PILOT: An Interactive Tool for Learning and Grading Senior Thesis, Presented May 1, 2000[*]

Ryan Shaun Baker
Brown University
rsb@cs.brown.edu

Abstract

We describe an interactive system, called PILOT, for testing computer science concepts. The strengths of PILOT are its use as an algorithm visualization tool, 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 and give meaningful feedback where appropriate. We also discuss issues involved in its use in the classroom, and some empirical results we have obtained through that usage.

1 Introduction

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 [9]). 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. Through the use of some of the technologies of Intelligent Tutoring Systems (ITS) and Algorithm Animation, such systems can be made considerably more useful and powerful. In particular, we believe that these systems can be extended to include more sophsticated questions such as "execute-this-algorithm" questions, a type of homework problem assigned many lower-level Computer Science classes, and that we can, for this sort of problem, 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 makes it feasible to assign different problems to different students and still grade each problem efficiently, thus reducing issues of cheating and plagiarism. It also allows students to practice with a virtually unlimited number of different problems, far more than a teaching assistant staff could conveniently create. Algorithm Animation can be used to provide support for students whose learning styles tend towards visual learning, and to provide students with a model of a correct answer process.

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 the visualization of questions, errors, and answers.

My specific contributions and enhancements to the version of PILOT discussed in the earlier conference paper [8] are as follows:

1. The addition of interactive feedback while the student is learning the algorithm
2. The use of algorithm animation techniques for several purposes, including demonstrating the correct functioning of an algorithm and demonstrating to a student how their answer was incorrect both as the student is learning the algorithm and during grading.
3. The preparation of PILOT for its first use in CS016, including security issues
4. Conducting the empirical research discussed below, including designing the problems used in both PILOT and the traditional homework, implementing the logging features within PILOT to gather the other data, and statistical analysis.

[*]Research supported in part by the National Science Foundation under grants CCR-9732327 and CDA-9703080, and by the U.S. Army Research Office under grant DAAH04-96-1-0013.

1.1 Previous Work

Several previous software systems have been designed with online testing in mind [15, 16]. Blackboard.com [4] provides automatic grading for quizzes with multiple choice and true/false questions. Systems such as QUIZIT [21], WebCT [22], and ASSYST [13] 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.

There has been considerable research into the question of interactive and intelligent tutoring systems. The earliest tutoring systems (called CAI, for Computer Aided Instruction) simply provided immediate, interactive feedback to their users after they completed a given problem. This help varied from simply telling the user whether they were right or wrong to sophisticated explanations and animations. First-generation Intelligent Tutoring Systems (ITS), produced in the 1970s and 1980s, used Expert Systems to analyze the steps towards a correct solution in comparison to the steps the student chose [18]. Later ITS used sophisticated models of student understanding and utilized artificial intelligence to determine what sort and difficulty of problems to offer to students – one example is the tutors produced in the Advanced Computer Tutoring Project at Carnegie Mellon University[10]. Ongoing research on ITS addresses many possible improvements, such as how tutors must adapt to the needs of different students, software engineering and ease of use concerns, and the possibility of "third-generation" tutoring systems which engage in natural language dialogue with students.

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. There is a rich literature that describes the benefits of concept visualization in education settings. Algorithm animation has been successfully used for visualizing graph algorithms, sorting, and searching, to name a few examples [20]. Similarly, program code animation also helps in the learning of new programming languages. Finally, concept animation has also been successful in communicating difficult concepts such as finite state automata [6]. Tools for creating animations of data structures and algorithms have also been developed [19]. Interactive tutorials have been designed and their positive impact on student learning evaluated [3]. Electronic books have been proposed and developed, in which hypertext, interactive animations, audio and video parts are integrated in a web-based standalone educational resource [5].

1.2 Our Results

We have designed `PILOT` with several goals in mind. First, we would like to offer an interactive tool that can be used in class to aid in exposition. Furthermore, there are numerous problems that students learn best by example, 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 instant feedback as the student learns the algorithm and automated grading when the student completes a problem for a grade. Thus, `PILOT` allows for:

- generation of interesting 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) through Kruskal's algorithm and Prim's algorithm.

Another very important feature of `PILOT` is the detailed feedback it can give the student either as they work the problem or while it grades the problem. For example, in creating a MST, if an edge is chosen incorrectly, `PILOT` will highlight the edge in red, inform the student how that choice was incorrect, and, as appropriate suggest how the student could improve their choice.

As currently written, `PILOT` can be securely used to grade homework either with a TA designated problem or a randomly generated problem. Grading automation has the potential to save a large amount of TA effort, especially in a large class. In the Spring of 2000, `PILOT` was used in homework in Brown University's Data Structures and Algorithms class, CS016. Our experiences were quite positive.

2 Using `PILOT`

In the current scenario, the user chooses a problem type and mode from pull-down menus and clicks the "generate" button to create a random instance of that problem. Figure 1 shows the result of generating an instance of MST-Prim — a minimum spanning tree problem to be solved 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, in order, belonging to the MST. They have unlimited levels of undo and redo to allow them to change

