PILOT: An Interactive Tool for Learning and Grading

Stina Bridgeman	Michael T. Goodrich	Stephen G. Kobourov	Roberto Tamassia
Brown University	Johns Hopkins University	Johns Hopkins University	Brown University
ssb@cs.brown.edu	goodrich@jhu.edu	kobourov@cs.jhu.edu	rt@cs.brown.edu

Abstract

We describe a Web-based interactive tool, called PILOT, for testing computer science concepts. The strengths of our system are its universal access and platform independence, its ability to test algorithmic concepts, its support for graph generation and layout, its automated grading mechanism, and its ability to award partial credit to proposed solutions.

1 Introduction

Interactive World Wide Web (WWW)-based learning tools have become the focus of research for a great deal of computer science educators [4, 5]. Interaction and animation in and out of the classroom offer the chance to actively engage students in the learning process. Several interactive educational tools have been developed over the last few years. Many of these, however, quickly become obsolete as hardware/software platforms and operating systems change. With the advent of platform-independent applications, there are far greater possibilities for creating more useful educational tools. While many computer science courses offer online access to handouts, syllabi, homework assignments solutions and other static documents, only a few have begun to exploit the full potential of the new technology available to us.

Online testing systems can be useful in distance learning, virtual universities, and online classes, and several systems that allow for online testing have been developed in the last decade (e.g., see [7]). Such systems tend to support multiple-choice questions, which provide a natural class of questions that can be automatically graded online. While such questions can be used to provide useful measures of student learning, we believe there are significant additional learning and testing opportunities available that have yet to be fully exploited. In particular, because of the ability to formally define input and output specifications, there are other more complex questions that should also allow for automatic online grading, at least in theory. Some of the immediate advantages of online grading for richer sets of questions are the ability to test students' answer *creation* abilities rather than simply their answer *choosing* abilities. In addition, online grading also provides fast and consistent grading, provably correct solutions, and pointers to information relevant to the question. Moreover, online grading also provides the possibility of assigning different questions to the students, thus reducing issues of cheating and plagiarism.

We are therefore interested in interactive online automated grading tools that aid student learning and test answer creation abilities, not just answer choosing skills. In addition, we are interested in tools can be used in class to visualize complex problems and concepts.

1.1 Previous Work

Several previous software systems have been designed with online testing in mind [10, 11]. In addition, systems, such as QUIZIT [14], WebCT[1], and ASSYST [9], have been designed to perform online testing of answers whose correct syntax can be specified as regular expressions. Previous systems that allow for richer types of answers have needed assistance from the course graders and the instructor to perform the actual answer checking. In addition to the difficulty of dealing with sophisticated forms of answers, another area where these previous systems have trouble is in their lack of ability to provide partial credit to answers that are "almost" correct.

Since our notion of answer specification and checking involves a strong visualization component, it is also related to previous work on the visualization of algorithms and data structures. Algorithm animation has been successfully used for visualizing graph algorithms, sorting, and searching, to name a few examples [13]. Similarly, program code animation also helps in the learning of new programming languages. Finally, concept animation has also been successful in in communicating difficult concepts such as finite state automata [4]. Tools

[1] www.webCT.com

for creating animations of data structures and algorithms have also been developed [12]. Interactive tutorials have been designed and their positive impact on student learning evaluated [2]. Hypertextbooks have been proposed and developed, in which hypertext, interactive animations, audio and video parts are integrated in a web-based standalone educational resource [3]. Thus, there is a rich literature that describes systems and benefits of concept visualization in education settings.

1.2 Our Results

We have designed a Platform-Independent Learning Online Tool, `PILOT`, with several goals in mind. First of all we would like to offer an interactive tool that can be used in class to aid in exposition. We would like the students to be able to actively interact with `PILOT` and gain something from the interaction. Furthermore, there are numerous problems that students learn best by examples, and we would like a tool that can generate random instances of a problem and allow the student to create the solution online. Finally, we would like to allow for automated grading so that the student can receive immediate feedback on her work. Thus, `PILOT` allows for:

- WWW access and platform independence
- generation of random instances of a problem
- user interaction to specify a solution
- online submission of solutions for evaluation
- evaluation of solutions, providing a score and comments
- generation of correct solutions to the problem

At this time, `PILOT` supports graph problems such as finding the minimum spanning tree (MST), tree search algorithms (breadth first search (BFS) and depth first search (DFS)), and shortest path algorithms.

