8-2012

# Sketchmate: A Computer-Aided Sketching and Simulation Tool for Teaching Graph Algorithms

Kristy Sue Van Hornweder
kvanhorn@utk.edu

To the Graduate Council:

I am submitting herewith a dissertation written by Kristy Sue Van Hornweder entitled "Sketchmate: A Computer-Aided Sketching and Simulation Tool for Teaching Graph Algorithms." I have examined the final electronic copy of this dissertation for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Doctor of Philosophy, with a major in Computer Science.

Bradley T. Vander Zanden, Major Professor

We have read this dissertation and recommend its acceptance:

Lynne E. Parker, James S. Plank, Christopher H. Skinner

Accepted for the Council:
Dixie L. Thompson

Vice Provost and Dean of the Graduate School

(Original signatures are on file with official student records.)

# Sketchmate: A Computer-Aided Sketching and Simulation Tool for Teaching Graph Algorithms

A Dissertation
Presented for the
Doctor of Philosophy
Degree
The University of Tennessee-Knoxville

Kristy Sue Van Hornweder

August 2012

# Acknowledgements

I would like to thank my research committee members, Dr. Brad Vander Zanden, Dr. Lynne Parker, Dr. Jim Plank, and Dr. Chris Skinner for serving on my committee and providing support and guidance for my research.

I would also like to extend a special thank you to Dr. Vander Zanden for being willing to serve as my advisor upon my decision to change research direction. During our weekly meetings over the last few years, he has offered much advice and direction for various activities such as Java program design, dissertation writing, teaching assistant duties, teaching style and philosophy, job searching, and non-academic pursuits, including hiking in the Great Smoky and Blue Ridge Mountains.

In addition, I would like to thank Dr. Chris Skinner for providing much guidance in designing and conducting my research experiments. I also appreciate the assistance of Dr. Skinner's colleague Mike O'Neil in providing support for statistical analysis of my experimental data.

I would also like to thank Dr. Plank and Dr. Vander Zanden for allowing me to use one of their class periods to conduct my research experiments. I also thank the students in those classes for their participation.

I also owe a special thank you to Dr. Timothy Colburn and Dr. Robert McFarland, my former advisors from the University of Minnesota-Duluth. Dr. Colburn directed my Computer Science Masters thesis, as well as supervised my work as a teaching assistant, an experience that led to my desire to pursue a career in academia. Dr. McFarland directed my Mathematics Masters project, taught several of my undergraduate courses, and inspired much interest in attending graduate school. Both Dr. Colburn and Dr. McFarland have played a major role in helping me define my career path.

Finally, I would like to extend gratitude to my friend Jill Torgerson, whom I met at the University of Minnesota-Duluth. She has served as a confidant for many years and has provided me invaluable feedback on my teaching while I was a Masters student. She has played an instrumental part in motivating my interest in teaching and pursuing a PhD so that I could ultimately become a professor.

# Abstract

In this dissertation, we developed and tested a sketching, visualization, and simulation tool called Sketchmate for demonstrating graph algorithms commonly taught in undergraduate computer science courses. For this research, we chose to focus on shortest path and network flow algorithms. Two versions of this tool have been implemented: 1) an instructor tool that supports computer-aided manual simulations of algorithms that augment traditional whiteboard presentations, allowing lectures to be more dynamic and interactive, and 2) a student tool that supports computer-aided manual practice of algorithms that enables students to work through homework problems more quickly while providing detailed incremental feedback about their performance and about how to solve a problem when they get stuck. Previous algorithm simulation systems have essentially forced instructors to narrate an algorithm as though they were describing an automated set of slides. In contrast, our tool allows instructors to manually manipulate attributes of a graph as they demonstrate an algorithm.

A set of experiments was conducted using the tools. The results for the student tool showed that there was no statistically significant difference in test score improvement between Sketchmate and paper and pencil students, although they did show that Sketchmate students scored roughly one letter grade higher than paper and pencil students. Based on survey data, the students preferred using the tool to using paper and pencil. The results of the experiment involving the instructor tool showed that although there was no statistically significant difference in learning between Sketchmate and the whiteboard, both the instructor and the students preferred a Sketchmate lecture to a whiteboard lecture.

# Contents

# List of Tables

# List of Figures

# Chapter 1

# Introduction

It is common practice to use illustrations when teaching abstract concepts in computer science to help make these concepts more concrete. The instructor will often draw a given data structure on the whiteboard and use it to step through the process of performing an algorithm on that data structure. Textbooks often include a series of diagrams of data structures to accompany the textual description and explanation of the given algorithm. While use of a whiteboard for free-drawing pictures is convenient and easily accessible, there are disadvantages of using this technique. These pictures are static and do not clearly show transitions between states of the given data structure. The diagrams can be cumbersome to draw by hand and use of this technique is prone to making errors. These diagrams lack domain-specific knowledge, which means that errors introduced by an instructor will not be caught. With static illustrations drawn on the whiteboard, it is more difficult to revert to previous stages or to save these diagrams for later use. A Smartboard (SMART-Technologies, 2012) allows one to save diagrams, but that requires the instructor to remember when to save the diagram. In some cases, instructors will erase portions of the data structure and redraw new components as they step through an algorithm. This makes the presentation more difficult to follow as the history of the algorithm's progression is lost. To preserve information, instructors have the option of recopying the data structure for each stage, however, most of the components of the data structure will not change in a given step, and so this method is time consuming, wasteful in terms of whiteboard space, and more prone to errors being introduced.

Another illustration strategy is the use of computerized simulations for presenting more complex concepts and algorithms. Several studies [(Hundhausen et al., 2002), (Narayanan and Hegarty, 2002), (Laakso and Salakoski, 2004), (Kumar, 2004), (Kumar, 2005b), (Baloukas, 2009)] have suggested that student learning may be improved by using simulations as opposed to traditional paper and pencil methods. Since simulations are dynamic, it is possible to watch the algorithm in action as it manipulates a data structure, and the transitions between steps is much more pronounced. Since the entire process is automated, these transitions can be demonstrated without needing to manually erase or recopy the data structures. A computer simulation is pre-programmed with the algorithm, thus making it easier to prevent errors when presenting the simulation to students. The example data

structures used in computer simulations also can be saved for later review and practice, either for instructors to present in their lectures, or for students to study the workings of the algorithms.

While computer simulations can be a significant improvement over static illustrations, they limit an instructor to narrating what is essentially an automated slide show that unfolds in a manner prescribed by the simulation tool, instead of allowing instructors to present the simulations in their own way. In this dissertation, we have developed a manual simulation tool that allows instructors to manually manipulate graph data structures as they would on a whiteboard, rather than helplessly watching as the computer manipulates the data structure for them. This tool essentially functions as an enhanced whiteboard with "smart objects", which correspond to key elements of graph diagrams, such as labeled vertices and edges. Previous tools such as MatrixPro (Karavirta et al., 2004) and Sketchmate (Orsega, 2009) provide the instructor with a library of pre-defined high-level operations that can be performed on a data structure. In contrast, our tool defines a set of graph attributes, such as vertex state and edge selection, and allows the instructor to manually manipulate these attributes. Hence, the instructor has finer-grained control over the simulation, since the instructor can choose the order in which the attributes are manipulated. Several additional features are also included. Users are allowed to create a custom graph on which to perform the simulation and are not limited to built-in examples. The application provides a history list of the graph algorithm steps to allow the user to revert to any previous step for review. Finally, the tool provides a notepad feature where the instructor can annotate a simulation with key concepts, which is similar to writing these concepts on a whiteboard.

In addition to a lecture aid for instructors, we also address the computer-aided simulation of practice exercises and completion of homework assignments for graph algorithms. Existing computer simulations allow a student to passively watch but do not actively engage the student. Hundhausen's meta-study (Hundhausen et al., 2002) suggests that student learning outcomes are typically optimized when a student is actively engaged in a simulation. This requires a manual simulation tool. A student using a modified version of the instructor's tool to study an algorithm and work practice exercises could obtain detailed feedback at each step that will explain what the student did incorrectly as opposed to simply stating that the student's action was incorrect. Such a tool could also facilitate the grading of student homework, since much of the process can be automated and not require hours of instructor time to carefully read student solutions. In this dissertation, we have also developed the student version of the manual simulation tool described here.

To design our tools, we conceptually built on Michael Orsega's Sketchmate tool [(Orsega, 2009), (Orsega et al., 2011), (Orsega et al., 2012)]. Orsega's Sketchmate tool supports an instructor version and a student version, it visualizes splay tree operations, and it implements several of the features mentioned in the preceding paragraphs, including "smart" tree objects, manual simulation for students, and detailed immediate feedback for students. It does not support manual simulation for instructors, which is one of the main contributions of this research.

In this dissertation, two versions of Sketchmate for prioritized graph search algorithms

have been implemented: a teaching version for instructors to teach graph algorithms during their lectures, and a tutoring version for students to practice examples and complete homework exercises. Our Sketchmate software for graphs contains several features beyond allowing the basic demonstration of graph algorithms. Users are able to input a custom graph, such as that found in a course textbook, rather than relying solely on built-in examples or random graphs. A history window of operations can be displayed with the instructor tool to enable users to refer to any previous step of the algorithm. A diagram of the graph structure at the previous step of the algorithm is displayed along with a diagram of the current graph structure to allow for easier viewing of changes between steps. An instructor is able to manually simulate the process of the algorithm, as well as annotate the simulations using a notepad feature.