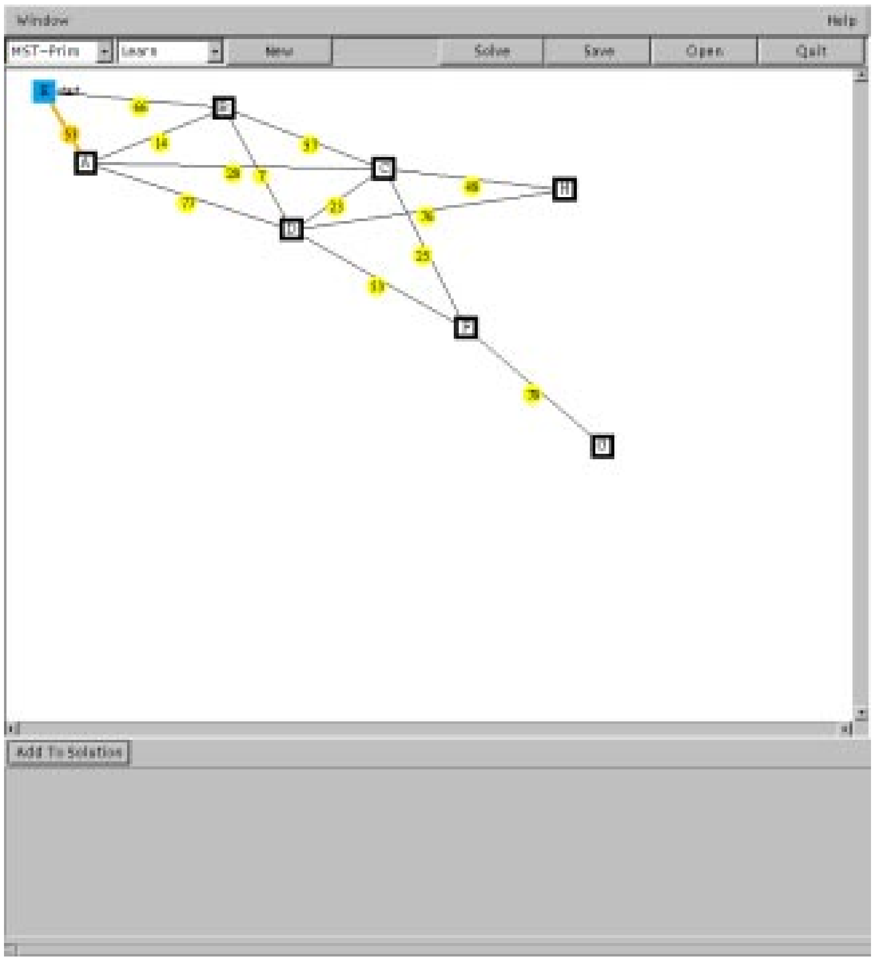


Figure 1: randomly generated graph for MST-Prim

their solution if they realize it is incorrect.

The student can place PILOT into three different modes of interaction. In the Learn mode, the student executes the algorithm, and after every step PILOT checks to see if that step was correct. If not, a message is generated and in some cases an animation displayed, informing the student how they were wrong. The student then has the option of undoing to try again or pressing on with the algorithm.

In the Exam Practice mode, the student creates their solution with no feedback (figure 2), and then, once the user is satisfied that she has entered the correct solution, clicking the "check" button will correct and grade the solution. (figure 3) The system will go through each of the student's actions in turn, explaining where and why the student's choice was incorrect. After it has finished, it gives an overall explanation of the student's performance, including the type of errors made, and a detailed grade breakdown (as shown in figure 4).

In the Learn and Exam Practice modes, an animation of the problem being solved can be obtained at any point by clicking the "solve" button. This allows a student unable to figure out what to do on a specific problem to see what to do.

The Exam mode is intended for students to use to submit a solution for a grade. When the student clicks to check the problem, the student's solution and performance is immediately logged in a place where it can be retrieved by a TA.

3 User interface

One extremely important part of any educational software package is its user interface. The best intentioned project will be quite useless if students cannot figure out how to use it. In designing our user interface, we have attempted to be sensitive to the problem of different learning styles. Since some students learn better with symbolic and natural language feedback whereas other students learn better from graphics and animation, we have attempted to accomodate both groups of students by giving simultaneous feedback of both types.

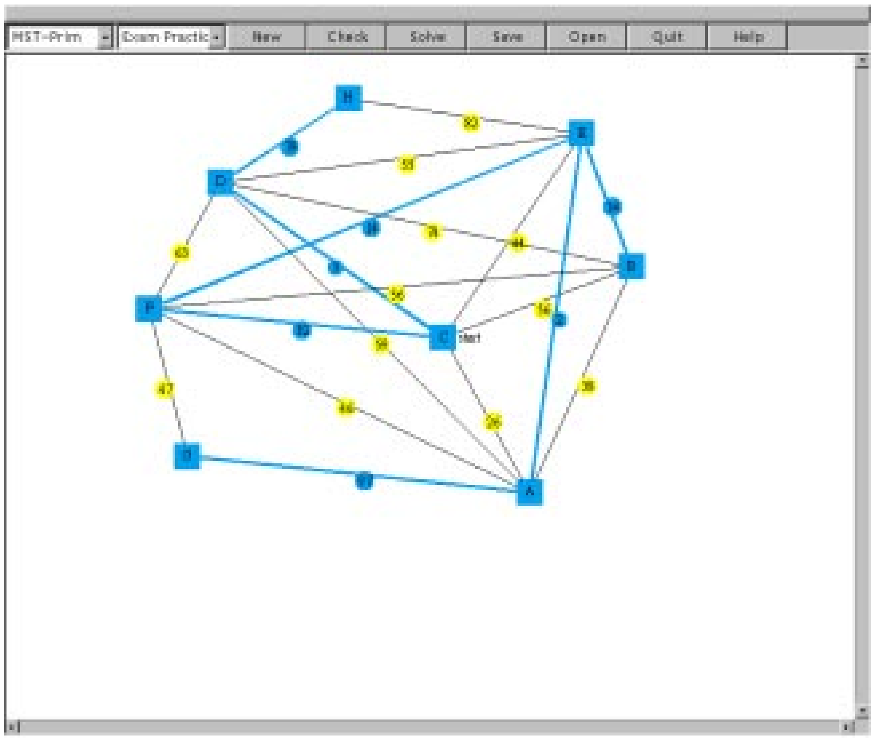


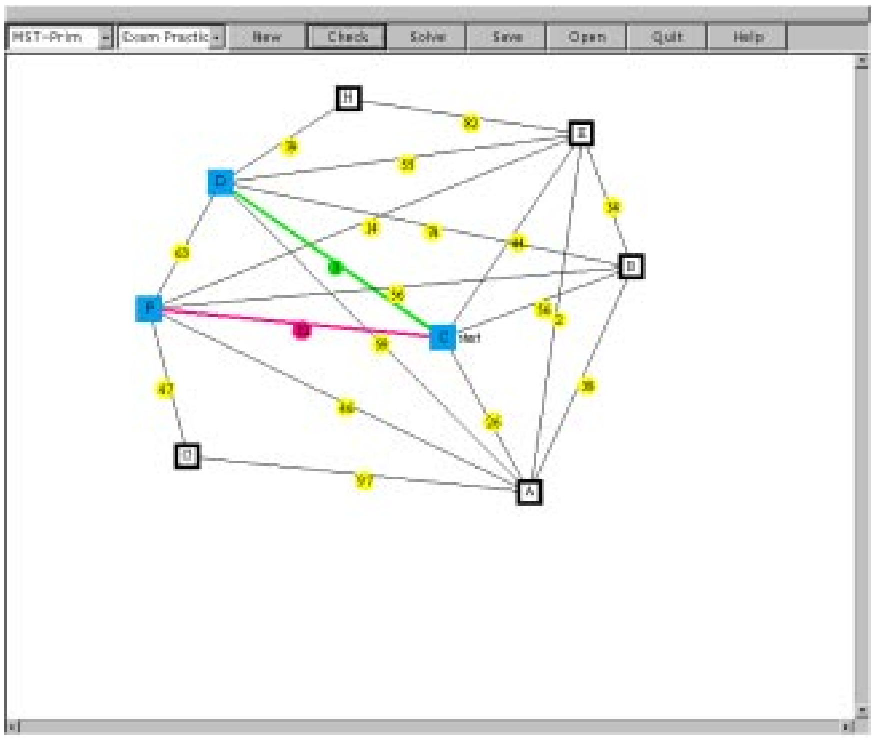
Figure 2: A student's answer

For the overall layout of `PILOT` we chose an approach similar to that used in a previous project, the JDSL Visualizer[1]. The JDSL Visualizer had one main window, which held the structure which the user interacted with and buttons to make changes to the structure. That main window also had a sub-window for the history of the structure and the student's actions. Menus in the main window allowed the student to create a new structure or to switch between structures. Smaller windows held messages from the JDSL Visualizer to the user, and help text.