One of the main advantages of `PILOT` is that it is a platform independent client/server based applet that can be run from a browser such as Netscape or Internet Explorer. Another even more important feature is its capability to successfully interact with the student. Getting any feedback for a given solution is a good idea, but it is even better if the feedback is more specific. For example, in creating a MST, if an edge is chosen incorrectly, `PILOT` will highlight the edge and suggest how to correct it. Grading automation also has great potential especially in graph-based problems, tree-traversals, sorting and searching problems.

With additional security, `PILOT` can be used for grading by allowing graders to input both problems and students' solutions. (In the current system, `PILOT` can only be used to check problems generated on the spot.) Such security could take the form of password protection or encryption, to allow only authorized users to connect. Additional security would also allow the use of `PILOT` in testing situations, where it is important to ensure that each student only submits one version of the answer.

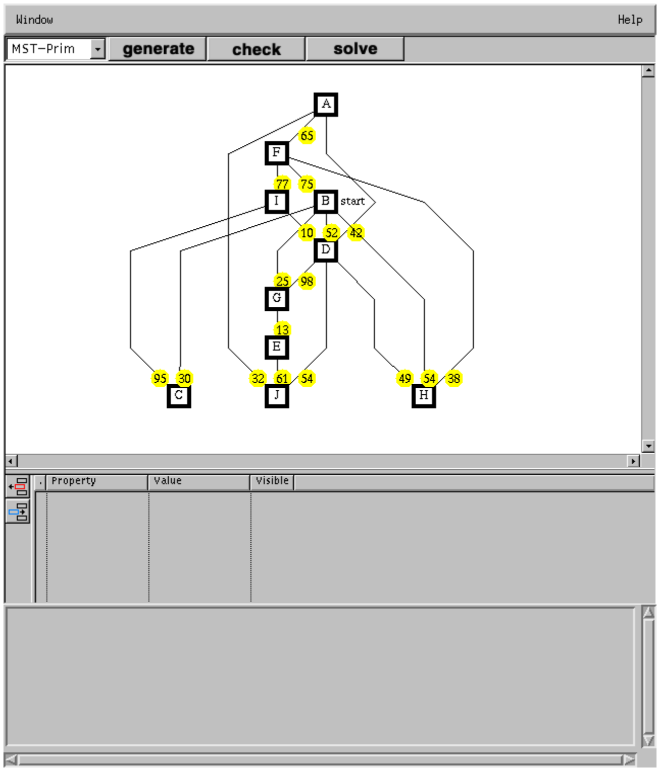
2 Using `PILOT`

In the current scenario, the user chooses a problem type from a pull-down menu and clicks the "generate" button to generate a random instance of that problem. Figure 1(a) shows the result of generating an instance of MST-Prim — a minimum spanning tree problem to be solved using Prim's algorithm. `PILOT` easily allows testing of both general concepts ("find a minimum spanning tree of the given graph") and specific algorithms ("find a minimum spanning tree of the given graph, using Prim's algorithm"). For MST-Prim, the user is to execute Prim's algorithm, starting with the vertex marked "start"; the solution is a numbering of the edges in the order in which they were added to the MST. To indicate the solution, the user clicks on the edges belonging to the MST. The order can be adjusted in the "Edge Ordering" window; by default, the edges are listed in the order in which they are selected.

