Vol. 10, No. 1, September 2009, pp. 34–40
ISSN 1532-0545 | 09 | 1001 | 0034

INFORMS Transactions on Education

informs

DOI 10.1287/ited.1090.0026

Teaching Note

An Animated Demonstration of the Uncapacitated Network Simplex Algorithm

Thanasis Baloukas, Konstantios Paparrizos
Department of Applied Informatics, University of Macedonia, GR-540 06 Thessaloniki, Greece
{thanasis@uom.gr, paparriz@uom.gr}

Angelo Sifaleras
Department of Technology Management, University of Macedonia, GR-592 00 Naousa, Greece,
sifalera@uom.gr

Operations Research (OR) instructors use visualizations to help teach graph algorithms and data structures. Algorithm visualization is an illustration of abstract concepts included in computer algorithms, making use of either static or dynamic (animated) computer graphics. In this paper we present new software that includes an animated demonstration of the uncapacitated primal network simplex algorithm. The aim of the animation software is twofold. First, it aims to help OR students understand the algorithm. The algorithm is difficult to grasp for many students because it generates a sequence of rooted trees. Second, it aims to enable OR instructors to explain each iteration of the algorithm visually with minimal effort. The software can be used in combinatorial optimization, graph theory, and similar courses. The software has been implemented as a Java applet, is freely available and highly interactive, and can be accessed through the Web. The software shows the solution process through textual information and depicts the relevant steps in pseudo code using multiple views.

Key words: OR education; combinatorial optimization; educational software
History: Received: April 2008; accepted: September 2008.

1. Introduction

Algorithm visualization (AV), a subcategory of software visualization (SV), is a scientific discipline that uses multimedia (graphics, animation, and sound) to facilitate the understanding of abstract notions in computer algorithms. AV can be either static or dynamic (Stasko et al. 1998). Static AVs depict discrete graphical snapshots that explain the computational steps of an algorithm while it is being executed. In these snapshots, AV programmers illustrate difficult points of the algorithm that they believe must be explained more effectively. On the other hand, dynamic AVs or algorithm animations use smooth animations, to convey the temporal evolution of an algorithm execution. Many AV tools also provide program visualization (another subcategory of SV; Stasko et al. 1998) by highlighting program code and variables.

AV tools are often implemented in Java (Naps 1997, Boroni et al. 1997) because this permits them to be used on every computer, independent of its operating system, as long as the free Java Runtime Environment (JRE, distributed by Sun Microsystems, http://java.sun.com) has been installed. Furthermore, a special category of Java programs, known as Java applets, can be embedded in Web pages and can be run in common Web browsers, making them well-suited for distance learning courses.

The network simplex algorithm, hereinafter called algorithm, is included in the syllabi of various postgraduate courses such as network flows, linear programming, and combinatorial optimization. These courses are taught in such university departments as operations research, mathematics, computer science, and electrical engineering. The algorithm is considered complex by most students, compared to standard graph and network algorithms.

In this paper, we present a smooth animation of the algorithm through visualization software that we have developed. We know of no other animation software for the network simplex algorithm, either for the uncapacitated or the capacitated version. Apart from animation, software also provides static visualization of the algorithm (Baloukas and Paparrizos 2006) and

static visualization for many other graph and network algorithms (Baloukas et al. 2005). In tandem with the visualization, the software displays algorithm data for the current iteration and all previous iterations. An important feature of the software, not present in similar tools, is that it allows users to draw arbitrary graphs and then to watch visualizations of the implemented algorithms applied on those graphs. Other tools typically only provide visualizations for specific and predefined problem instances. Furthermore, OR instructors benefit from visual and interactive educational tools that allow students to achieve a deeper understanding than by simply reading a textbook. Most OR instructors are interested in improving their teaching (Scott 2001, 2002), and e-learning technologies are one means that operations researchers have investigated recently to improve OR education (Papamanthou et al. 2005). Broadband networks and services (Varkas et al. 2005) have proved to be of great assistance in the advancement of e-learning methods, including enabling the use of Java applets like the one we have developed.

The software can be used either by instructors to explain the algorithm or by students as they struggle to comprehend the algorithm. The software is available as either a Java applet or a standalone application.[1]

The paper is structured as follows. In §2, we review several AV tools that have been developed by other researchers. In §3, we describe the algorithm and the data structures that are necessary for its effective visualization. Section 4 provides details concerning the animation of the algorithm. Section 5 describes student reactions and comments after they watch the animation and discusses future research.

2. Related Work

Many AV tools have been developed for educational purposes. These tools include static and dynamic visualizations for sorting and searching, graph and network algorithms, tree traversal, string matching, distributed and parallel computing, computational geometry, and automata theory.