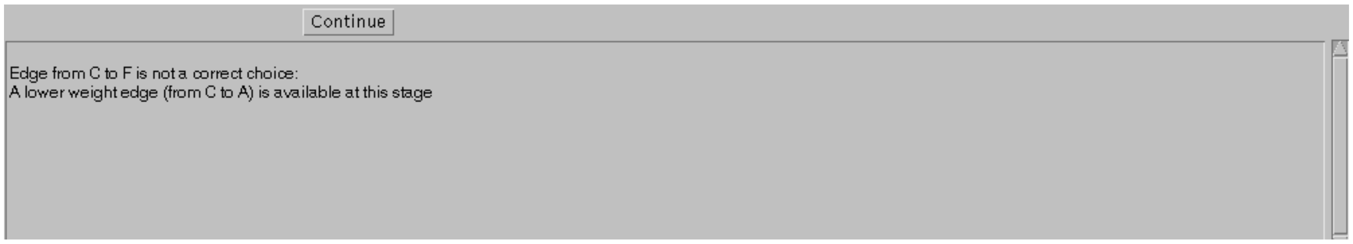
In `PILOT` we chose much the same approach. A large window contains the structure and allows the user to interact with the structure. Smaller windows held messages from `PILOT` to the user and a tutorial to help the students use PILOT. One difference is that the message window is also used by the student to undo and redo (thus putting history-keeping there), and to step through animations and grading. We chose this because the student will be looking at the messages in that window when they choose those actions. Examples of the main window and the message window can be seen in figure 3.

The student highlights an edge by clicking it or its edge weight label – this allows the student to see the edge and its weight more clearly (and also eliminates any danger of ambiguity in which label refers to which edge). Then the student can add it to their solution by clicking on it again. The student can remove it from their graph by clicking undo. The student can also drag vertices around to improve the readibility of the graph (see the further discussion of this in section 4.1).

The main window relies in part on code created by Robinson Mason in his Graph Editor[17]. This approach allowed us to implement `PILOT` more quickly, since many of the essential tasks of displaying a structure (such as converting an abstract drawing of a graph to boxes, lines, and circles on a screen) and of interacting with such a structure (recognizing clicks and drags on those objects) were already handled by the Graph Editor. One unexpected drawback to this approach comes with the use of older, "legacy" code – the Graph Editor was written in an older graphics library, BONGO [14], which has not been maintained by its implementors in three years (an eternity in the history of JAVA). Because BONGO is not fully compatible with newer versions of JAVA, we needed to use older virtual machines to run `PILOT`, making it necessary to use older versions of some software packages such as JDSL, the Library of Data Structures for JAVA [12]. We also found that we had to cope with a number of documented bugs in JAVA virtual machines which had published fixes in later versions of JAVA.



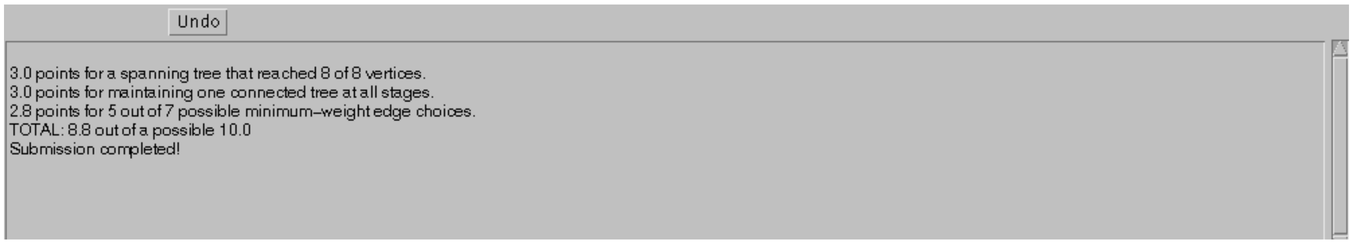
(a) Main window



Edge from C to F is not a correct choice:
A lower weight edge (from C to A) is available at this stage

(b) Message window

Figure 3: In the process of grading the student's solution



3.0 points for a spanning tree that reached 8 of 8 vertices.
3.0 points for maintaining one connected tree at all stages.
2.8 points for 5 out of 7 possible minimum–weight edge choices.
TOTAL: 8.8 out of a possible 10.0
Submission completed!

Figure 4: The student's final grade for this graph

3.1 Algorithm animation in PILOT

Algorithm Animation has been used in PILOT in several areas. The most substantive is in allowing the student to see an animation of the problem at hand being solved. As previously mentioned, if the student is having too much difficulty solving a specific problem, they can (so long as they are not working that problem to turn in for a grade) click a button to request such an animation.

In the case of Prim's algorithm, at each step of the algorithm's execution, the edges that connect to the spanning tree thus far are highlighted in orange – the edge among those with the lowest weight is highlighted in green. Those edges are also detailed in a message window, allowing users who prefer a linguistic explanation to gain the same information (see figure 5).

Another area where animation has been added is during PILOT's Learn mode. In this mode, when the student chooses an incorrect edge while executing the algorithm – the edge they chose turns red and in some cases there is animation. One instance is upon the choice of an edge of non-minimal weight – an edge they could have chosen with lower weight will flash momentarily in blue and a message posted (see figure 6). Another instance occurs when the user has created a cycle – in this case, each of the edges along the cycle is flashed blue in turn.

A third area where animation has been added is during the grading of a student's solution in the exam and exam practice modes. Rather than just returning an itemized explanation of how the student performed (as in the version discussed in [8]), it goes through the solution step-by-step, animating incorrect choices as in the learning mode.

Thus, Algorithm Animation is used in PILOT in two ways: to provide a quick illustration of how the problem should be executed and to add support for students who learn visually rather than verbally in PILOT's interactive learning mode and during grading.

3.2 The use of color

Color is an important tool in any user interface – it can add considerable information if used properly, but can be confusing or misleading if misused. We have attempted to use color to add information and context to PILOT in the following ways:

1. During animation to indicate incorrect edges, the edge the student should have chosen, the pieces of a cycle, and possible choices the student could choose when PILOT is demonstrating a solution.