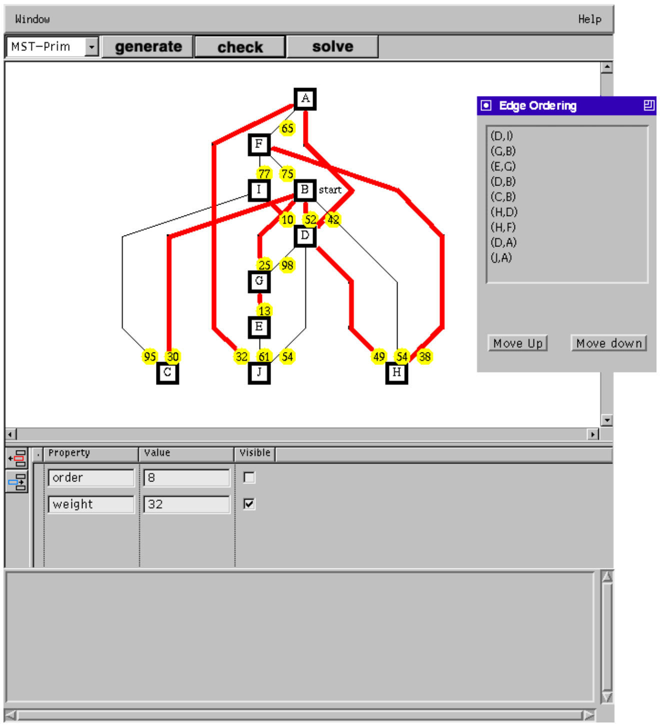
Once the user is satisfied that she has entered the correct solution (Figure 1(b)), clicking the "check" button will correct and grade the solution. The system will display the graph with the incorrect edges highlighted, along with a score and an explanation of the errors made; see Figure 1(c). Note that the actual solution is not displayed — this is because the checker may not actually compute the solution in the process of grading the user's input. A solution can be obtained at any point by clicking the "solve" button; see Figure 1(d).

3 `PILOT` Architecture

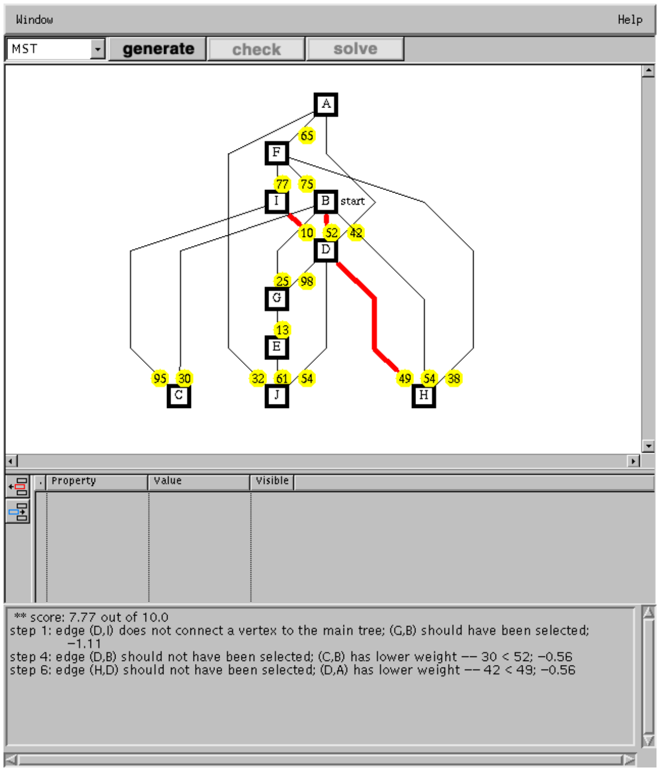
`PILOT` uses a client-server architecture, and is built on top of GeomNet [1]. In the GeomNet model, the client is responsible for maintaining the user interface and all of the algorithm-related computation is done on the server. For `PILOT`, the client is implemented as a Java applet and the server side contains the problem generators, checkers, and solvers, currently also implemented in Java. The main motivation for choosing the client-server architecture was flexibility — the server is not constrained by the security restrictions placed on applets and is not limited to running Java programs, making it possible to take advantage of existing tools. The graph generator, for example, uses the Graph Drawing