JHAVE (Naps et al. 2000) is a software tool that provides Web-based algorithm visualizations and also displays the underlying pseudo code in a separate graphical window. It is based on client-server architecture and encompasses static and dynamic visualizations. JHAVE interrupts visualizations with "stop-and-think" questions to encourage students not only to watch visualizations passively but also to engage actively, by trying to predict what will happen next. JHAVE contains visualizations for hashing, searching, sorting, tree traversal, graph search, shortest paths, and minimum spanning tree algorithms.

ANIMAL (Rößling et al. 2000) is a similar didactic tool featuring animations for a wide range of algorithms and data structures. The animations are produced with a visual editor, with a scripting language, or through application programming interface (API) calls.

PILOT (Bridgeman et al. 2000) is a Web-based interactive visualization program employed to test students on their understanding of algorithms. It can produce random instances of a problem, and it allows students to solve the problem online. In addition, PILOT can be used as an AV tool for minimum spanning tree, shortest path, and graph search algorithms.

EVEGA (Khuri and Holzapfel 2001) is a standalone Java application that includes visualizations for graph algorithms. One of its strengths is its high degree of interactivity. Most AV systems allow users to start or stop a visualization and adjust its execution speed. In EVEGA users can also manipulate graphical objects through a built-in graph editor. The tool includes visualizations for breadth-first search, depth-first search, and a maximum flow algorithm.

Andreou et al. (2005) describe a novel Java application that provides a static visualization of a specific algorithm (Achatz et al. 1991) for the assignment problem. Karagiannis et al. (2006) describe an educational platform for solving network optimization problems using Apache Web Server, PHP, and Matlab Web Server. Lazaridis et al. (2007) describe Visual LinProg, a didactic tool (Java applet) for the visualization of the revised simplex algorithm.

In most algorithm animation systems, students watch animations developed by an expert programmer. An alternative technique allows students to create animations using a scripting animation language. SAMBA (Stasko 1997) is a software tool that belongs to the latter category. None of the software tools mentioned above includes a static or a dynamic visualization of the network simplex algorithm.

3. Algorithm Description

3.1. The Minimum Cost Network Flow Problem (MCNFP)

Let $G = (N, A)$ be a network with n nodes ($n = |N|$) and m arcs ($m = |A|$). The vector b consists of supplies or demands $b(i)$ of all nodes in the network. In addition, there is a cost c_{ij} of shipping one unit along arc (i, j). The amount (flows) of a commodity shipped directly from node i to j will be denoted by x_{ij}. In this paper, the network simplex algorithm is applied to uncapacitated and balanced MCNFPs.

[1] The Java applet can be obtained from http://eos.uom.gr/~thanasis/NetworkSimplex.html and the standalone version is at http://eos.uom.gr/~thanasis/NetworkSimplex.jar

The MCNFP aims to find the least expensive way to ship prescribed amounts of a commodity from supply nodes ($b(i) > 0$) to demand nodes ($b(i) < 0$) and satisfy the total demand; see Ahuja et al. (1993). Furthermore, we denote by w_i the dual variables and by s_{ij} the reduced-cost variables. The mathematical formulation of the balanced MCNFP is as follows:

$$\min \sum_{(i,j)\in A} c_{ij}x_{ij}$$

$$\text{s.t.} \sum_{\{k:(i,k)\in A\}} x_{ik} - \sum_{\{j:(j,i)\in A\}} x_{ji} = b(i), \quad i \in N$$

$$x_{ij} \geq 0, \quad (i,j) \in A$$

$$\sum_{i=1}^{n} b_i = 0$$

3.2. The Network Simplex Algorithm

The network simplex algorithm starts with a feasible tree solution T, meaning that $x_{ij} \geq 0$ only for arcs that are part of T. These arcs are known as basic arcs, whereas the rest arcs, having zero flow, are referred to as nonbasic arcs. The dual slack variables for the basic arcs are each equal to 0 ($s_{ij} = 0$).

Each iteration of the algorithm generates another feasible tree T' that has a corresponding basic feasible flow vector x'. At each iteration, also called a pivot, the algorithm adds an entering nonbasic arc to the tree, creating a single cycle. One arc, the leaving arc, is removed from the cycle according to a rule explained below. Table 1 illustrates the algorithm pseudocode. Any characters following "%" are comments/documentation.