A student using the student tool is able to obtain detailed feedback at each step that explains what the student did incorrectly as opposed to simply stating that the student's action was incorrect. Users have the option of saving their work for future reference and review. Students are able to complete a typical homework assignment using the tool, submit their work, and receive immediate feedback and a score. As noted in the related work chapter, very few previous tools support manual simulation and none exists that implements all of the above features as applied to graph algorithms.

Unlike most of the existing simulation tools, we have performed a formal evaluation of both the instructor and the student tool. The experiment performed with the instructor Sketchmate tool involved a group of undergraduate students observing a lecture on network flow problems. Half of the students observed the lecture presented using Sketchmate, and the other half of the students observed a traditional whiteboard lecture on the same problems. The instructor Sketchmate tool was evaluated based on student learning as measured by the difference between pre-test and post-test scores, as well as overall satisfaction of observing a lecture that uses Sketchmate.

The student Sketchmate tool experiment involved a group of undergraduate students working through practice exercises for shortest path and network flow. In one part of the experiment, half of the students worked through the shortest path exercises using Sketchmate, while the other half of the students worked through the same exercises using a traditional paper and pencil method. For the other part of the experiment, the students who worked through the shortest path exercises using paper and pencil worked through network flow exercises using Sketchmate, and the students who worked through the shortest path exercises using Sketchmate worked through the same network flow exercises using paper and pencil. The evaluation of the student Sketchmate tool was based on the amount of learning achieved from performing the exercises as measured by the difference between pre-test and post-test scores, as well as students' scores on the practice exercises, and overall satisfaction of using Sketchmate.

The remainder of this dissertation is organized as follows. Chapter 2 surveys related work in algorithm visualization and general teaching tools. Goals of this research are discussed in Chapter 3. Chapter 4 reviews basic graph terminology and explains the shortest path and network flow algorithms in terms of example problems. In Chapter 5, a detailed description

of the instructor and student Sketchmate simulation tools is given. Chapter 6 presents the experimental results of the study with the student tool, while Chapter 7 presents the experimental results of the study with the instructor tool. Conclusions and directions for future work are included in Chapter 8.

# Chapter 2

# Related Work

In this chapter, existing work in algorithm visualization (AV) and computerized teaching tools is surveyed. Many aspects of these tools are closely related to those of the tools developed in this dissertation. This chapter first discusses research on the general effectiveness of AV tools. It then surveys existing AV tools for teaching computer science. Next, it provides a brief summary of tools that can be used for general presentation and annotation. It then analyzes AV tools that are specifically aimed at simulating and visualizing graph algorithms. Finally, it summarizes features of the previous Sketchmate tool for splay trees, along with the similarities and differences of the Sketchmate tool for graphs developed in this dissertation.

## 2.1 Effectiveness of Algorithm Visualization Tools

There have been numerous studies analyzing the effectiveness of algorithm visualization tools. Research by Mayer et. al. [(Mayer and Anderson, 1991), (Mayer and Anderson, 1992), (Mayer and Sims, 1994)] has shown that presenting text along with images results in better recall and problem solving skills than text presented before images or images presented without text. Furthermore, animation presented without explanatory text or narration is no different than receiving no instruction at all.

Hundhausen (Hundhausen et al., 2002) conducted a meta-analysis of 24 algorithm visualization (AV) studies and obtained mixed results. The main conclusion was that how students use an AV has a greater impact than what the AV demonstrates, and that students learn better if they are actively engaged in the learning process. Narayanan (Narayanan and Hegarty, 2002) also concluded that learning is enhanced if students are actively involved and create their own ideas and explanations. He also found that the content and structure of the visualization is more important for comprehension and learning than the media or modality by which the algorithm is presented. Palmiter (Palmiter and Elkerton, 1991) notes that it is important to carefully choose the text to accompany animations; otherwise, the animations may not provide instructions that are useful for later recall and application. Pane (Pane et al., 1996) points out that the use of animations and simulations does not guarantee improvement in learning and that well-designed static text and images may be just as effective

for learning. However, visualizations may be more appealing and thus more motivating, so students may spend more time with them, and thus learn more. The findings of Kumar (Kumar, 2005b) were that students learn more using an online tutor than a textbook, presenting images and text together is more effective than presenting only images, and using images alone results in better learning than no instruction at all.

Kumar (Kumar, 2004) also analyzed results from a survey asking students what they think of the use of online tutors. Students generally felt they learned better with text and graphics. They think that tutors that provide feedback are more effective, although tutors that give verbose feedback are less effective. Kumar also found that it might be necessary to consider variations among different demographic groups when assessing instructional methods for computer science courses (Kumar, 2009), and that female students tend to evaluate online tutors more positively than male students (Kumar, 2008).

To summarize, various studies on the effectiveness of AV tools have produced mixed results. The major conclusions are that simulations and visualizations are more effective if students are actively engaged in the learning process, text and images presented together results in better learning outcomes than when text and images are presented separately, the text presented with an image must be chosen carefully to ensure that it actually enhances understanding of the image, and AV tools that include detailed feedback are more effective than those that do not, although tools that give verbose feedback are less effective.

## 2.2 Animation and Visualization Tools for Teaching Computer Science

The first animation tool for teaching computer science concepts dates back to 1981 with the movie *Sorting Out Sorting* (Baecker, 1981). Since then, numerous animation and visualization systems have been developed, each with their set of features and limitations. Various general purpose software systems such as graphical editors, PowerPoint, and Flash can be used to simulate animations. However, creating animations using these tools is time consuming and these tools do not contain any domain-specific knowledge about the data structure or algorithm at hand.

Systems such as XTANGO (Stasko, 1992) can function as a visual debugger where students replace existing function calls with calls to non-standard libraries. However, its focus on visual debugging differs from our focus on conceptual understanding of the algorithm. Programs such as BALSA (Brown, 1988) allow students to create generic animations. However, students need to have deep knowledge of the algorithm, as well as the programming language and application interface. A programmatic tool for building visualizations that focuses on conceptual understanding of the algorithm is Visualiser (Naps, 1998). This tool involves students annotating their code to produce a visual slide show that displays the state of the data structures at each step. While these visualizations are actively created by students, the visualizations do not allow the student to manually manipulate the components of the data structure, nor do they provide a detailed explanation of what is occurring at

each step. Tools such as JAWAA (Pierson and Rodger, 1998) and ANIMAL (Robling et al., 2000) allow users to create animations visually through a drawing editor using primitive graphical objects, and easy drag and drop functions. While little knowledge of the algorithm is required, these systems do not have built-in operations specific to the data structures of interest.

The Problets website (Kumar, 2010) contains a collection of fully automated tutors for teaching basic computer programming concepts, such as programming scope (Kumar, 2005a) and *for* loops and parameter passing (Kumar, 2006). These tutors automatically generate problems, answers, grades, and feedback. The problems are generated using templates or a model of the problem domain, and answers are generated by solving the problems using tree traversal algorithms. The feedback can be minimal (diagnostic) or detailed (step-by-step explanation). Studies have found that detailed feedback is more effective for learning and that succinct feedback is more effective than verbose feedback. The tutors are also more effective when they are adapted to the needs of the individual learner, such as focusing on concepts the student finds more difficult.

Based on the above discussion, the goal of this dissertation project was to develop a tool that: contains domain-specific knowledge and built-in operations for the data structure and algorithm of interest, allows the user to focus on conceptual aspects of the algorithm without becoming bogged down in coding details, includes an intuitive and easy to learn interface, and provides detailed feedback and step-by-step explanation of the algorithm.

## 2.3 General Presentation and Annotation Tools

The tools discussed thus far were designed for animating and visualizing concepts and algorithms for computer science. Another feature needed to teach these concepts during lecture is a way to annotate and make notes on presentations. A number of lecture aids have been designed to help fulfill this purpose. ScreenCrayons (Olsen et al., 2004) allows the user to make annotations on a screen capture and highlight portions of a document or image. The user can create notes using this tool and save them to a file system for later reference. The Lecturer's Assistant (Buckalew and Porter, 1994) enables communication between the lecturer and the students in that the student can make annotations and ask questions without the need to leave their seat. Several tools are designed to work with pen-based computers and tablet PCs where slides are annotated with text or notes [(Berque et al., 2001), (Anderson et al., 2004), (Golub, 2004), (Wilkerson et al., 2005), (Berque, 2006)]. Tools have also been developed for PDAs and handheld devices [(Myers et al., 1998), (Myers, 2001), (SMART-Technologies, 2012)], which allows for multiple devices to be connected to a shared whiteboard application.

In summary, various presentation tools have been developed for annotating slides with notes, enhancing communication between lecturer and students, and enabling students to simultaneously connect to an electronic whiteboard. These tools are domain-independent and hence not restricted to computer science. They operate at a higher level than the domain-specific tools described in this dissertation and could complement these tools. However, the

instructor tool developed in this dissertation incorporates an annotation feature by providing a notes window for annotating graph simulations demonstrated during lecture. This feature is a significant enhancement over the existing AV tools for simulating graph algorithms.

