Updates During Iterations

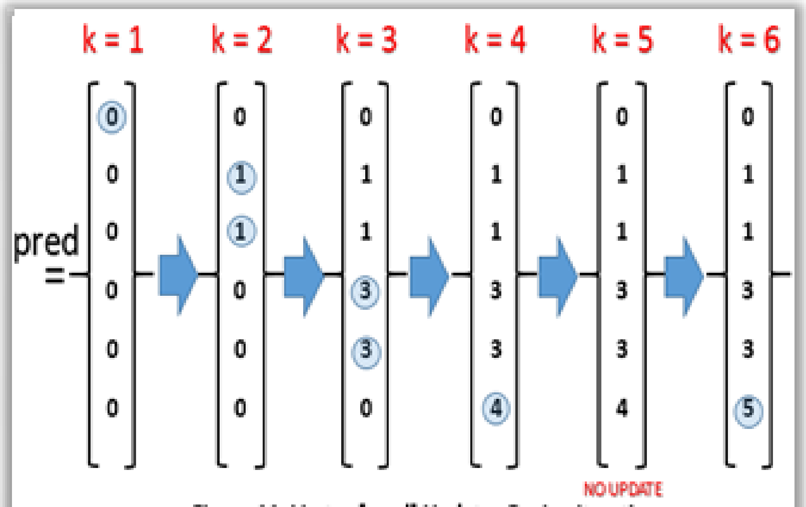


Figure 11: Vector [pred] Updates During Iterations

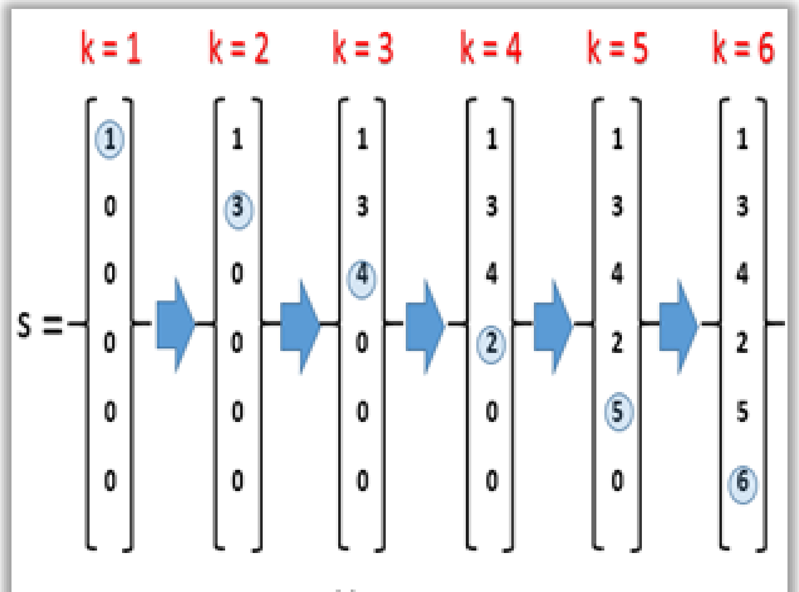


Figure 12: Set {s} Updates During Iterations

The updates for each iteration for Vector [d], Vector [pred], and Set {s} are displayed in Figures 10, 11, and 12, respectively. For each iteration, the highlighted circle (light blue circle) shows the corresponding values being updated during that iteration.