If the algorithm first removed the leaving arc instead of adding the entering arc, then the basis tree would split into two subtrees. We denote by T_g the subtree that contains nodes k and g and by T_h the subtree that contains nodes h and l. These two sets can be used to update the w_i and s_{ij} variables. We use the artificial tree of the big-M problem (Ahuja et al. 1993) as a starting feasible tree. Interpreting the results of the big-M problem, we can draw conclusions about the solution of the original problem. We select the leaving arc using Cunningham's anticycling rule (Chvatal 1983).

3.3. Data Structures Used for the Visualization

To efficiently visualize the algorithm, we use the following data structures to store each rooted tree T:

1. The *predecessor array* p stores for each node i its father $p(i)$ in the tree T. That is, if $p(i) = j$, then node j is the predecessor (father) of node i. We let $p(root)$ be equal to -1.

2. The *direction array* t stores for each node i the number 0 if the arc $(p(i), i) \in T$ or the number 1 if the arc $(i, p(i)) \in T$. In the former case the arc $(p(i), i)$ is oriented away from the root, whereas in the latter case the arc $(i, p(i))$ is oriented toward the root. We let $t(root)$ be equal to -1.

3. The *depth array* d stores for each node i its depth $d(i)$ in the tree T. We let $d(root)$ be equal to 0. Using the arrays p, t, d and taking into account Cunningham's anticycling rule, we can identify the leaving arc f.

4. The *array* y, which is known as preorder or thread, stores for each node i its successor $y(i)$ in the preorder traversal of tree T. If i is the last node in the preorder, we let $y(i)$ be the root. Array y is required to update array d.

5. The *subtree* T^*. As mentioned before, the removal of the leaving arc from T splits T into two distinct subtrees. One of these contains the root; the other, which will be denoted by T^*, is necessary to properly update the dual variables w_i, the dual slack variables s_{ij}, and the array d. The subtree T^* can be easily identified using the arrays d and y.

6. The *pivot stem* or *backpath* z. The endpoints of the entering arc e will be labeled as e_1, e_2, and the endpoints of the leaving arc f will be labeled as f_1, f_2 in such a way that $e_1 \in T^*$ and $f_1 \in T^*$. The path in T joining the nodes e_1 and f_1 is called the *pivot stem*

Table 1 Network Simplex Algorithm Pseudocode

Start with a feasible tree T
Compute x_{ij}, w_i, s_{ij}
% Optimality check
while there exist arcs $(i,j) \notin T$ such that $s_{ij} = \delta < 0$
  Choose an arc $(g,h) \notin T$ such that $s_{gh} < 0$
  % $e = (g,h)$ is the entering arc
  Let C be the unique cycle of the network $T \cup (g,h)$
  $C^+ = \{(i,j) \in C$: (i,j) has the same direction as $(g,h)\}$
  $C^- = C - C^+$
  if $C^- = \varnothing$
    STOP % The problem instance is unbounded
  else
    $\varepsilon = x_{kl} = \min\{x_{ij}$: $(i,j) \in C^-\}$
  endif
  % $f = (k,l)$ is the leaving arc
  % Pivoting and data update
  $T' = T \cup (g,h) - (k,l)$ % or $T' = T + e - f$

$$x'_{ij} = \begin{cases} x_{ij} + \varepsilon, & (i,j) \in C^+ \\ x_{ij} - \varepsilon, & (i,j) \in C^- \end{cases}$$

  Compute the subtrees T_g, T_h, and the sets of arcs $[T_g, T_h]$
    (arcs heading from the T_g towards to the T_h set) and $[T_h, T_g]$
    (arcs heading from the T_h towards to the T_g set)

$$w'_i = \begin{cases} w_i - \delta, & i \in T_h \\ w_i, & \text{otherwise} \end{cases}$$

$$s'_{ij} = \begin{cases} s_{ij} - \delta, & (i,j) \in [T_g, T_h] \\ s_{ij} + \delta, & (i,j) \in [T_h, T_g] \\ s_{ij}, & \text{otherwise} \end{cases}$$

  $T \leftarrow T'$ % Substitute T with T'
endwhile

or *backpath* and will be stored in array z. It is always $z(1) = e_1$ and $z(u) = f_1$, where u is the length of z. Array z is required to properly update the arrays p, t, d, and y. It must be mentioned that the previous notation of the entering arc (g, h) cannot be used in this point, because node g may not always belong in T^*. The same applies for the leaving arc (k, l).

For a detailed description of the algorithm and data structures we used, see Ahuja et al. (1993), Chvatal (1983), Bazaraa et al. (2005), and Bertsekas (1998).

4. An Illustrative Example

In this section, we describe the animation of the algorithm as it is carried out in our program. To help students gain a comprehensive understanding of the algorithm, the following elements, we believe, should be visualized: the tree T produced at each iteration, the entering arc (g, h), the arcs that compose the cycle C, the arcs in set C^+, the arcs in set C^-, the leaving arc (k, l), the nodes belonging to the subtree T^*, and the nodes belonging to the pivot stem z. These elements are illustrated as follows.

