Journal of Computer Science 1 (1): 19-23, 2005
ISSN 1549-3636

Application of a New Network-enabled Solver for the Assignment Problem in Computer-aided Education

Andreou Dimitrios, Paparrizos Konstantinos, Samaras Nikolaos and Sifaleras Angelo
Department of Applied Informatics, University of Macedonia
156 Egnatia Str., 54006 Thessaloniki, Greece

Abstract: This study present a new learning tool developed for the visualization of an algorithm for the assignment problem. We show how the teaching of an algorithm can be greatly enhanced and the tutor's effort decreased with a visualization tool that provides an interactive and animated view of the subject being taught to the students. This tool makes use of the Java technology, therefore it is platform independed and can be used efficiently in Distance Education. The "Achatz, Kleinschmidt and Paparrizos" algorithm features significant tree modifications and furthermore, it is the first time that this visualization is being made. This tool has a friendly user interface, thus enabling the user - student to familiarize quickly. It can be used efficiently in courses like Graph Theory or Network Optimization. Benefits and drawbacks are thoroughly described in order to support the significance of this tool in computer-aided education.

Key words: Computer-Aided Education, Learning Technology, Visualization, Assignment Problem

INTRODUCTION

It is well known that teaching can be vastly amplified when it is not done purely theoretically, but visually and interactively instead. This way, students can obtain a deeper understanding of the subject being taught, rather than a merely theoretical knowledge. Algorithmic visualization is based on the basic graphic capabilities of a personal computer to demonstrate using animation of the algorithmic functionality, [1]. Teaching network algorithms is no exception and even lends itself to such approaches. Vertices and edges that compose a graph are intuitively depicted and are easily interpreted by the user - student. We present a visualization tool for teaching a specific graph algorithm [2] for the assignment problem, which provides visual and animated representation of the algorithm's execution and its details. From now on this specific algorithm, it will be referenced simply as "algorithm".

Related Work: One might ask why we implemented visualization for this specific algorithm, while there are more wide - spread algorithms for the same problem, like the Hungarian method and the primal simplex algorithm. These can be found on almost all textbooks about *Analysis of Algorithms* [3, 4]. It is very important to emphasize that, to our knowledge; this specific algorithm hasn't ever been visualized in the past.
The most difficult part of this animation is the fact that this algorithm features significant tree modifications. This difficulty is due to the complex movements of the nodes. Teaching these algorithms up to the point that the student understands their logic is only a small proposion of the effort needed to elaborately describe these algorithms detailed enough so they can actually be programmed. For once, tree structures cannot be avoided in the implementation, although that part is *not* obvious at these algorithm's general descriptions. Some general statements in these descriptions, understandable by humans, for example "*draw crosses on the matrix till all cells are spammed"*, is not at all obvious how they will be translated into such instructions, that the computer can understand.
A short description of the algorithm, for the assignment problem is discussed. The tree structures which are needed are well-defined, while statements like "*cut this part of the tree and put it there"* are also accompanied by detailed instructions, of how this can be done, so the algorithm can be programmed in straight-forward fashion.

Algorithm: An assignment problem consist of two set of nodes and one set of directed weighted edges from the first set towards the second. Given such a bipartite graph, (Fig. 1), the problem is to find a set of edges, which will contain every node of the second set *once only and* minimize (or maximize) the total weights of the edges chosen.
Note that this example defines a *sparse* assignment problem, as not all possible edges from the first set to the second exist. The problem to be solved is making the best matching of suppliers and (equal in number) demands. Each supplier makes an offer for every demand, at a specified cost. The "*best matching"* means that every demand must be met, at the lowest total cost. A specific node represents each supplier and demand. There are n supplier nodes and n demand nodes. There

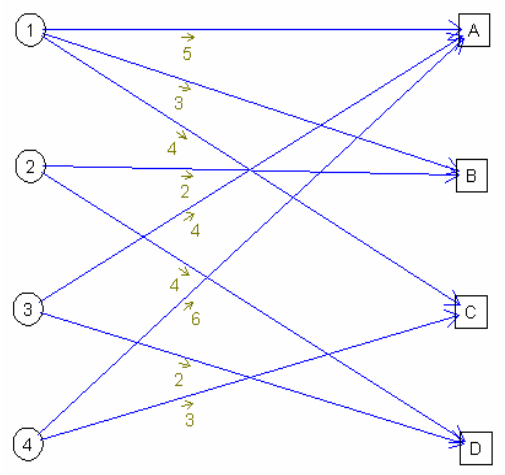

Fig. 1: A Bipartite Graph that Defines an Assignment Problem

is also a neutral node. Each supplier node has an edge *to* the neutral node and each demand node has an edge *from* the neutral. The whole combination of those is the dual tree with which the algorithm starts.

