Visualization and Animation for Teaching Frank-Wolfe Transportation Network Equilibrium

Zhi Li, Ivan Makohon
Old Dominion University (Graduate Students)
Norfolk, Virginia
zxxli001@odu.edu, imako001@odu.edu

Masha Sosonkina#, Yuzhong Shen^,
Duc T. Nguyen#
Old Dominion University (^/#: Assoc./Full Prof)
Norfolk, Virginia
msosonki@odu.edu, yshen@odu.edu,
dnguyen@odu.edu

ABSTRACT

The popular Frank-Wolfe (FW) algorithm for solving the "network equilibrium" problems has been well-documented in the literature. Other (more efficient) variations of the FW algorithm (such as Conjugate FW, Bi-Conjugate FW algorithms) have also been extensively studied by the research communities.

In this paper, the basic FW algorithm is re-visited, with the ultimate goal of developing a useful, user-friendly, and attractive Java computer animation for "effectively teaching" this basic/important (transportation network equilibrium) algorithm. Since the shortest path (SP) algorithm (such as the well-known Dijkstra algorithm) is a basic building block within the FW algorithm, the readers are assumed to have a working knowledge about Dijkstra SP algorithm.

The final product from this work will help both the students and their instructor not only to master this technical subject, but also to provide a valuable tool for obtaining the solutions for homework assignments, class examinations, self-assessment studies, and other coursework. Engineering educators who have adopted "flipped class-room instruction" can also utilize the developed JAVA animation software for students to "self-learn" these algorithms at their own time (and at their preferable locations), and use valuable class-meeting time for more challenging (real-life) problems' discussions.

ABOUT THE AUTHORS

Zhi Li is a graduate student pursuing a master's of science degree in Modeling, Simulation and Visualization Engineering Department at Old Dominion University. He has received his bachelor of science degree in computer science and technology from Shandong University in 2012. He is currently working on the development of transportation education software.

Ivan Makohon is a current ODU graduate student pursuing his Masters in Modeling and Simulations. He has obtained 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.

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.

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. Hehas 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 ISIHighly Cited.com list of most highly cited researchers in Engineering.

Visualization and Animation for Teaching Frank-Wolfe Transportation Network Equilibrium

Zhi Li, Ivan Makohon
Old Dominion University (Graduate Students)
Norfolk, Virginia
zxxli001@odu.edu, imako001@odu.edu

Masha Sosonkina#, Yuzhong Shen^,
Duc T. Nguyen#
Old Dominion University (^/#: Assoc./Full Prof)
Norfolk, Virginia
msosonki@odu.edu, yshen@odu.edu,
dnguyen@odu.edu

I. INTRODUCTION

Knowing the equilibrium state of a given/large transportation network is an important, fundamental problem in transportation modelling. Transportation planners/engineers have often used the network's equilibrium for assisting them to make "wise decisions" in allocating limited resource to improve his/her network's performance. Efficient Deterministic User Equilibrium (DUE) algorithms, such as the Frank-Wolfe (FW), and its improved versions of Conjugate Frank-Wolfe (C-FW), and Bi-Conjugate Frank-Wolfe (Bi-C-FW) have been developed, tested and well documented in the literatures. Teaching the FW algorithms, however, can be a difficult/challenging task !
While some teaching information/lecture/tool/animation for FW algorithms have existed/appeared in the literatures, 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:

(a) The developed software/tool should be user friendly (easy to use)
(b) Graphical/colorful animation should be extensively used to display equations, and/or intermediate/final output results
(c) Clear/attractive computer animated instructor's voice should be incorporated into the software/tool
(d) The instructor's voice for teaching materials can be in different/major languages, such as English, Chinese, Spanish, etc.
(e) User's input data can be provided in either interactive mode, or in edited input data file mode, or by graphical mode.
(f) Options for partial (or intermediate) results and/or complete (final results) are available for the user to select
(g) Options for displaying all detailed intermediate results in the first 1-2 iterations, and/or directly show the final answers are available for users
(h) 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 of this article is organized as following. The basic FW algorithm is reviewed in Section II. JAVA software and useful animations for teaching the FW algorithms are explained in Section III (including some computer screen shots). Conclusions and future research works are summarized in Section IV.

