Graphical Extensions to Online Problem Sets and Tutors

by

Yuval Mazor

B.S., Electrical Engineering and Computer Science (2002)

Massachusetts Institute of Technology

Submitted to the Department of Electrical Engineering and Computer Science
in Partial Fulfillment of the Requirements for the Degree of Master of
Engineering in Electrical Engineering and Computer Science

at the

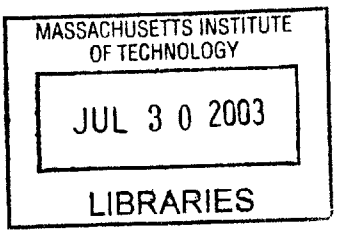
Massachusetts Institute of Technology

June, 2003

Author _____
Department of Electrical Engineering and Computer Science
May 21, 2003

Certified by _____
Tomas Lozano-Perez
Professor of Computer Science and Engineering
Thesis Supervisor

Accepted by _____
Arthur C. Smith
Chairman, Department Committee on Graduate Theses



GRAPHICAL EXTENSIONS TO ONLINE PROBLEM SETS AND TUTORS

by

YUVAL MAZOR

Submitted to the
Department of Electrical Engineering and Computer Science

May 21, 2003

In Partial Fulfillment of the Requirements for the Degree of Master of Engineering in Electrical Engineering and Computer Science

ABSTRACT

Adding a graphical interface to online problem sets and tutors enables the addition of a variety of different types of graphical problems to interactive learning systems. The graphical interface allows professors and students to sketch graphs, draw shapes, and label objects using a mouse-based interface. A text-based I/O allows professors to create graphical problems, set up the solutions, and provide interactive feedback. Students working on a problem can be given hints, check their solution, or see the correct solution. The graphical extension also lays the groundwork for more complex graphical problems involving connectivity or other rules between graphical objects.

Thesis Supervisor: Tomas Lozano-Perez

Title: Professor of Computer Science and Engineering

TABLE OF CONTENTS

LIST OF FIGURES

Chapter 1

INTRODUCTION

In 1999, a collaboration between MIT and Microsoft initiated iCampus, a 5-year campus-wide campaign to improve the use of technology in the educational process. Although new technology is always being incorporated into lessons, this was the first attempt to fundamentally change the way classes were taught and information exchanged between professors and students. One of the projects under iCampus was the Technologically Enhanced Education in Electrical Engineering and Computer Science project, headed by Tomas Lozano-Perez. His project focuses on building an on-line interactive tutor for the Electrical Engineering and Computer Science Department. This EECS tutor will let students work at their own pace, and help professors and teaching assistants spend less time review and grading, and more time presenting new material and interacting with students. [4]

Although the TEE project was a first for MIT's Electrical Engineering and Computer Science Department, other classes both at MIT and elsewhere have witnessed significant benefits from using online tutors. For example, the Cybertutor is a web-based system developed for 8.01, the Introductory Newtonian Mechanics course in the Physics Department at MIT. This highly interactive tutor presents questions using pictures, graphs, and text, and relies on students to ask for hints when they are stumped. By including a great deal of information in the problems, as well as links to background information, the tutor has improved student performance in the class. In addition, by retaining statistical data of students' responses, it allows for easy assessment of both the students' skills and the effectiveness of the curriculum. [6]

David Arnow's WebToTeach program is another interactive tutor intended for classroom use. WebToTech is designed specifically for teaching programming languages over the web, focusing on immediate correctness feedback and repetition of exercises. [2] The tutor includes a large library of questions and answers and there is a minimal amount of overhead on the part of the professor. In addition, response data is collected from all participating schools, to be used in a Clinical Trial that will assess the effectiveness of interactive tutors in education. [2]

A commercial product, David and Paul Gries' ProgramLive, is perhaps the most thorough teaching system we explored. Like WebToTeach, ProgramLive focuses on teaching computer programming. However, beyond presenting problems and solutions, ProgramLive includes step-by-step learning and reference sources. Using a variety of interactive means, hints, flash videos, etc. ProgramLive is effective as a stand-alone teaching program, rather than a complementary tutor. [3] Other companies, like WebCT, move even further towards managing a technology-based classroom environment, but do not deliver as much of the question and answer based components as the other tutors. [8]

The successes of these four products served as a blueprint for the TEE project to create the EECS tutor as a similar web-based program. As with the ProgramLive and WebToTeach systems, much of this project's effort has been aimed at integrating the tutor into a computer science curriculum. Two specific classes at MIT have been targeted: 6.001 Structures and Interpretations of Computer Programs, an introductory class taught in the scheme programming language, and 6.034 Artificial Intelligence, a more advanced class that uses many of the skills from 6.001.

The initial version of the EECS tutor, first used in 6.001 in the spring of 2000, provides online problem sets and practice problems to students via a web browser. The tutor allows for a number of different types of questions including multiple-choice, true-false, short answers, and scheme programming. Answers are stored and checked and students are given immediate feedback.

Although the tutor eventually proved successful in both classes, it is fundamentally limited in the types of questions that can be presented. The most significant constraint is the lack of graphical problems. While text and code-based problems are important, the family of graphical problems includes everything from vector

diagrams and simple graphs, to flow charts and shape-based drawings, and eventually to subject-specific problems like circuit diagrams. The purpose of this work is to expand the EECS tutor to include the basic graphical problems involving vectors, curves, and shapes, and lay the groundwork for the more complicated graphical problems.

Chapter 2

BACKGROUND

When the first tutor was developed, a number of important decisions were made that served as the backbone for future extensions. High-level goals were set to ensure that integrating the EECS tutor into a class would be beneficial to the curriculum. The initial version of the tutor sought to meet these goals in the limited scope of the spring 2000 version of 6.001.

2.1 System Goals

In order for the EECS tutor to be accepted as a viable contribution to the 6.001 curriculum, it had to achieve five primary goals: accessibility, interactivity, ease of setup, ease of use, and gradability.

1. Accessibility - One of the motivations behind the iCampus initiative was to take advantage of the prevalence of technology on campus. While human resources like professors and TA's are constrained by time and availability, computers and Internet access are available anywhere, any time at MIT. Using computer resources should help students learn and work at their own convenience and pace.
2. Interactivity - One of the drawbacks of replacing human interaction with computers is the loss of a back-and-forth dialogue. However, computers are able to give immediate correctness information which humans cannot provide. By checking a student's work and giving immediate feedback an interactive online environment can replace, if not improve on some of the face-to-face interactions that are lost.

3. Ease of setup – It is time consuming for professors to modify their curriculum to use new technology. Even if the long-term benefits are clear, the overhead cost must be as low as possible. It should be as easy as possible for professors to find computer-based analogs to the questions they want to ask. Furthermore, setting up questions using the new system should be as simple and intuitive as possible.
4. Ease of use – One of the main purposes of the EECS tutor is to improve the learning process for students. However, if it becomes more difficult to use the tutor than normal paper-based problem sets, or if the complexity of the technology detracts from the intended lesson, then the students suffer. It is important to make the system both intuitive and user friendly so it does not distract from the actual material being taught.
5. Gradability – In order for the EECS tutor to be effective, it must be able to take a student's graphical input and convert it into a format that can be stored and graded automatically. Furthermore, it must be possible to encode a solution and determine a correspondence between the correct solution and a given student answer.

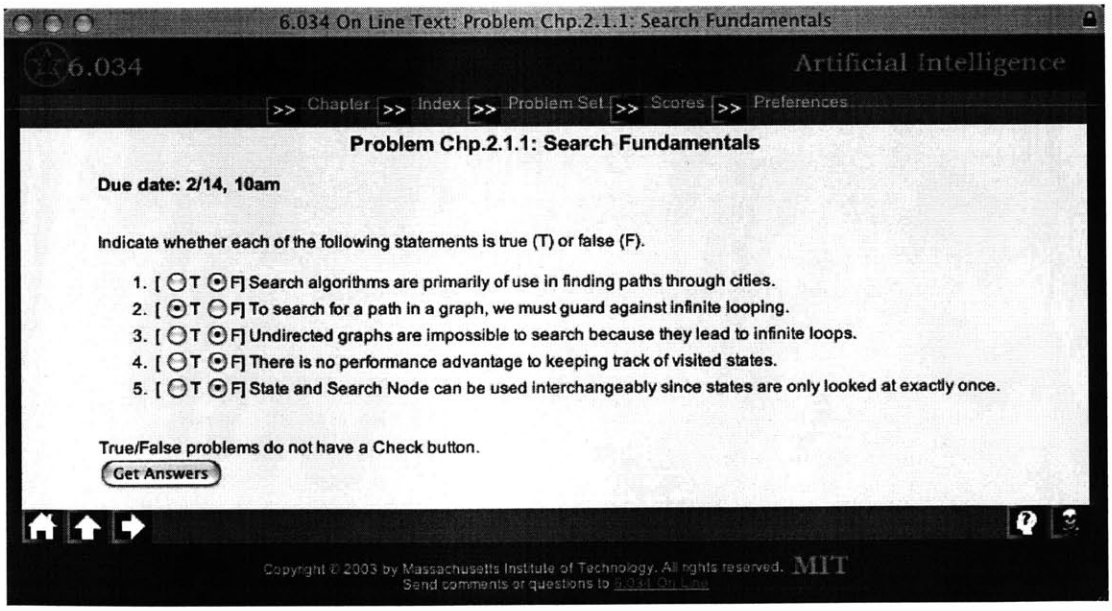


Figure 2.1. A screen shot of multiple choice problems from the spring 2003 Artificial Intelligence class

2.2 Initial Implementation

With these five guidelines in mind, it was decided to use web pages as the primary interface for the tutor. Web browsers are easily accessible, and since they function the same regardless of the hardware or operating system (for the most part), they require minimal overhead. HTML is dynamically generated and form submission, embedded JavaScript and Java applets contribute to make an interactive system. Once a new type of problem or entire problem set is completed, it can serve as a template to simplify future work so professors will not have to spend time creating web pages. Lastly, most students are already familiar with interactive web pages so the learning curve should be relatively shallow.

Using HTML and JavaScript for the front-end, the initial version was first used in

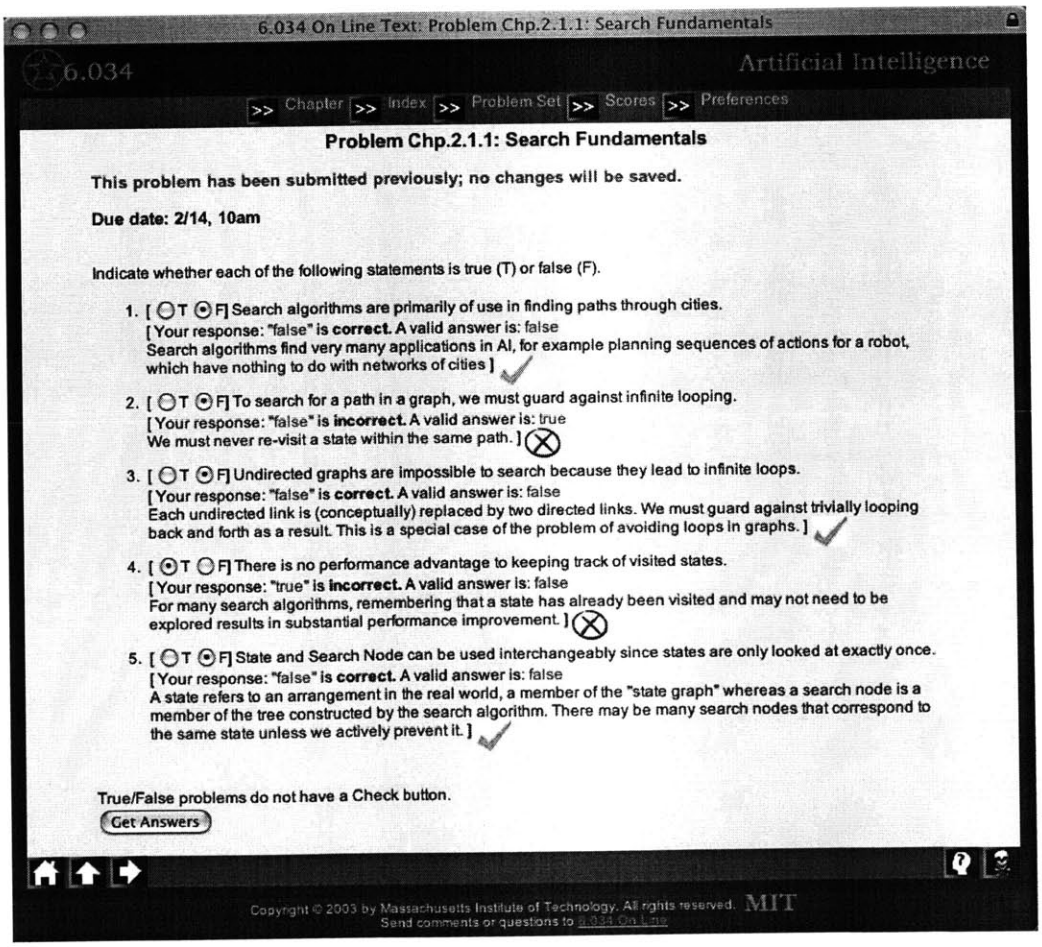


Figure 2.2. The feedback returned after a partially incorrect attempt to solve the multiple choice problems.

the spring 2000 semester of 6.001. Taking advantage of standard HTML form features – radio buttons, drop down lists, input fields – the EECS tutor was able to ask multiple-choice, true-false, fill in the blank, and short answer questions. Upon submission, the student's answers are checked and the results stored on the back-end. The students' answers are then redisplayed with the correct answer, a message indicating whether or not the student was correct, and an explanation of the correct solution.