2. By changing edge colors depending on whether they are highlighted, added to the tree, correct, and incorrect.

3. By changing a vertex's color once it connects to the tree.

Although these additions are important, and are helpful to students, it is important to realize that some students may have partial or complete color-blindness. Therefore no information should be transmitted through color alone. All color-transferred information is duplicated by text messages. Furthermore, we provide support for color customization, to allow adjustment away from difficult-to-see colors, or adjustment of color intensity. The student who wishes to change their colors scheme needs only to modify a provided dotfile and place it in their home directory.

4 PILOT Architecture

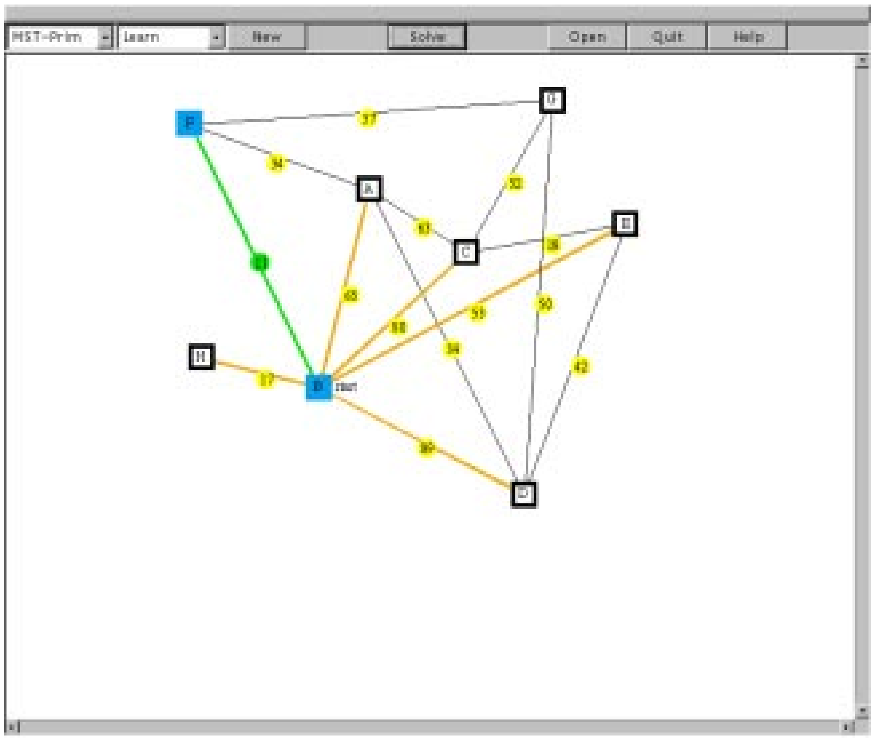
PILOT uses a client-server architecture, and uses GeomNet [2] to support some of its functionality. In the GeomNet model, the client is responsible for maintaining the user interface and all of the algorithm-related computation is done on the server. For PILOT, the client is implemented as a Java application and the server side handles the generation and drawing of the problems. The main motivation for choosing the client-server architecture in these cases was flexibility — the server 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 Server [7] component of GeomNet to compute a layout for the automatically generated graphs. The server can be run either remotely or on the same machine as the client – running locally can be more demanding on slower machines but eliminates network delays as a run-time factor.

We do not use GeomNet for PILOT's interaction, feedback, and grading. Instead, we place those components on the client side. Although this reduces modularity, it eliminates the danger of a huge numbers of transactions flying back and forth between client and server, causing considerable slowdown.

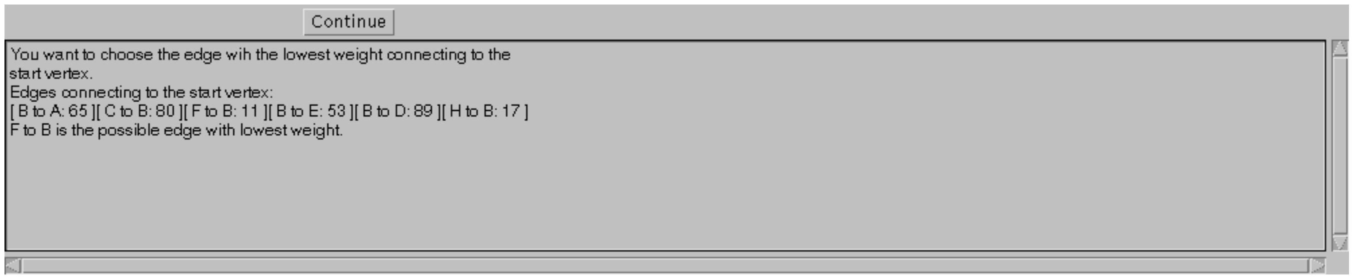
We now look at the graph generator and interaction/grading components of PILOT in more detail, focusing on minimum spanning tree problems as an example; the problem solvers are straightforward implementations of the appropriate problem solving algorithms and are not considered further.

4.1 Graph Generator

The graph generator uses a method similar to that of [11] to generate "realistic" graphs for experimental purposes. Graphs are built from a single vertex by repeatedly applying three operations — (1) insertion of a vertex and a random number of adjacent edges, (2) insertion of an edge between two existing vertices, and (3) splitting of an existing edge by replacing it with a



(a) Main window



You want to choose the edge wih the lowest weight connecting to the start vertex.
Edges connecting to the start vertex:
[B to A: 65][C to B: 80][F to B: 11][B to E: 53][B to D: 89][H to B: 17]
F to B is the possible edge with lowest weight.

(b) Message window

Figure 5: PILOT here seen solving a problem for the student.

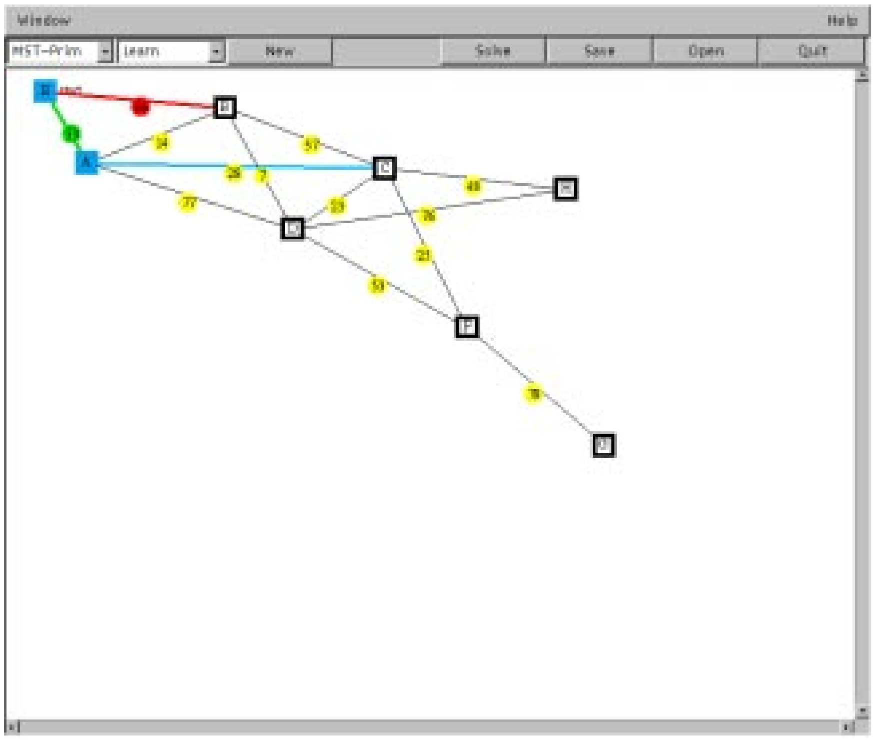


Figure 6: The user has chosen an edge of non-minimal weight.