## 2.4 Animation and Visualization Tools Capable of Demonstrating Graph Algorithms

The following sections discuss several tools that are capable of working with graph algorithms such as Dijkstra's shortest path and network flow algorithms. These AV tools are discussed with the following desired features in mind:

- Supporting automatic continuous display of algorithm steps from beginning to end, without pausing between steps, which the user passively views as if they are watching a movie
- Providing a discrete step-by-step mode that pauses after each step, and allows a user to click a button to continue
- Stepping back to previous steps to reverse the visualization process
- Providing a history list of operations performed, with capability of reverting to any previous step
- Presenting animations of objects moving to a new location or flashing objects to draw attention to them
- Allowing the user to create a custom graph from scratch
- Displaying a textual description of what is occurring at each step
- Allowing the user to manually practice a problem step by step
- Grading the user's solution in terms of the number of correct steps
- Providing detailed feedback of the user's solution when the user is incorrect

All of the surveyed AV tools allow the user to proceed through the algorithm one step at a time, pausing between steps. Each of the above features is supported by at least one existing AV tool, except for a history list of operations where the user can select a step to revert back to for review. No existing tool for visualizing graph algorithms supports such a history list. Sketchmate for splay trees (Orsega, 2009) does support a history list feature, and this feature has been extended to our Sketchmate for graphs. To help organize the above features, we have created a taxonomy of AV tools, as shown in Figure 2.1, based on three criteria:

1. Automatic simulation of graph algorithms (for instructor) vs. manual practice of graph algorithms (for student). "Automatic simulation" means that the computer displays the simulation on its own, and "manual practice" means that a human user simulates the algorithm manually.

2. Continuous display of steps vs. discrete step-by-step display

3. Graph creation capabilities vs. toolkit-provided or random graphs

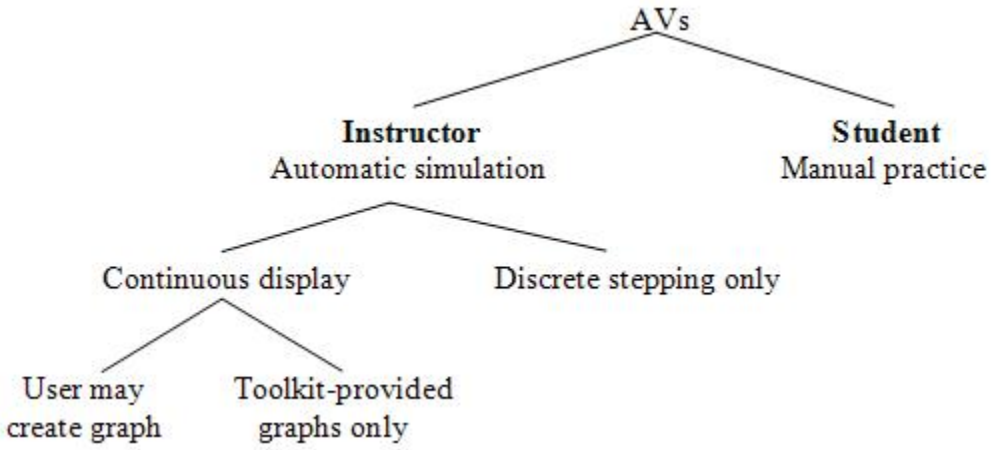


Figure 2.1: Taxonomy of AV tools

Each classification as based on the leaves of the tree in Figure 2.1 is discussed below, and examples of each class are analyzed. Most of these tools can be found in the AlgoViz Wiki website catalog (AlgoViz, 2010). An overview of AlgoViz and the state of the field is available in (Shaffer et al., 2010).

### 2.4.1 Tools with Automatic Simulation, Continuous Display, and Graph Creation Capability

All of the tools in this category are capable of automatic simulation but not manual practice, continuous display of the algorithm steps, and the custom creation of a graph. These include GALGO (Hoebel and Burrer, 2003), Dijkstra's Shortest Path (DSP) (Laffra, 1996), EVEGA (Khuri and Holzapfel, 2001), Swan (Yang et al., 1996), Network Flow (Chalidabhongse, 1996), and JAVENGA [(Baloukas, 2009), (Athanasios, 2009)]. Each of these is examined in turn.

**GALGO** The GALGO applet (Hoebel and Burrer, 2003) does not have any of the other desired features shown in the list at the beginning of this section. It allows users to create and edit graphs by clicking and dragging vertices and edges. Users are able to choose a

start vertex. During execution of the algorithm, the vertex and edges currently being visited flash once in different colors, but none of the components actually move. The adjacency matrix is displayed in another window. GALGO supports the visualization of a number of graph algorithms, including Dijkstra's shortest path, Floyd-Warshall, spanning trees, graph traversal, Kruskal, and Prim, as well as algorithms for trees. The applet does not work on the Ubuntu system and it runs somewhat slowly on a PC. The interface is a little difficult to learn how to use, partly because some of the components are in German, such as file choices, warning messages, and help information.

**Dijkstra's Shortest Path (DSP)** DSP (Laffra, 1996) is an applet that includes a textual description of the algorithm steps. The user can create or edit graphs by adding nodes and edges or changing the weights of the edges. The toolkit also includes built-in example graphs. Nodes currently being visited in the simulation are displayed in a different color. None of the components are animated, although the simulation does pause between steps. DSP is specific to Dijkstra's shortest path algorithm; it is not capable of performing any other algorithms. The applet does not appear to run on the Ubuntu system, but it runs smoothly on a PC machine.

**EVEGA** EVEGA (Khuri and Holzapfel, 2001) is a standalone Java application that is specialized for simulating graph algorithms. It uses colors, multiple views, and a textual description of each step of the algorithm. It allows for direct manipulation and creation of graphical objects with a drawing editor. The package also includes built-in graph generators. The tool is able to step through an algorithm and pause between steps. The speed of the simulation can be controlled by the user. The tool provides online help and is designed to prevent most possible user errors. EVEGA is capable of simulating maximum flow algorithms. EVEGA includes the capability of analyzing the runtime and number of operations of the algorithm. The tool allows an instructor to change or create a graph in class to answer "what if" questions asked by students, as well as create a simulation in advance and save it for a later time.

**Swan** Swan (Yang et al., 1996) is a standalone Java application that includes a few built-in examples of graphs that the user can modify by adding and deleting nodes and edges. Both the graph representation and its corresponding adjacency list are displayed during the simulation. The user can step through the algorithm in discrete steps or run it continuously until the end, although the steps are displayed in succession very rapidly. There is no actual animation of graph components but different colors and thicknesses are used for the edges to indicate their current state in the simulation. A brief textual description of each step appears below the graph. The user can zoom in or zoom out, save the screen to a file, and change the attributes of the graph or its components such as color, thickness, position, and layout. The application is capable of working with several different algorithms and data structures including binary search, heapsort, Huffman encoding, string matching, network flow, finding the minimum and maximum elements of a list, red black trees, topological sort,

and vertex cover.

The project is no longer active and has not been updated since the mid-1990's. The program could only be successfully run on a Windows system; it could not be run on a LINUX box. The software also contains a few bugs such as closing the entire application when step mode completes and after clicking the cancel button of the attribute dialog box. The application also requires users to supply the exact filename containing the graph example rather than allowing them to locate the file via a file chooser.

**Network Flow** Network Flow (Chalidabhongse, 1996) is an applet that allows users to create and edit a graph from scratch. The simulation can be run in discrete step mode or in continuous mode, although the steps are executed so quickly that the simulation appears to jump to the end state and the intermediate states are not able to be viewed. None of the components of the graph are animated and the nodes and edges do not change colors; only the flows and costs along the edges are changed. The user is able to restart the algorithm in order to replay the simulation from the beginning.

**JAVENGA** JAVENGA [(Baloukas, 2009), (Athanasios, 2009)] is an applet capable of simulating a number of graph algorithms, including Dijkstra's shortest path, depth-first search, breadth-first search, topological sort, Bellman-Ford, Prim, Kruskal, and network simplex. In addition to displaying a textual description for the history of algorithm steps, JAVENGA also allows the user to step forward or backward through the algorithm, and the speed of execution can be controlled. The values of variables are displayed at each step. Although none of the components are animated, the nodes currently being visited flash in different colors. The tool provides a window to create a graph by clicking to add nodes and edges, and it allows the user to type in the initial edge and node costs. The adjacency matrix can be viewed, although not at the same time as the graph. The tool also provides help files and windows.

There are several issues with using JAVENGA. The graph creation process requires reading and scrolling through a window of instructions instead of being intuitive. It is nearly impossible to slow down or pause the simulation, and the entire graph window flashes rapidly as well as the PC desktop items outside the applet window. The graph window flashes continuously, and users cannot stop the simulation without closing the window. The textual information printed to the text box flashes and disappears, making it impossible to read. The tool will not run at all on the Ubuntu system.

JAVENGA researchers have tested their tool on a small group of students who were given either a Powerpoint presentation on Dijkstra's shortest path algorithm or a JAVENGA visualization of the algorithm. The students who used JAVENGA performed better on test questions involving the application or analysis of the algorithm and worse on questions involving fact-based knowledge of the algorithm. In other experiments, the same researchers also found that visualization has significantly improved learning of difficult concepts, such as the network simplex algorithm.

### 2.4.2 Tools with Automatic Simulation, Continuous Display, and Toolkit-Provided Graphs

The tools in this grouping allow for automatic simulation but not manual practice, continuous display of algorithm steps, but do not allow users to create their own graphs. Instead, users operate on built-in graphs or randomly generated graphs. These tools include Data Structure Visualization (DSV) (Galles, 2006), Auckland (Ng et al., 1998), Project Links (Holmes et al., 1999), and Animal [(Brodowski, 1999), (Robling et al., 2000)]. A discussion of each of these tools follows.