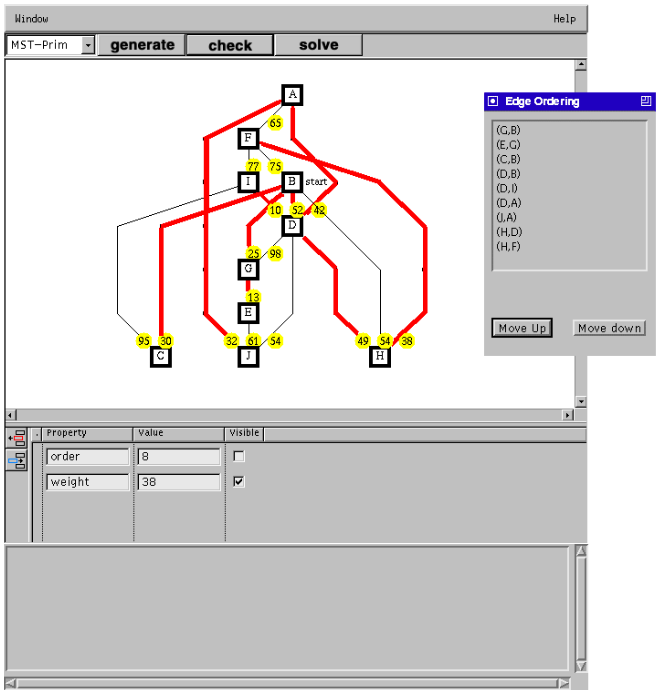
(a) random instance of MST-Prim



(b) user's solution



(c) automatically corrected solution, with incorrect edges highlighted



(d) system-generated solution

Figure 1: Example of user interaction with `PILOT`.

Server [6] component of GeomNet to compute a layout for the automatically generated graphs. The modularity of the GeomNet system also makes it easy to add new components — both interfaces and problem generators/checkers/solvers — to `PILOT`.

We now look at the graph generator and problem checker components of `PILOT` in more detail, focusing on minimum spanning tree problems as an example; the problem solvers are straight-forward implementations of the appropriate problem solving algorithms and are not considered further.

3.1 Graph Generator

The graph generator uses a method similar to that used in [8] to generate "realistic" graphs for experimental purposes. Graphs are built from a single vertex by repeatedly applying three operations — insertion of a vertex and a random number of adjacent edges, insertion of an edge between two existing vertices, and splitting of an existing edge by replacing it with a new vertex and two new edges. Graph properties such as the density can be controlled by adjusting probabilities assigned to each of the operations and the degree of newly inserted vertices.

3.2 Problem Checkers

There are four main challenges in designing problem checkers: determining what constitutes a solution, handling non-unique solutions, assigning appropriate partial credit, and returning meaningful comments.

The format of the solution fundamentally affects the structure of the checker. For example, the MST problem simply tests whether or not the user can construct a minimum spanning tree, and so the solution is a list of the edges belonging to the MST. The checker simply verifies whether or not the right edges were selected. In MST-Prim, the goal is to test the user's knowledge of a specific algorithm and so more information is needed in the solution. In this case, the order in which the edges are added to the MST is sufficient to verify that the user executed the algorithm correctly, and the checker must check this order.

The last three problems are related. It is relatively easy to compute a solution and compare the user's input to it, simply returning "correct" or "incorrect" (or "full credit"/"no credit"). However, this unfairly penalizes a student who understands the concept but makes a small mistake, and is of limited use to a student who is trying to master a concept. More appropriate responses for MST problems, for example, would be something like "Edge (a,c) should be replaced by the lower-weight edge (a,b)" and a 1-point penalty for each incorrect edge. The solve-and-compare approach also runs into problems when the solution is not unique, since the user may have a correct solution but be marked wrong because the system generated a different one. Non-unique solutions can easily occur in MST problems when multiple edges have the same weight.

One approach is to verify properties of the user's solution, to ensure that it is valid. This is the approach taken in the MST checker — for each edge in the MST, that edge should be the lowest-weight edge of any connecting the two vertex partitions created by the removal of the edge from the spanning tree. Each time an edge violates this property, it is marked incorrect and the appropriate replacement edge can be indicated to the user. Partial credit can be assigned according to the number of incorrect edges. (If the user's input is not a spanning tree, cycles are broken by removing the highest-weight edge in the cycle and trees are joined by adding the lowest-weight edge between the trees. The checker then proceeds with the spanning tree produced, adding an additional penalty for non-tree input.)

This approach partially addresses the problem of meaningful comments and partial credit, but is not appropriate for problems where an early mistake can be compounded. For example, if the user chooses the wrong edge in the first step of Prim's algorithm but otherwise executes the algorithm properly, the one mistake may cause several other edges to be selected incorrectly. It is unfair to penalize the user for every edge that is wrong since it was actually only one mistake, and the system's comments may be similarly misleading. A checker can solve this problem by taking an incremental approach and stepping through the solution of the problem, taking into account the user's choices as they happen. The MST-Prim checker considers the user's edges in order, testing each edge to determine if it is valid as the next choice. An edge is valid if it connects a new vertex to the spanning-tree-in-progress and has the lowest weight of any edge connecting a new vertex to the tree. A penalty is assessed if the user's edge is not valid, with a higher penalty if the edge does not connect to the tree. The internal data structures are then updated to include the new edge, and the checker continues with the next edge.