II. REVIEW OF FRANK-WOLFE(FW) DETERMINISTIC USER EQUILIBRIUM(DUE)

To facilitate the discussion in this section, the following notations are defined (please also refer to Figure 1, and Table 1).

$t_a(.)$ = link performance (BPR = Bureau Public Road) function
$f_k(r,s)$ = flow on path "k" connecting the OD (Origin-Destination) pairs (r,s)
x_a = flow on link "a"
delta_a,k(r,s) = 1, if link "a" is on path "k" connecting the OD pairs (r,s) = 0, otherwise
Ca = road capacity (in # vehicles/hour)

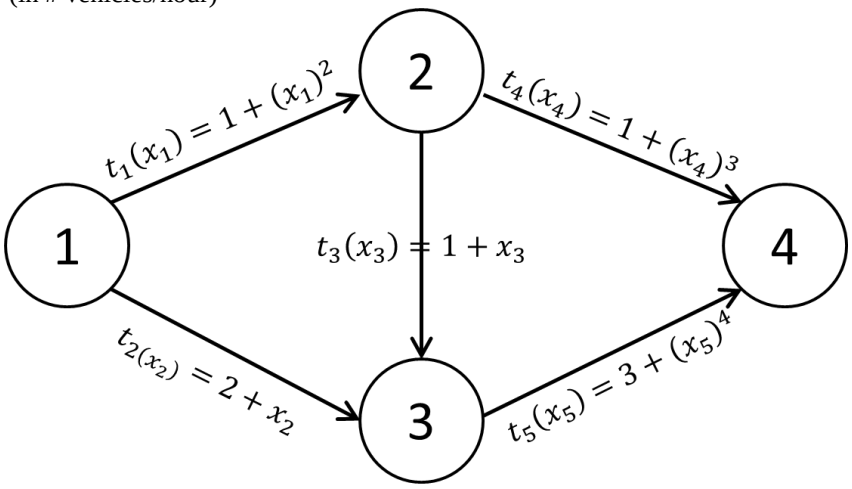

Figure 1. A Simple (4-node/5-link) Network with Defined Links' Travel Costs

Remarks

(a) The link performance (time) function shown in the above figure/example basically stated that the travel time t_i on link " i " is proportional to the link flow "x_i ".

(b) In practical applications, the Bureau Public Road (BPR) function/data is usually employed, such as [see Table 1]

$$t_a(x_a) = t_a 0\left[1 + alfa\left(\frac{x_a}{Ca}\right)^{beta}\right]$$

and the DUE objective function can be defined as:

$$\min Z(x) = \sum_a \int_0^{x_a} t_a(w)dw$$

Table 1. A Simple (4-node/5-link) Network with BPR Parameters Defined

Link #	Link	t_a0	ca	alfa	beta	travel-time
1	1-2	3	3	0.15	4	25
2	1-3	1	3	1.00	3	35
3	2-3	1	2	0.15	2	15
4	2-4	1	2	0.15	2	45
5	3-4	2	2	1.00	4	15

For the above example, the Origin-Destination (OD) matrix can be given as [see Table 2]

Table 2. Original-Destination (OD) Matrix For a 4-node/5-link Network

		1	2	3	4
[OD] =	1	0	0	0	10
	2	0	0	0	20
	3	0	0	0	0
	4	0	0	0	0

In other words, we assume that:

q_{14} = 10 = # cars travel from origin node 1 to destination node 4
q_{24} = 20 = # cars travel from origin node 2 to destination node 4

The Frank-Wolfe (FW) algorithm can be explained through the following step-by-step procedures, based on the given network described in Figure 1 and Table 1.

Step 0: Initialization ….