2.3 Scheme Coding

This basic functionality allowed the EECS tutor to handle a large percentage of the questions in a typical problem set. However, since a major focus of the curriculum is writing and modifying scheme code, an applet was developed to act as a GUI for on-screen scheme coding. From the user's perspective, text input is automatically tabbed, and parentheses are flashed to show where they match. On the back end, the student's

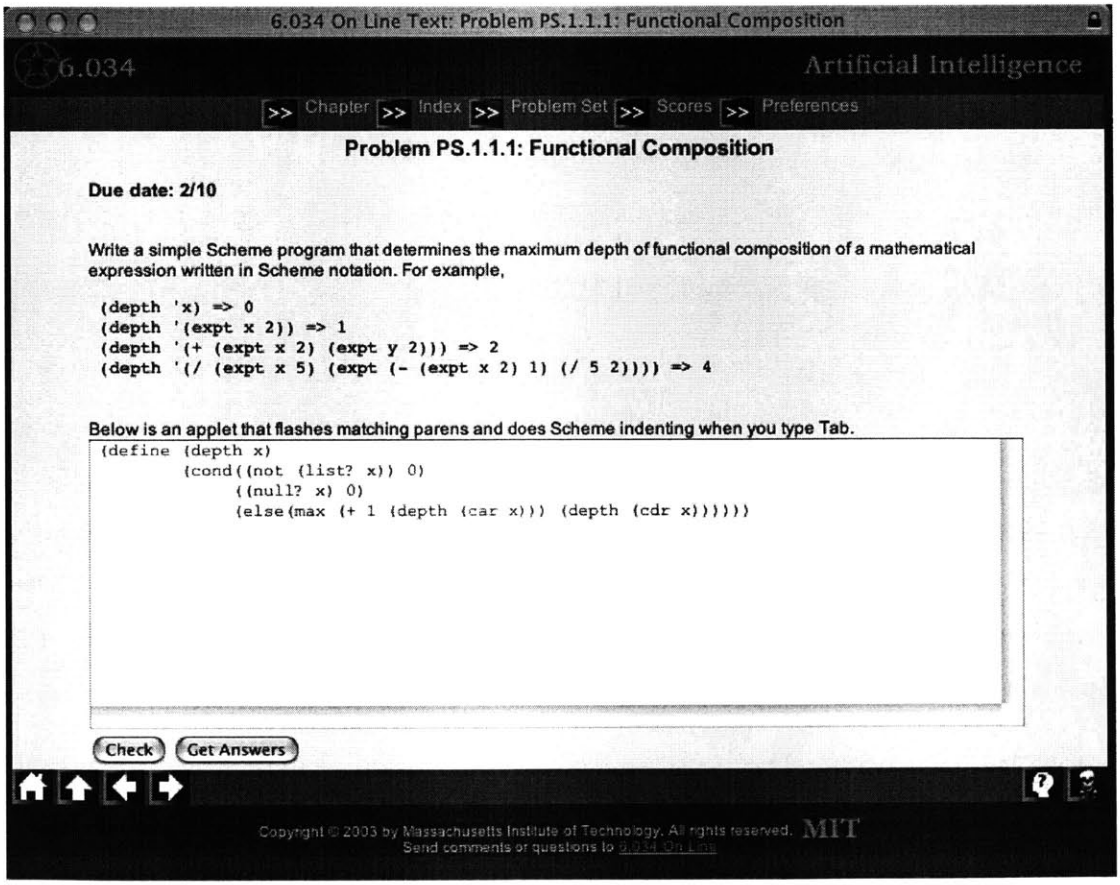


Figure 2.3. A screen shot of a scheme coding problem from the same class

code is interpreted and run against a number of test cases to determine correctness. As with the simpler cases, after submission the browser redisplays the student's answer in the GUI applet, as well as the appropriate feedback and correctness information.

Including the scheme applet in the EECS tutor accomplished a number of important goals. Above all it made the tutor robust enough to permanently include it in 6.001 and led to its inclusion in 6.034, as well. However, it also was an example of how different types of problems could be added to the tutor, and suggests that with similar additions, the EECS tutor may be useful for a variety of other classes.

Chapter 3

MOTIVATION

One of the major deficiencies of the original version of the EECS tutor was the lack of an interface for graphical problems. Many classes in the Electrical Engineering and Computer Science Department rely heavily on problem sets requiring students to sketch graphs, draw diagrams or charts, or other graphical solutions. For example, many electrical engineering problem sets include problems involving drawing or analyzing circuit diagrams or graphing the time and frequency responses to an input signal. Similarly, tree and graph analysis, flow charts, and vector diagrams, are all common types of problems in the math and computer science sides of the department. In order for the EECS tutor to be useful throughout the department, it is essential to develop another

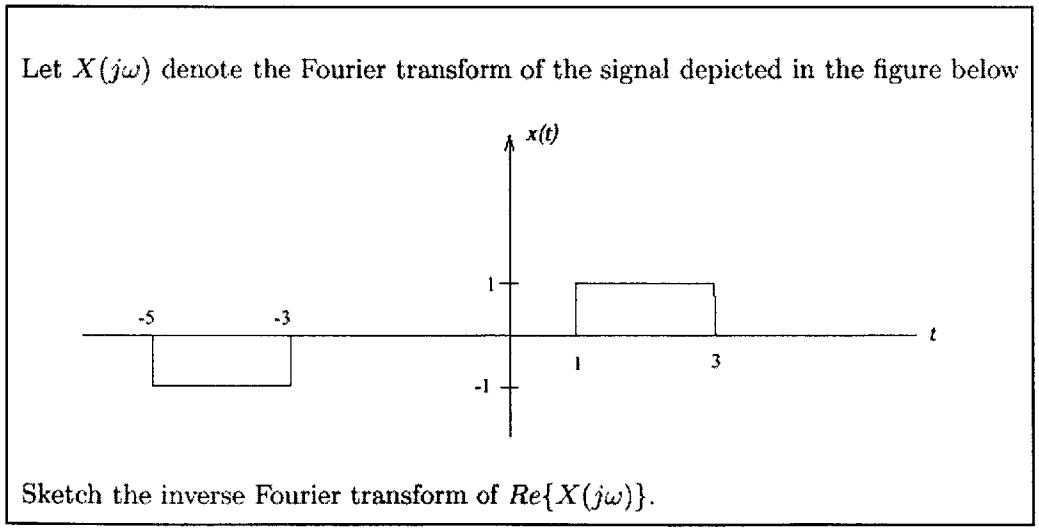


Figure 3.1. A graphical question from the Signals and Systems class, asking students to sketch a graph.

extension that provides for these various types of graphical problems.

3.1 Commercial Solutions

One possible graphical solution is to incorporate an existing program into the EECS tutor. For example, commercial software like MATLAB and MAPLE already provide graphing functionality and many students and professors are familiar with them. Similarly, programs like Microsoft Visio provide functionality for drawing flowcharts and other diagrams. In fact, many classes already expect students to use these complementary programs on paper-based problem sets. There are a number of advantages for trying to incorporate third-party software into the EECS tutor, especially since applets for some of these programs already exist.

3.2 Drawbacks

The limitation of relying on commercial products is that they could fundamentally limit the ability to customize and expand the EECS tutor. There must be a means to access the data from the graphical extension, store it, and return feedback and grade information to the user. None of the commercial products provide such features and creating our own interface to do so would be extremely time consuming and difficult, if not altogether impossible. Creating our own graphical extension would allow for much tighter interaction between the data students enter, the back-end processing, and feedback and solution checking.

Another strike against the commercial products is the extended amount of features that they provide that would not be used. The commercial systems are designed for scientific use, and although the classes that will use the tutor are inherently scientific, problem sets will not expect students to use many of the features that the commercial software includes. By developing our own graphical interface we can keep it "light-weight" providing only the features necessary for the academic curricula of the classes involved. Furthermore, one of the intents of the TEE initiative is to give professors the means to use technology to improve how they present material. By creating an in-house extension, managed by the MIT community, professors will have the freedom and flexibility to change and add to the graphical components.

Chapter 4

DESIGN

A number of important design decisions had to be made to define the requirements for an initial implementation of the graphical extension. In this section we describe some of the key design decisions that went into the graphical extension and the reasoning behind the decisions.

4.1 Problem Scope

Knowing that it would not be feasible to address all the new types of graphical problems at once, we decided to narrow the initial goal to a group of simple, yet useful problems. Our focus for the initial implementation was on three basic families of problems: graphs, vector diagrams, and simple drawings. While this restriction omits a number of important types of problems – circuit diagrams, connected flow charts, etc. – the EECS tutor will still handle a significant percentage of the graphical problems found on problem sets in the Electrical Engineering and Computer Science Department.

4.2 Back-End I/O

Another important design choice was made regarding the architecture of the new extension. Since graphical problems are fundamentally different from the text-based problems previously implemented, the method of interaction between the front and back ends has to be reevaluated. In the scheme applet, the code was extracted as a text string and sent to a back-end evaluator function for processing. The code is run against test cases and compared to results of correct code run on the same test cases. Any feedback or other information to be displayed is passed back as a text string.

This method of text messaging may not be ideal for graphical information, but in order to keep it consistent with the rest of the tutor, we decided to use it as best we could. The text string that is passed between the front and back ends for the graphical extension is a list of variable-value pairs that describe, among other things, how the applet has been configured, as well as the location of all the graphical objects created by the user. These pairs can be stored in a database on the back-end, along with the individual user's status on a problem – still working, complete, seeking feedback, etc. The database can store each student's problem state, allowing students to leave in the middle of the problem without losing their work. Similarly, maintaining state allows the system to log changes to analyze how students are working and monitor the effectiveness of the problems.

Another advantage of using text strings to pass graphical information back and forth is that the syntax of variable-value pairs is simple to process and format. This makes it much easier to write code to operate on the graphical information and check correctness.

Although converting graphical data to text allows the system to maintain problem state and simplifies operations on the graphical answers, it does not completely solve the larger issue of gradability. It is unreasonable to expect student answers to be identical to the correct answer; in fact in many cases, there is a wide range of acceptable answers. Unfortunately, using test cases, as was done with the scheme problems, is not a reasonable solution for graphical problems. Instead of seeking exact correctness, the graphical evaluator function should attempt to determine whether the answer is within an acceptable distance from the correct answer. Furthermore, it should identify the incorrect regions or missing objects in a student's answer and return feedback distinguishing between the correct and incorrect parts of the student's answer.

4.3 Front-End I/O

Another important design choice had to be made regarding the front end user interface. Graphical questions are fundamentally more complex and come in more varieties than the types of questions available in the current version of the EECS tutor. It is simpler and more consistent for the user to have the same type of applet all the time, and always have access to all the functionality. However, as we develop more types of graphical problems, the complexity of the applet will grow. This can lead to a situation

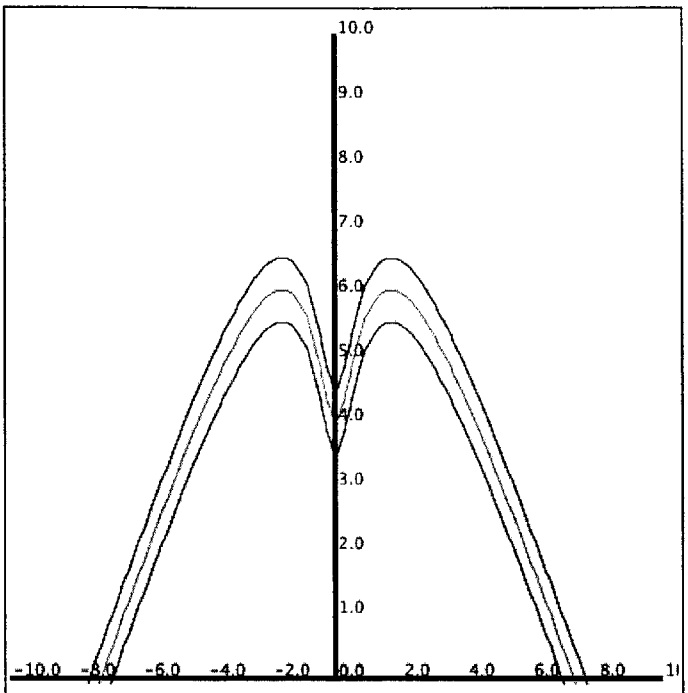


Figure 4.1. An example of acceptable ranges: The red graph is the correct answer. Graphs that stay within the bounds of the blue graphs should be accepted.

where figuring out what features are necessary to solve the problem becomes more difficult than solving the problem itself.