The tree T is drawn as a rooted tree using a drawing algorithm that makes optimal use of the graphical window. The drawing algorithm is a modification of a level-order traversal of a binary tree (described in Standish 1997). The nodes of T are light blue and the arcs are black.

The entering arc (g, h) is illustrated with a dotted and a curved green arrow. A curve was chosen instead of a straight line to prevent intersecting the many nodes and arcs that may be lying between nodes g and h, thus making the visualization more aesthetically appealing.

The arcs in cycle C are illustrated in green. The arcs in C^+ (which are also green because C^+ is a subset of C) are indicated with a red "+" to symbolize that these arcs are in the same direction as the entering arc. In contrast, the arcs in C^- are indicated with a red "−" to symbolize that these arcs are of the opposite direction of the entering arc. The leaving arc (k, l) is illustrated with two parallel red line segments that are perpendicular to the arc. The nodes in subtree T^* are drawn in red. After deleting the leaving arc, the subtree T^* is cut from tree T, and in the next iteration of the algorithm it is placed under the entering arc. The nodes in the pivot stem or backpath z are drawn in red as well because they belong to subtree T^*; but to distinguish them from the remaining nodes in T^*, the pivot stem nodes are drawn inside an outer blue circle.

In Figure 1 we provide a weighted directed graph that was created with the graph editor in our tool.