new vertex and two new edges. Graph properties such as the ratio of edges to vertices can be controlled by adjusting probabilities assigned to each of the operations and the degree of newly inserted vertices.

After the graph has been generated, it is then layed out by an appropriate graph-drawing algorithm. After considering many alternatives, we chose GEM, a force-directed program written in C, for its relatively quick, well-spaced, and easy-to-read drawings. Using GeomNet and the Graph Drawing Server will allow users of PILOT on different platforms to still use GEM's drawings, despite the fact that GEM is platform-specific (running in UNIX), by placing the server on a UNIX machine.

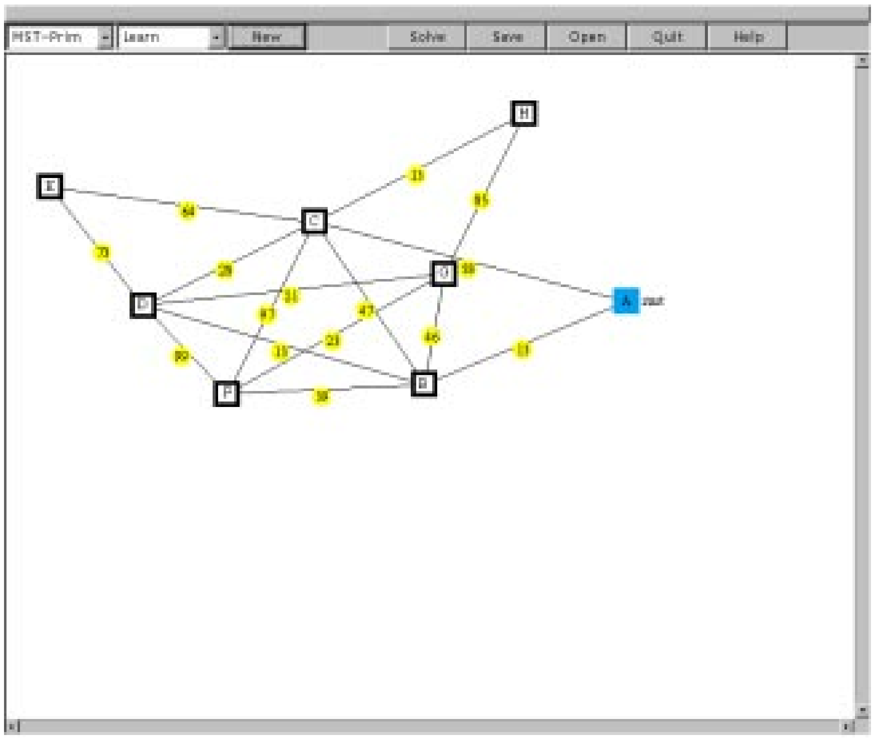
The graph is then passed to the client, where it is drawn onscreen. Students can at this point drag vertices around the screen to improve the graph's readability. Edges are pulled along with the vertex – see figure 7 for an example of this.

4.2 Grading and Interaction

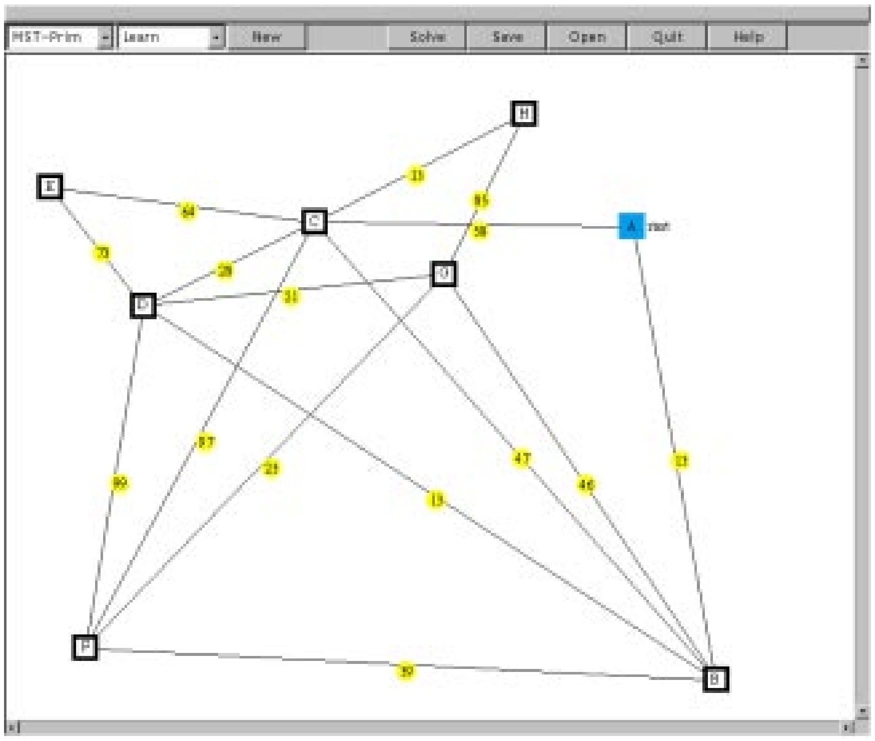
There are three main challenges in checking the correctness of a student's solution of an algorithm: handling non-unique solutions, assigning appropriate partial credit, and returning meaningful feedback.

There are several possible ways to design a checker for MST-Prim. It is relatively easy to compute a solution to the entire problem and compare the user's input to it, returning a grade based on how many of the edges are the same. However, this approach 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, very different solutions can easily occur in MST problems when multiple edges have the same weight. Some handling of non-uniqueness is possible, such as accepting alternate edges of the same weight, but this does not address the large changes to the ordering of a solution that can arise from switching one edge – if we were solely concerned with whether the student could generate a correct MST, this would not be a problem, but it is not appropriate where we are concerned with their process of creation, and thus their ordering.