Since the initial implementation will include only a small subset of the eventual features, we decided that consistency outweighed the potential for confusion. However, in the interest of extensibility, as more varied types of problems are added to the EECS tutor, it will make sense to separate among distinct modes – circuit building, graph sketching, free-flow drawing, etc. – where each mode consists of a subset of the total available features. The different modes should look and feel as similar as possible, but by separating features that will never be used together, overall performance and utility should improve. Unfortunately, this type of design puts a burden on implementation, as the feature set must be highly modular to allow for the different overlapping problem modes.

4.4 Features

With the important design decisions made, the next step was to decide what features to include. In order to make the EECS tutor as intuitive as possible, we chose to model the feature set on the commercial drawing software described above. With three problem types in mind - graphs, vector diagrams, and simple drawings – we formulated two groups of the high-level functionality for the graphical extension. The first group is the back-end interface necessary to set up a problem from the text inputs to the applet. This group depends on text input from the HTML parameters in the web page containing the applet. The second group is the user interface and user-side functionality. Most of the functionality in this group relies on graphical interactions using the mouse and keyboard within the applet:

Setting up a Problem (back-end interface):

- A means to determine the size of the graphic applet
- A means to designate which mode of the applet to use
- A means to disable features included in the current mode
- A means to display a picture file in the background of the drawing area
- A means to display or hide the axes, and define the scale of the axes
- A means to display or hide tick marks, and determine the number of ticks
- A means to display or hide gridlines
- A means to place predefined objects in the drawing area
- A means to designate predefined objects as read-only

Using the Apple (front-end):

- A means to create curved lines, straight lines, vectors, circles, rectangles, multi-sided regions, and text labels
- A means to create multiple instances of each of the objects listed above
- A means to place points graphically by clicking on the screen
- A means to select an entire object and move it using the mouse
- A means to select a point in an object and move it using the mouse
- A means to cut, copy, and paste an object using the keyboard
- A means to change the color of an object
- A means to name an object
- A means to delete an object
- A means to delete a point in an object

Chapter 5

FRONT-END IMPLEMENTATION

Although the design for the graphical extension is fairly straightforward, in order to make the new code as robust and extensible as possible, a large emphasis was placed on modularity in the class hierarchy. The applet itself acts mainly as a buffer between the web page and lower level implementations of the Problem Mode interface. Since only one problem mode exists for the initial release of the graphical extension, the benefits of this type of set up are minimal. However, as more problem modes are added, it should be easier to integrate them into the existing code.

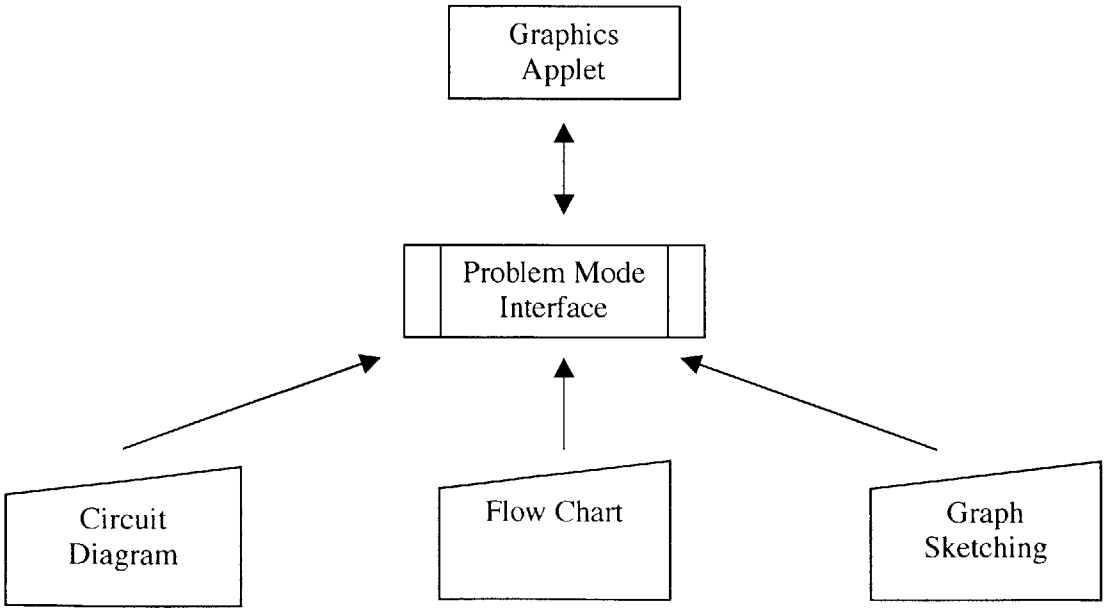


Figure 5.1. An implementation diagram of the high-level graphical interface.

5.1 Initialization

Each problem mode sets up the graphical interface that the user will have available. That interface consists of two parts – a toolbar and a drawing area. The toolbar runs down the left side of the applet and contains the buttons available to the user. These buttons are inherent to the problem mode, but can be disabled during setup if so desired. At the bottom of the toolbar is a color palette. The drawing area lies to the right of the toolbar and contains the graphical representation of the problem.

During setup, the applet is passed a series of parameters that define or customize the toolbar and drawing area. As mentioned above, one of these parameters denotes which buttons are disabled. The drawing area has parameters to specify axes, tick marks, and grid lines. In addition, a URL to a picture file can be passed as a parameter. The picture file will appear as the background of the drawing area. A more detailed explanation of the different parameters is available in Appendix A.

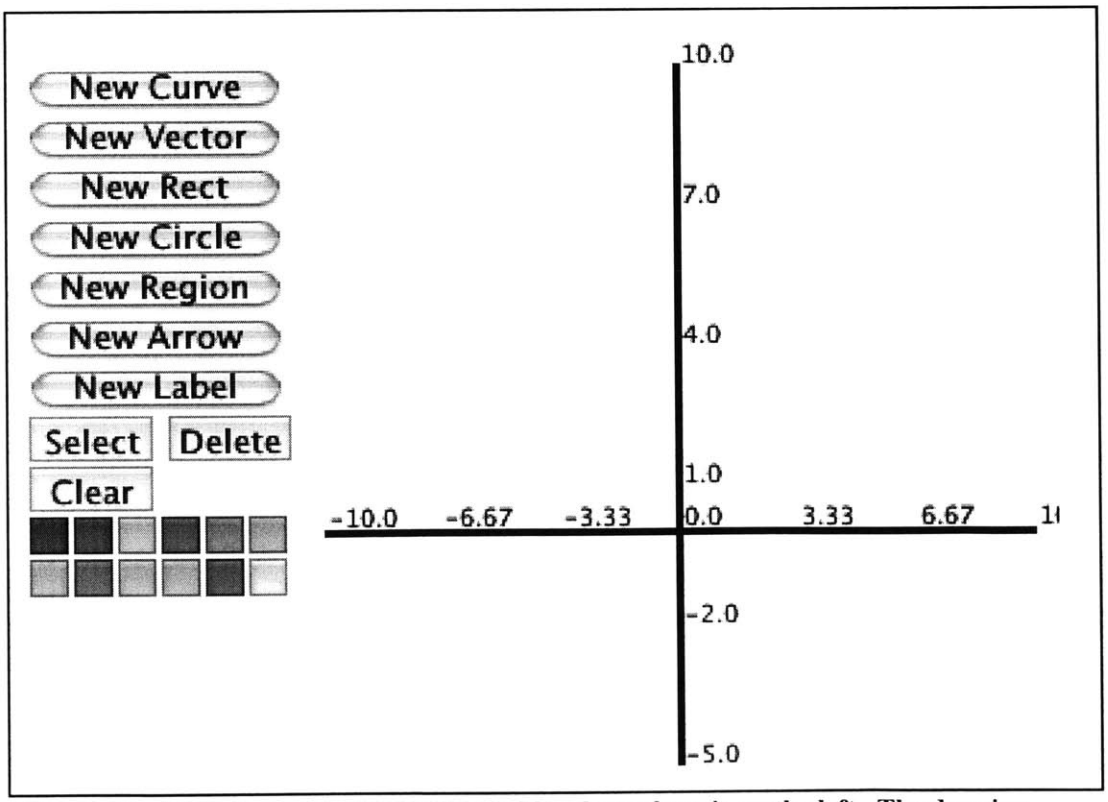


Figure 5.2. A blank applet. The toolbar with color palette is on the left. The drawing area with visible axes is on the right.

5.2 Creating Drawable Objects

A hierarchical structure similar to the one used at the high level of the applet, was used to implement the different types of objects that can be drawn. A high level class of Drawable Objects handles the functionality shared across all the different graphical objects. Each individual object extends from the Drawable class, overwriting methods when necessary, and adding the specific details that distinguish one type of object from another. In our initial version we included 7 objects – Curves, Lines, Rectangles, Circles, Regions, Vectors, and Labels. As more objects are added it will probably make sense to add another level of hierarchy as the differences between the types of objects grow.

Contained within the descendants of the Drawable class is the mathematical representation of the graphical objects. At a high level, all the objects are defined by two arrays of coordinates – one for x and one for y - of each point in a Drawable Object. When a new object is created, the corresponding Drawable Object descendant is instantiated with two blank arrays. As the user clicks to place points, the coordinates of the points are added to the arrays. Moving an entire object adds an offset to all the members of the coordinate arrays, while moving and deleting specific points affects all the indexes involved.

Although the high-level storage among all the objects is the same, each one has its own method of onscreen display. In the simplest cases, Vectors and Regions, a line segment is drawn from each point to the next; in a Region a final line is drawn from the last point back to the first. For Arrows, only two points are allowed. The points are connected with a line segment and an arrowhead at the second point. For Rectangles

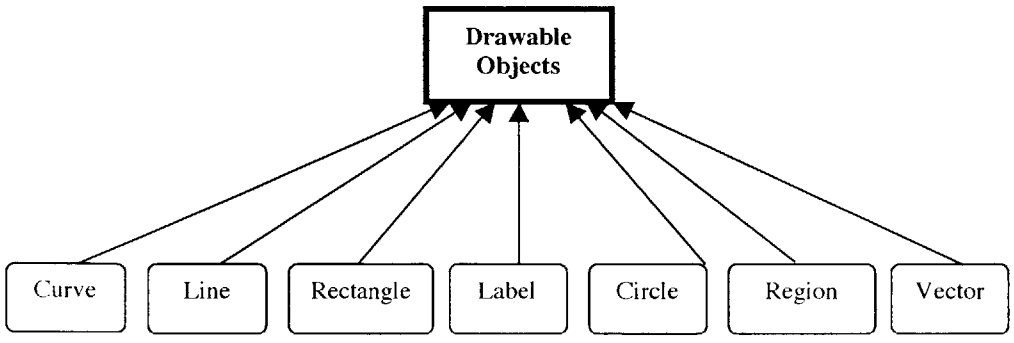


Figure 5.3. The current drawable object hierarchy,

only two points are allowed again, this time representing a pair of opposite corners in the rectangle. The other corners are calculated and the four points are stored in counter-clockwise order. Line segments connect the four points as they would the Region.

For Curves and Circles, it was necessary to draw best-fit spline curves to connect the points. There are a number of different methods to formulate the spline between points. We adopted the code for spline fitting from a pre-existing model, which creates arcs between each pair of adjacent points. Each arc is dependant on all the points in the curve, but points closer to the endpoints of an arc have a greater effect on the curvature of that arc than those further along the curve. As a result, adding a new point to the end of a curve affects all previous arcs, but the effect is dampened as it travels down the curve. [7] The points on a Curve are connected using this spline-fitting formula. Graphically, the curved lines are actually drawn as very small lines that visually appear to be curved.