**Data Structure Visualization (DSV)** DSV (Galles, 2006) is a downloadable Java application that does not support any of the other desired features shown in the list at the beginning of this section. It does include a few annotations and the user can view the adjacency list or matrix, although it cannot be viewed at the same time as viewing the graph. In addition to the ability to step through the simulation one step at a time, the user is also able to skip to the end result. In automated continuous display mode, the user is able to pause the simulation and control the simulation speed. Users can move vertices and edges around to improve readability and they can choose the start vertex. During each step in the simulation, a copy of a vertex number animatedly moves from a table of vertex state information to a path list for a vertex to the right of the table, but nothing in the actual graph representation is animated. The vertex and edges currently being visited are highlighted in both the graph and table representations. The package includes smaller graph problems and larger, more complex ones. DSV is capable of visualizing several graph algorithms, including Dijkstra's shortest path, depth-first search, breadth-first search, topological sort, Floyd-Warshall, Kruskal, Prim, and connected components, as well as many other data structures and algorithms, such as lists, stacks, queues, sorting, trees, heaps, hashing, Huffman encoding, and dynamic programming. One minor issue is that the final path in the graph is not highlighted so it is a little more difficult to see the solution.

**Auckland** Auckland (Ng et al., 1998) is an applet that includes annotations and limited textual descriptions of the steps, as well as pseudocode with the current step highlighted. During the simulation, the nodes and edges currently being visited blink, but nothing in the graph actually moves. Nodes are displayed in different colors, depending on their current state. The speed of the simulation can be controlled by the user. Two different graph examples are provided. Other data structures and algorithms simulated by Auckland include sorting, trees, dynamic programming, Huffman encoding, heaps, hashing, and minimum spanning trees.

**Project Links** Project Links (Holmes et al., 1999) is an applet that is capable of simulating the Dijkstra shortest path and Bellman-Ford graph algorithms. The user can click the Go, Pause, and Reset buttons to control the simulation. It is not possible to step through the algorithm one step at a time, but the Pause button can be used to simulate the discrete

step mode. The colors of nodes and edges are changed throughout the simulation, but no components of the graph are actually animated. A brief textual description of the current step is given above the graph. When running the applet on a Windows machine, there are issues with refreshing the screen between steps of the simulation. The graph and information bar appear duplicated and the user must scroll up or down the web page in order to eliminate these ghosting effects. This display issue makes it difficult to read the graph and the textual descriptions.

**Animal** As with Auckland, Animal [(Brodowski, 1999), (Robling et al., 2000)] also includes textual descriptions and highlighting of the current step in the pseudocode, however, these descriptions and pseudocode are very high level. Animal is also capable of navigating backwards to any previous step of the algorithm. The nodes and edges currently being visited in the graph are highlighted, although there is nothing that animates. The user can zoom in on the graph for easier viewing. The software is capable of handling many types of data structures and algorithms, including sorting, searching, compression, cryptography, trees, hashing, and graph algorithms such as Dijkstra's shortest path and Floyd-Warshall. Animal is a downloadable Java application. The tool appears to allow the user to change vertices and edges, however, it does not seem to reload the user's new graph to use in the simulation.

### 2.4.3 Tools with Automatic Simulation and Discrete Stepping

Each tool referred to in this class is capable of automatic simulation but not manual practice, and only allows discrete stepping through the algorithm and no automated continuous display of steps. The tools described here include Shortest Path Problem (SPP) (Ikeda, 2004), Minimum Routes Finder (MRF) (Papagelis, 1997), tutORial (Sniedovich, 2000), ALVIE (Crescenzi, 2009), and JHAVE [(Naps, 2005), (Teviotdale and Naps, 2008)].

**Shortest Path Problem (SPP)** SPP (Ikeda, 2004) is a very simple applet that does not include any of the other desired features listed at the beginning of this section. Several graph examples are provided by the toolkit. Although there is nothing that animates, the simulation uses different colors for nodes and edges that are currently being visited. The tool can also simulate other algorithms, such as simplex, Prim, Kruskal, and Ford-Fulkerson.

**Minimum Routes Finder (MRF)** MRF (Papagelis, 1997) is an applet that does not have any of the other features of interest. The changing graph is displayed next to the initial graph. The nodes are colored based on their current state, but nothing in the graph actually moves. Users can skip to the solution, but only the final result is shown, none of the intermediate steps are displayed. There is a bug in that if the user clicks "next step" at the end of the simulation, the graph is cleared and therefore the solution is no longer viewable. The tool is specific to shortest path algorithms; it is not capable of simulating any other algorithms. While the tool runs correctly on a PC, it will not run on the Ubuntu system.

**tutORial** tutORial (Sniedovich, 2000) is an applet that simulates a map of Australia with routes between cities. It includes detailed textual descriptions as the user steps through the algorithm, and the user is able to choose the start and end vertices. The nodes and edges currently being visited are colored, but nothing in the graph actually moves. The tool is also capable of simulating other algorithms such as dynamic programming, queuing networks, linear programming (simplex algorithm), linear algebra algorithms, integer programming, and topological sorting.

**ALVIE** ALVIE (Crescenzi, 2009) is a Java-based application that includes a textual description of algorithm steps, the ability to step backwards or to the end, and a display of pseudocode with the current step highlighted. Nodes and edges currently being visited are colored, but nothing in the simulation is animated. The tool can also simulate other algorithms, including Bellman-Ford, breadth-first search, depth-first search, Hamiltonian paths, graph coloring, independent sets, Kruskal, Prim, vertex covering, nearest neighbor, hashing, Fast Fourier Transformations, Huffman encoding, dynamic programming, linear programming, matrix algorithms, sorting, and searching. It takes a little time to learn how to open a graph problem as the interface includes icons instead of words. It appears that ALVIE may allow the user to create graphs, however, the interface is difficult to use and this precludes us from getting it to work properly.

**JHAVE** JHAVE [(Naps, 2005), (Teviotdale and Naps, 2008)] provides a more direct way to revert to previous steps than the other tools discussed thus far. Users click a tick mark on a timeline to revert to a previous step; however, there is no textual description of the history of steps. Another difference from previously discussed tools is that JHAVE engages users by popping up a quiz-like question after each step, and after completion of the algorithm, the user receives a score indicating the number of questions answered correctly. A window with pseudocode is displayed next to the graph. The user can zoom in or out. None of the components in the simulation are animated. The tool displays a vertical listing of vertices with their costs and predecessors. JHAVE is able to simulate many other algorithms, including searching, sorting, trees, Huffman encoding, hashing, dynamic programming, depth-first search, breadth-first search, Prim, Kruskal, topological sorting, and Floyd-Warshall. The tool can be run directly on the website or it can be downloaded as a Java application.

### 2.4.4 Tools with Manual Practice and Automatic Simulation

None of the tools discussed thus far allow the user to manipulate the graphs and practice a problem manually. There are three AV tools that include this capability: Trakla2 [(Korhonen et al., 2003), (Laakso and Salakoski, 2004), (Karavirta et al., 2006), (Myller et al., 2007)], MatrixPro (Karavirta et al., 2004), and PILOT [(Bridgeman et al., 2000), (Baker, 2000)]. These tools are discussed in more detail in the following paragraphs.

**Trakla2** When students solve problems using the Trakla2 applet [(Korhonen et al., 2003), (Laakso and Salakoski, 2004), (Karavirta et al., 2006), (Myller et al., 2007)], they can immediately obtain a score indicating the number of steps they have performed correctly. This grading only includes a score; it does not include any explanation or detailed feedback of their incorrect steps. Trakla2 does not appear to include animation capabilities, but in addition to the ability to step forward, it is also possible to step backwards to any previous step of the algorithm for review. The user can also undo or redo a step in the manual-practice mode. The user can change the font size to apply a zoom function to the graph. Trakla2 is also capable of simulating other data structures and algorithms such as searching, traversing, sorting, heaps, trees, hashing, breadth-first search, depth-first search, and Prim.

Trakla2 displays a list of nodes, where the list item for each node gives its cost and a list of nodes that it is connected to, but it is static and difficult to relate to the graph. Code is displayed next to the graph, but it is static and the current step is not highlighted. Trakla2 does not display any textual descriptions of the algorithm steps; it only shows the final solution. The user cannot create a custom graph. In manual-practice mode, the user's answer can be reviewed step by step, but there is no feedback provided at each step, so the user will not know whether or not a given step is correct. However, if the user becomes stuck, then the user has the option of viewing the model solution to see the correct answer, although the edge costs are not shown in the graph, making it more difficult to understand the solution.

Despite its shortcomings, student feedback for Trakla2 has been positive. They think it is more elegant than paper and pencil teaching methods and they feel it helps give them more motivation to learn the subject. They think that the tool concretizes the actions and operations of the algorithm, therefore making it easier to learn. Researchers have found that the tool is most helpful for students who are struggling, in that it helps them get over the hump and pass the course. While there has been a formal evaluation of Trakla2 applied to binary heap problems, there has not been a formal evaluation of the tool applied to graph problems.