Additionally, it 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. Additionally, it is very diffuclt to generate meaningful feedback with this scheme, since it cannot differentiate between different types of errors, and to assign more sophisticated partial credit which



(a) Before



(b) After

Figure 7: Dragging a vertex to a more appropriate location

addresses the varying importance of different sorts of errors.

A better approach is to verify properties of the user's solution, to ensure that it is valid. One such approach for checking an MST would be as follows: 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 still does not address the situation where early mistakes can be compounded. An additional problem with this approach is that it grades the student's final solution rather than the process the student went through to get that answer. This means that it is vulnerable to errors where the student ended up with a correct result but did not apply the algorithm properly, adding edges in an incorrect order and/or violating structural properties. For example, if a student executed Kruskal's algorithm for Minimum Spanning Trees, their final answer would be a correct spanning tree, but it would not demonstrate understanding of Prim's algorithm.

To address these concerns, our checkers take an incremental approach and step through the solution of the problem, taking into account the user's choices as they happen. At each step, `PILOT` generates a correct solution for that step and then compares it to the student's solution to see if the student's solution is equally as valid. It's important that it give credit to alternate, equally valid answers – otherwise, we run into the problem of non-unique solutions again. Since this approach looks at each individual step, it is not vulnerable to taking off many points for one mistake. Similarily, since it checks at each step for correctness it can ensure the student used correct process as well as coming to a correct final solution.

In the case of MST-Prim, this step of the student's answer could be inferior to the correct solution in one of the following ways:

- Creating a cycle
- Disconnecting the spanning tree
- Not adding another edge where an edge could be added
- The edge the student chose is of higher weight than the edge `PILOT` chose

If the student's answer is equally as good as `PILOT`'s answer (it is possible for two different edge choices to both be correct if both do not create a cycle, maintain the spanning tree, and have the same weight), the student gets credit for a correct answer. If `PILOT` determines the student's answer is wrong, it notifies the student using both text and animation and also takes away points from the student. The seriousness of the error (and whether the student has already made that error) affects how many points the student loses.

One example of the partial credit system is the one currently being used for grading MST-Prim problems. In that system, the student's grade is computed as follows:

- 3.0 points for ending with one connected spanning tree, with no cycles, that includes most or all of the vertices
- 3.0 points for maintaining a connected spanning tree at all steps of execution (given that the previous condition is met)
- plus 4.0 points
- minus 4.0/(total number of edges in student's solution) for each edge of greater than minimum weight or extraneous edge in student's solution

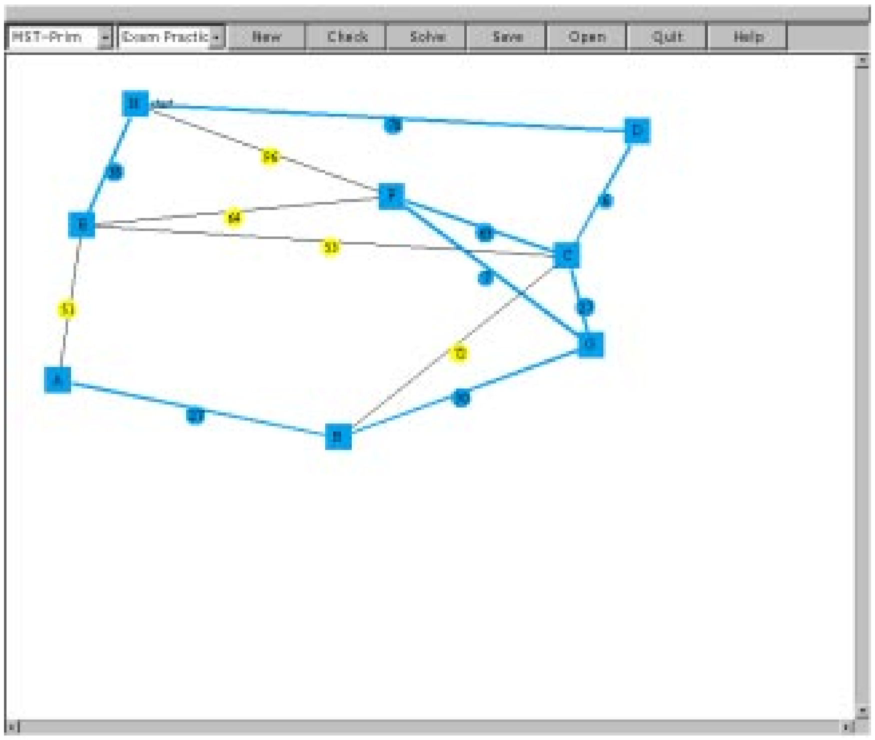
The student will gain 10.0 points in total if their execution is perfect. See figure 8 for an example of a grade report where the student had a cycle and one edge of greater than minimum weight.

Once a system has been created for grading each step of a pre-created solution, it was extremely easy to adapt to immediate interaction; rather than going through each step of a created solution at once, it simply grades each step as the student enters it. It is important while doing this to make sure that cached information about the overall structure of the graph (such as whether the graph has a cycle) is re-generated in the event the student undoes a wrong answer.

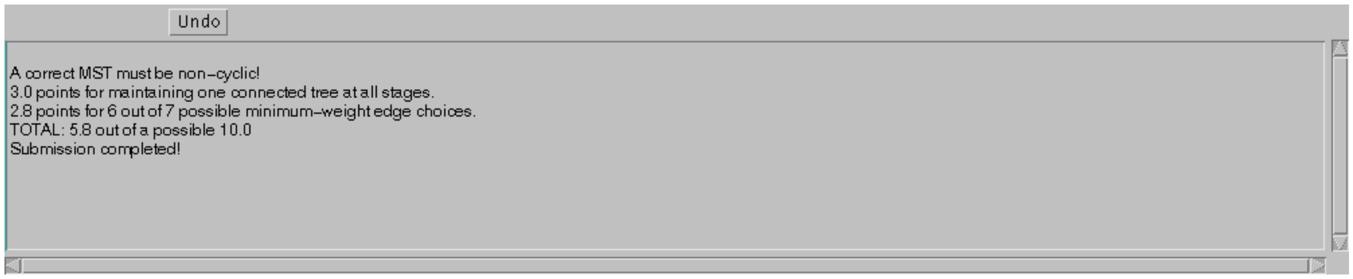
5 Use in CS016

5.1 Classroom use

`PILOT` was used for the first time in a large-scale classroom setting this spring in CS016 at Brown University. It was introduced during lecture on April 10th, and students were required to complete and turn in a designated problem by April 14th, shown in figure 9. Then, a comparable traditional homework problem (with an identical number of vertices and edges) was assigned to be turned in on April 21st, shown in figure 10. Both



(a) Main window



```
A correct MST must be non-cyclic!
3.0 points for maintaining one connected tree at all stages.
2.8 points for 6 out of 7 possible minimum-weight edge choices.
TOTAL: 5.8 out of a possible 10.0
Submission completed!
```

(b) Message window

Figure 8: A student's final grade

problems were for the execution of Prim's Algorithm for Minimum Spanning Trees on a specific graph.