Figure 1 A Weighted Directed Graph

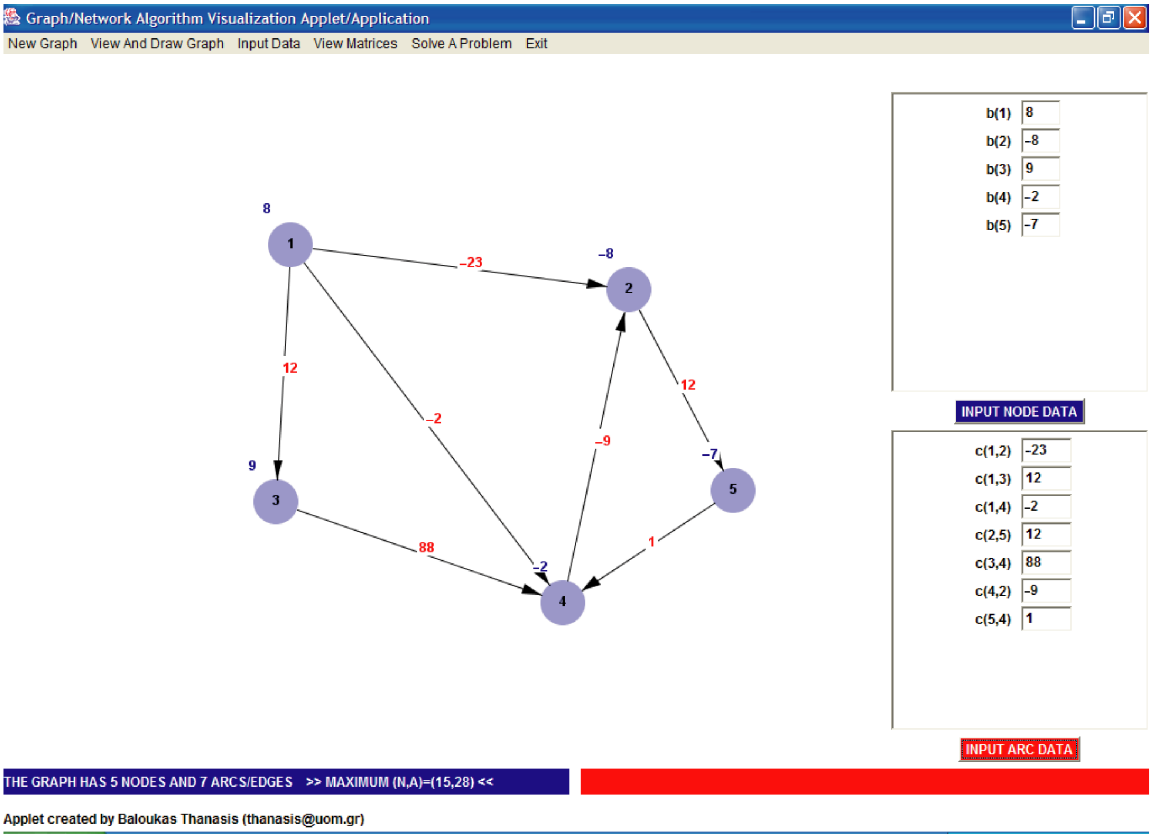
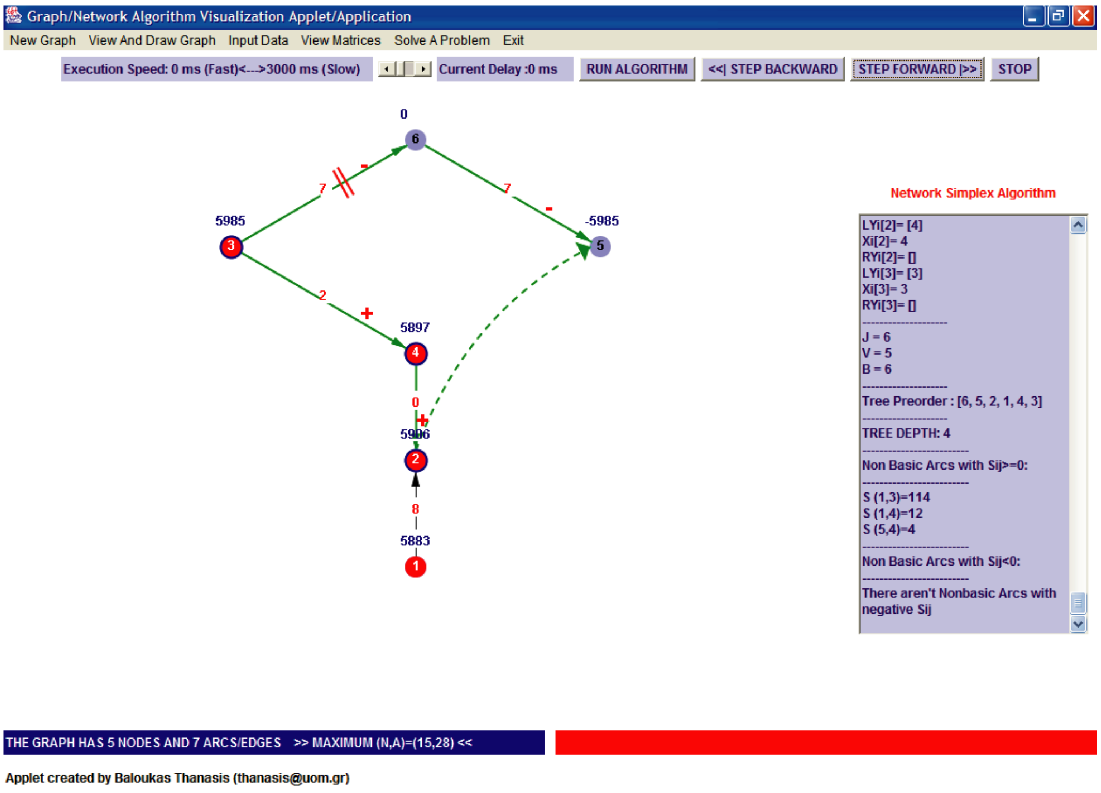


Figure 2 **Algorithm Visualization at Step 5**



The software allows users to enter arc costs and node supplies/demands (the sum of the node supplies/demands must equal to zero).

The *Solve A Problem* menu (Figure 1) allows one to visualize a specific graph or network algorithm. To see a static visualization of the network simplex algorithm, one chooses *Solve A Problem* → *Network Simplex*. To see the animation of the algorithm, one chooses *Solve A Problem* → *Network Simplex Animation*. After choosing either of these visualizations, one sees a toolbar containing four buttons, one slider, and two labels, as shown in Figure 2. Using this toolbar, one can opt to see either a nonstop visualization (by clicking the button labeled RUN ALGORITHM and specifying an execution speed) or a stepwise visualization. The stepwise visualization can be run either step-by-step forwards by pressing the button labeled STEP FORWARD |>> or step-by-step backwards by pressing the button labeled <<| STEP BACKWARD. In Figure 2 we provide a snapshot of the algorithm visualization after a user has stepped forward five times.