The algorithm's execution will be best understood using an example. An *n* x *n* matrix containing numbers, which correspond to the bids of the suppliers for each demand must be provided by the user. The set of suppliers' nodes at the *left* side of the tree is called *Fs* and the set of demands' nodes at the *right* side of the tree is called *Fd*. At each iteration, the minimum number from the matrix, which at the same time belongs to Fs and Fd, is picked. The tree is modified in two ways: by inserting an edge, which corresponds to the position of the number picked and removing another, according to certain principles, without producing cycles. This implies that a sub - tree changes position at every step, i.e. connects to the rest of the tree through a different connection. We will refer to this sub - tree as "*T**". Also, at every step of the algorithm, some rows and/or columns of the matrix must be updated, i.e. added with some number [5, 6].

Implementation Issues: The tool is fully implemented in *Java*, since object - orientation really lends itself for network - producing tools, like this in particular. Due to the nature of the language, Java is widely used for the implementation of algorithmic teaching purposes [7, 8]. It is far more intuitive to imagine a node which "knows" its name, position, shape, etc and an edge that knows the two nodes it connects, etc, rather than to have plain global arrays for each of these attributes.

Also, Java provides convenient drawing functions and primitives, as is Bezier curve [9], of which the usage in the tool is shown below and a multi-threaded environment with the appropriate synchronization features, which are necessary for step-by-step execution of the algorithm.

Finally, the tool, as a Java program, can run not only as an applet, but also as a stand-alone mode, for the most popular platforms like Linux, Windows and Mac, making it easily available for the students. However, it is the fact that it can be ran as an applet, that makes it possible for many people to ran it from any remote place (e.g. their home). This way any student can benefit from this tool, as long he / she has access to the Internet.

The tree's structure, its modifications, the "T*" movement and the matrix updates are the key points of the algorithm that the tutor has to point out to the students. The tool can vastly facilitate the tutor in this task. As the tool makes it very straight - forward to illustrate examples of the algorithm, we will use examples using it to show its functionality. First of all, the user has to enter the data of the problem. A 4 x 4 input example is shown in Fig. 2.

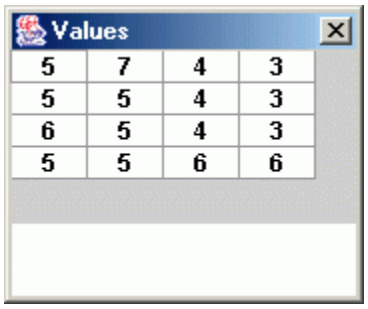

Fig. 2: A 4 x 4 Input Example

In the next step, a dual tree is being constructed, as described above. An example of an initial tree is depicted in Fig. 3.

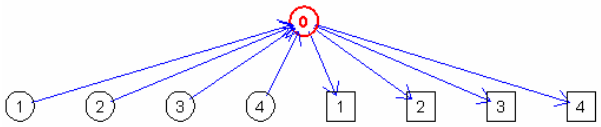

Fig. 3: A 4 x 4 Problem Initial Tree

Obviously, this is the starting tree for any 4 x 4 problem. One can easily observe the dual nature of the tree. The nodes are drawn with different shapes, so to distinguish the difference of row -and column- nodes (round and square shaped, respectively). First of all, the two sets *Fs* and *Fd* are indicated to the user using different colors, (Fig. 4).

According to the algorithm, the chosen cell of the matrix specifies a row - node and a column - node. These two nodes, say "N - start" and "N - end", define the incoming edge. An example of the above calculations follows in Fig. 5, while the incoming edge is pointed out in Fig. 6.

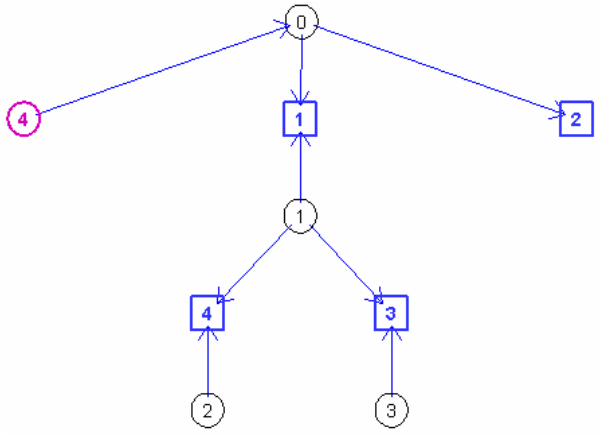

Fig. 4: The Fs (Pink) and Fd (Blue) Sets

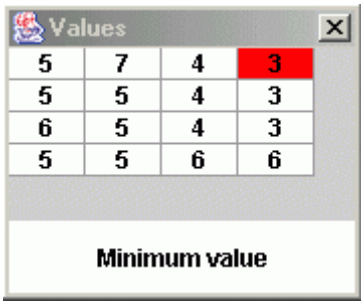

Fig. 5: The Minimum Value of the Matrix. It Defines the Edge from "1" Row-node to "4" Column-node