3. Java Computer Animated Software Tool for Teaching Forward Dijkstra Algorithm

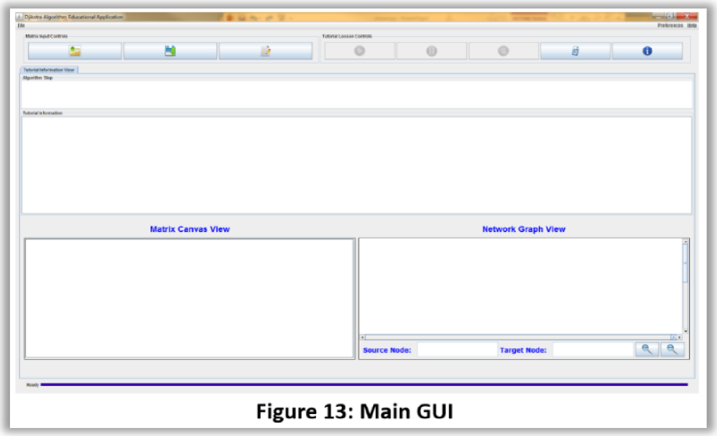
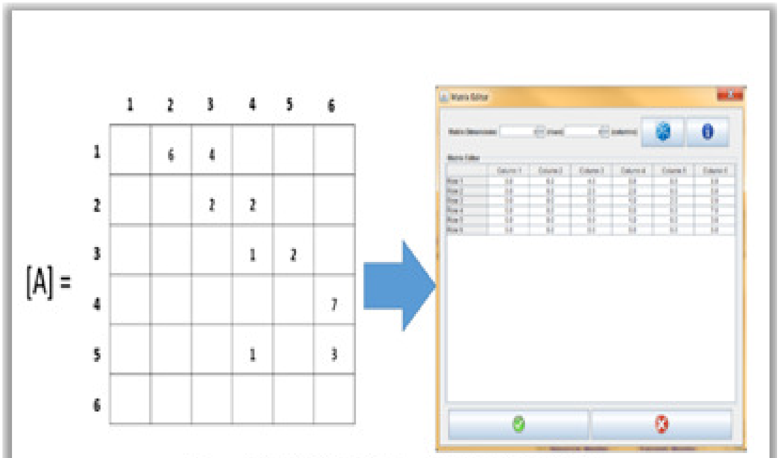


Figure 13: Main GUI

The previous section's small-scale 6 x 6 Matrix [A] graph data will be used for the Java [7-9] computer animated software tool for teaching the Dijkstra algorithm. The users/learner will initialize and run the Java software. Once initialized, the application's "Main Control" Graphical User Interface (GUI) is displayed (Figure 13). 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.


[A] =

	1	2	3	4	5	6
1		6	4			
2			2	2		
3				1	2	
4						7
5				1		3
6						

Figure 14: Matrix Editor (Matrix [A] Data)

The users/learners 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 14) 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.

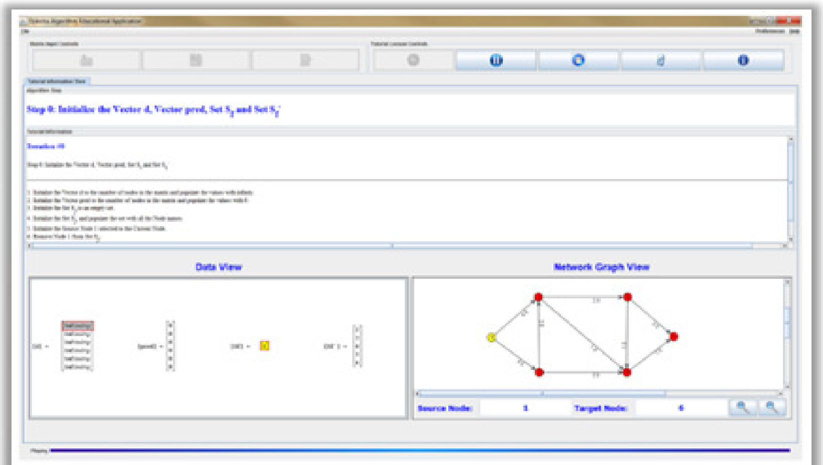
After the Matrix data [A] is entered into the "Matrix Editor" and "Ok" is selected, the focus is returned back to the "Main Control" GUI. The matrix data [A] is shown in to viewable formats, the “Matrix Canvas”, "Data", and "Network Graph" views. Both views will be updated accordingly throughout the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

Once the matrix data [A] is inputted, the "Play" button is enabled. 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 is lectured to the user/learner by a computer animated voice. The algorithm steps handled within the Java software are as follows:

3.1. Step 0 – Initialization

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/learning to select what the source and destination nodes.