Circles are perhaps the most complex objects to draw. As with Rectangles, only two points are allowed, and they define the opposite corners of a Rectangle. Again, the other two corners are calculated and the four points are stored counter-clockwise. A curve is then fit connecting the four corners, and back to the first. However, due to nuances in the spline-fitting formula, this Circle is very flat, looking more like a rounded

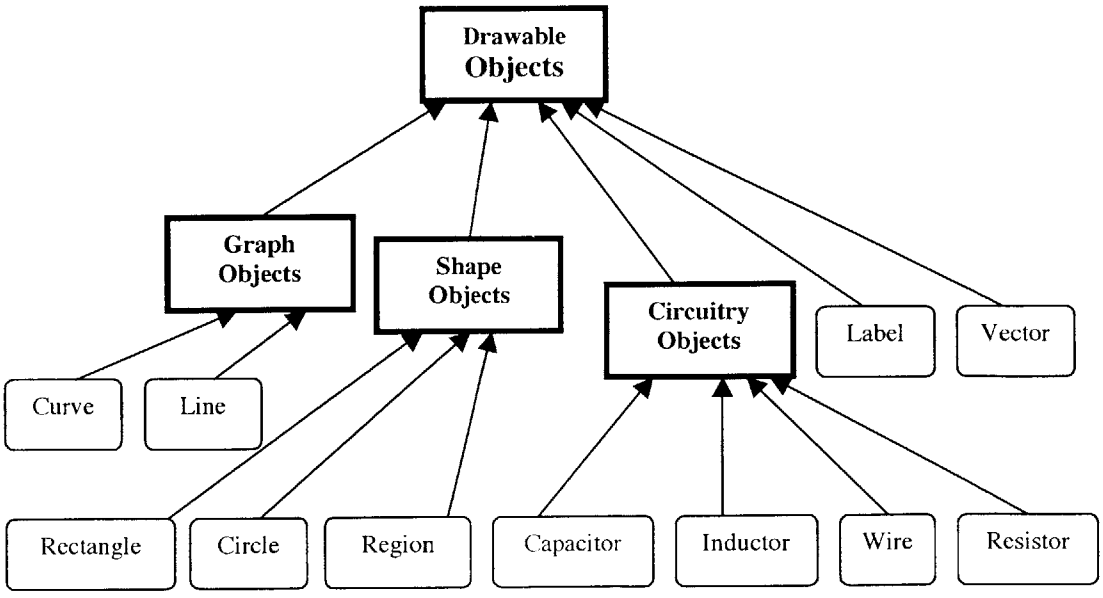


Figure 5.4. A 2-tiered drawable object hierarchy for future versions of the graphics applet.

rectangle than a circle. In order to give the Circle more curvature, the four stored points are duplicated three times, so the Circle contains 12 points. The middle four points are the only points that are drawn, and they appear much rounder than the first four points. Any changes to the actual four points of the Circle are echoed to the corresponding duplicates,

Labels are fundamentally different since they require both mouse and keyboard input. Graphically, Labels can be thought of as text strings inside an invisible Rectangle. When the user clicks the first time, the Label is anchored, with its bottom left and right corners at the point clicked. The top two corners have the same x-coordinate, but the y-coordinate is offset by the font size. As the user types, a key listener adds the characters to a text string until the user types the Return key. As characters are added, the Label's right two corners add the offset of each character, expanding the box to contain the string.

5.3 Manipulating Drawable Objects

Manipulating the different objects is done both at the Drawable Object level and within the individual objects. When the user presses the Select button in the toolbar and clicks within the drawing area, each object checks to see how close that point is to the area it comprises. For line based objects – Vectors, Rectangles, Arrows, and Regions – the object goes through each line segment, and checks the distance from the line to the point clicked. For the spline-based objects – Curves and Circles – a similar check goes on, except instead of using the lines between points, it calculates the distance between the point clicked and the small lines making up the arcs of the spline. Labels are the easiest to handle, as they check if the point clicked lies anywhere inside the bounding rectangle. After all the objects are checked, the closest one is stored as the most likely choice to be selected. If the distance to the closest object is within a threshold, then that object is selected; otherwise nothing is selected.

Once an object is selected dragging the mouse will move it around the drawing area. As the mouse is dragged, the coordinates of the object are constantly updated by the offset from the current mouse position to the location of the original click. When the mouse is released, the updated offset coordinates are set, and the offset returns to zero.

A selected object can also be cut, copied, or pasted using the keyboard, mimicking those commands as found in commercial drawing programs. When an object

is selected, CTRL-C copies it, CTRL-X cuts it, and CTRL-V pastes it at a small offset to the lower right of its original location. A selected object can also be permanently removed by pressing the remove button in the toolbar.

Selecting and moving individual points involves similar checks to selecting and moving entire objects. Once an object has been selected, clicking on it again causes the object to check the distance between the mouse position and all of its points. Again, if the distance to the closest point is under some threshold, then that point is selected. When a selected point is dragged, its coordinates are updated by the offset from the current mouse position. All other points remain static unless they have direct constraints with the point being moved. Moving a point in a Vector, Curve, Region, or Arrow has no effect on the other points. The adjacent corners of a Circle or Rectangle, however, are adjusted to maintain the shape when a point on those objects is moved. Individual points on a Label cannot be selected. Although individual points cannot be cut, copied, or pasted, they can be deleted using the Remove button in the toolbar. Specific instructions on how to use the graphical components of the EECS tutor are available in Appendix B.

Chapter 6

BACK-END IMPLEMENTATION

The back-end development consists of three major parts – data storage, the solution engine, and the I/O between the user interface and the stored data. Graphical data is converted into a text string, which is then stored in the storage database. Depending on the user's actions, the data is run through the solution engine, and the results and feedback are updated in the database. The current state is then downloaded from the database, and sent as a text string back to the front-end where the text is processed and converted back into graphical data.

6.1 Graphics-To-Text Interface

Since the front-end implementation relies heavily on graphical and mathematical constructs, and the back-end database and solution engine rely on textual representations, it was necessary to devise methods to translate from the text-based graphical interface to the back-end and vice versa. As described in the Front-End Implementation, each Drawable Object can be viewed at a high level as an array of coordinates. Other information, like the object's type, color, name, and label text are all stored within the object. Each object has a method that returns a list of name-value pairs that uniquely define the object. In addition, the applet itself can export its own defining list of name-value pairs, containing information about the axes, grid lines, tick marks, background pictures, and applet dimensions.

When a user saves the page, checks his current answer, or submits his final answer, a JavaScript command is called which asks the applet to return a text definition of its current state. The applet produces three strings: a list of all the read-only objects, a

list of all the non read-only objects, and the applet's own definition list. The complete string of all three lists is sent to the back-end for storage and processing.

The reverse process takes place when a page is loaded. Three parameter lists are passed to the front-end which augments the graphical information with the name-value pairs of the different lists. The Applet Settings list, as described in the Front-End Implementation section, defines the applet's size, toolbar settings, and overall look. The other two parameters, UserObjects and OverlayObjects provide the graphical information for the objects in the drawing area. The two lists differ in that OverlayObjects are read-only, while the user is free to modify UserObjects. The syntax of each input object is the same list of name-value pairs that the applet outputs, as described above. A detailed explanation of the different parameters and Drawable Object syntax is available in Appendix A.

The following piece of code, similar in format to XML, was used on an example problem in the spring 2003 6.034 class. When the page is loaded, a scheme interpreter creates the list format required of OverlayObjects from the data set definition. The remainder of the code sets up the Applet Settings and OverlayObjects parameters, as well as the problem solution:

```
(define data
   (apply append
      (map (lambda (point)
         (let ((class (data-point-class point))
         (feat (data-point-features point)))
      `((label (color ,(if (= 1 class) -16776961 -65281))
                  (text ,(if (= 1 class) '+ '-))
                  (coords ,@feat))
         (circle (color ,(if (= 1 class) -16776961 -65281))
                  (coords ,@(map (lambda (a) (- a 0.05)) feat)
                           ,@(map (lambda (a) (+ a 0.05)) feat))))))
         *data*)))

(define-problem
   '((type graphics)
      (title "1 Nearest Neighbor Boundaries")
      (intro "Draw in the 1-nearest-neighbor decision boundary for these points.")
      (settings
         (sample 100) (axes on) (grid on) (ticks on) (xmin 0) (ymin 0) (ymax 7) (xmax 7)
         (xticks 8) (yticks 8) (buttons vector))
      (background @data)
      (initial-display)
```

```
(answer
    "(line (color -65536) (name Answer) (coords 0 5  2 3  2.5 3  3 3.5  3 5.5  3.75 7))
    (line (color -65536) (name Answer) (coords 5 7  5 3.5  5.3 2.9  4.75 2.25 7 0))"
    (rationale "That's the nearest neighbor decision boundary.")
    (compare (lambda (input ans) (test-graphics input ans 0.5))))))
```

6.2 Data Storage

One of the important features of the EECS tutor was the concept of maintaining problem state. Many of the other online systems have only a minimal notion of problem state – either correct or incorrect. By including information about the student's action history, the latest version of his answer, the most current feedback, in addition to correctness information, the EECS tutor can be more interactive and more useful both to the student and the professor. In addition, it gives feedback about how students are using the tutor, which is useful in improving the problems and the tutor, itself.

The actual back-end database consists of two log files – one for cumulative changes and one for the current state. When a student first submits a problem, an entry is made in each log. Further submissions result in updates to the entry in the current state log, and new entries into the cumulative log. Each entry in a log contains the following information:

Time Code – the date and time that the entry was submitted
User ID – each user of the system has a unique identifier
Problem ID – each problem in the system has a unique identifier
Hints - some problems give hints in the form of text or graphs
Graphical State – what the applet looks like: Applet Settings, OverlayObjects, and UserObjects
Trace Data – a list of the user's actions since the last entry, including button presses, mouse clicks, and key presses.
Correctness – the current score and the maximum score for the problem, e.g. 8 out of 10
Graphical Feedback – a list of objects, in the same format as OverlayObjects, to return as feedback
Textual Feedback – a text string to return as feedback

When a user begins work on a problem, the databases have no entries for him. Saving the problem adds an entry to both logs containing only the fields available from the front-end i.e. no Correctness, or Feedback fields. When a user checks the problem, a new entry is inserted into the cumulative log and the entry in the current state log is updated. The data is sent to the solution engine, which returns values for the Correctness and Feedback

fields. These values are inserted into the two logs and the web page is reloaded with the data from the current state log. Submitting a problem results in the same database manipulation as checking, except the database entries are then locked, preventing further changes to that problem.

Appendix C includes an excerpt from the Cumulative Log containing three entries of work on a sample problem. The first entry is the initial entry for the problem, and occurs after a Save, so it has no feedback values. The second entry is a check immediately after the Save; there is no TraceData, but there is textual feedback indicating missing edge segments. The last entry is another check, after further manipulation. In this case there is both textual and graphical feedback.

Although either table alone would be sufficient to maintain problem state, using the dual-table format is useful for a number of reasons. When a student loads a problem, the load comes from his information in the current state table, so minimizing the number of entries keeps access times low. In the event that data gets corrupted or lost in the current state table, a more time-consuming query can be made to the cumulative table. Furthermore, by keeping all entries in the cumulative log, the professor can determine usage statistics to monitor the difficulty of problems or the trends of specific students or topics.

6.3 Solution Engine

Perhaps the most flexible part of the back-end is the solution engine itself. While gradability is a key component of a successful graphical tutor, it is difficult to devise a method to check correctness that is applicable across all the different types of graphical problems. By having the graphical data go into storage between the front-end graphics and the back-end solution engine, the exact implementation of the solution engine will not affect the rest of the system. As a result, different methods of correctness checking can be used for different problems.

In Chapter 4 we discussed one possible algorithm to grade student answers by checking to see if they fall within an acceptable upper and lower bounds of correct answers. While this method may prove to be the most effective, we used a simpler point-to-point scheme in our initial implementation of the EECS tutor. Instead of comparing the student's answer to see if it falls in a bounded region, we compare it to the exact

answer. Both the student's answer and the correct answer are divided into small sub-segments. The end points of corresponding sub-segments are compared to determine the maximum distance between the two sub-segments. Any sub-segments of the correct answer that are too far from a student answer indicate missing regions in the student's answer. Similarly, sub-segments of the student's answer that are too far from the correct answer indicate incorrect regions in the student's answer. In our implementation, the first type of mistake results in only textual feedback, as was shown in the second entry of the log file in Appendix C. The second type results in both textual and graphical feedback as shown in the third entry in the log file.

Chapter 7

TESTING

There are innumerable instances where a graphical interface will be useful in an online problem set, so the only obstacle to including it in the EECS tutor is the potential that it be too distracting or difficult to use. While it is tricky to determine a numerical or statistical measurement of usability, it is important to have some quantitative metrics. With that in mind, a sample problem was included as an optional problem on the final problem set in the spring 2003 6.034class.

7.1 Problem Make-Up

The problem we chose to include is a nearest-neighbor decision boundary question, where students are given a set of graphical data points and asked to draw in the function separating the two different types of data. On loading the problem set, a piece of XML code sets up the web page with the Applet Settings and OverlayObject data, as well as feeding the correct answer to the solution engine. When the page loads, the applet starts up with the given settings and the student is ready to work.