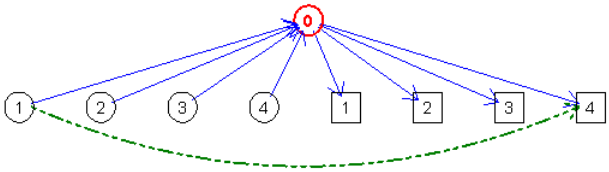

Fig. 6: The Incoming Edge as Pointed out by the Tool

The incoming edge is drawn using a *Bezier curve*, [10, 11] with end - points the positions of these two nodes and a single control point which is positioned at a small distance from the center of the "N - start" and "N - end" line. As can be easily imagined, if the incoming edge was simply drawn as a line, the depiction in many cases would be confusing, as the line could intersect many nodes before reaching "N - end". This is especially true for the starting position.

The next step is to choose the outgoing edge and also update the matrix; which means modifying the values of specific rows and / or columns by adding or subtracting a specific number. The selection of the outgoing edge depends on the in - degree of the incoming edge's destination node, as well as the matrix modification, which can occur in two different ways. In Fig. 7, the outgoing edge is depicted.

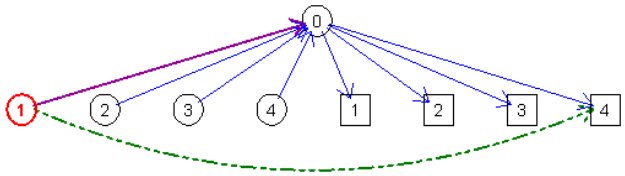

Fig. 7: The Outgoing Edge is Shown together with the Incoming One

The tool distinguishes these two cases for the student. The student also needs to understand what rows and/or columns are modified and with which row or column the modification starts. All these are highlighted effectively by the tool as depicted in the following examples. Note that for every edge between row - and column - nodes, at the corresponding cell there is no number but the symbol "[ ]" instead. A series of sequential matrix updates can be seen in Fig. 8.

Since this study is focused in the educational value of this tool, it is rather preferred not to explain thoroughly how the matrix updates at each step. After all, the user may read the pseudo code from his / her textbook. After the incoming edge has been selected and the matrix has been updated, the tool highlights a very important entity for the algorithm: The "T*" tree which will change position and possibly root and structure. An example of one possible transition that can happen follows in Fig. 9.