The students were assigned comparable problems in both PILOT and traditional homework for two reasons: First, just in case PILOT turned out to be of no benefit, the students would still have the opportunity to learn the subject material – and second, because that would make it possible to make correlations between performance in the two media in order to determine whether PILOT is of benefit.

5.2 Classroom concerns

To prepare an online homework for a classroom requires attention to several concerns, including fairness and security.

Regarding fairness, students were encouraged to go through as many random problems as they wished for their own learning process, but since guaranteeing equal difficulty between two different problems is quite difficult, anything involving a student's grade cannot differ between students – and thus, in order to guarantee fairness, all students were given the same problem for the problem they would eventually turn in for a grade.

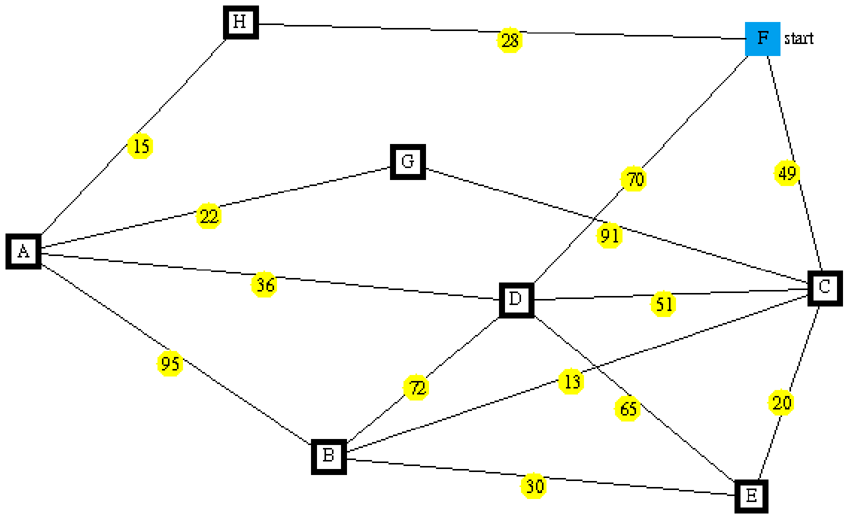


Figure 9: The `PILOT` problem designated for handin.

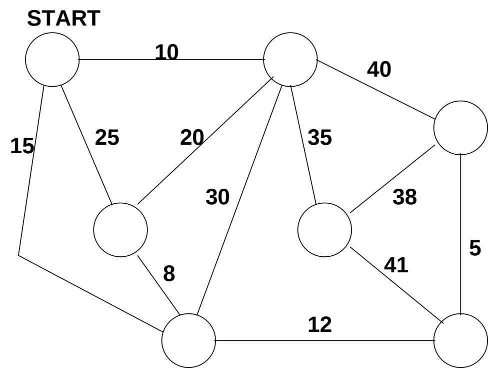


Figure 10: The traditional MST-Prim homework problem used for comparison.

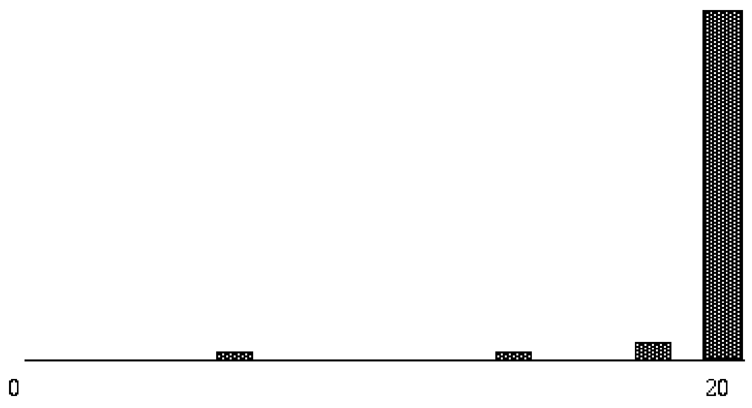
In order to guarantee the security of student handins, logs of student actions were placed into a password-protected directory accessible only to authorized course staff both as soon as the student clicked to Submit a homework problem and after grading had been completed. These logs make it very difficult for one student to copy another's work without being caught, by using timestamps for when the log was created and when it is handed in – if one student were to save their solution and give it to another student, the logs would reflect that the student had handed in a correct answer without taking the steps necessary to build it. Autosave snapshots were also taken after every student action to prevent the loss of a student's work in the event of a crash.

5.3 Empirical results

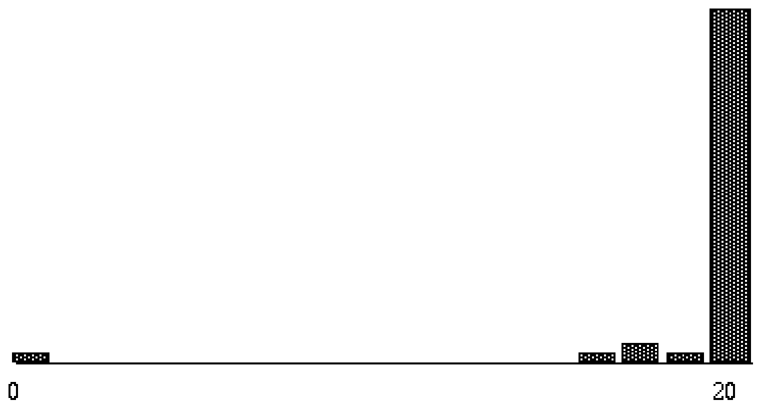
We took the following variables to examine for correlation between them:

- **P** Performance in `PILOT` (grade on figure 9): 0.0 to 20.0
- **H** Performance on traditional homework problem (grade on figure 10): 0.0 to 20.0
- **T** Amount of time spent using `PILOT` : 0 minutes to 238 minutes
- **W** Number of practice/learn problems student began in `PILOT` : 0 to 68
- **C** Percentage of correct steps taken while using `PILOT` : 0 to 100

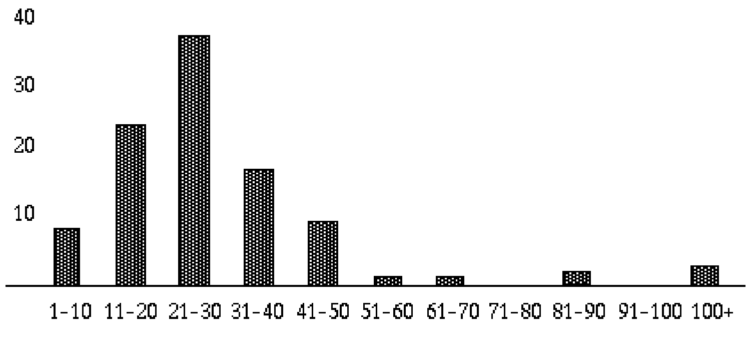
T,**W**,**C**, and **P** were calculated by `PILOT` as the student used it. **H** is based upon the traditional TA grading scheme used in CS016, where the TAs assigned to grade the problem determine the grading rubric that evening. This system was chosen to match as closely as possible to the way grading has been done in CS016 in past years.