As students work on the problem they are allowed to check their answer against the correct answer and receive feedback. As described in Chapter 6, two forms of feedback were available in this initial implementation. Solution attempts that were too far from the correct answer were given textual feedback and graphical feedback where the correct and incorrect regions of their answer were redrawn in green and red, respectively. Solution attempts that left out regions from the correct answer were given only textual feedback. Figures 7.1-7.5 illustrate the workflow of the problem

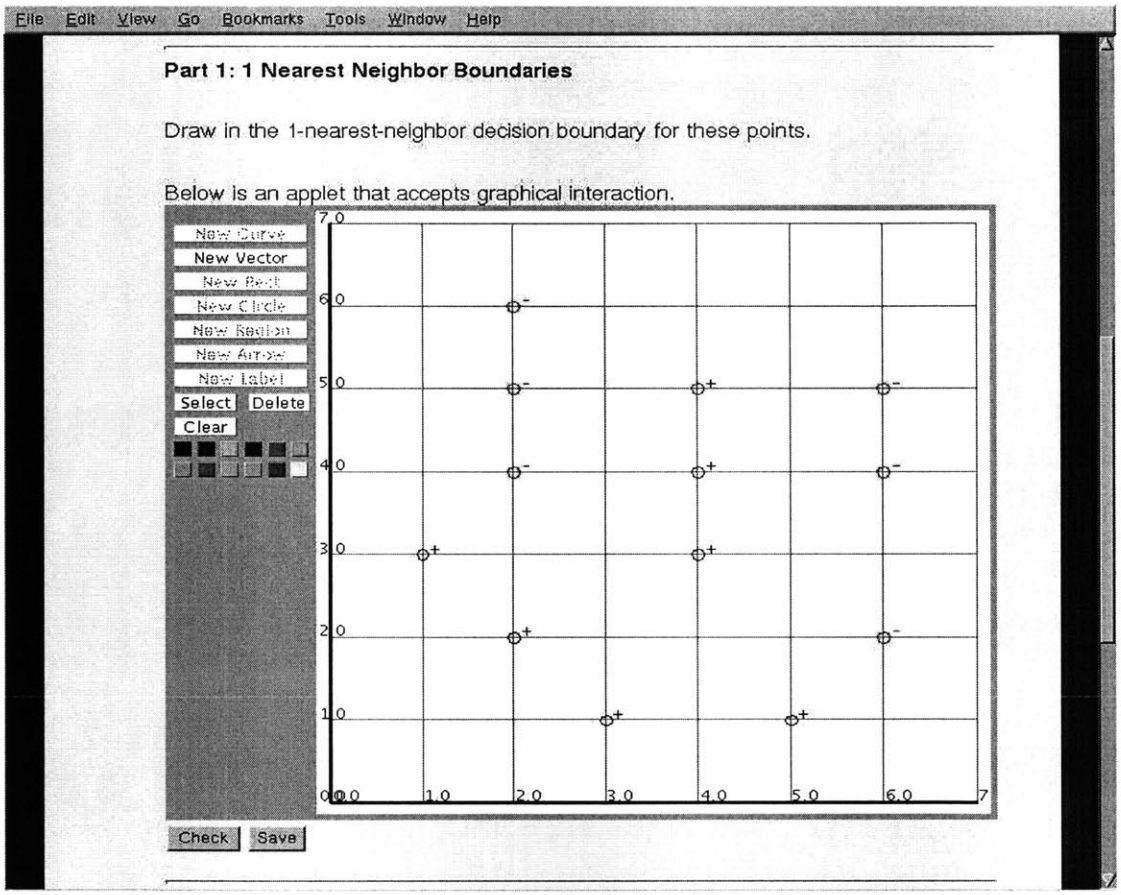


Figure 7.1. A graphical problem from an Artificial Intelligence problem set.

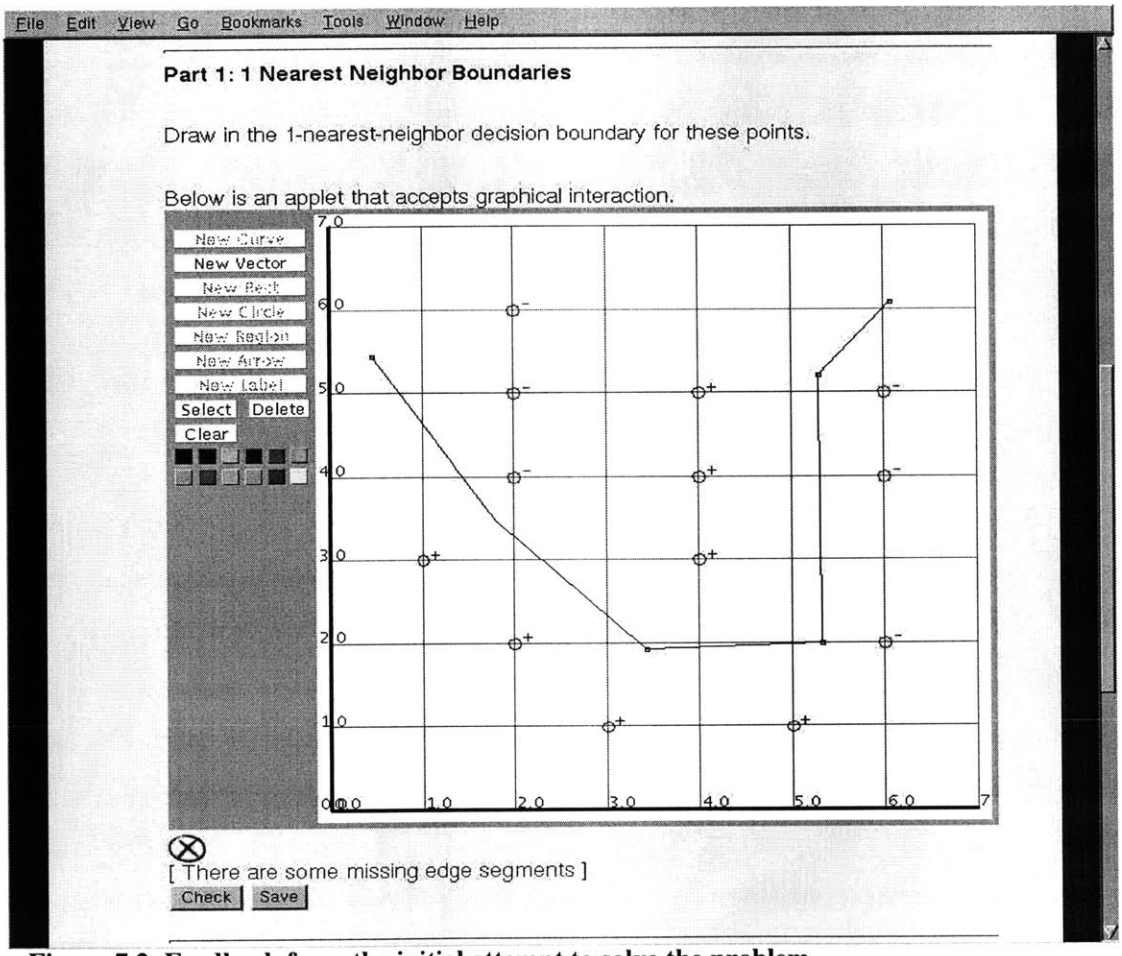


Figure 7.2. Feedback from the initial attempt to solve the problem.

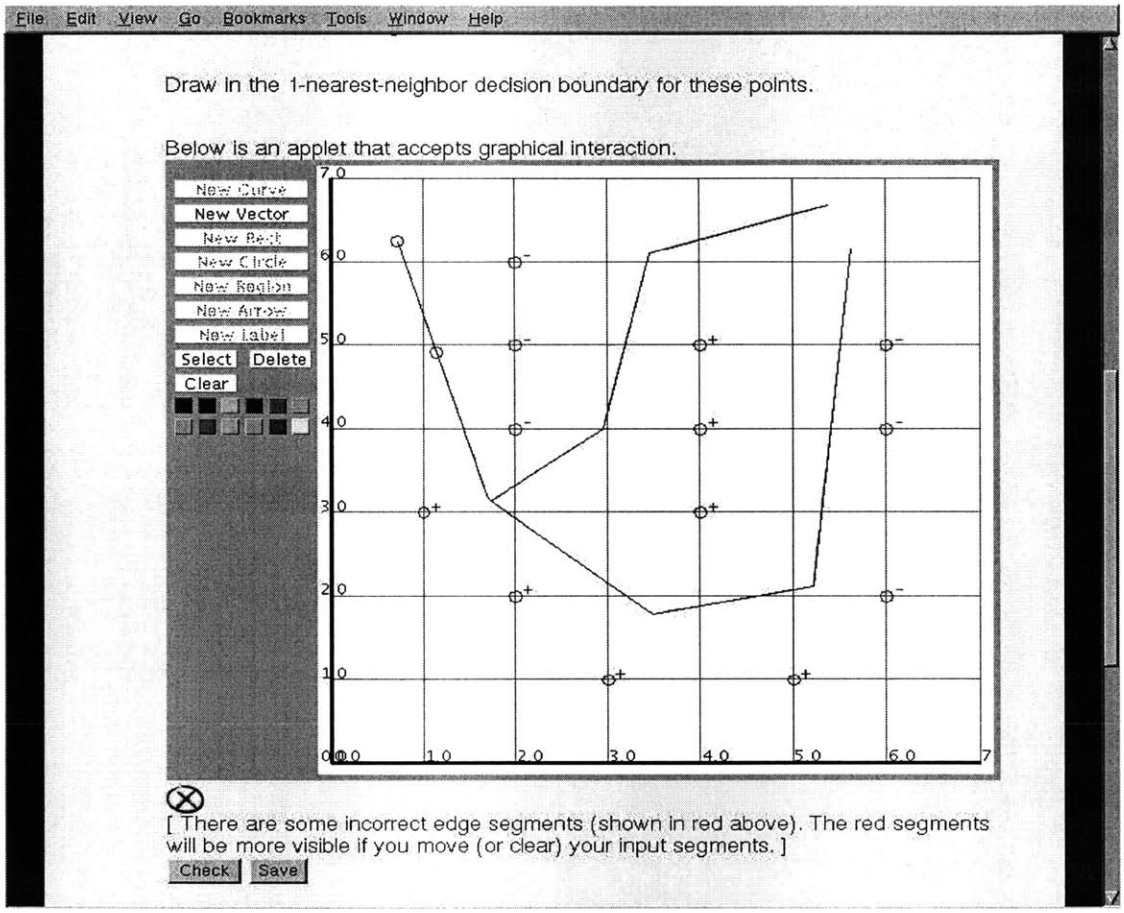


Figure 7.3. Feedback from the second attempt to solve the problem. The red circles indicate regions that are too far from the correct answer.

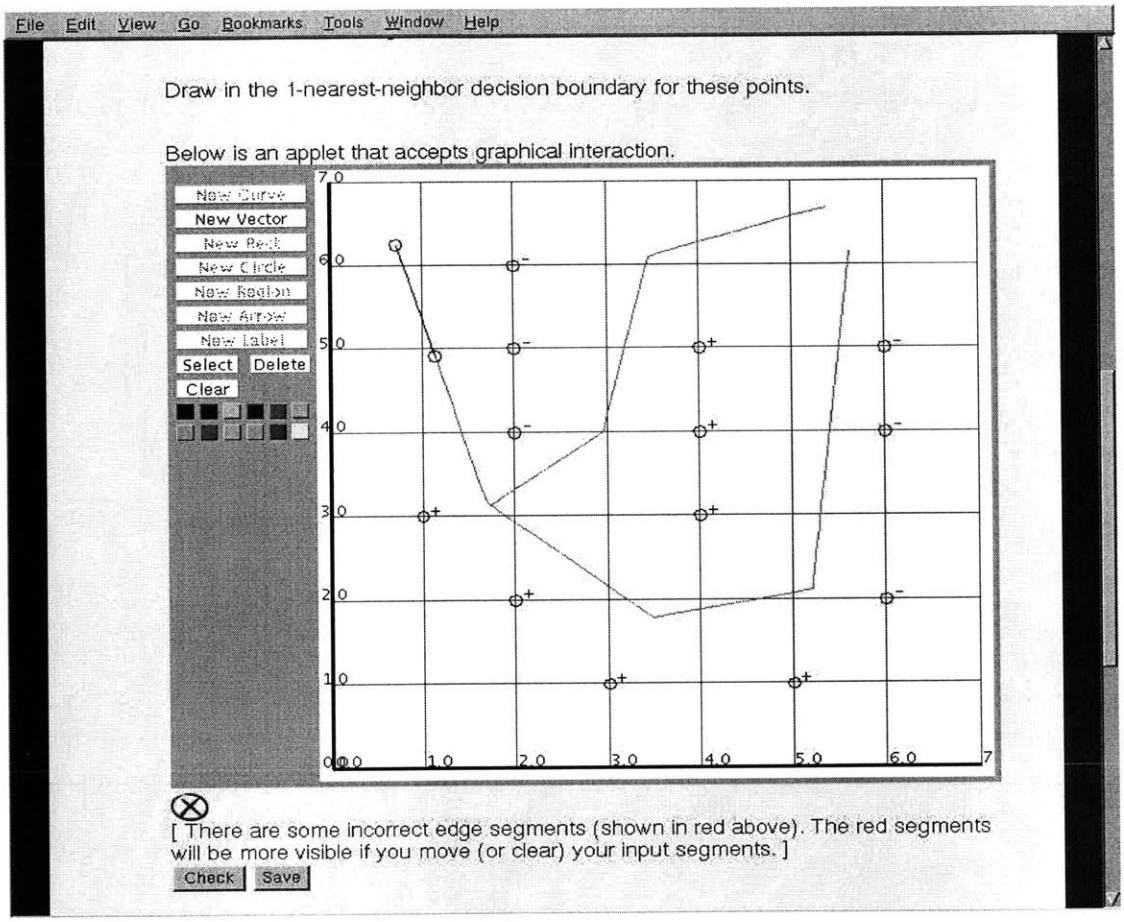


Figure 7.4. Feedback from the second attempt to solve the problem. The student's answer has been erased to show only the graphical feedback. Green lines indicate correctly placed regions of the student answer. Red lines indicate incorrectly placed regions.