Iteration counter k = 1
Assuming each link flow "x_a" is zero initially, hence [see Figure 2]

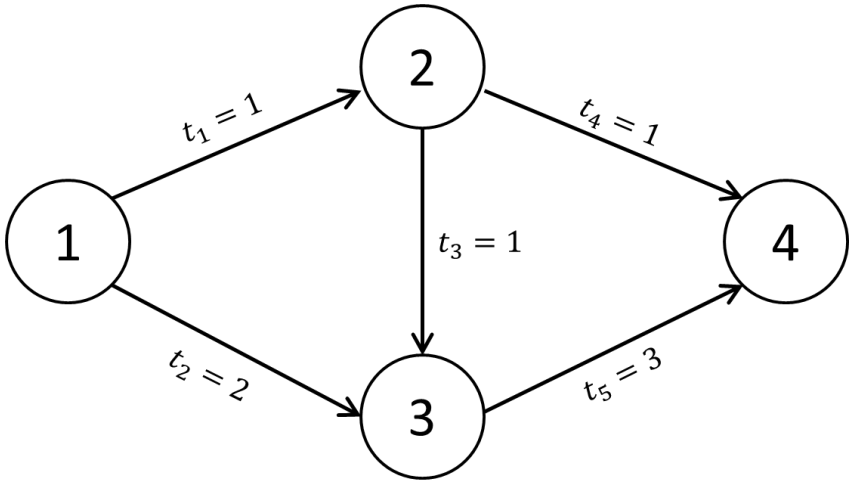

Figure 2. A Simple (4-node/5-link) Network with Free Flow Initially

Using any SP algorithms, such as the Dijkstra method, we can find the shortest path from source nodes 1 and 2 to all remaining destination nodes (including the destination node 4, in this example).

Thus, we found the shortest path from node 1 to node 4 is 1-2-4 (10 cars travel on link 1-2, and 10 cars travel on link 2-4), and we found the shortest path from node 2 to node 4 is 2-4 (20 cars travel on link 2-4)

Therefore, there will be a TOTAL of 10 cars travel on link 1-2 (or L1), and 10+20 = 30 cars travel on link 2-4 (or L4). The link flow vector {x_1} can be computed/updated as:

{x_1} = {10 cars, 0, 0, 30 cars, 0}

Step 1 (beginning of a do-loop):

With the above revised link flow, we re-calculate the travel time for each link, as follows [see Figure 3]

$\{t_1, t_2, t_3, t_4, t_5\}$ = {101sec, 2sec, 1sec, 27001sec, 3sec}

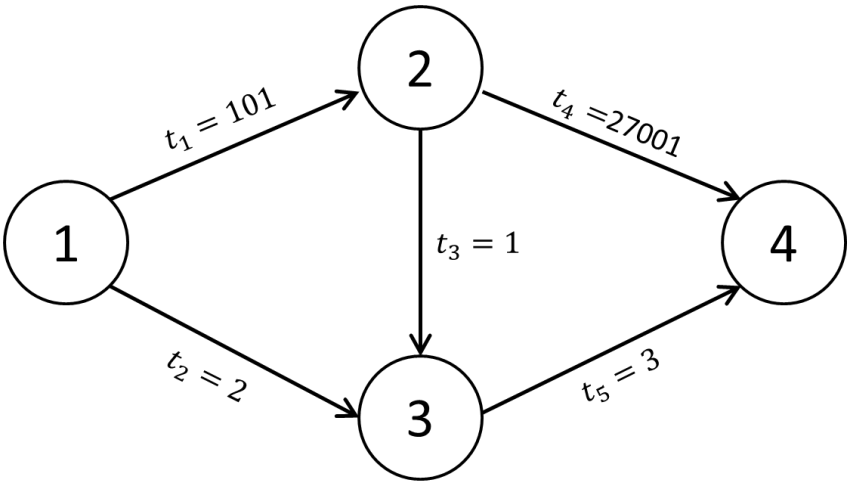

Figure 3. Revised Links' Travel Costs Based On Current Link Flows

Applying any SP algorithms [see the above Figure 3] to " find the best direction to travel = { y_k – x_k } " (for achieving optimal user equilibrium). The SP results are:
The shortest path from node 1 to node 4 is 1-3-4 (10 cars travel on link 1-3, and 10 cars travel on link 3-4),
and the shortest path from node 2 to node 4 is 2-3-4 (20 cars travel on link 2-3, and 20 cars travel on link 2-4).

