Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science
ELSEVIER
Electronic Notes in Theoretical Computer Science 224 (2009) 151–158
www.elsevier.com/locate/entcs

PathFinder: A Visualization eMathTeacher for Actively Learning Dijkstra's Algorithm

M. G. Sánchez-Torrubia[1,2], C. Torres-Blanc[3] and M. A. López-Martínez[4]

Applied Mathematics Department
Universidad Politécnica de Madrid
Madrid, Spain

Abstract

PathFinder is a new eMathTeacher for actively learning Dijkstra's algorithm. In [12] the concept of eMathTeacher was defined and the minimum as well as some additional requirements were described. The tool presented here is an enhanced paradigm of this new concept on Computer Aided Instruction (CAI) resources: an application designed following the eMathTeacher philosophy for active eLearning. The highlighting new feature provided by this application is an animated algorithm visualization panel showing, on the code, the current step the student is executing and/or where there is a user's mistake within the algorithm running. PathFinder also includes another two interesting new features: an active framework area for the algorithm data and the capability of saving/retrieving the created graph.

Keywords: eMathTeacher, eLearning, Active learning, Constructive learning, Interactive Java applications, Computer Assisted Instruction (CAI).

1 Introduction and Preliminaries

Graphical and dynamic web based tools are more appealing for students than traditional learning materials. It has been confirmed that learners spend much more study time when visualization is involved; however, there has been some skepticism about the real value of visualizations as a pedagogical tool. Many educators think that visual tools enhance their lectures and significantly increase student's comprehension, but such tools are of little effectiveness when students are not actively engaged in the learning process [6]. Furthermore, when students are not required to wonder about the concepts, to provide answers and to predict what is happening

[1] This work has been partially supported by UPM (Spain) under Project No. IE07 1010-029.
[2] Email: gsanchez@fi.upm.es
[3] Email: ctorres@fi.upm.es
[4] Email: lopez.martinez.ma@gmail.com

doi:10.1016/j.entcs.2008.12.059

next, they might adopt a passive attitude that is not beneficial at all, and may even be harmful for their training. The analysis presented by Hundhausen et al. [3] asserts that "*how* students use AV technology, rather than *what* students see, appears to have the greatest impact on educational effectiveness" and their study "suggests that the most successful educational uses of AV technology are those in which the technology is used as a vehicle for actively engaging students in the process of learning algorithms". They concluded that, those who are actively engaged with the visualization have consistently outperformed the other ones who passively viewed them. Thus, in order to avoid a passive attitude, during the execution, the program should interact continuously with the users, forcing them to predict the following step.

In [1], the authors state that a "consciously designed approach informed by a constructivist view holds the most potential for effective online learning designs"; and that "learners must construct their understanding through an active process building on past experiences and knowledge and that knowledge cannot be simply accepted from others". According to this opinion, we believe that active learning is the only effective way of acquiring knowledge and that students cannot only look how the processes evolves, but they must get involved in the process itself.

The challenge of discovering new ways to motivate students in active learning encouraged us to develop a new kind of web based tools. Our main goal has been to get students involved, as actively as possible, in their learning process. With this objective in mind, we have been developing several interactive Java applets, that allow visualized execution of algorithms ([13], [10] & [14]). Those applets have been designed under the eMathTeacher philosophy and are being used as complementary material for blended learning (bLearning) [13] both for teachers on classroom lectures and students when learning by themselves. This way, the power and effectiveness of face to face teaching are boosted with the flexibility and technical capabilities of eLearning, turning out the students into the protagonists of their own learning progression.

In the next sections, we review the definition of eMathTeacher as well as its main requirements. This kind of application introduces a new concept in computer aided education as they can act as genuine virtual trainers extending the teacher's hand through the Web.

1.1 eMathTeacher Definition

An eLearning tool is eMathTeacher compliant [12] if it works as a virtual maths trainer. In other words: if it is an on-line self-assessment tool that help users to actively learn math concepts or algorithms by themselves, correcting their mistakes and providing them with clues to find the right solution.