**MatrixPro** MatrixPro (Karavirta et al., 2004) builds on Trakla2 by allowing an instructor to create custom animations with custom input data sets. The instructor can demonstrate algorithms on the fly by using different input, and this allows for the handling of "what if" questions asked in class, thus enhancing instructor and student interaction. The features of this tool allow the instructor to present algorithms from a conceptual perspective and suppress the coding details. Animations can be prepared before or during lecture. The user is able to customize the look and feel of the interface by controlling what operations are displayed in the toolbar, font sizes, the layout of the interface and the graph data structures, and how the data structures are visualized (e.g., whether or not node labels are shown). A graph can be created by dragging and dropping pre-defined labels from an array into a window that displays the graph structure, however, it is somewhat difficult to figure out how to add edges to the graph. It is possible to cut, copy, and paste different components of the data structures. The user can walk through an algorithm by invoking pre-defined operations

on the data structures, such as inserting a node into a tree. The user is not able to walk through an algorithm by manipulating the data structure at the level of vertices and edges. The speed of the simulation can be adjusted and the granularity of the visualized execution can also be controlled. Breaks in the animation can be inserted to fine tune the granularity of the animation and steps and substeps in the animation can be joined or split. The user can undo and redo operations and rewind the series of steps and reapply them. The user is able to step backward or forward and jump to the beginning or the end of the animation, but there is no support for a history list of operations. It is not clear whether the tool supports continuous simulation of steps, or whether any components of the data structure are moved during simulation. There is no textual description of what is occuring at each step of the simulation. An animation can be loaded, saved, or printed. MatrixPro is capable of handling many types of data structures and algorithms, such as searching, traversing, sorting, search trees, priority queues, hashing, string matching, spatial data structures, and several graph algorithms including depth-first search, breadth-first search, Prim, and Dijkstra's shortest path. As with Trakla2, the user is able to solve problems manually by selecting nodes and edges. There are several built-in example exercises for the user to work through. If users become stuck, they can view a model solution to see the correct answer. A user's work can be graded, although only the number of correct steps is given; the tool does not provide any detailed feedback about what the user has done incorrectly, nor does it provide detailed feedback at each step as the user works through the example.

**PILOT** PILOT [(Bridgeman et al., 2000), (Baker, 2000)] is a Java-based AV tool specific to graph problems that is similar to Trakla2 in that it assigns a grade to a student's submitted solution. The main difference is that it also provides the student with detailed feedback of what they did incorrectly. It also provides immediate feedback at each step of manually practicing the problem by alerting the student to incorrect choices with colors, flashing, and animations. PILOT also awards the student partial credit where appropriate. As with Trakla2, it allows the user to view a model solution, but enhances Trakla2's capability by including textual explanations of each step. Users can undo and redo actions, as well as drag vertices around to improve readability. The tool provides three different modes: Learn, Exam Practice (grade does not count), and Exam (grade counts for the course). Graph algorithms that can be simulated are minimum spanning trees, breadth-first search, depth-first search, and shortest path. Limitations of this tool are that it does not allow the user to create a graph from scratch, it does not provide a history list of operations, and detailed feedback is only given after completion of the problem as opposed to being given at each step. PILOT also does not appear to be publicly available for download.

The developers of PILOT conducted an empirical study where the tool was used in a CS data structures course and compared with results of students solving problems involving Prim's algorithm for minimum spanning trees using a traditional paper and pencil method. It appears that students answered only one problem with each method and that over 90 percent of the students in both the computer and paper and pencil groups achieved a perfect score on those problems, which essentially prevented any conclusions from being drawn about

the effectiveness of the computer tool. The authors of PILOT suggested that it is possible that this outcome is a result of the given problem or algorithm being too easy to learn, and posed the question of whether a similar result would occur with a more challenging problem, such as maximum flow.

### 2.4.5 Summary

The AV tools discussed in this chapter each have their own set of strengths and limitations. Table 2.1 summarizes the capabilities of each of these tools. Discrete step-through mode is not included since all of the tools have this capability, and history list is not included since none of the tools support this feature.

Table 2.1: Summary of AV tool capabilities. Entries with a question mark (?) were unable to be determined with certainty. "N" = "Nodes" and "E" = "Edges".

| Tool | Cont. disp. | Anim. | Step back | Create graph | Text descr. | Man. pract. | Grade | Det. fdbk |
|---|---|---|---|---|---|---|---|---|
| **GALGO** | Yes | N,E blink | No | Yes | No | No | No | No |
| **DSP** | Yes | No | No | Yes | Yes | No | No | No |
| **EVEGA** | Yes | No? | No | Yes | Yes | No | No | No |
| **Swan** | Yes | No | No | Yes | Yes | No | No | No |
| **Net. Flow** | Yes | No | No | Yes | No | No | No | No |
| **JAVENGA** | Yes | N,E blink | Yes | Yes | Yes | No | No | No |
| **DSV** | Yes | N num's | No | No | No | No | No | No |
| **Auckland** | Yes | N,E blink | No | No | Yes | No | No | No |
| **Proj. Links** | Yes | No | No | No | Yes | No | No | No |
| **Animal** | Yes | No | Yes | No | Yes | No | No | No |
| **SPP** | No | No | No | No | No | No | No | No |
| **MRF** | No | No | No | No | No | No | No | No |
| **tutORial** | No | No | No | No | Yes | No | No | No |
| **ALVIE** | No | No | Yes | No | Yes | No | No | No |
| **JHAVE** | No | No | Yes | No | No | No | No | No |
| **Trakla2** | No | No | Yes | No | No | Yes | Yes | No |
| **MatrixPro** | Yes? | No? | Yes | Yes | No | Yes | Yes | No |
| **PILOT** | Yes? | N,E blink | Yes? | No | Yes | Yes | Yes | Yes |

Since the automated "slide show" approach has been used for most of the existing tools, our Sketchmate software for graphs focuses on the implementation of a manual simulation environment for both the student and instructor versions. Users can manually manipulate graph components, which more actively engages the user than simply clicking through a series of representations of the graphs. The automated solution approach is actually implemented for shortest path and network flow problems, but behind the scenes as a means of verifying

that the user's solution is correct. The student Sketchmate tool can also be used as an automated discrete stepping tool if the user repeatedly clicks the "Submit" and "Continue" buttons without manipulating the graph components in any way. In this manner, the student tool can be used as an automated instructor tool which provides detailed feedback at every step. An additional feature of the instructor tool is a notepad where the instructor can interactively add explanations of each step of the algorithm. Thus, the instructor tool functions as an enhanced whiteboard, where an instructor is able to manually simulate the steps of an algorithm and add verbal explanations as the graph structure is being worked on. As mentioned previously, the instructor tool also includes a Revert pane for reviewing previous steps of the algorithm. The research performed in this dissertation also differs from the work done with other AV tools in that we performed formal experimental studies of the effectiveness of both the instructor tool and the student tool. PILOT and JAVENGA were the only two projects in which the researchers formally evaluated learning outcomes of experiments with graph problems (Trakla2 researchers formally evaluated learning outcomes, but for experiments with binary heaps, not graphs). Our study also differs from these previous studies in that it focuses on both learning rates and learning outcomes.

## 2.5 Sketchmate for Splay Trees

Orsega's Sketchmate for splay trees [(Orsega, 2009), (Orsega et al., 2011), (Orsega et al., 2012)] includes an instructor tool for demonstrating splay tree operations during lecture, and a student tool for working practice exercises. The instructor tool allows a user to create a custom splay tree on which to operate. The tool performs operations on the created splay tree and displays this process as a series of animations with pauses between steps. Users can also revert to previous operations by choosing the desired operation from a history list. The student tool contains several built-in exercises in which the student can manually perform operations on the given splay tree. An image of the previous splay tree structure is displayed beside the splay tree the student is currently working on. When the student performs an incorrect operation, detailed feedback is given on what the student has done incorrectly. Students also can receive a grade for their work.

Experiments with the instructor tool studied its usability in a lecture setting, and experiments involving the student tool analyzed student learning outcomes and learning rate. The results of the experiments with the student tool were that learning outcome was slightly increased when using Sketchmate versus traditional paper and pencil methods, although the increase is not statistically significant. However, using Sketchmate did result in learning rate being increased significantly. As stated in Orsega's dissertation, "students were able to complete the exact same exercises nearly 40% faster when using Sketchmate versus paper and pencil."

Sketchmate for graphs shares several similarities with Sketchmate for splay trees. Sketchmate for graphs also includes both instructor and student tools. The instructor tool allows users to create a custom graph, simulate algorithms operating on the graph, and revert to previous steps of the algorithm's process via a history list. The student tool enables a stu-

dent to practice graph algorithm problems, and provides a grade for their work, as well as detailed feedback explaining what the student has done incorrectly. Sketchmate for graphs also includes studies on the usability of the instructor tool in a classroom environment, and studies exploring Sketchmate's effect on student learning outcomes and learning rate.

Sketchmate for graphs differs from Sketchmate for splay trees in a number of ways. The most fundamental difference is that our Sketchmate is applied to a data structure other than splay trees. Another major difference is that the instructor tool allows the user to manually simulate graph algorithms in a type of "enhanced whiteboard" environment, as opposed to viewing a slide show presentation of algorithm steps. An image of the previous graph is displayed concurrently with the graph currently being worked on for both the instructor and student tools. The instructor tool also includes a notepad feature, which allows instructors to annotate their simulations with notes explaining what is occurring at a given step. The student tool allows the user to input any graph, and thus does not restrict the student to work with built-in exercises. A graph problem is solved by the computer in the background, and detailed feedback is generated from this automated solution. The automated solution can also be used to verify the instructor's solution. The experiment for the instructor tool also evaluated student learning rate and learning outcome in addition to usability in the classroom.