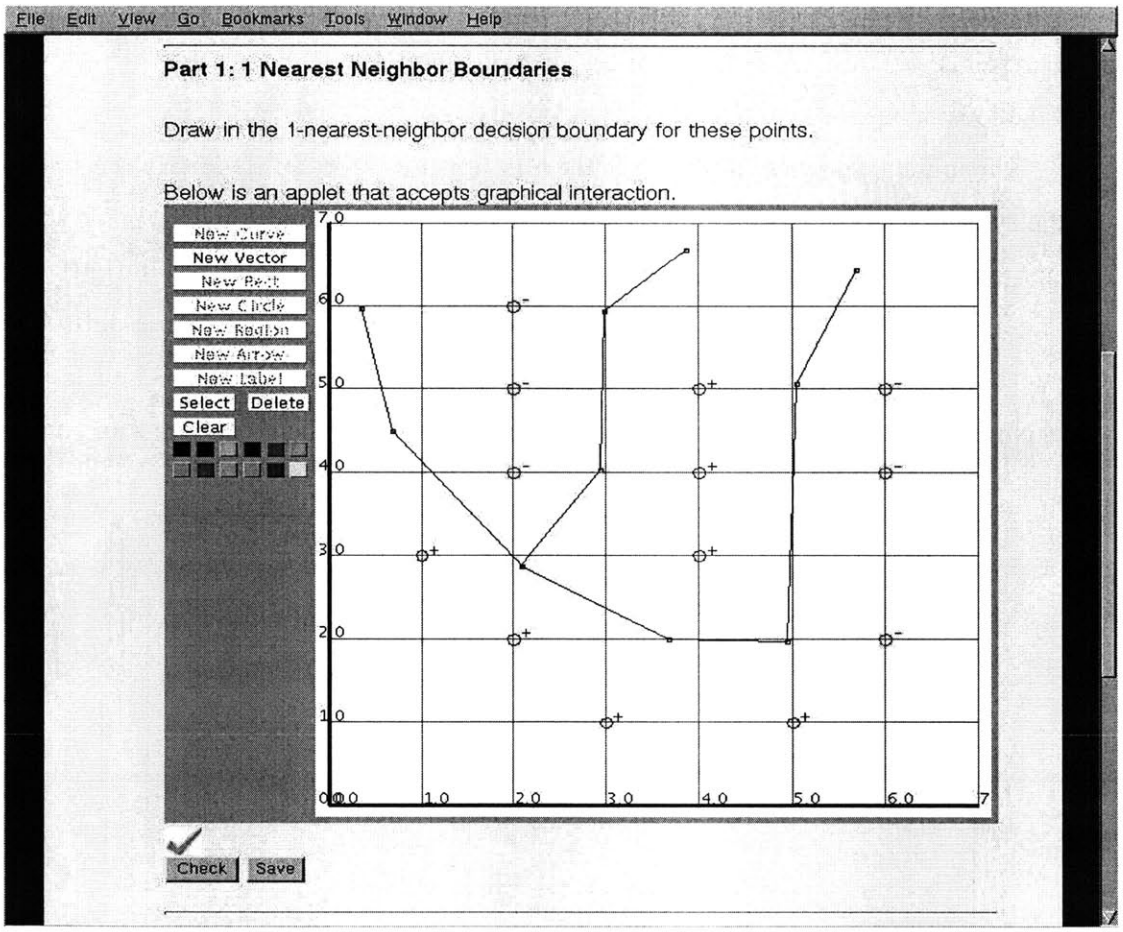


Figure 7.5. Feedback from a third attempt at a solution. The check mark indicates that this is indeed the correct answer.

7.2 Results

Results of the first run of the graphical interface on a live system were mixed, but promising. We obtained results in two ways – first by asking for student reaction and opinion after they used the system; second by logging how effectively the students were able to use the system. The general response data revealed that a large percentage of students were unable to load the applet. While we had anticipated that there would be some systems that would be problematic, we were surprised at the number of different combinations of hardware, operating systems, and browsers that failed.

The logged data was significantly more promising. Over 30 students were able to use the system effectively enough to check their answer at least once, and 19 made some attempt to actually solve the problem. The usage statistics are summarized in figure 7.6 and are available in full in Appendix D. While it is difficult to determine definitive answers from the limited sample space, a number of trends do emerge. 11 of the 19 students were able to solve the problem rather easily – in fewer than 4 attempts and under

User	Trials	Mouse Clicks	Removes	Clears	Solved
USER1	1	14	0	0	YES
USER2	1	20	2	1	YES
USER3	1	26	3	1	YES
USER4	1	33	4	1	YES
USER5	1	57	2	15	YES
USER6	1	59	8	3	YES
USER7	2	21	0	0	YES
USER8	3	56	0	1	YES
USER9	3	72	14	3	YES
USER10	4	31	1	3	YES
USER11	4	40	0	1	YES
USER12	4	66	0	0	NO
USER13	8	132	16	4	YES
USER14	9	196	7	2	YES
USER15	10	144	12	25	YES
USER16	11	223	12	0	NO
USER17	12	130	19	16	YES
USER18	15	224	18	16	YES
USER19	16	202	25	5	YES
19	**107**	**1746**	**143**	**97**	**17**

Figure 7.6. A chart of usage statistics from the graphical problem on an Artificial Intelligence problem set.

75 clicks. Of the remaining 8 students, 2 gave up, and the other 6 averaged over 11 attempts and 175 clicks to solve the problem. Looking at the more detailed statistics suggests that the main difficulty was that the feedback was insufficient. It appears that students who did not get it right immediately wasted a lot of effort trying to do the same thing over and over, because they were unsure where they were in error. If this is the case, then the graphical and textual feedback must be more precise or illustrative than previously thought.

The 11 students who solved the problem relatively easily give a decent sense of the learning curve. The large numbers of clears and removes from users 5, 6, and 9 suggest that they had difficulty manipulating the vectors at first, but were able to figure it out within 50 clicks. While a 50-click learning curve for a "fast" student may seem steep, given the fact that the students were unprepared for the system and had little guidance in how it worked, it is acceptable. As the graphical interface becomes a regular part of the EECS tutor, the wasted clicks per problem should go down quickly as students become accustomed to using the new interface.

Chapter 8

CONCLUSIONS

The work on adding a graphical extension to the interactive EECS tutor has been extremely productive. Unfortunately, a number of unexpected pitfalls were discovered in the later stages that have slowed the integration of the graphical extension into the system as a whole. The majority of the effort was spent generating a front-end user interface that is intuitive to the user and provides enough features to be useful for a number of classes. For the most part, that aspect has been accomplished. The graphical extension to the EECS tutor correctly handles problems involving sketching, boundary graphs, and simple drawings. Most of the graphical interaction is intuitive and similar enough to commercial products that students should not be distracted or confused.

Most of the emphasis up to this point has been concentrated on the front-end implementation. The back-end, especially the solution engine, is very skeletal. In our first experimental problem in 6.034, neither the solution checking nor feedback was optimal and that appears to have contributed to the difficulties students had using the graphical components of the tutor. Another problem that was more significant than we anticipated is the lack of compatibility with various operating systems and web browsers. There were known incompatibilities with the original system, but the graphical extension is incompatible with more hardware-software combinations, particularly when run over SSL. The most immediate work on the system should address the incompatibility issues and at the very least determine what combinations work and what do not, as well as what changes can be made to increase the number of supportive systems.

Assuming the access issues are manageable, the next step for the EECS tutor is to provide a number of rough templates for graphical solutions in the solution engine. Since

each problem is different from the next and professors want varying degrees of feedback, there will always be some level of overhead in setting up the problems, solutions, and feedback interactions. Over time, a library of different problems and different ways to take advantage of the graphical interactivity will be built, so that new problems can be created quickly and easily on the backs of earlier ones.

A third direction for future development lies in extending the classes of graphical problems supported in the tutor. One of the goals for the graphical interface is to include a notion of connectivity. Including elements that can be connected allows for more complete flow charts, and opens the door for circuit diagrams. Being able to set up and solve circuit problems graphically with instant feedback would be especially useful, as it would give classes in the electrical engineering side of the department an opportunity to take advantage of the EECS tutor.

Appendix A

PROBLEM SETUP

Setting up a problem:
A problem definition consists of 4 parts: applet type, applet settings, overlay objects, and user objects. The applet parameters describe the type of problem (graphs, circuit diagrams, etc.). This parameter is mandatory.

Any associated values associated with that type of problem are included in the applet settings (axes, grids, x-y ranges, available buttons, pictures etc.) These parameters are all mandatory.

The overlay objects are optional pre-defined objects (graphs, shapes, circuits, labels) that appear in the applet. The objects are read-only, so the student cannot modify them.

The user objects are also optional pre-defined objects but they are not read-only so the student can modify them.

Applet Tag Parameters:
Name - user's choice

Code - Plot2D.class

Archive - Plot2D.jar

Dimensions - The left control margin is always 100 pixels wide, by 290. The minimum size of the graphing area is 275 pixels wide by 290 pixels high. If there is room, the width of the graphing area is the total width minus 115 pixels, and the graphing height is the total height minus 10. So, in order to have a square graphing area of NxN, the applet should be N+115 wide by N+10 high.

Example - <Applet name ="MyApplet" code = "Plot2D.class" width =615 height = 510>

AppletType Parameters:
Only "graph" is accepted at the moment.

Example - <PARAM name="AppletType" value="Graph">

AppletSettings Parameters:
A scheme list of inputs including the following:

axes on/off - are the axes visible (default true)

grid on/off - is the grid visible (default true)

ticks on/off - are the tick marks and values visible (default true)

naming on/off - can objects be named (default false)

xticks - the number of ticks along the x axis (default 5)

yticks - the number of ticks along the y axis (default 5)

round - the number of digits after the decimal to round tick values (default 2)

xmin/xmax - the minimum and maximum x values of the graph (default -5, 5)

ymin/ymax - the minimum and maximum y values of the graph (default -5, 5)

buttons - what buttons are enabled for the user. Possible values (curve, vector, rect, circle, region, arrow, label). If the button is not listed, it is disabled.

picture - the exact URL of a background picture file

Example - <PARAM name="AppletSettings" value="((sample 100) (round 2) (naming on) (axes on) (grid off) (ticks on) (xmin -10) (ymin -10) (ymax 10) (xmax 10) (xticks 11) (yticks 11) (buttons curve vector label) (picture http://www.yahoo.com/image.gif))">

OverlayObjects Parameters:
A scheme list of objects that the user cannot modify, which will load automatically. Each Object is a sub-list containing the color, coordinates, name (optional), and in the case of labels, text. Objects are of the form:

(Curve (color color_int) [(name text_string)] (coords x y x y x y ...))

(Vector (color color_int) [(name text_string)] (coords x y x y x y ...))

(Rect (color color_int) [(name text_string)] (coords x y x y)) - two opposing corners of the rectangle.

(Circle (color color_int) [(name text_string)] (coords x y x y)) - two opposing corners of the circle.

(Region (color color_int) [(name text_string)] (coords x y x y x y...))

(Arrow (color color_int) [(name text_string)] (coords x y x y)) - tail coordinate, head coordinate

(Label (color color_int) [(name text_string)] (text text_string) (coords x y)) - bottom left point of the label.

Example - <PARAM name="OverlayObjects" value="((arrow (color -1677209) (name This is a sample arrow) (coords 0 2 8 0)) (label (color 0) (text +) (coords 2 2.5)) (label (color 0) (text -) (coords 4 2.5)) (label (color 0) (text -) (coords 6 2.5)) (label (color 0) (text +) (coords 8 2.5)))" >

UserObjects Parameters:
Identical to OverlayObjects except names are not available.