They can also be applied as bLearning complementary material for being used both by teachers on classroom lectures and by students when learning maths by themselves. However, the most important feature of these tools is the feasibility of being used for practicing with maths methods or algorithms while the system guides the user towards the right answer.

1.2 Minimum requirements for an eMathTeacher

These, as described in [12], are the minimum conditions we establish a tool must fulfil to be considered an eMathTeacher :

- Step by step inquiring: for every process step, the student should provide the solution while the application waits in a stand by mode, expecting the user's input.
- Step by step evaluation: just after the user's entry, the eMathTeacher evaluates it, providing a tip for finding the proper answer if it is wrong or executing it if ok.
- Visualization of every step change that happens.
- Easy to use.
- Flexible and reliable: allowing the user to introduce and modify the example and to repeat the process if desired.
- Clear presentation within a nice and friendly graphic environment, helping insight.
- Platform independency and continuous availability (anytime, anywhere).

1.3 Additional requirements for an eMathTeacher

The requirements listed above are mandatory for an application to be considered an eMathTeacher. In addition, there are some other desirable conditions, containing those described in [14], that these tools should meet. The main features to be included are:

- Algorithm visualization panel.
- Framework panel showing the current state of the algorithm data structures. It should also allow the user to update those structures.
- Samples library and/or saving/retrieving capabilities. This requirement should include saving, for later analysis, both the basic structure the user is working with, and also the user's interaction history.
- Language menu.
- Integration of different tools as a complete suite, able to cover the whole topic.
- Automatic execution process (optional), especially designed for very complex exercises.
- Capability to find and show alternative solutions once the problem has been solved.
- A theoretical introductory part.
- No installation or maintenance tasks required and light downloading weight.

Other authors (see e.g. [4], [5], [9], [6] & [7]) have already highlighted most of the above listed features as being required for a learning tool to be effective. In [11], we perform a literature search and study, which allowed us to identify a

number of systems designed under Java Technology and oriented to help students on learning different Computer Science topics (e.g. IDSV, JHAVÉ, TRAKLA2, JFLAP or AulaWeb Self-Assessment Module). Though those tools seem similar in terms of functionality, there is actually a feature that makes eMathTeachers (i.e. eLearning tools that are eMathTeacher compliant) different: they only execute the current step in case the input is ok and return a customized error message, providing a tip for finding the proper answer, otherwise. This feature is specially meant for (our) novice students whose background on algorithms, data structures and programming is virtually nil in most cases. Thus, these kind of tools are aimed at helping them to learn what an algorithm is and how an algorithm works, which means to understand, for instance, how to execute an *if... then* sentence. The unique characteristic mentioned above provides eMathTeachers with full interactive learning capabilities, and distinguishes them from other systems so far.

2 PathFinder: an eMathTeacher for Dijkstra's algorithm

Dijkstra's algorithm [2], commonly known as *shortest path algorithm*, solves the problem of identifying the shortest path in a weighted graph from an initial vertex to the other vertices. The central idea behind the algorithm is that each subpath of the minimal path is also a minimum cost path.

Keeping the features described in the preliminaries, we have designed and implemented an application, available at `http://www.dma.fi.upm.es/java/matematicadiscreta/dijkstra`, (see Figure 1) for active learning of Dijkstra's algorithm, including both *with* and *without final node* possibilities. Our PathFinder has been designed under the eMathTeacher philosophy (section 1.1) and meets all the minimum requirements listed in section 1.2 as well as nearly all the additional ones (section 1.3).