# Chapter 3

# Research Goals

This research has two broad objectives: 1) developing a tool that supports computer-aided manual simulations of algorithms to augment traditional whiteboard presentations, allowing lectures to be more dynamic and interactive, and 2) improving student learning rate and accuracy by developing a tool that supports computer-aided student practice of algorithms by enabling students to work through homework problems more quickly while providing detailed incremental feedback about their performance and about how to solve a problem when they get stuck. Between the instructor tool and the student tool, all features in Table 2.1 except for the first two features were implemented. The Sketchmate instructor tool for graphs also includes several features not present in existing AV tools for graphs, as discussed below.

- Supporting manual simulation of the algorithms in addition to the automated solution computed in the background
- Including a revert pane that lists all steps performed on the graph and allows the viewing of any previous step in the simulation
- Displaying a diagram of the previous step along with a diagram of the current step so that students can easily see the changes between steps
- Providing a notepad feature for adding explanations for each step
- Using the automatic solution in the instructor tool to check the manual solution and ensure that all state information is updated correctly before the instructor advances to the next step
- Developing an AV tool that is easy, fast, and convenient to use, requires a low learning curve to use, and uses class time more efficiently
- Exploring mixed modes of delivery where the student can view the computer-aided manual simulation of an algorithm as the instructor modifies various parts of the screen and simultaneously hear the instructor's explanation of each step

As with the instructor tool, the student tool displays diagrams of both the previous and current graphs simultaneously, and an effort has been made to develop the student tool to be as easy and fast to use as possible. An additional feature includes allowing the user to practice with any custom-made graph and not be restricted to built-in exercises.

Another goal in developing the student tool was to increase student learning rate compared with that obtained from using paper and pencil. Learning rate refers to how much a student learns in a given amount of time, as illustrated in Figure 3.1. In our experiments, students were given an open-ended set of test problems and practice exercises to see how many points they could accumulate and how many problem steps they could successfully complete in a fixed amount of time. The hope was that students could earn higher scores for number of points and number of steps using Sketchmate (thus an increased learning rate), and that this additional practice would lead to improved learning outcomes on a more complicated algorithm, such as the maximum flow network flow problem.

An earlier study (Baker, 2000) had suggested that manual simulations might not lead to improved learning outcomes on a simpler algorithm, such as shortest path, but that it might lead to improved learning outcomes on a more complicated problem like network flow. We thus built shortest path and network flow problems into our student tool and performed an experiment that measured student learning rates and learning outcomes on each algorithm. We hoped that learning rates and learning outcomes would improve on network flow, and could not be sure what would happen with shortest path.

A recent study (Babcock and Marks, 2010) has shown that students currently spend 27 hours a week studying or completing homework, while 50 years ago they spent 40 hours a week studying. If our tool indeed increases a student's learning rate, then it can help compensate for the reduced amount of time students spend studying.

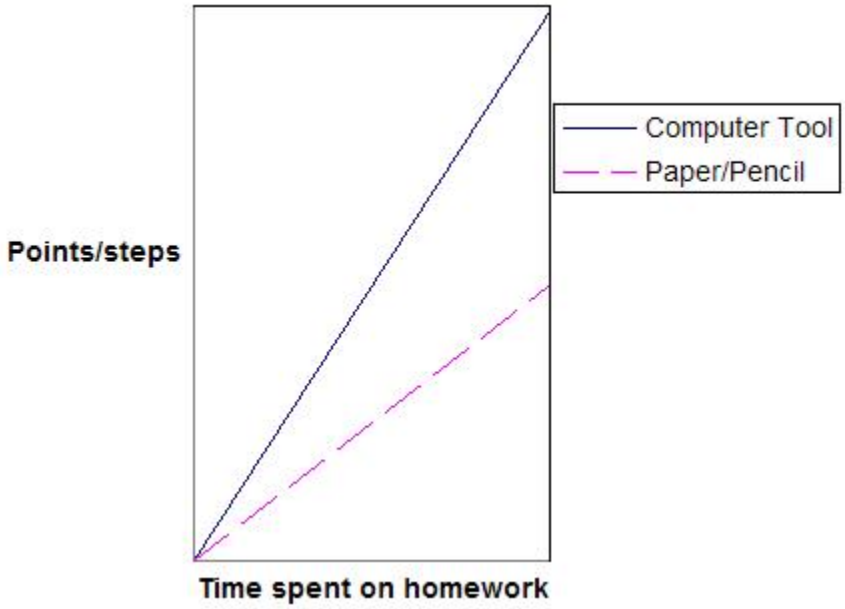


Figure 3.1: Hypothesized learning rate for Sketchmate compared with learning rate for paper and pencil. The y-axis indicates the number of points accumulated or the number of steps successfully completed. Points/steps increase as one moves up the y-axis and time increases as one moves rightward along the x-axis. We did not necessarily expect to double the learning rate but we hypothesized that the learning rate would be increased for network flow problems.

# Chapter 4

# Overview of Graph Algorithms

This chapter first gives a brief overview of basic graph terminology. Next it presents a brief description of the shortest path and maximal network flow algorithms and illustrates each algorithm performing on an example graph. A full presentation of these algorithms can be found in textbooks such as (Weiss, 2006) and (Cormen et al., 1990).

## 4.1 Graph Terminology

A graph is a collection of vertices and edges. An edge connects two vertices. Vertices can be thought of as cities on a map, and edges can be thought of as road between the cities. A graph may be directed or undirected. If the graph is directed, then each edge is a one-way edge that is represented as an arrow that points from the source vertex to the destination vertex. Edges may have a cost or weight associated with them, that can represent measurements such as the distance between two cities or the capacity of a pipe. Figure 4.1 shows an example of a directed graph with five vertices. The numbers along the edges are the edge costs.

## 4.2 Dijkstra's Shortest Path Algorithm

Dijkstra's shortest path algorithm involves finding the shortest path from a start vertex to each of the other vertices in the graph, where the edges in the graph must have positive cost. A path is given by a sequence of vertices. Initially each vertex in the graph is marked as unseen. The start vertex s is assigned a cost of 0 and the remaining vertices are assigned a cost of infinity. A vertex's cost represents the length of the shortest path found thus far from the start vertex to this vertex using only visited vertices. The algorithm proceeds in steps, with the first step visiting the start vertex and each subsequent step visiting a previously unvisited vertex. The algorithm selects the lowest cost unvisited vertex (i.e., the unvisited vertex with the shortest path from the start vertex) as the next vertex to visit. When the algorithm visits a vertex v, it marks the vertex as visited, and then examines each of v's neighbors to determine whether there is a shorter path to each neighbor w which includes v. This is done by adding v's cost and the edge cost from v to w and checking whether

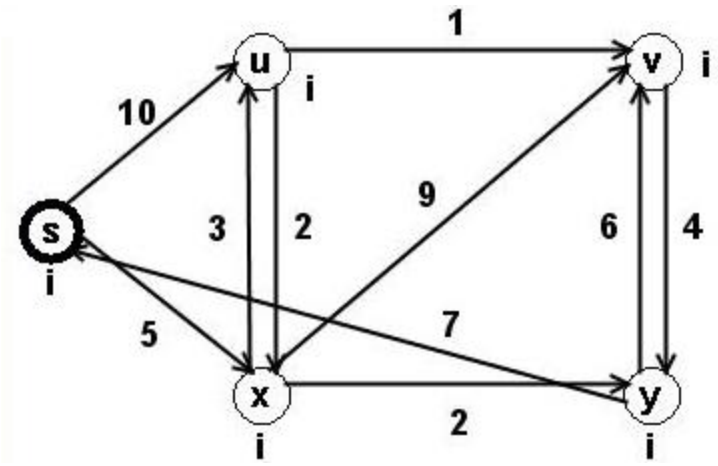


Figure 4.1: Directed graph with five vertices

the sum is less than the current cost of w. If so, then there is a shorter path to w through v that includes the edge v-w. The cost of w is decreased to the cost of this new shortest path and the edge v-w is selected as being on the shortest path from the start vertex to w. Some textbooks use another state to either denote vertices that have been seen but not yet visited or to denote the vertex that is currently being visited. Our Sketchmate tool supports these various notations by allowing a vertex to be in one of four states: visited, seen but not visited, unseen, and currently being visited.

As an example of Dijkstra's algorithm, consider how it would operate on the graph in Figure 4.1. The start vertex is visited first (marked with a thickened border in the graph). The result of this step is shown in Figure 4.2. The start vertex s has been colored gray to denote it as visited. Then, each of the neighbors of s, which are u and x, are examined one at a time. These two vertices are colored green to denote that they are vertices that have been seen but not visited. Edge s-u has a cost of 10, so the cost to go from s to u is 10, thus the cost of u is updated to 10. Similarly, the cost of x is updated to 5. Note that "i" denotes a cost of infinity. The edges s-u and s-x are now shaded to mark them as being on the shortest path, since the shortest path to u and to x is from s. The step involving visiting vertex s is now complete, and the algorithm moves to the next step.