At this point we consider it useful to provide some implementation details regarding the tree T animation. For the tree T to be smoothly animated, we have to compute appropriate coordinates for its nodes and arcs. To accomplish this, we followed the next steps:

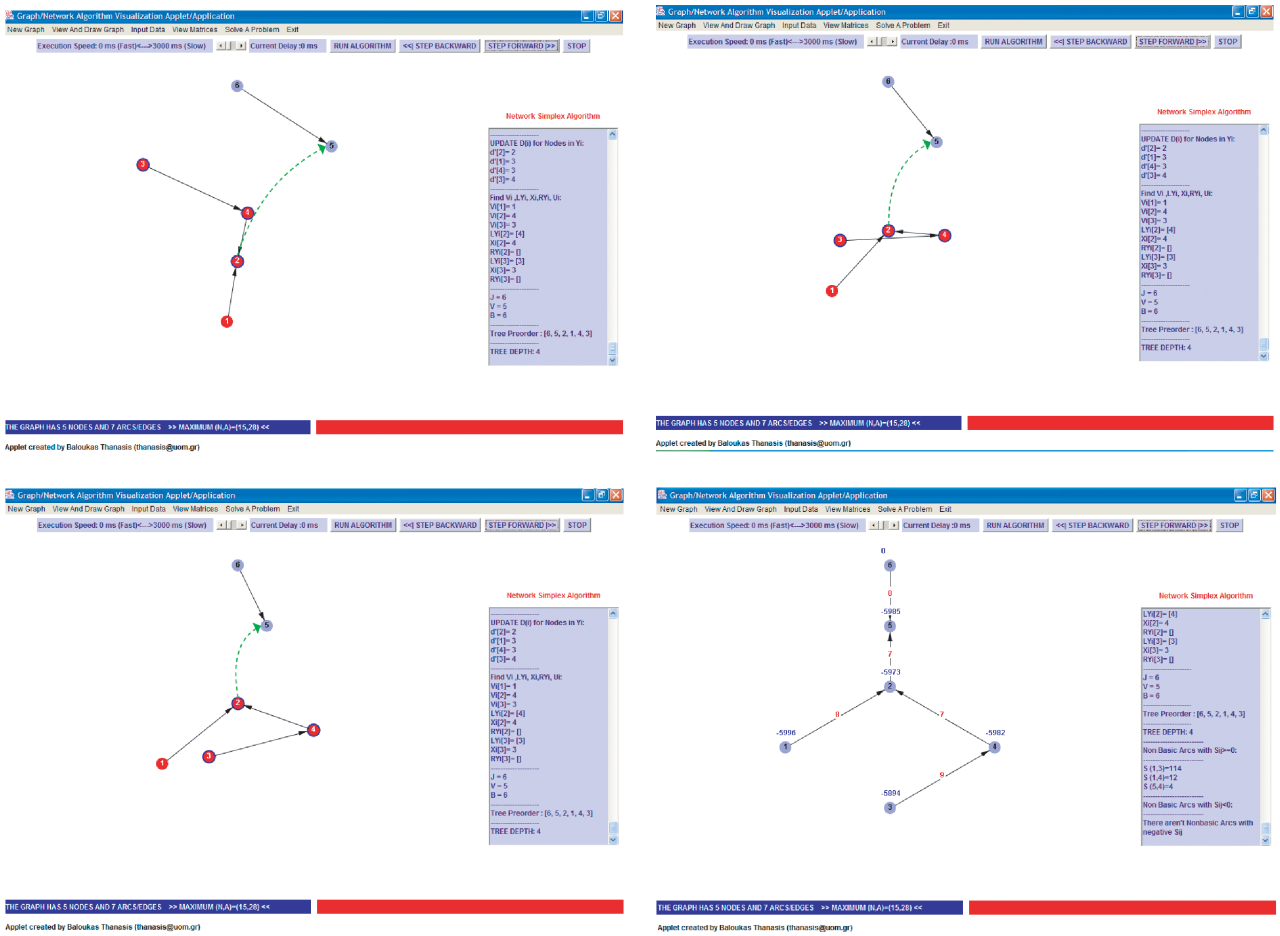
- We stored the coordinates of the arcs and nodes in the current tree T.
- We calculated the coordinates of the arcs and nodes of the next tree T'.
- For each pair of elements in the two trees, we calculated coordinates for 50 intermediate points. We used these intermediate points to display a sequence of 50 images that gives an impression of continuous motion from T to T'.

Initially, the subtree T^* is under the leaving arc. After the leaving arc is deleted, the subtree T^* is placed under the entering arc. This removal entails a number of visual changes:

- The parent nodes $p(i)$ for all nodes i on the pivot stem change value.
- The arcs $(i, p(i))$ or $(p(i), i)$ that connect some nodes i in the pivot stem with their parents $p(i)$ change direction. That is, some arcs that in the current iteration are oriented toward the root will be oriented away from it in the next iteration and vice versa.
- The above changes suggest that between two successive algorithm iterations, a rotation of the pivot stem z is carried out.