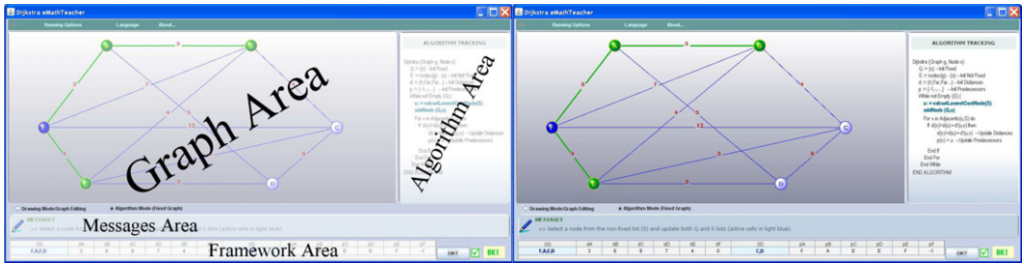

Fig. 1. PathFinder: an eMathTeacher for Dijkstra's algorithm

2.1 Detailed description

The application runs in a Java Web Start window. The window is split into four areas: graph, algorithm, messages and framework areas (see Figure 1), where **graph** and framework panels are the main working areas. The first one is a panel where the graph is displayed and allows the user to create and edit nodes and weighted

edges. The second one is the **framework** area. A table, presenting the current state of the algorithm structures (fixed and unfixed nodes, distances to the initial node and predecessors list), is displayed in this panel.

The **algorithm** area displays the execution code, showing in blue the current step or in red the point where the user's mistake is located, while the **message** panel provides clues to find the right solution or indicates the next step to be done. This panel also offers useful hints when the graph is being edited.

The menu bar presents a graph saving/retrieving option, three execution options and a language selector, currently implemented in Spanish and English.

2.2 Editing the graph

The graph nodes are drawn by left clicking the mouse, and the edges by right dragging between two nodes. When an edge is being created, a text cell appears in the message panel for entering the edge's weight. The nodes can be moved or deleted at any time and the edges can be erased or their weigh modified. The first created node is predefined as the initial node (e.g. A in Figure 2) but the user has also the possibility of changing the initial node and/or defining a final node (e.g. D in Figure 2).

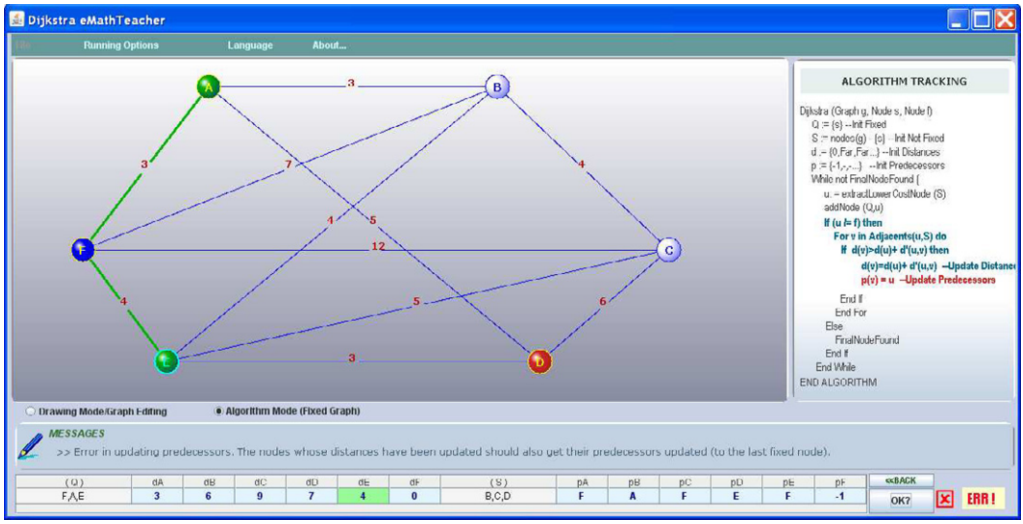

Fig. 2. The PathFinder showing an error on predecessors update

2.3 Executing the algorithm

Once the graph has been introduced and the algorithm mode has been selected, the application checks whether it can be executed or not (as stated in [2] "we restrict ourselves to the case where at least one path exists between any two nodes", i.e. to connected graphs). If the graph is not connected and thus the algorithm cannot be executed, an error message indicating this fact is displayed, otherwise the algorithm starts.