Figure 4.3 shows the result of the second step of the algorithm. The vertex to be visited is chosen as the vertex with the lowest cost that has not yet been visited. In this case, that vertex is x, as indicated by its gray color in the figure. The algorithm then examines each neighbor of x: u, v, and y. Each of these neighbors is colored green if it has not been colored green previously. We will first examine vertex u. The cost of x-u is 3, while the cost from s to x is 5. Adding these together yields a total cost of 8, which is less than the previous cost of u, which was 10. The cost of u is now updated to 8, since s-x-u is a shorter path than s-u. The edge x-u is now shaded since it is on the new shortest path from s to u, and the edge s-u is unshaded since it is no longer on the shortest path from s to u. Next we will examine

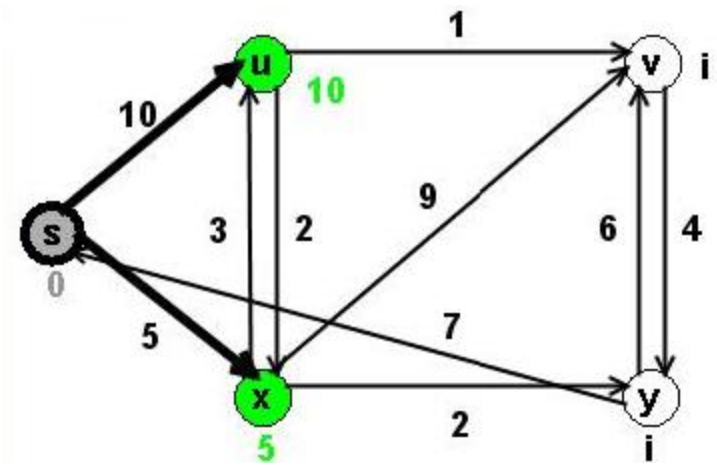


Figure 4.2: Shortest path after visiting vertex s

vertex v. The cost of edge x-v is 9, while it is a cost of 5 from s to x, thus the total cost of v is updated to 14. The edge x-v is shaded to mark it as being on the shortest path. In a similar manner as for vertex v, the cost of vertex y is updated to 7, and the edge x-y is added to the shortest path.

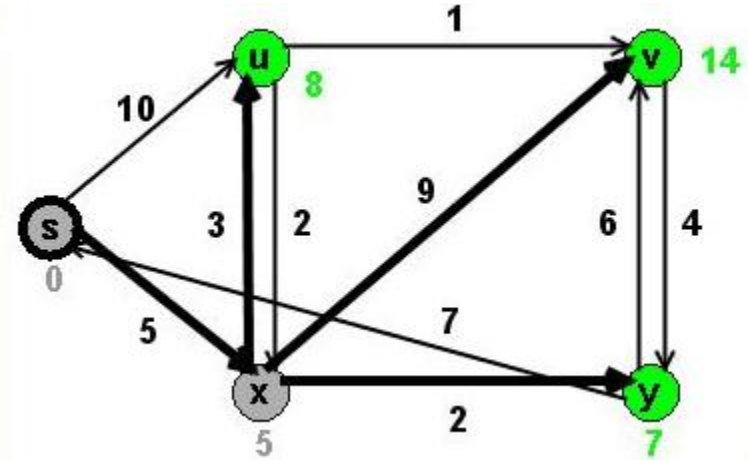


Figure 4.3: Shortest path after visiting vertex x

The algorithm proceeds in this manner until all vertices have been visited. Note that when a vertex's neighbors are examined, only the neighboring vertices that have not yet been visited are examined. The final graph, which is illustrated in Figure 4.4, gives the shortest path from the start vertex s to each of the other vertices in the graph.

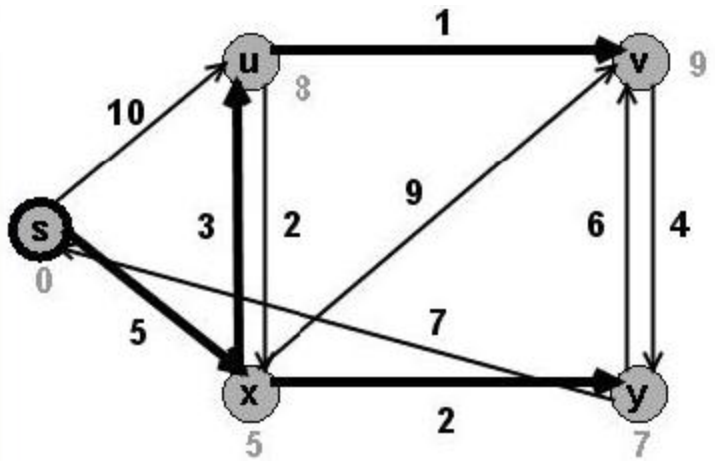


Figure 4.4: Shortest path after visiting all vertices

## 4.3 Maximal Network Flow Algorithm

The maximal network flow algorithm involves finding the maximal amount of flow that can be pushed through a network. A graph in this case can be thought of as a network with water flowing through a system of pipes. The network contains a source vertex and a sink vertex. The purpose of the algorithm is to find the maximal amount of flow from the source vertex to the sink vertex. Flow passes through the edges in the network. Each edge is assigned a capacity to indicate the maximal amount of fluid that can flow through that edge. The combined flow out of a vertex must equal the combined flow into the vertex. Figure 4.5 shows the flow through a graph initially on the left, and the maximal flow through that graph after performing the network flow algorithm on the right. The left number associated with each edge is the amount of flow through that edge, and the right number is the capacity of that edge.

The network flow algorithm involves two graphs: a flow graph to keep track of the flows along each of the edges, and a residual graph to keep track of the amount of flow that can still be added to each edge. An example of a flow graph and a residual graph is given in Figure 4.6. In the flow graph on the left, the flow values are initialized to 0. In the residual graph on the right, the numbers along the edges denote the free capacity of the respective edge. These capacities are the same as the corresponding capacities in the flow graph at this point, since all of the flow can still be added to the edges.

As the algorithm starts to allocate flow to various edges, backedges start being added to the residual graph, which allow flow to be "pushed back" along the forward edge, thus reducing the amount of flow allocated to that edge. For example, in Figure 4.7, there is a flow of 3 units along the edge a-d and there is a backedge in the residual graph with 3 units of flow from d to a, which indicates that up to 3 units of flow can be pushed back along the edge d-a, thus reducing the flow allocated to edge a-d by up to 3 units. The thick magenta edges will be explained shortly.

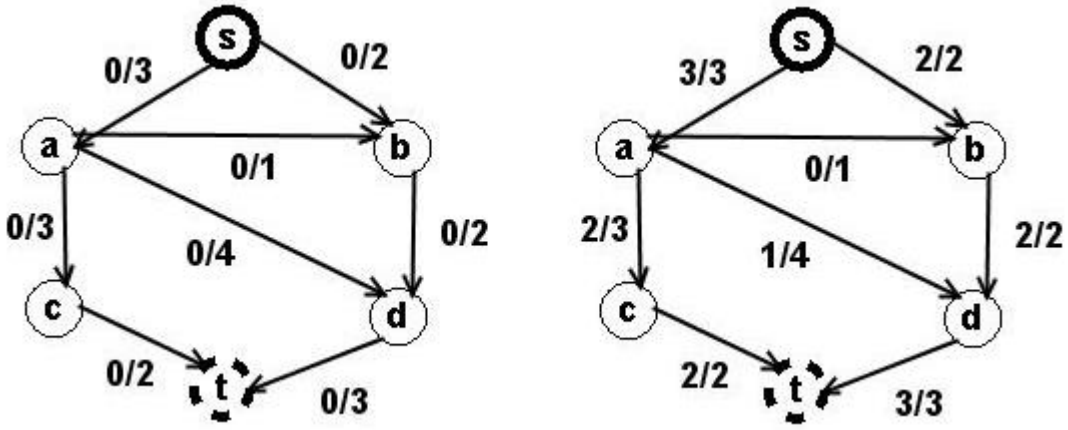


Figure 4.5: Initial and final flow through a graph for a network flow problem. The flows along the edges are initially 0. The final graph shows the maximal flow along each edge.

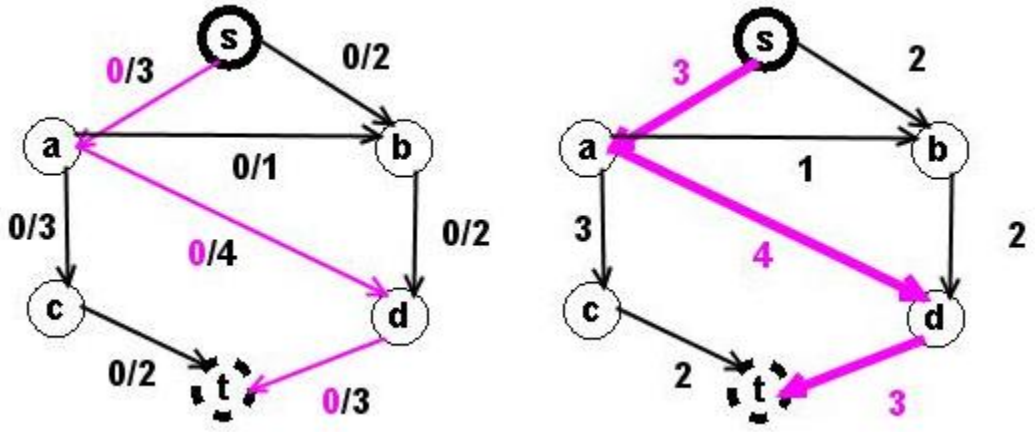


Figure 4.6: Flow and residual graphs of a flow network

Each step of the network flow algorithm consists of two parts: first, finding the maximal augmenting path through the network, and second, updating the flow and residual graphs along this path. An augmenting path is a path from the source vertex to the sink vertex that takes into account the amount of flow that can still be pushed through the edges. Thus, the residual graph is used to find the augmenting path. The flow for a path through the network from source to sink is given by the smallest capacity of all the edges on that path. To find the augmenting path that will yield maximal flow, a variant of the Dijkstra's shortest path algorithm is used, where the goal is to find a maximal flow rather than a smallest cost. According to (Weiss, 2006), choosing a maximal flow augmenting path at each step will minimize the number of steps required to find a maximal flow through the network. The augmenting path in the first step using our example graph is denoted by the magenta shaded edges in Figure 4.6. This path contains a flow of 3, which is the maximal flow that can be obtained among any of the paths from source vertex s to sink vertex t.