Accessing Problems:
The Applet has a method getText() which can return different amounts of information.
Calling getText(inputs) will return the AppletSettings scheme list.
Calling getText(graphs) and getText(overlayGraphs) will return the corresponding scheme lists.
Calling getText() will return a scheme list of all three lists as well as the getData() list..

The Applet also has a getData() function that will return a string of all the button presses and mouse clicks, and key presses.

Appendix B

USING THE APPLET

Creating a new Curve - To create a new curve, click the "New Curve" button. This will begin a new series of connected curve segments. Click once in the drawing area to place the first point. Each additional click will add a new point and connect it to the existing segments using a best-fit curve.

Creating a new Vector - To create a new vector, click the "New Vector" button. This will begin a new series of connected line segments. Click once in the drawing area to place the first point. Each additional click will add a new point and connect it to the existing segments.

Creating a new Rectangle - To create a new rectangle, click the "New Rect" button. A rectangle is defined by two opposing corners. Click once in the drawing area to place one corner of the rectangle. Click again to place the opposite corner of the rectangle.

Creating a new Circle - To create a new circle, click the "New circle" button. A circle is defined by the rectangle it circumscribes. Click once in the drawing area to place the first corner. Click again to place the opposing corner of the rectangle. The circle intersects this rectangle at all four corners.

Creating a new Region - To create a new region, click the "New Region" button. This will begin a new series of line segments, closed to contain a region. Click once in the drawing area to place the first point. Each additional click will add a new point and connect it to the existing segments with a line segment. Right-click to place the final point. This point will be connected to the previous point and the initial point to close the region

Creating a new Arrow - To create a new arrow, click the "New Arrow" button. Two points define an arrow. Click once in the drawing area to place the first point (tail). Click again to place the second point (head). The two points will be connected with an arrow from tail to head.

Creating a new Label - To create a new label, click the "New Label" button. Click once

in the drawing area to place the label. The bottom left corner of the text will begin at the point clicked. Begin typing the text of the label, which should appear on screen. Use the delete key to delete the last character typed. Press return when the text is completed.

Selecting an Object- To select an object, click the "Select" button and then click on the Object you want to select. The end points of the object will highlight to show that it has been selected.

Selecting a Point - To select a point, first select the Object as described above. Click again on the highlighted point to be selected. The other points will be deselected leaving only the desired point highlighted. Points on a label cannot be selected.

Moving an Object- To move an Object in the drawing area, first select the Object as described above. Click on the Object again and hold the mouse button. Drag the mouse to move the Object and release it to place it. The entire Object will move as one – it will not be reshaped.

Reshaping an Object - To reshape an Object, first select the point to move. Click on the selected point and hold the mouse button. Drag the mouse to move the point, and release to place it.

Changing Colors - To change the color of n Object, first select it as described above. Once the Object is selected click on any of the colors in the color palette to change the Object color. The color of individual line segments and points cannot be changed.

Removing an Object- To delete an Object, select the Object as described above. Then click the "Delete" button to remove the Object.

Removing a Point - To delete one point in an Object, select the point as described above. Then click the "Delete" button to remove the point. If an Object becomes under-defined by removing the point, the entire object will be deleted. Deleting a point from a circle, rectangle, or arrow will always remove the entire Object. Curves and Regions must have at least three points. Vectors must have at least two points.

Clearing the Drawing Area - To clear the drawing area click the "Clear" button. All vectors will be removed.

Naming an Object - This is only available when setting up a problem. First select an Object, then click the "Name" button. A text box should appear next to the origin point of the Object. Begin typing the name, using the delete key if necessary, and press return when finished. The name will appear alongside the origin point of the Object only when the Object is selected.

Copying an Object - To copy an Object, select the Object as described above. Once the Object is selected, press CONTROL-C on the keyboard to copy.

Cutting an Object - To cut an Object, select the Object as described above. Once the Object is selected, press CONTROL-X on the keyboard to cut.

Pasting an Object – To paste an Object, it must first be cut or copied as described above. Once an Object is cut or copied, press CONTROL-V to paste it. The pasted Object will appear slightly below and to the right of original Object.

Appendix C

DATA STORAGE LOG

The following are three entries from the Data Storage Log. The initial entry occurs after the problem is saved after some work. There is TraceData but no feedback. The second entry is after the problem is immediately checked. Note the lack of any new TraceData, but there is feedback text indicating that there are missing entries. The last entry is after further changes and another check. Once again there is feedback text indicating that incorrect regions have been drawn in red, as well as feedback graphics.

```
(1053019486 ps-problem "PS.13.2.1" () () ((hints ())
(coding-hints ()) (subs (0))) (("((AppletSettings ((sample
100) (grid on) (ticks on) (axes on) (xmin 0.0) (ymin 0.0)
(xmax 7.0) (ymax 7.0) (xticks 8.0) ( yticks 8.0)))
(OverlayObjects ((circle (color -16776961) (coords
0.9479167 2.9604168 1.0354167 3.0625)) (circle (color -
16776961) (coords 1.9395833 1.9541669 2.0416667 2.05625))
(circle (color -16776961) (coords 2.9458334 0.9625001
3.0479167 1.0500002)) (circle (color -16776961) (coords
3.9375 2.9604168 4.039583 3.0625)) (circle (color -
16776961) (coords 3.9375 3.9520833 4.039583 4.054167))
(circle (color -16776961) (coords 3.9375 4.958333 4.039583
5.0604167)) (circle (color -16776961) (coords 4.94375
0.9625001 5.045833 1.0500002)) (circle (color -65281)
(coords 1.9395833 3.9520833 2.0416667 4.054167)) (circle
(color -65281) (coords 1.9395833 4.958333 2.0416667
5.0604167)) (circle (color -65281) (coords 1.9395833 5.95
2.0416667 6.0520835)) (circle (color -65281) (coords
5.9354167 1.9541669 6.0375 2.05625)) (circle (color -65281)
(coords 5.9354167 3.9520833 6.0375 4.054167)) (circle
(color -65281) (coords 5.9354167 4.958333 6.0375
5.0604167)) (label (color -16776961) (text  +) (coords
0.9916667 3.0041666)) (label (color -16776961) (text  +)
(coords 1.9979167 2.0124998)) (label (color -16776961)
(text  +) (coords 2.9895833 1.0062499)) (label (color -
16776961) (text  +) (coords 3.9958334 3.0041666)) (label
```

(color -16776961) (text +) (coords 3.9958334 4.010417))
(label (color -16776961) (text +) (coords 3.9958334
5.0020833)) (label (color -16776961) (text +) (coords
4.9875 1.0062499)) (label (color -65281) (text -) (coords
1.9979167 4.010417)) (label (color -65281) (text -)
(coords 1.9979167 5.0020833)) (label (color -65281) (text
-) (coords 1.9979167 6.008333)) (label (color -65281) (text
-) (coords 5.99375 2.0124998)) (label (color -65281) (text
-) (coords 5.99375 4.010417)) (label (color -65281) (text
-) (coords 5.99375 5.0020833)))) (UserObjects ((line
(color -16777216) (coords 0.10208333 4.666667 1.4583334
3.3833334 2.3770833 2.6541667 3.325 2.1 4.725 2.31875
5.2791667 4.010417 5.1916666 6.4166665 5.38125 7.0875))
(line (color -16777216) (coords 2.58125 2.6979165 2.7854166
5.2645836 2.975 6.7229166 2.975 6.8541665)))) (TraceData
VectorButton (MousePress 1175.7142 -11167.143)
(MouseRelease 1175.7142 -11167.143) (MousePress 7552.857 -
17201.428) (MouseRelease 7758.5713 -17338.572) (MousePress
11872.857 -20630.0) (MouseRelease 12215.714 -20698.572)
(MousePress 16330.0 -23235.715) (MouseRelease 16330.0 -
23235.715) (MousePress 22912.857 -22207.143) (MouseRelease
22912.857 -22207.143) (MousePress 25518.572 -14252.857)
(MouseRelease 25518.572 -13910.0) (MousePress 25107.143 -
2938.5715) (MouseRelease 25244.285 -2732.8572) (MousePress
25998.572 215.71428) (MouseRelease 25998.572 215.71428)
VectorButton (MousePress 12832.857 -20424.285)
(MouseRelease 12832.857 -20424.285) (MousePress 13792.857 -
8355.714) (MouseRelease 13792.857 -8355.714) (MousePress
14684.286 -1498.5714) (MouseRelease 14684.286 -1224.2858)
(MousePress 14684.286 -881.4286) (MouseRelease 15095.714 -
58.571426)))" (0 1)) ("" ())))

(1053019617 ps-problem "PS.13.2.1" () () ((hints ())
(coding-hints ()) (subs (0))) (("((AppletSettings ((sample
100) (grid on) (ticks on) (axes on) (xmin 0.0) (ymin 0.0)
(xmax 7.0) (ymax 7.0) (xticks 8.0) (yticks 8.0)))
(OverlayObjects ((circle (color -16776961) (coords
0.9479167 2.9604168 1.0354167 3.0625)) (circle (color -
16776961) (coords 1.9395833 1.9541669 2.0416667 2.05625))
(circle (color -16776961) (coords 2.9458334 0.9625001
3.0479167 1.0500002)) (circle (color -16776961) (coords
3.9375 2.9604168 4.039583 3.0625)) (circle (color -
16776961) (coords 3.9375 3.9520833 4.039583 4.054167))
(circle (color -16776961) (coords 3.9375 4.958333 4.039583
5.0604167)) (circle (color -16776961) (coords 4.94375
0.9625001 5.045833 1.0500002)) (circle (color -65281)

(coords 1.9395833 3.9520833 2.0416667 4.054167)) (circle
(color -65281) (coords 1.9395833 4.958333 2.0416667
5.0604167)) (circle (color -65281) (coords 1.9395833 5.95
2.0416667 6.0520835)) (circle (color -65281) (coords
5.9354167 1.9541669 6.0375 2.05625)) (circle (color -65281)
(coords 5.9354167 3.9520833 6.0375 4.054167)) (circle
(color -65281) (coords 5.9354167 4.958333 6.0375
5.0604167)) (label (color -16776961) (text +) (coords
0.9916667 3.0041666)) (label (color -16776961) (text +)
(coords 1.9979167 2.0124998)) (label (color -16776961)
(text +) (coords 2.9895833 1.0062499)) (label (color -
16776961) (text +) (coords 3.9958334 3.0041666)) (label
(color -16776961) (text +) (coords 3.9958334 4.010417))
(label (color -16776961) (text +) (coords 3.9958334
5.0020833)) (label (color -16776961) (text +) (coords
4.9875 1.0062499)) (label (color -65281) (text -) (coords
1.9979167 4.010417)) (label (color -65281) (text -)
(coords 1.9979167 5.0020833)) (label (color -65281) (text
-) (coords 1.9979167 6.008333)) (label (color -65281) (text
-) (coords 5.99375 2.0124998)) (label (color -65281) (text
-) (coords 5.99375 4.010417)) (label (color -65281) (text
-) (coords 5.99375 5.0020833)))) (UserObjects ((line
(color -16777216) (coords 0.0875 4.68125 1.4583334
3.3979166 2.3625 2.6687498 3.325 2.1 4.710417 2.31875
5.2791667 4.025 5.1770835 6.4166665 5.366667 7.102083))
(line (color -16777216) (coords 2.5666666 2.6979165
2.7708333 5.2791667 2.9604166 6.7229166 2.9604166
6.8541665)))) (TraceData))" (0 1) ((graphics ()) (text
"There are some missing edge segments"))) ("" ()))))

(1053019707 ps-problem "PS.13.2.1" () () ((hints ())
(coding-hints ()) (subs (0))) (("((AppletSettings ((sample
100) (grid on) (ticks on) (axes on) (xmin 0.0) (ymin 0.0)
(xmax 7.0) (ymax 7.0) (xticks 8.0) (yticks 8.0)))
(OverlayObjects ((circle (color -16776961) (coords
0.9479167 2.9604168 1.0354167 3.0625)) (circle (color -
16776961) (coords 1.9395833 1.9541669 2.0416667 2.05625))
(circle (color -16776961) (coords 2.9458334 0.9625001
3.0479167 1.0500002)) (circle (color -16776961) (coords
3.9375 2.9604168 4.039583 3.0625)) (circle (color -
16776961) (coords 3.9375 3.9520833 4.039583 4.054167))
(circle (color -16776961) (coords 3.9375 4.958333 4.039583
5.0604167)) (circle (color -16776961) (coords 4.94375
0.9625001 5.045833 1.0500002)) (circle (color -65281)
(coords 1.9395833 3.9520833 2.0416667 4.054167)) (circle
(color -65281) (coords 1.9395833 4.958333 2.0416667