4 Future Work

The current `PILOT` system can be extended in many ways. Of particular use in a teaching tool would be to allow greater interactivity — as the user works through the problem, the system can immediately provide feedback as to whether or not the user is doing the right thing. Also, if the user is unsure of what step to take next, the system can provide hints or outright statements about what to do.

Another issue is the generation of problems of approximately equal difficulty (and, related to this, the generation of appropriate special cases). This is particularly relevant if `PILOT` is used in a testing situation, since it is undesirable for one student to get an easy case when another is faced with a much harder example. Dealing with this involves looking more carefully at the properties of the graphs produced by the graph generator.

Problem checkers can pose challenging problems of their own. The issues are the same as those mentioned in Section 3.2 — determining an appropriate format for the solution and handling partial credit and comments. Partial credit is one of the most "human" tasks of grading, and one that is very subjective, and so determining appropriate ways to handle it automatically is an important task. Implementing checkers to assign partial credit can be significantly harder than the corresponding problem solvers.

Finally, `PILOT` can be extended to handle additional problem types and algorithms. The mechanism for doing this is straightforward — many other graph problems, such as maximum flow, can be supported by the current interface so all that is required are additional checkers and solvers. Adding new problem types, such as sorting, requires more work to create a new interface in addition to generators/checkers/solvers. In both cases, however, the server remains the same so adding new components is only a matter of plugging in a new front- or back-end.

References

[1] Barequet, G., Bridgeman, S. S., Duncan, C. A., Goodrich, M. T., and Tamassia, R. Geometric computing over the Internet. *IEEE Internet Computing 3*, 2 (March/April 1999), 21–29.

[2] Barnett, L., Casp, J., Green, D., and Kent, J. Design and implementation of an interactive tutorial framework. In *Proceedings of the 29th SIGCSE Technical Symposium* (1998), pp. 87–91.

[3] Boroni, C., Goosey, F., Grinder, M., Lambert, J., and R.Ross. Tying it all together creating self-contained, animated, interactive, web-based resources for computer science education. In *Proceedings of the 30th SIGCSE Technical Symposium* (1999), pp. 7–11.

[4] Boroni, C., Goosey, F., Grinder, M., and R.Ross. Weblab! A universal and interactive teaching, learning, and laboratory environment for the World Wide Web. In *Proceedings of the 28th SIGCSE Technical Symposium* (1997), pp. 199–203.

[5] Boroni, C., Goosey, F., Grinder, M., and R.Ross. A paradigm shift! The Internet, the Web, browsers, Java, and the future of computer science education. In *Proceedings of the 30th SIGCSE Technical Symposium* (1999), pp. 145–152.

[6] Bridgeman, S., Garg, A., and Tamassia, R. A graph drawing and translation service on the WWW. *Internat. J. Comput. Geom. Appl.*. to appear.

[7] Carrasquel, J. Teaching CS1 on-line: the good, the bad, and the ugly. In *Proceedings of the 30th SIGCSE Technical Symposium* (1999), pp. 212–216.

[8] Di Battista, G., Garg, A., Liotta, G., Tamassia, R., Tassinari, E., and Vargiu, F. An experimental comparison of four graph drawing algorithms. *Comput. Geom. Theory Appl. 7* (1997), 303–325.

[9] Jackson, D., and Usher, M. Grading student programs using ASSYST. In *Proceedings of the 28th SIGCSE Technical Symposium* (1997), pp. 335–339.

[10] Mason, D., and Woit, D. Integrating technology into computer science examinations. In *Proceedings of the 29th SIGCSE Technical Symposium* (1998), pp. 140–144.

[11] Mason, D., and Woit, D. Providing mark-up and feedback to students with online marking. In *Proceedings of the 30th SIGCSE Technical Symposium* (1999), pp. 3–6.

[12] Pierson, W., and Rodger, S. Web-based animation of data structures using JAWAA. In *Proceedings of the 29th SIGCSE Technical Symposium* (1998), pp. 267–271.

[13] Stasko, J., Domingue, J., Brown, M. H., and Price, B. A., Eds. *Software Visualization: Programming as a Multimedia Experience*. MIT Press, 1998.

[14] Tinoco, L., Fox, E., and Barnette, D. Online evaluation in WWW-based courseware. In *Proceedings of the 28th SIGCSE Technical Symposium* (1997), pp. 194–198.