Indeed, as we observe in Figure 2, the parent nodes $p(i)$ of nodes i composing the pivot stem z are $p(2) = 4$, $p(4) = 3$, $p(3) = 6$. However, in Figure 3 the new parent nodes $p'(i)$ become $p'(2) = 5$, $p'(4) = 2$, $p'(3) = 4$. In addition, we notice a change in the orientation of arcs $(3, 4)$ and $(4, 2)$. For example, we observe that for arc $(3, 4)$ in Figure 2, $t(4) = 0$, whereas

Figure 3 A Sequence of the Subtree T^* Rotations



Note. The images depicting the sequence of the subtree T^* rotations can be found online (in Figure3images.zip) at http://ite.pubs.informs.org/.

in the last screenshot of Figure 3, $t'(3) = 1$. Also for arc $(4, 2)$, whereas in Figure 2 $t(2) = 0$, in Figure 3 $t'(4) = 1$.

A static visualization simply displaying the trees T and T' might not be so helpful for students to comprehend the rotation of the pivot stem that occurs at the same time with the movement of the subtree T^*. A more effective way for students to understand this rotation would be through a smooth animation of tree T.

For students to distinguish the above operations during the animation, the nodes in T^* are displayed in red; the nodes belonging to the pivot stem are indicated with an extra outer blue circle; and the remaining nodes of T are blue.

In Figure 3 we display graphical snapshots of the animated transition from T to T'. In parallel with the animation, the pivot stem (consisting of nodes 2, 3, and 4) is rotated. During the animation we also display the entering arc, to show the final position where the subtree T^* will be placed. Finally, in the lower-right screenshot of Figure 3, one can see the next tree T' generated by the algorithm.

During the visualization process, the values of the algorithm variables are also displayed inside the blue vertical window that lies beside the main visualization window. The values of these variables are shown not only for the current algorithm iteration but also for all iterations from the beginning of the algorithm execution.

5. Conclusions and Future Work

Many other algorithms included in the syllabus for our network optimization course (which is taught to second-year students at our department) can be visualized using similar techniques. These algorithms solve transportation and assignment problems; like the network simplex algorithm, they start with a feasible tree T. At each iteration they select an entering and a leaving arc. Subsequently, the leaving arc is deleted, and the subtree T^* moves from its original position and is placed under the entering arc. In the future, we intend to visualize more algorithms for the MCNFP by means of animation, using similar implementation techniques. For example, we could include an animation for new network exterior point

simplex algorithms (see Paparrizos et al. 2009). Also, in this algorithm type there are many tree modifications from iteration to iteration. It would additionally be useful to animate the capacitated network simplex algorithm, but this is complicated by the need to distinguish between arcs that are nonbasic at their upper bound from arcs that are nonbasic at their lower bound. This future improvement must be very carefully designed so that the proposed software maintains its simple and easy-to-understand look.

Several empirical studies (Lawrence et al. 1994, Byrne et al. 1999, Kehoe et al. 2001) have assessed the educational effectiveness of algorithm visualizations. Although some of these studies have yielded contradictory results, it is commonly believed among AV researchers that visualizations do help students comprehend algorithms in a more effective way. At the Department of Applied Informatics at the University of Macedonia, we have demonstrated the animation of the algorithm to a number of postgraduate students who were assigned theses related to the algorithm. The students' comments about the proposed animation were positive; most appreciated that they could choose to see either a static or an animated visualization of the algorithm. Furthermore, they remarked that smooth animation enabled them to understand the rotation of the pivot stem as well as the movement of the subtree T^*. Students mentioned that the use of colors contributed positively to their comprehension of the algorithm.

In the future, we intend to conduct a thorough empirical study to evaluate the pedagogical usefulness of our tool. While assessing the educational tool JHAVE (Naps et al. 2000), the researchers found that some students preferred a smooth animation, whereas other students preferred a static visualization. As part of our empirical assessment, we plan to pay special attention to the relative merits of dynamic and static visualization.

Supplementary Files

Files that accompany this paper can be found and downloaded from http://ite.pubs.informs.org/.

Acknowledgments

We thank the anonymous referees for some very valuable comments and suggestions that helped us to improve the quality of the paper.

References

Achatz, H., P. Kleinschmidt, K. Paparrizos. 1991. A dual forest algorithm for the assignment problem. *DIMACS Ser. Discrete Math. Theoret. Comput. Sci.* **4** 1–12.

Ahuja, R. K., T. L. Magnanti, J. B. Orlin. 1993. *Network Flows: Theory, Algorithms and Applications*. Prentice-Hall, Upper Saddle River, NJ.

Andreou, D., K. Paparrizos, N. Samaras, A. Sifaleras. 2005. Application of a new network-enabled solver for the assignment problem in computer-aided education. *J. Comput. Sci.* **1**(1) 19–23.