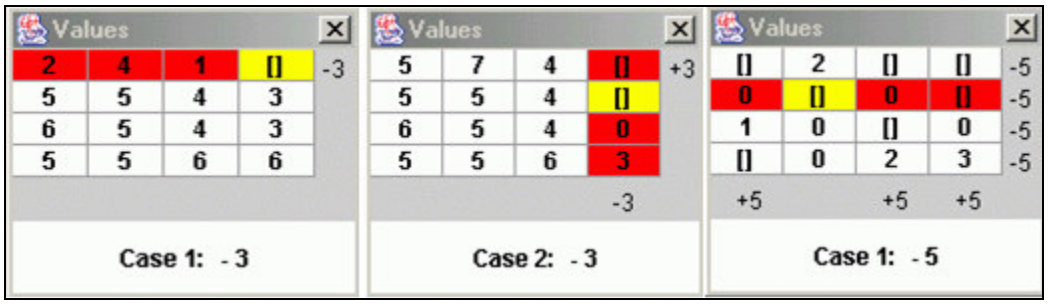

Fig. 8: A Series of Matrix Updates

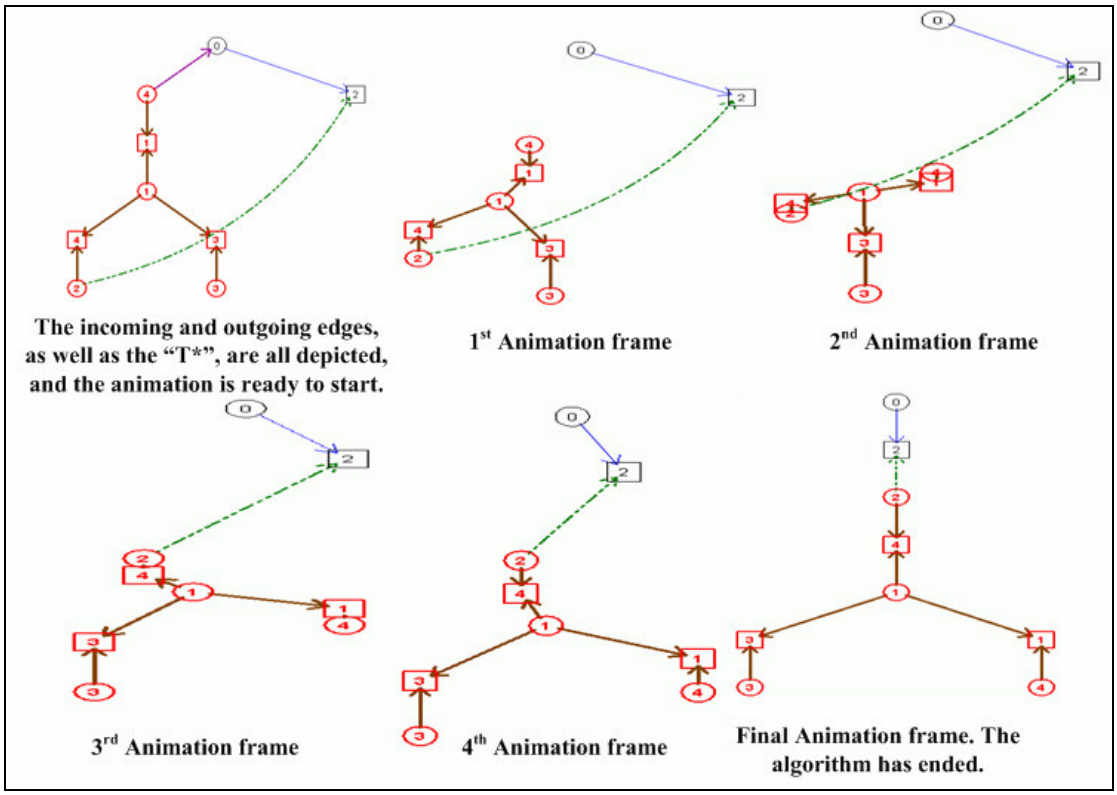

Fig. 9: A Series of Animation Frames

The tool provides a smooth view of the whole transition of the tree. We report a short explanation of how this animation works. It has been developed a simple and effective algorithm to display a tree structure (with no intersecting edge), the presentation of which is not important to the user. It is sufficient to point out that every *leaf* of the tree structure will get equal width space after this algorithm finishes. The tree drawing algorithm is used to display every tree shown in the tool. Before the modification of the dual tree, the tool calculates with this algorithm the position of every node after the change takes place. So, every node at that point has a starting point and a destination point. Twenty frames are shown, with small pauses between them and at each step every node moves the 1/20 of the distance to its destination point. At the twentieth frame, every node has gone to their final positions. This simple approach to the problem, using linear node - movement, can be used by other researchers to implement visualization support for network simplex algorithms, [12] and hopefully for an even wider spectrum.

During this animation, the "T*" may turn upside down, change root, along with the consequence (for its appearance) that these changes have. This simple animation mechanism addresses most if not all the difficulties that the tutor encounters when it comes to explain what happens when the "T*" is cut from the dual tree and reconnected to it in another place. Surely it is cumbersome enough to endeavor to show what node goes where, for every node, on a blackboard. This same approach may be also implemented in similar algorithms, which need a sub - tree movement.

Estimating the Educational Value of the Tool: As was demonstrated, the tool does not concentrate to some specific part of the algorithm, like the "T*" animation, but for completeness' sake it handles all the details of the algorithm and provides the user, with the graph and the arithmetic data of the algorithm's execution, simultaneously. So, the user does not have to be distracted or use study to make notes or calculations, as he / she can find all the relevant information of the algorithm's execution integrated in the tool.

This educational tool (applet in Java programming terms), due to the Java Technology is platform independed. Therefore every student who wishes to learn the specific algorithm may use it from his home. It doesn't matter what is his / her operating system, or which browser. It only requires that the Java Virtual Machine is installed into his / her system. Moreover, it is our belief that such applets can prove to be helpful to all the people, in general, who cannot attend a course in a University.