Therefore, there will be a TOTAL of 10 cars travel on link 1-3 (or L2), and 20 cars travel on link 2-3 (or L3), and 10+20 = 30 cars travel on link 3-4 (or L5). The link flow vector {y_1} can be computed/updated as:

$$\{\bar{x}\_k\} = \{\bar{x}\_1\} = \{0, 10 \text{ cars}, 20 \text{ cars}, 0, 30 \text{ cars}\} = \{y\_k\} = \{y\_1\}$$

Step 2:

To find the "step size", we use the following standard formula

$$x\_k+1 \;=\; x\_k \;+ (\text{step size}) * \{ yk – x\_k \}$$

In the above formula, the vector {y_k – x_k} can be interpreted as the "direction to travel" vector. The step size can be computed by the simple MSA (Method of Successive Average), such as step size = 1/k (where k = iteration counter). Thus, as # iterations increased, the computed step size will be smaller & smaller. This simple approach for step size calculation makes sense, because near/at the optimum, the step size should be very close to zero.

$$x\_k+1 \;=\; x\_k \;+ (\text{step size} = 1/k) * \{ yk – x\_k \}$$

Step 3:

Calculating the new link flows

x_2 = x_1 + (step size = 1/1) * {y_1 – x_1}
x_2 = {10, 0, 0, 30, 0}' + {step size = 1} * {0-10, 10-0, 20-0, 0-30, 30-0}
x_2 = {0, 10, 20, 0, 30}'

Step 4:Convergence Test

If "CONVERGE", then stop (and print all the link flows).
If "NOT converge", then (set iteration counter k = k+1) and go back to Step 1

III. JAVA COMPUTER ANIMATED SOFTWARE/TOOL FOR TEACHING FRANK-WOLFE ALGORITHMS

As we already introduced, the software is written in Java meant to create 2D animations and voice explanations for Frank-Wolfe Algorithms. The whole software is developed from the scratch and with lots of functions and features. We will start with its neat and friendly user interface.