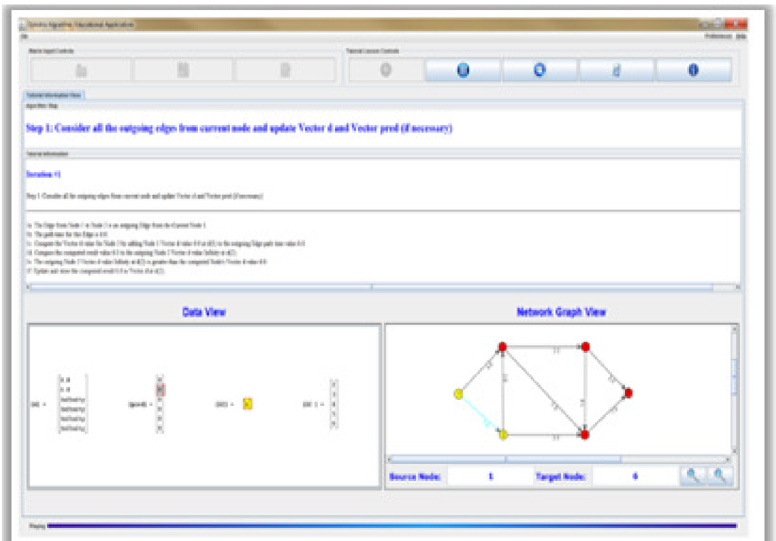
Step 0: Initialize the Vector d, Vector pred, Set S_1 and Set S_2'

Figure 15: Main GUI (Step 0)

The Java Computer Animation for Teaching the Forward Dijkstra Algorithm will initialize the data vectors and sets and set the source node (Figure 15).

3.2. Step 1 – Consider all Outgoing Edges from the Current Node

The Java Computer Animation for Teaching the Forward Dijkstra Algorithm will consider all outgoing links from the current node and 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.



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

Iteration #1

[illegible]

Data View

Network Graph View

Source Node: 1

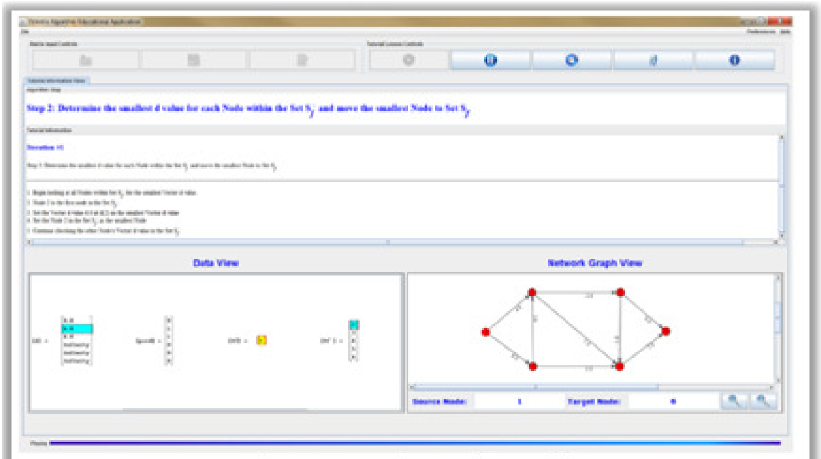
Target Node: 6

Figure 16: Main GUI (Step 1)

The Java Computer Animation for Teaching the Forward Dijkstra Algorithm will have animated each step and provide detailed animated voice and text for each step (Figure 16).

3.3. Step 2 – Determine the Smallest "d" Value for each Node within Set $S_{final`}$ (Prime) and Move the Smallest Node to Set S_{final}

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}.



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

[illegible]

Data View

Network Graph View

Source Node: 1

Target Node: 6

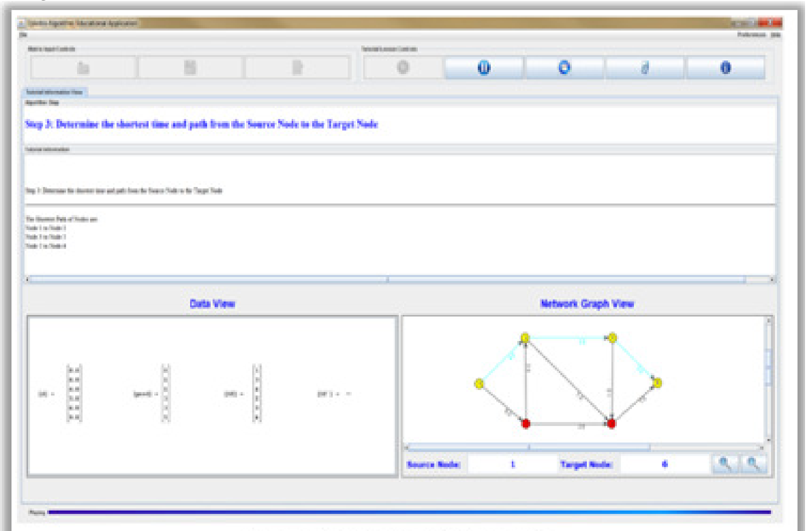
Figure 17: Main GUI (Step 2)

The Java Computer Animation for Teaching the Forward Dijkstra Algorithm will update the current node as the source node for the next iteration (Figure 17).