The execution has three performance options: two modes of interactive running (step by step and iteration verification) and an option of direct execution, able to directly provide the final result (restricted to help on verifying results or to simplify the flow process in case of very complex exercises).

In **step by step verification** option, for every algorithm iteration, the user must update first the fixed and unfixed nodes, then check whether it is correct or not and finally update distances and predecessors and check again the input correctness. **Iteration verification** option is aimed at more advanced students as the user should perform a whole iteration before checking the input correctness. For every verification (in both interactive running modes), when any of the updates is not right, the message panel shows the event: an error message pointing out the problem and providing hints for rectifying is displayed. Simultaneously, in the algorithm panel, the step where the mistake is located changes into red (see Figure 2). If all the updates are right, the application changes the node and/or edge's color and waits for the next user's entry.

2.4 Enhancing the previous eMathTeachers

While using the previous eMathTeachers as complementary material for bLearning, we have realized the necessity of implementing an algorithm visualization panel inside the tool for enhancing the student's consciousness of each of the algorithm steps. Also, while encouraging them to write down the algorithm structures, we found the necessity of including those structures inside the tools. As a consequence, when designing PathFinder, we incorporated several new features that, in our opinion, entail huge pedagogical enhancement:

- The algorithm visualization panel (see Algorithm area in Figure 1) provides a better understanding of the algorithm execution code.
- The framework panel (see Framework area in Figure 1) allows users to practice every algorithm step exactly as if they were implementing it *by hand*, but including the graphic panel as well as the feasibility of being corrected and advised by the tool when a mistake happens (this is the main feature of eMathTeacher philosophy: acting as a virtual maths teacher).
- The retrieval option offers the instructor the possibility of preparing selected exercises for the students, and gives the learner the advantage of saving the graph for later review.

In [3], the authors assert that "AV technology has been successfully used to actively engage students in such activities as what-if analysis of algorithmic behavior, prediction exercises and programming exercises". The eMathTeacher philosophy has been mainly inspired by getting the students involved in the two first activities: what-if analysis of algorithmic behavior (what means understanding the algorithm design in-depth) as well as predicting the algorithm outcomes (that entails a good understanding of the algorithm process). In our opinion, as a consequence of the above described improvements, PathFinder will get the users even more engaged in the two first activities, obtaining as a result an active learning of this algorithm and

thus, hopefully, successful educational results.

3 Conclusions and next steps

In this paper, PathFinder, a new tool for actively learning Dijkstra's algorithm, has been presented. Moreover, some new additional requirements for eMathTeacher tools have been introduced. PathFinder is an enhanced paradigm of this new concept on CAI resources. The highlighting new feature provided by this application is an animated algorithm visualization panel. It shows, on the code, the current step the student is executing and also where there is a user's mistake within the algorithm running. Other important new attribute is the active framework area for the algorithm data, designed for encouraging students to active learn the algorithm process, as implemented *by hand*, while the application verifies the correctness of each user's input. This way, novice students must study and understand the algorithm in advance, or look it over while they are practising, as it is nearly impossible to happen to fill in the structures panel correctly. Finally, the application runs in a Java Web Start window and this technology allows it to read and write in the client's hard disk, which gives us the feasibility of saving and retrieving graphs. This way, the instructor can load an examples library and the students can save the edited graphs for later revision or modification.

PathFinder has recently been finished, which means that it has not been used by any students yet. Though there are not impact evaluations of it, based on our previous research, we have high expectations regarding its potential effectiveness on helping learning activities. As part of that research, the previous graph eMathTeachers impact has already been evaluated by comparing the rates obtained on the graphs exercise in a Discrete Mathematics final exam. As detailed in [12], the results showed a deeper understanding of algorithms process in the study group, even considering that those tools offered neither the algorithm visualization panel nor the framework panel.