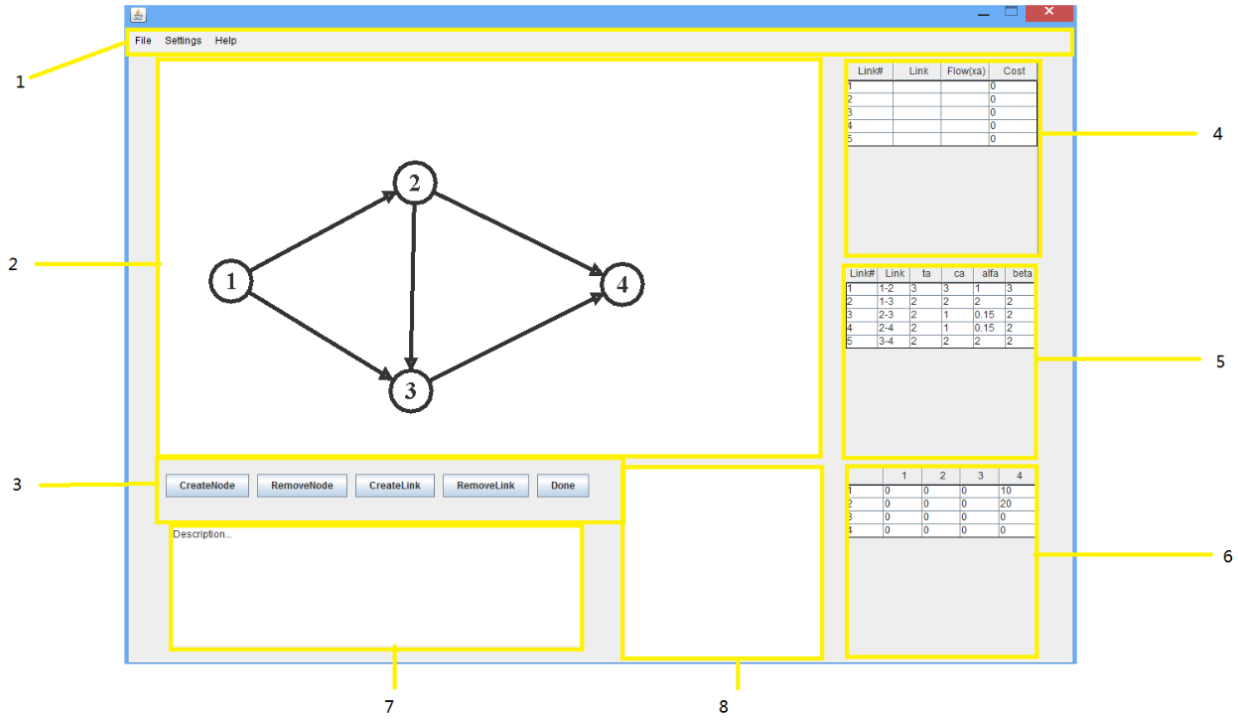

Figure 4. User Interface

The tool's window is developed with AWT (Abstract Window Toolkit) package in Java standard API. It consists of eight main components as shown in Figure 4.

- The top of window is the menu bar, it includes some basic options, like open and save, and several special options, such as shortest path methods and animation modes. We will introduce these special options in the following paragraph.
- Component 2 is the most important area in this software. Users can easily draw any network by several mouse clicks with buttons in component 3 as long as it has less than 8 nodes and 20 links. Besides that, the animation is also played in this area after we create network.
- Component 5 and 6 are two tables responsible for input data, BPR and OD pairs respectively. The structures of tables are determined by software automatically according to the network topology we draw in Component 2. In other words, users don't need to worry about the numbers of rows in these tables due to a self-correcting system in them.
- After we finish creating network, a 0-1 adjacency matrix is shown in component 8 to represent the network topology.
- While playing animations, not only text tutorials are printed in component 7, but also some intermediate data, including traffic flows and travel times on each iteration, are displayed in component 4.

As we mentioned above, the software has several special options, such as animation modes and shortest path methods. The animation mode is one of important features in this tool. Software supports two modes, steps animation and color map.

In steps animation mode, the software will play a detailed step-by-step animation for a given network or problem. This mode aims to help students understand the working mechanism of Frank-Wolfe algorithm. To achieve this goal, program highlights relative nodes and links to simulate the link flow going as well as prints formulas and intermediate data to the screen at each iteration step. In addition to visible perception, auditory perception is introduced as well. A gentle computer generated voice is accompanied while the animation is playing.

However, the second mode, dynamic color map, has a different objective. We assume that students already have a main idea on how the Frank-Wolfe algorithm works after they watch animations in the first mode several times. They might not be clear on the variation and status of the whole network during iterations. To solve this problem, we come up with an idea of using a continuous color map to describe the status of Frank-Wolfe algorithm. In this mode, we don't emphasize the algorithm steps; we adapts a blue-red spectrum to represent the volumes of link flow.

In terms of shortest path methods, this software supports both P-LCA (Polynomial Label Correcting Algorithm) and Dijkstra Algorithm. Users can choose either of them in these two modes to compare their final results.

The previous paragraphs introduced the main features of Frank-Wolfe Algorithm teaching software, in the following sections, we will focus on the Java algorithms which implement Frank-Wolfe Algorithm's animation.