(a) H (performance on traditional homework problem)



(b) P (performance in `PILOT`)



(c) T (number of minutes spent using `PILOT`)

Figure 11: Distribution of some variables in our empirical study

Our intention was at this point to correlate **H** to each of the other metrics in order to determine whether performance on PILOT correlated to performance on the written homework (telling us that PILOT is a valid and fair replacement for grading purposes) and whether students who use PILOT for a longer amount of time perform better on the future written homework (telling us that PILOT is indeed teaching them something).

Unfortunately, this proved impossible, for the reason that the students did considerably better on the assignment than expected. On **H**, 94 percent of the students achieved perfect scores and only 2 students out of 102 received less than 16 out of 20 on the assignment, giving an average of 19.71 and a standard deviation of 0.56. On **P**, 93 percent of the students achieved perfect scores (not including 5 students who did not use PILOT and received a grade of 0 for the assignment). Furthermore, no relationship was found between those students who did not perform well on the two assignments. The small percentage of students who performed imperfectly – or poorly – on the homework made it impossible to find a significant relationship between it and any other variable. You can see diagrams of the distributions on some of the variables in figure 11.

We suspect that the cause of this inconclusive data is that the graded problems we gave to the students, both in PILOT and on the homework, were too easy. We are not certain whether the problem was that the specific problems we assigned were too easy or whether Prim's algorithm is too easy to learn, although we suspect the second is a better explanation. We can probably reject the hypothesis that this extremely positive performance is the result of the students' use of PILOT , since the performance on **H** included several students who either used PILOT very briefly or did not use PILOT at all, and took a grade of 0 on that assignment. (5 of the 6 students who did not use PILOT still got perfect grades on the homework)

Therefore, the essential step we must take in experimenting with PILOT in the future is to do our experiment on a considerably more conceptually difficult type of problem. One possibility under consideration is Maximum Flow algorithms. Next year, when we repeat the experiment with this change, we hope to gain considerably more conclusive data. Another future direction once we have resolved this problem is a full-scale study with a randomly assigned control group to ensure that any corrleations between **T** or **W** and **H** are the result of using PILOT rather than measures of how dilligent the student is. This will require considerable care and perhaps require conducting the study over multiple years or outside of the classroom.

6 Future Work

The current PILOT system can be extended in many ways. Of particular importance is the generation of problems of approximately equal difficulty (and, related to this, the generation of appropriate special cases). For example, in Prim's algorithm the addition of an edge and vertex to the spanning tree may result in a new, lower-weight connection for an unconnected vertex and thus change the best choice for the next vertex/edge pair added to the tree. Problems with many instances of this case may be viewed as harder than problems without, since they require knowledge of particular cases in the algorithm. 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. One idea for dealing with this is the generation of isomorphic graphs with different edge weights but identical order of edge choice, drawn in a different configuration.

Additionally, 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. Adding new algorithms will also require the creation of problem checkers with appropriate partial credit.

Finally, we believe that further empirical study of PILOT is necessary in order to demonstrate its usefulness (or uselessness, as the case may be). As discussed above, this will include comparing with more difficult types of problems, and possibly the use of random control groups.

References

[1] Baker, R., Boilen, M., Goodrich, M. T., Tamassia, R., and Stibel, B. A. Testers and visualizers for teaching data structures. *Proceedings of SIGCSE* (March/April 1999).

[2] Barequet, G., Bridgeman, 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.

[3] Barnett, L., Casp, J., Green, D., and Kent, J. Design and implementation of an interactive tutorial framework. In *Proc. 29th SIGCSE Tech. Symp.* (1998), pp. 87–91.

[4] Blackboard Inc. www.blackboard.com.

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

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

[7] Bridgeman, S., Garg, A., and Tamassia, R. A graph drawing and translation service on the WWW. *Internat. J. Comput. Geom. Appl. 9*, 4 & 5 (1999), 419–446.

[8] Bridgeman, S., Goodrich, M. T., Kobourov, S. G., and Tamassia, R. Pilot: an interactive tool for learning and grading. *Proceedings of SIGCSE* (March/April 2000), 139–143.

[9] Carrasquel, J. Teaching CS1 on-line: the good, the bad, and the ugly. In *Proc. 30th SIGCSE Tech. Symp.* (1999), pp. 212–216.

[10] Corbett, A., Koedinger, K. R., and Anderson, J. Intelligent tutoring systems. In *Handbook of Human-Computer Interaction*, M. Helander, T. Landauer, and P. Prabhu, Eds. Elsevier Science, Amsterdam, The Netherlands, 1997, pp. 849–874.

[11] 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.

[12] Goodrich, M. T., Handy, M., Hudson, B., and Tamassia, R. Abstracting positional information in data structures: Locators and positions in jdsl. *Oopsla '98 Technical Notes* (1998).

[13] Jackson, D., and Usher, M. Grading student programs using ASSYST. In *Proc. 28th SIGCSE Tech. Symp.* (1997), pp. 335–339.

[14] Marimba. Bongo sets the rhythym for java development. Press Release, 1996. http://www.marimba.com/news/releases/bongo-oct7.html.

[15] Mason, D., and Woit, D. Integrating technology into computer science examinations. In *Proc. 29th SIGCSE Tech. Symp.* (1998), pp. 140–144.

[16] Mason, D., and Woit, D. Providing mark-up and feedback to students with online marking. In *Proc. 30th SIGCSE Tech. Symp.* (1999), pp. 3–6.

[17] Mason, R. Graph editor: An applet interface to graph drawing server. Master's thesis, Brown University, March/April 2000.

[18] McArthur, D. Some possible futures for artificial intelligence in mathematics education. *Proceedings of the Fifth Annual Conference on Technology in Collegiate Mathematics* (1992). http://www.rand.org/hot/mcarthur/Papers/aied.htm.

[19] Pierson, W., and Rodger, S. Web-based animation of data structures using JAWAA. In *Proc. 29th SIGCSE Tech. Symp.* (1998), pp. 267–271.

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

[21] Tinoco, L., Fox, E., and Barnette, D. Online evaluation in WWW-based courseware. In *Proc. 28th SIGCSE Tech. Symp.* (1997), pp. 194–198.

[22] WebCT, Inc. `www.webCT.com`.