3.4. 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 18).



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

Figure 18: Main GUI (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. CONCLUSIONS

In this article, the basic Dijkstra algorithm has been firstly summarized. Then, Java Computer Animated Software Tool has been developed to enhance student's learning. The developed Java animated software has all the following desirable features/capabilities, such as:

1. The developed software tool should be user friendly (easy to use).
2. Graphical/colourful 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 can 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 his/her own data, and compare his/her hand-calculated results with the computer software's generated results (in each step of the algorithm) for enhancing/improving his/her learning abilities.

ACKNOWLEDGEMENTS

The partial support provided by the NSF grant # ACI-1440673 (ODU-RF Project # 100507-010) to Duc T. Nguyen is gratefully acknowledged. The work of Sosonkina was supported in part by the Air Force Office of Scientific Research under the AFOSR award FA9550-12-1-0476, and by the National Science Foundation grants NSF/OCI---0941434, 0904782, 1047772.

REFERENCES

[1] 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

[2] 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.

[3] 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.

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

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

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

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

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

[9] Java Platform Standard Edition.http://www.oracle.com/technetwork/java/javase/downloads/index.htmllishers.

Authors

Ivan Makohon is a current Old Dominion University graduate student pursuing his Masters in Modeling and Simulations. He has received his Bachelors of Science in Computer Science from Christopher Newport University (CNU). He has been working as a contractor (and now, a civil servant) as a Senior Software Engineer, and he is also a Private Pilot.

Duc T. Nguyen has received his B.S., M.S. and Ph.D. (Civil/Structural engineering) degrees from Northeastern University (Boston), University of California (Berkeley), and the University of Iowa (Iowa City), respectively. He has been a Civil Engineering faculty at Old Dominion University since 1985. He has published 4 undergraduate/graduate textbooks. Over 160 research articles and nearly $ 4 million funded projects have been generated by him. He has received Cray Research, NASA, ODU shining star, and Rufus Tonelson Distinguished Faculty Awards. His name has been included in the ISIHighlyCited.com list of most highly cited researchers in Engineering.

Masha Sosonkina has received her B.S. and M.S. degrees in Applied Mathematics from Kiev National University in Ukraine, and a Ph.D. degree in Computer Science and Applications from Virginia Tech. During 2003-2012, Dr. Sosonkina was a scientist at the US Department of Energy Ames Laboratory and an adjunct faculty at Iowa State University. She is currently a professor of Modeling, Simulation and Visualization Engineering at Old Dominion University. She has also been a visiting research scientist at the Minnesota Supercomputing Institute, at CERFACS and INRIA French research centers. Her research interests include high-performance computing, large-scale simulations, parallel numerical algorithms, performance analysis, and adaptive algorithms.

Yuzhong Shen received his B.S. degree in Electrical Engineering from Fudan University, Shanghai, China, M.S. degree in Computer Engineering from Mississippi State University, Starkville, Mississippi, and Ph.D. degree in Electrical Engineering from the University of Delaware, Newark, Delaware. His research interests include computer graphics, visualization, serious games, signal and image processing, and modeling and simulation. Dr. Shen is currently an Associate Professor of the Department of Modeling, Simulation, and Visualization Engineering and the Department of Electrical and Computer Engineering of Old Dominion University. He is also affiliated with Virginia Modeling, Analysis, and Simulation Center (VMASC). Dr. Shen is a Senior Member of IEEE.

Manwo Ng is currently an Assistant Professor of Maritime and Supply Chain Management at Old Dominion University (ODU). He obtained his Ph.D. degree in Transportation from The University of Texas at Austin. His research focuses on port operations, container shipping, and transportation network modeling.