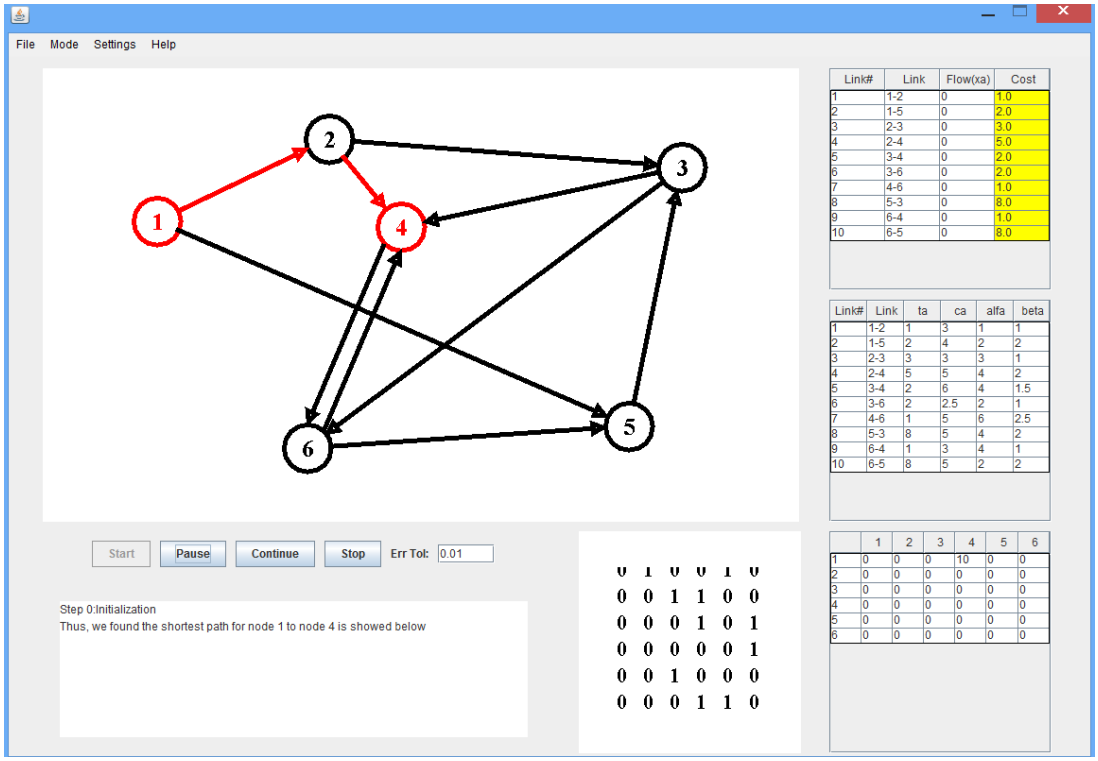

Figure 5. Steps animation mode uses highlighted nodes and links to describe every step of algorithm.