5.0604167)) (circle (color -65281) (coords 1.9395833 5.95
2.0416667 6.0520835)) (circle (color -65281) (coords
5.9354167 1.9541669 6.0375 2.05625)) (circle (color -65281)
(coords 5.9354167 3.9520833 6.0375 4.054167)) (circle
(color -65281) (coords 5.9354167 4.958333 6.0375
5.0604167)) (label (color -16776961) (text +) (coords
0.9916667 3.0041666)) (label (color -16776961) (text +)
(coords 1.9979167 2.0124998)) (label (color -16776961)
(text +) (coords 2.9895833 1.0062499)) (label (color -
16776961) (text +) (coords 3.9958334 3.0041666)) (label
(color -16776961) (text +) (coords 3.9958334 4.010417))
(label (color -16776961) (text +) (coords 3.9958334
5.0020833)) (label (color -16776961) (text +) (coords
4.9875 1.0062499)) (label (color -65281) (text -) (coords
1.9979167 4.010417)) (label (color -65281) (text -)
(coords 1.9979167 5.0020833)) (label (color -65281) (text
-) (coords 1.9979167 6.008333)) (label (color -65281) (text
-) (coords 5.99375 2.0124998)) (label (color -65281) (text
-) (coords 5.99375 4.010417)) (label (color -65281) (text
-) (coords 5.99375 5.0020833)))) (UserObjects ((line
(color -16777216) (coords 0.21875 3.0625 1.6041666
1.7645831 2.49375 1.0354166 3.4708333 0.4666667 4.85625
0.6854167 5.425 2.40625 5.3229165 4.7833333 5.5125
5.46875)) (line (color -16777216) (coords 2.5520833
2.6979165 2.75625 5.29375 2.9458334 6.7229166 2.9458334
6.8541665)))) (TraceData SelectButton (MousePress 9472.857
-18504.285) (MouseRelease 9472.857 -18504.285) (MousePress
9472.857 -18504.285) (MouseRelease 10158.571 -26184.285)
))" (0 1) ((graphics ((line (color -16711936) (name
feedback) (coords 0.21875 3.0625 0.48381037391313874
2.814180224690405)) (line (color -16711936) (name feedback)
(coords 1.1985809531997491 2.144552897806288
1.4648679258551366 1.8950839928868115)) (line (color -
16711936) (name feedback) (coords 1.4648679258551366
1.8950839928868115 1.4816859935910462 1.8793281153119485))
(line (color -16711936) (name feedback) (coords
1.4816859935910462 1.8793281153119485 1.6041666
1.7645830999999998)) (line (color -16711936) (name
feedback) (coords 3.294182792587309 0.5694931001911789
3.4708332999999997 0.46666669999999994)) (line (color -
16711936) (name feedback) (coords 3.4787499828200774
0.46791670252046335 3.786858083875697 0.5165653488850657))
(line (color -16711936) (name feedback) (coords 4.85625
0.6854167 5.218511143490608 1.7814888565093907)) (line
(color -16711936) (name feedback) (coords 5.218511143490608
1.7814888565093907 5.264625274235074 1.9210136596489633))
(line (color -16711936) (name feedback) (coords

5.264625274235074 1.9210136596489633 5.311970699446897
2.0642639177714485)) (line (color -16711936) (name
feedback) (coords 5.3119706994469 2.0642639177714534 5.425
2.40625)) (line (color -16711936) (name feedback) (coords
5.425 2.40625 5.403604902907624 2.904449297635455)) (line
(color -16711936) (name feedback) (coords 5.403604902907625
2.9044492976354554 5.394163899142862 3.124289451866553))
(line (color -16711936) (name feedback) (coords
5.394163899142862 3.124289451866553 5.393243466332785
3.145722352008187)) (line (color -16711936) (name feedback)
(coords 5.393243466332785 3.145722352008187
5.373651431811311 3.6019361189148546)) (line (color -
16711936) (name feedback) (coords 5.373651431811311
3.6019361189148546 5.3229165 4.783333300000001)) (line
(color -16711936) (name feedback) (coords 5.3229165
4.783333300000001 5.5125 5.46875)) (line (color -16711936)
(name feedback) (coords 2.5520833 2.6979165000000002
2.572302365481434 2.954987449941401)) (line (color -
16711936) (name feedback) (coords 2.572302365481434
2.954987449941401 2.5753763936902287 2.994071519010123))
(line (color -16711936) (name feedback) (coords
2.5753763936902287 2.9940715190101233 2.5823170544701294
3.0823170544701294)) (line (color -16711936) (name
feedback) (coords 2.5823170544701294 3.0823170544701294
2.629830603802923 3.6864178361449413)) (line (color -
16711936) (name feedback) (coords 2.629830603802923
3.6864178361449413 2.75625 5.29375)) (line (color -
16711936) (name feedback) (coords 2.75625 5.29375
2.787351645685533 5.528208466399468)) (line (color -
16711936) (name feedback) (coords 2.7873516456855322
5.528208466399468 2.9458333999999997 6.7229165999999995))
(line (color -16711936) (name feedback) (coords
2.9458333999999997 6.7229165999999995 2.9458333999999997
6.854166500000001)) (circle (color -65536) (name feedback)
(coords 0.43381037391313874 2.764180224690405
0.5338103739131387 2.8641802246904045)) (circle (color -
65536) (name feedback) (coords 1.1485809531997481
2.094552897806289 1.2485809531997482 2.1945528978062887))
(line (color -65536) (name feedback) (coords
0.48381037391313874 2.814180224690405 1.1985809531997482
2.144552897806289)) (circle (color -65536) (name feedback)
(coords 1.5541665999999998 1.7145830999999997 1.6541666
1.8145831)) (circle (color -65536) (name feedback) (coords
2.44375 0.9854165999999999 2.5437499999999997 1.0854166))
(line (color -65536) (name feedback) (coords 1.6041666
1.7645830999999998 2.49375 1.0354166)) (circle (color -
65536) (name feedback) (coords 2.44375 0.9854165999999999

```
2.5437499999999997 1.0854166)) (circle (color -65536) (name
feedback) (coords 3.244182792587309 0.5194931001911788
3.3441827925873086 0.619493100191179)) (line (color -65536)
(name feedback) (coords 2.49375 1.0354166 3.294182792587309
0.5694931001911789)) (circle (color -65536) (name feedback)
(coords 3.7368580838756972 0.466565348885066566
3.836858083875697 0.5665653488850657)) (circle (color -
65536) (name feedback) (coords 4.80625 0.6354166999999999
4.90625 0.7354167)) (line (color -65536) (name feedback)
(coords 3.786858083875697 0.5165653488850657 4.85625
0.6854167)))) (text "There are some incorrect edge segments
(shown in red above).  The red segments will be more
visible if you move (or clear) your input segments."))) (""
()))))
```

Appendix D

USAGE DATA

The following is a detailed table of the usage statistics of students using the graphical interface:

User	Attempt	Mouse Clicks	Removes	Clears	Feedback
USER1	1	24	15	3	Incorrect
USER1	2	2	0	2	Incorrect
USER1	3	3	0	2	Incorrect
USER1	4	5	0	1	Incorrect
USER1	5	14	1	2	Incorrect
USER1	6	10	0	1	Incorrect
USER1	7	12	0	1	Incorrect
USER1	8	19	3	0	Incorrect
USER1	9	0	0	0	Incorrect
USER1	10	15	0	2	Incorrect
USER1	11	14	0	1	Incorrect
USER1	12	12	0	1	Correct
USER2	1	11	0	0	Missing
USER2	2	25	0	0	N/A
USER2	3	15	0	0	Incorrect
USER2	4	17	2	1	Missing
USER2	5	9	0	1	Incorrect
USER2	6	17	2	1	Missing
USER2	7	8	1	2	Incorrect
USER2	8	18	8	0	Incorrect
USER2	9	7	0	0	Incorrect
USER2	10	18	8	0	Incorrect
USER2	11	4	0	0	Missing
USER2	12	10	0	0	Missing
USER2	13	2	0	0	N/A
USER2	14	14	0	0	Incorrect
USER2	15	7	2	0	Missing
USER2	16	20	2	0	Correct
USER3	1	14	0	0	Correct
USER4	1	11	1	2	Missing

USER4	2	2	0	0	Missing
USER4	3	8	0	1	Missing
USER4	4	10	0	0	Correct
USER5	1	20	2	1	Correct
USER6	1	19	0	0	Missing
USER6	2	2	0	0	Correct
USER7	1	59	8	3	Correct
USER8	1	12	0	0	Missing
USER8	2	30	2	0	Incorrect
USER8	3	26	0	0	Incorrect
USER8	4	57	0	0	Incorrect
USER8	5	19	0	0	Incorrect
USER8	6	21	4	0	Incorrect
USER8	7	10	1	0	Incorrect
USER8	8	14	0	0	Incorrect
USER8	9	22	4	0	Incorrect
USER8	10	6	1	0	Incorrect
USER8	11	6	0	0	Incorrect
USER9	1	57	2	15	Correct
USER10	1	33	4	1	Correct
USER11	1	26	3	1	Missing
USER11	2	28	0	2	Incorrect
USER11	3	7	6	12	N/A
USER11	4	8	0	0	Missing
USER11	5	6	0	0	Incorrect
USER11	6	10	0	6	Missing
USER11	7	11	0	0	Incorrect
USER11	8	46	0	4	Incorrect
USER11	9	0	1	0	Missing
USER11	10	2	2	0	Correct
USER12	1	28	5	1	Missing
USER12	2	14	2	0	Incorrect
USER12	3	5	3	0	Incorrect
USER12	4	36	3	1	Missing
USER12	5	2	0	0	Missing
USER12	6	8	0	0	Missing
USER12	7	19	2	1	Missing
USER12	8	20	1	1	Correct
USER13	1	18	0	1	Missing
USER13	2	28	0	0	Missing
USER13	3	10	0	0	Correct
USER14	1	7	0	1	Incorrect
USER14	2	12	0	1	Incorrect
USER14	3	6	0	0	Incorrect
USER14	4	11	0	0	Incorrect
USER14	5	10	0	0	Incorrect

USER14	6	31	3	0	Incorrect
USER14	7	34	0	0	Incorrect
USER14	8	42	0	0	Incorrect
USER14	9	43	4	0	Correct
USER15	1	11	0	0	Missing
USER15	2	2	0	0	Missing
USER15	3	13	0	0	Missing
USER15	4	14	0	1	Correct
USER16	1	10	0	0	Incorrect
USER16	2	30	0	0	Incorrect
USER16	3	3	0	0	N/A
USER16	4	23	0	0	N/A
USER17	1	55	0	0	N/A
USER17	2	9	0	0	Incorrect
USER17	3	7	1	6	N/A
USER17	4	39	1	3	Missing
USER17	5	2	0	0	Missing
USER17	6	9	0	0	Missing
USER17	7	35	3	1	Missing
USER17	8	2	0	1	Incorrect
USER17	9	0	0	5	N/A
USER17	10	36	3	0	Missing
USER17	11	14	7	0	Missing
USER17	12	3	0	0	Incorrect
USER17	13	3	2	0	Missing
USER17	14	3	0	0	Incorrect
USER17	15	7	1	0	Correct
USER18	1	26	3	1	Correct
USER19	1	33	5	3	Missing
USER19	2	3	0	0	Missing
USER19	3	36	9	0	Correct

BIBLIOGRAPHY

[1] Arnow, David and Barshay, Oleg. "WebToTeach: An Interactive Focused Programming Exercise System." 29th ASEE/IEEE Frontiers in Education Conference, San Juan, Puerto Rico, November 1999.

[2] Arnow, David. *WebToTeach Project Page*, http://www.sci.brooklyn.cuny.edu/~arnow/WebToTeach, Brooklyn, NY, 1999

[3] Gries, David and Gries, Paul. "ProgramLive and Its Companion." http://webster.cs.uga.edu/~gries/programlive/plive.html, Athens, GA, 2001

[4] Lozano-Pérez, Tomás, Grimson, Eric, Kaelbling, Leslie, Terman, Chris, Winston, Patrick. *Technologically Enhanced Education in Electrical Engineering and Computer Science*, http://www.swiss.ai.mit.edu/projects/icampus/projects/eecs.html, Cambridge, MA, 2001

[5] *Mastering Physics Demo*, http://mpdemo.mycybertutor.com, *University Physics 11th ed.* Addison-Wesley, Boston, MA, 2002.

[6] Pritchard, David and Morote, Elsa-Sofia. "Reliable Assessment with CyberTutor, a Web-based Homework Tutor." World Conference on E-Learning in Corporate, Government, HealthCare, and Higher Education, E-Learn 2002. Montreal, Canada, October, 2002.

[7] Tesuya, Mizutori. *Plotspline Curve; Exercises on the Interpolation and Approximation Method*, http://www.bekkoame.ne.jp/~mizutori/java/spline/Plot2D.html, Tokyo, Japan, 1997

[8] *WebCT*, http://www.webct.com, WebCT, Inc. Lynnfield, MA, 2003