In the near future, we aim to perform a tool effectiveness evaluation, following the models proposed in ([6] & [8]), as well as measuring its impact on the students' learning. We are also preparing an users survey (covering both students and teachers population) added to a collection of opinions, suggestions for improvement, etc.

Currently, we are designing and developing a whole suite which actually integrates different types of graphs and graph algorithms, as well as several correction levels for each algorithm simulation, fitted in it. The design includes all four panels described in the PathFinder and will feature the above mentioned functions together with the possibility of saving the user's interaction history for later analysis and/or (automatic) assessment.

References

[1] Clear, T., A. Haataja, J. Meyer, J. Suhonen and S. A. Varden, *Dimensions of distance learning for computing education*, SIGCSE Bull. **33** (2001), pp. 101–110.

[2] Dijkstra, E., *A note on two problems in connexion with graphs*, Numerische Mathematik **1** (1959), pp. 269–271.

[3] Hundhausen, C. D., S. A. Douglas and J. T. Stasko, *A meta-study of algorithm visualization effectiveness*, J. Visual Languages & Computing **13** (2002), pp. 259–290.

[4] Jarc, D. J., M. B. Feldman and R. S. Heller, *Assessing the benefits of interactive prediction using web-based algorithm animation courseware*, SIGCSE Bull. **32** (2000), pp. 377–381.

[5] Korhonen, A. and L. Malmi, *Algorithm simulation with automatic assessment*, SIGCSE Bull. **32** (2000), pp. 160–163.

[6] Naps, T., G. Rößling, V. Almstrum, W. Dann, R. Fleischer, C. Hundhausen, A. Korhonen, L. Malmi, M. McNally, S. Rodger and J. Á. Velázquez-Iturbide, *Exploring the Role of Visualization and Engagement in Computer Science Education*, SIGCSE Bull. **35(2)** (2003), pp. 131–152.

[7] Naps, T., G. Rößling, J. Anderson, S. Cooper, W. Dann, R. Fleischer, B. Koldehofe, A. Korhonen, M. Kuittinen, C. Leska, L. Malmi, M. McNally, J. Rantakokko and R. J. Ross, *Evaluating the educational impact of visualization*, SIGCSE Bull. **35(4)** (2003), pp. 124–136.

[8] Reeves, T. C. and J. G. Hedberg, "Interacive Learning Systems Evaluation," Educational Technology Publication. Englewood Cliffs, 2003.

[9] Rößling, G. and T. L. Naps, *A testbed for pedagogical requirements in algorithm visualizations*, SIGCSE Bull. **34** (2002), pp. 96–100.

[10] Sánchez-Torrubia, M. G. and S. Gutiérrez-Revenga, *Tutorial interactivo para la enseñanza y el aprendizaje de los algoritmos de búsqueda en anchura y en profundidad*, in: *Proceedings of the XII J. de Enseñanza Universitaria de la Informática*, 2006, pp. 573–580.

[11] Sánchez-Torrubia, M. G. and C. Torres-Blanc, *eMathTeacher: Activating students attitude and engagement*, International Journal on Engineering Education (to appear).

[12] Sánchez-Torrubia, M. G., C. Torres-Blanc and J. B. Castellanos, *Defining eMathTeacher tools and comparing them with e&bLearning web based tools*, in: *Proceedings of the International Conference on Engineering and Mathematics (ENMA)*, 2007.

[13] Sánchez-Torrubia, M. G., C. Torres-Blanc and V. Giménez-Martínez, *An eMathTeacher tool for active learning Fleury's algorithm*, International Journal Information Technologies and Knowledge (IJ ITK) **2** (2008), pp. 437–442.

[14] Sánchez-Torrubia, M. G., C. Torres-Blanc and S. Krishnankutty, *Mamdani's fuzzy inference eMathTeacher: a tutorial for active learning*, WSEAS Transactions on Computers **7** (2008), pp. 363–374.