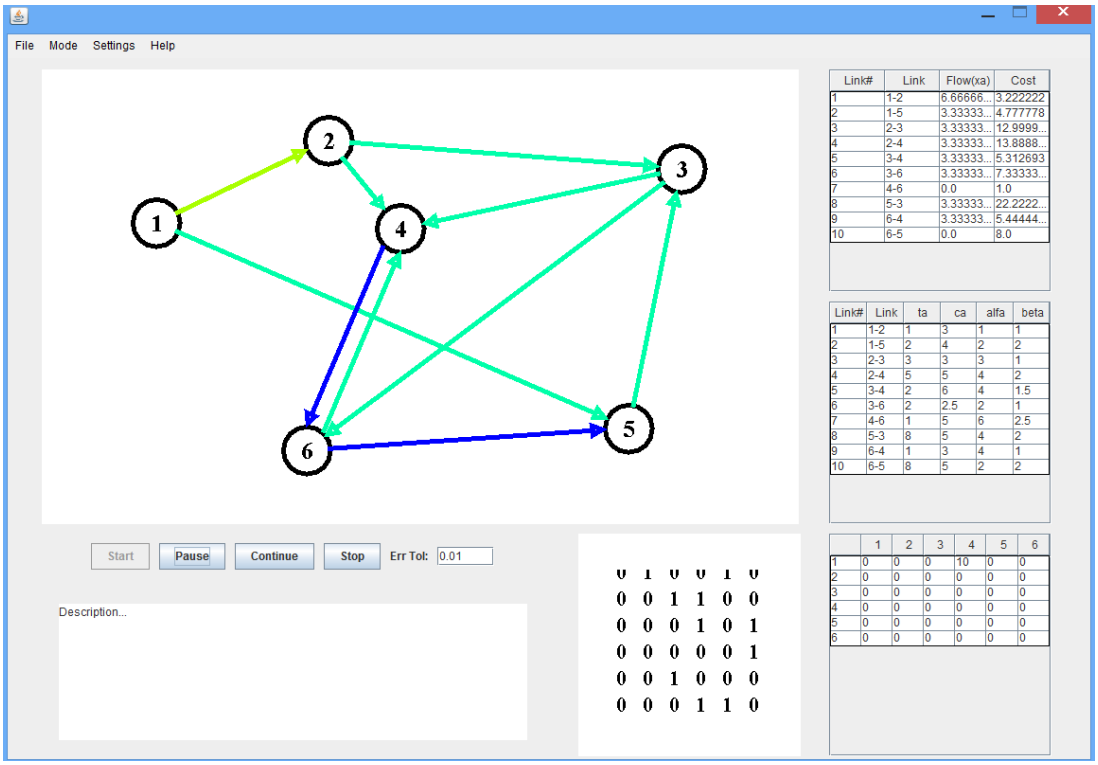

Figure 6. Color map mode adapts blue-red spectrum to show link flows change among iterations

In order to develop this software easier, the calculation and animation parts are not being done simultaneously. The program can be divided into three main modules, I/O module, calculation module and animation module. After users create network and input necessary data (BPR data, OD pairs and error tolerance), program's I/O module will collect and analyze input data, and then send to calculation module. Once calculation module receives data, it initializes the iteration as step 0 in section II and starts iteration loops. All the calculating processes are exactly same as in section II. When it converges, the iteration quits. We need to pay attention that calculation module not only stores the final results but also records other intermediate data, such as link flow, current error and travel time for all iterations, and send them to animation module. Actually, the animation module already gets the final results before it starts animation and generates animation by translating intermediate data.

IV. CONCLUSIONS

In this article, the basic Frank-Wolfe (FW) algorithm has been firstly summarized. Then, JAVA Computer Animated Software/Tool has been developed to enhance students' learning. The developed JAVA animated software has all the desirable features, which have been stated in the introduction section.

V. 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

Sheffi, Y. (1985). *Urban Transportation Networks: Equilibrium Analysis with Mathematical Programming Methods*, Prentice-Hall, Inc.

Lawson, G., Allen, S., Rose, G., Nguyen, D.T., Ng, M.W. (2013). Parallel Label Correcting Algorithms For Large-Scale Static and Dynamic Transportation Networks on Laptop Personal Computers, *TRB 2013 Annual Meeting; Session 844Presentation # 13-2103; Poster Presentation # P13-6655*.

Paul Johnson III, Duc T. Nguyen, and Manwo Ng. (2014). An Efficient Shortest Distance Decomposition Algorithm For Large-Scale Transportation Network Problems, *TRB 2014 Annual Meeting*.

Maria Mitradjieva, and Per Olov Lindberg. (2012). The Stiff is Moving – Conjugate Direction Frank-Wolfe Methods with Applications to Traffic Assignment. *Transportation Science, Volume 47, Issue 2*, pages 280-293.

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

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

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

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

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

Java Platform Standard Edition. http://www.oracle.com/technetwork/java/javase/downloads/index.html