Should we want to find the educational value of our tool; then a good idea might be to put a questionnaire online, in the same web - page with the tool. It is very interesting to learn from the students who used it, their opinions. It is more important to find how the remote

users - students, (probably from their home), make use of this specific software, rather than just explaining to them what might be the potential benefits, [13, 14]. Other researches show how to use algorithm animations in learning environments, [15]. This is the best way for measuring the efficiency of any software dealing with Algorithm Visualization, described by [16] as a subclass of software visualization, concerned with illustrating computer algorithms in terms of their high-level operations, usually for the purpose of enhancing computer science students' understanding of the algorithms' procedural behaviour.

To end with, this technology seems to be very promishing for the Distance Learning. However, we shouldn't forget the fact that there have been conducted researches which support that pedagogical advantages can be only partially attributed to Algorithm Visualization Technology, [17].

CONCLUSION

In this study we demonstrated how specialized software can add to the learning process of graph algorithms which include difficult – to – verbalize animations and significant tree modifications. The approaches and ideas demonstrated here can also be of use and implemented in many graph algorithms.

The tool was not created merely to visually present a single algorithm; it is complete enough by including the more common (but very instructive and educational) algorithms as *Dijkstra*, *Bellman* and *Topological ordering (*for shortest path problems*), Prim*, *Kruscal (*for minimum spanning trees) and others. These were not presented here as they are common in the international bibliography. Finally, the implementation of the visualization process for different variations of the specific algorithm, is an interesting issue.

REFERENCES

1. Stasko, J. T., 1997. Using student-built algorithm animations as learning aids. ACM SIGCSE Bulletin, 29: 25-29.
2. Achatz, H., P. Kleinschmidt and K. Paparrizos, 1991. A dual forest algorithm for the assignment problem. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, 4: 1-12.
3. Aho, A.V., J.E. Hopcroft and J.D. Ullman, 1974. The design and Analysis of computer Algorithms. Addison-Wesley.
4. Cormen, T. H., C.E. Leiserson, R.L. Rivest and C. Stein, 1999. Introduction to Algorithms 2nd Ed., MIT Press.
5. Paparrizos, K., 1988. A non-dual signature method for assignment problems and a generalization of the dual simplex method for transportation problems. RAIRO-Operations Research, 22: 269-289.
6. Paparrizos, K., 1991. An infeasible (exterior point) simplex algorithm for assignment problems. Math. Programming, 51: 45-54.
7. Boroni, C. M., F.W. Goosey, M.T. Grinder and R.J. Ross, 1998. A paradigm shift! The Internet, the Web, browsers, Java and the future of computer science education. ACM SIGCSE Bulletin, 30: 145-152.
8. Lazaridis, V., N. Samaras and D. Zissopoulos, 2003. Visualization and teaching simplex algorithm. Proc. of the 3rd IEEE International Conference on Advanced Learning Technologies (ICALT03), 09/07–11/07/2003, Athens, Greece, pp: 270-271.
9. Sun Microsystems, The Java Tutorial, Stroking and Filling Graphics Primitives, http://java.sun.com/docs/books/tutorial/2d/display/strokeandfill.html
10. Swinburne University of Technology (a), Bézier curves, Melbourne, Australia. http://astronomy.swin.edu.au/~pbourke/curves/bezier/
11. Swinburne University of Technology (b), Melbourne, Australia. Piecewise Cubic Bézier Curves, http://astronomy.swin.edu.au/~pbourke/curves/bezier/cubicbezier.html
12. Ahuja, K. R., L.T. Magnanti and B.J. Orlin, 1993. Network Flows: Theory, Algorithms and Applications. Published by Prentice Hall, Englewood Cliffs, NJ.
13. Hundhausen, D. C., A.S. Douglas and T.J. Stasko, 2002. A Meta-Study of Algorithm Visualization Effectiveness. J. Visual Languages and Computing, 13: 259-290.
14. Lawrence, A. W., A.N. Badre and J.T. Stasko, 1994. Empirically Evaluating the Use of Animations to Teach Algorithms. Proc. of the IEEE Symposium on Visual Languages, St. Louis.
15. Kehoe, C., J.T. Stasko and A. Taylor, 2001. Rethinking the evaluation of algorithm animations as learning aids: an observational study. Int. J. Human-Computer Studies, 54: 265-284.
16. Price, B.A., R.M. Baecker and I.S. Small, 1993. A principled taxonomy of software visualization. J.Visual Languages and Computing, 4: 211-266.
17. Byrne, M.D., R. Catrambone and J.T. Stasko, 1999. Evaluating animations as student's aids in learning computer algorithms. Computers and Education, 33: 253–278.