Baloukas, T., K. Paparrizos. 2006. A visualization software for the network simplex algorithm. *Proc. 2006 ACM Sympos. Software Visualization—Software Visualization Poster Session*, ACM, Brighton, UK, 153–154.

Baloukas, T., K. Paparrizos, A. Sifaleras. 2005. Promoting operations research education using a new Web—Accessible didactic tool. *Proc. 7th Balkan Conf. Oper. Res., Constanta, Romania*, 393–398.

Bazaraa, M. S., J. J. Jarvis, H. D. Sherali. 2005. *Linear Programming and Network Flows*, 3rd ed. John Wiley & Sons, Inc., Hoboken, NJ.

Bertsekas, D. 1998. *Network Optimization: Continuous and Discrete Models*. Athena Scientific, Belmont, MA.

Bridgeman, S., M. T. Goodrich, S. G. Kobourov, R. Tamassia. 2000. PILOT: An interactive tool for learning and grading. *ACM SIGCSE Bull.* **32**(1) 139–143.

Byrne, M. D., R. Catrambone, J. T. Stasko. 1999. Evaluating animations as student aids in learning computer algorithms. *Comput. Ed.* **33**(4) 253–278.

Chvatal, V. 1983. *Linear Programming*. W. H. Freeman, New York.

Karagiannis, P., I. Markelis, K. Paparrizos, N. Samaras, A. Sifaleras. 2006. E-learning technologies: Employing Matlab Web server to facilitate the education of mathematical programming. *Internat. J. Math. Ed. Sci. Tech.* **37**(7) 765–782.

Kehoe, C., J. Stasko, A. Taylor. 2001. Rethinking the evaluation of algorithm animations as learning aids: An observational study. *Internat. J. Human-Comput. Stud.* **54**(2) 265–284.

Khuri, S., K. Holzapfel. 2001. EVEGA: An educational visualization environment for graph algorithms. *ACM SIGCSE Bull.* **33**(3) 101–104.

Lawrence, A. W., A. M. Badre, J. T. Stasko. 1994. Empirically evaluating the use of animations to teach algorithms. *Proc. IEEE Sympos. Visual Languages, St. Louis, MO*. Institute of Electrical and Electronics Engineers, Washington, DC, 48–54.

Lazaridis, V., K. Paparrizos, N. Samaras, A. Sifaleras. 2007. Visual LinProg: A Web-based educational software for linear programming. *Comput. Appl. Engrg. Ed.* **15**(1) 1–14.

Naps, T. 1997. Algorithm visualization on the World Wide Web—The difference Java makes! *ACM SIGCSE Bull.* **29**(3) 59–61.

Naps, T., J. Eagan, L. Norton. 2000. JHAVE—An environment to actively engage students in Web-based algorithm visualizations. *ACM SIGCSE Bull.* **32**(1) 109–113.

Papamanthou, Ch., K. Paparrizos, N. Samaras. 2005. A parametric visualization software for the assignment problem. *Yugoslav J. Oper. Res.* **15**(1) 1–12.

Paparrizos, K., N. Samaras, A. Sifaleras. 2009. An exterior simplex type algorithm for the minimum cost network flow problem. *Comput. Oper. Res.* **36**(4) 1176–1190.

Rößling, G., M. Schüer, B. Freisleben. 2000. The ANIMAL algorithm animation tool. *ACM SIGCSE Bull.* **32**(3) 37–40.

Ross, R. J., C. M. Boroni, F. W. Goosey, M. Grinder, P. Wissenbach. 1997. WebLab! A universal and interactive teaching, learning and laboratory environment for the World Wide Web. *ACM SIGCSE Bull.* **29**(1) 199–203.

Scott, J. L. 2001. Education and a future for OR—A viewpoint. *J. Oper. Res. Soc.* **52**(10) 1170–1175.

Scott, J. L. 2002. Stimulating awareness of actual learning processes. *J. Oper. Res. Soc.* **53**(1) 2–10.

Standish, T. 1997. *Data Structures in Java*. Addison-Wesley, Boston.

Stasko, J. T. 1997. Using student-built algorithm animations as learning aids. *ACM SIGCSE Bull.* **29**(1) 25–29.

Stasko, J. T., J. B. Domingue, M. H. Brown, B. A. Price. 1998. *Software Visualization*. The MIT Press, Cambridge, MA.

Varkas, G., V. Kostoglou, K. Paparrizos. 2005. Broadband networks and services: Innovative means of human communication. *Proc. 4th Internat. Conf. New Horizons Indust. Ed., Corfu, Greece*, 52–57.