The next part of this step is to update the flow and residual graphs along this augmenting path. Figure 4.7 shows the updates to these two graphs (the magenta edges in these graphs denote the augmenting path of the next step, which can be ignored for the moment). The flow along the augmenting path is 3, thus, 3 is added to the flow for each of the three edges along the augmenting path in the flow graph. In the residual graph, the flow of 3 is subtracted from each forward edge along the augmenting path since their free capacity has been decreased by 3 units. If the free capacity of an edge becomes 0, then that edge is removed from the residual graph. A flow of 3 is added to the corresponding backedges of the augmenting path, to indicate that 3 units of flow can be "pushed back" along the edge to free up capacity in the pipe. Note that if a backedge did not previously exist, then it is added to the residual graph.

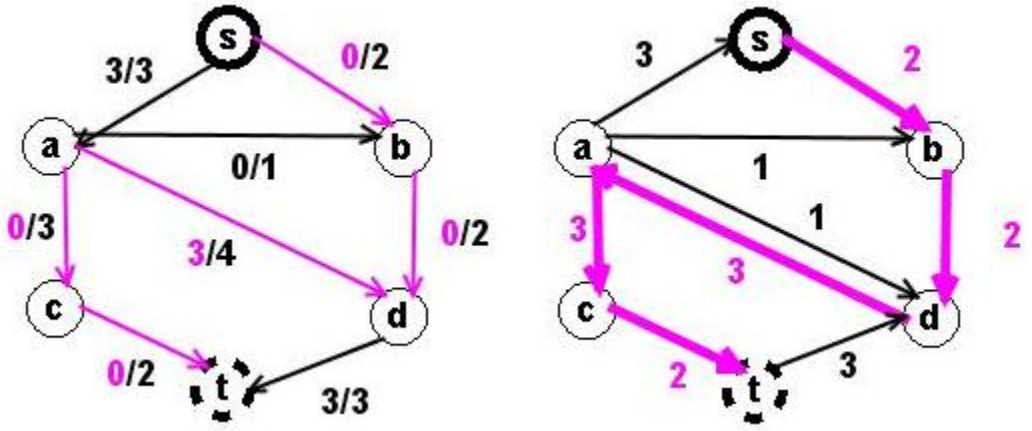


Figure 4.7: Flow and residual graphs after the first step of the network flow algorithm

At this point, the next step of the algorithm begins. The maximal augmenting path is shown in the residual graph of Figure 4.7. In this case, the edge d-a is actually a backedge

that was added in the previous step; this edge is not in the original graph. Because of this, the updating of the flow and residual graphs is altered slightly, as shown in Figure 4.8. The flow along the augmenting path is 2 units. In the flow graph, 2 units of flow are added to each of the edges, except for a-d. For the edge a-d, since its corresponding edge in the residual graph is a backedge, the flow of 2 units is *subtracted* from the flow of a-d, reducing its flow to 1. In the residual graph, the flow of 2 is subtracted from each of the edges along the augmenting path, and the flow of 2 is added to the backedge corresponding to each edge along the path. Note that the backedge of d-a is a-d, which is an edge in the original graph. In this case, since the algorithm is pushing 2 units of flow "back" through the backedge, it is actually increasing the free capacity of the pipe from a to d by 2 units.

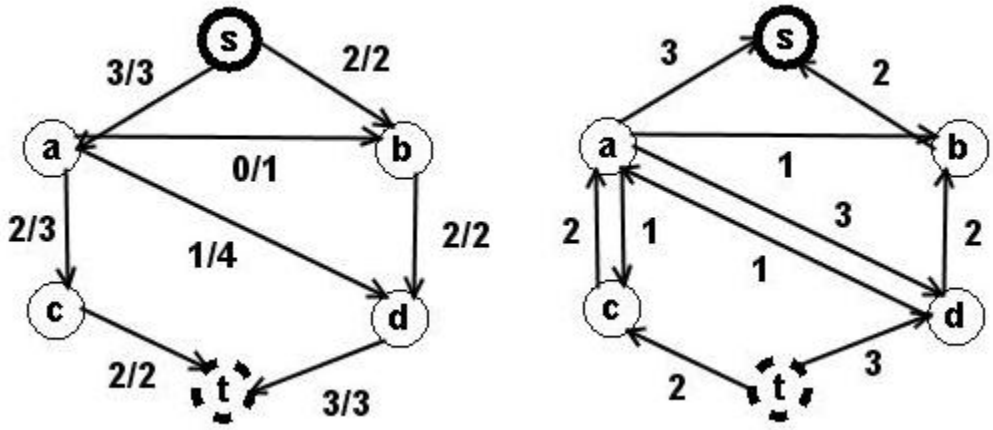


Figure 4.8: Flow and residual graphs after the second step of the network flow algorithm

After updating the flow and residual graphs in this step, the problem is now complete because there is no longer a path from the source vertex s to the sink vertex t in the residual graph. The final result is that the maximal flow of this network is 5 units, since 3 units flow out of s to a and 2 units flow out of s to b. Note that the amount of flow coming out of s is the same as the amount of flow coming in to t.

# Chapter 5

# Description of the Sketchmate Environment

Two versions of the Sketchmate tool have been implemented and tested: one for instructors to use during their lectures, and another for students to use for studying and completing homework exercises. Many of the same or similar features are included in both tools, which are focused on graphs. Graphs consist of vertices and edges, where the vertices contain a label and a cost in the case of shortest path, and the edges contain a cost in the case of shortest path, and a flow and/or capacity in the case of network flow. A flow in a network flow problem indicates the amount of substance currently flowing through its associated edge (or pipe) in the flow graph. A capacity in a network flow problem indicates the amount of substance that can still flow through its associated edge (or pipe) in the residual graph. Both tools support a number of domain-specific operations required by shortest path and network flow algorithms, including:

1. Setting the state of a vertex to one of visited, seen but not visited, unseen, and currently being visited.

2. Highlighting or unhighlighting edges that are part of certain paths, such as a shortest path or an augmenting path.

3. Modifying vertex costs to represent updates to a vertex's cost, such as the length of the current shortest path from the start vertex to that vertex.

4. Modifying edge labels, such as changing the flow of an edge in a flow graph or the capacity of an edge in a residual graph.

5. Adding edges to or removing edges from a residual graph.

Both the instructor and student versions are capable of working with both the shortest path and network flow algorithms. The instructor tool includes three different panes: Create, Simulate, and Revert, while the student tool includes the Create and Simulate panes.

This chapter is organized as follows. First a discussion of the Create pane for both the instructor and student tools is presented. Following is a description of the Simulate and Revert panes for the instructor tool. Next, a discussion of the Simulate Pane for the student tool is given. Finally, step-by-step sample Sketchmate sessions for both tools are presented: first appears a session of the student tool applied to shortest path, and second appears a session of the instructor tool applied to network flow.

## 5.1 Create Pane

Both the instructor and student tools contain the Create pane. The Create pane contains a blank area of the screen where the user can create a custom graph. It supports several operations for adding vertices and edges and editing their labels, costs, flows, and capacities. A screen capture of the Create pane in the instructor tool with a shortest path problem is shown in Figure 5.1. The upper left portion under the toolbar contains the graph that is being created (this example graph is from (Cormen et al., 1990)). Underneath the graph area is a set of operations for adding or modifying components of the graph. The panel to the right of the graph drawing area includes basic instructions for how to create a graph. The instructions vary, depending on which operation mode is currently selected. This example shows the instructions for the "Select" mode.

### 5.1.1 Adding and Modifying Vertices and Edges

A vertex can be added by selecting the "Add vertex" mode and clicking on the desired position in the graph area. A label will be automatically assigned and appear in an editable textbox inside the vertex. Another editable textbox will be added underneath the vertex to denote its cost for a shortest path graph (default value is infinity, which is denoted as "i" in the figure). If the network flow algorithm is selected, then no cost will appear underneath the vertex.

The user can add an edge by dragging a line from the source vertex to the destination vertex while in "Add edge" mode. If the "directed" radio button in the toolbar is selected (the default), an arrowhead will appear at the destination vertex. An editable textbox containing the cost or capacity (default value is 0) will also be added along the middle of the edge.

A vertex label or cost can be changed by selecting the "Change cost/capacity/label" mode and then clicking on the textbox associated with the vertex label or cost, editing the label or cost, and pressing the "Enter" key to commit the change. An edge cost or capacity can be changed in a similar way.

To delete a vertex, while in "Select" mode, the user can click on the chosen vertex and press the delete or backspace key. All associated textboxes and edges will also be deleted. Deleting an edge can be performed in a similar manner.

Also while in "Select" mode, the user can move a vertex by clicking on it and dragging it to its new location. All attached edges and all associated costs or capacities will move

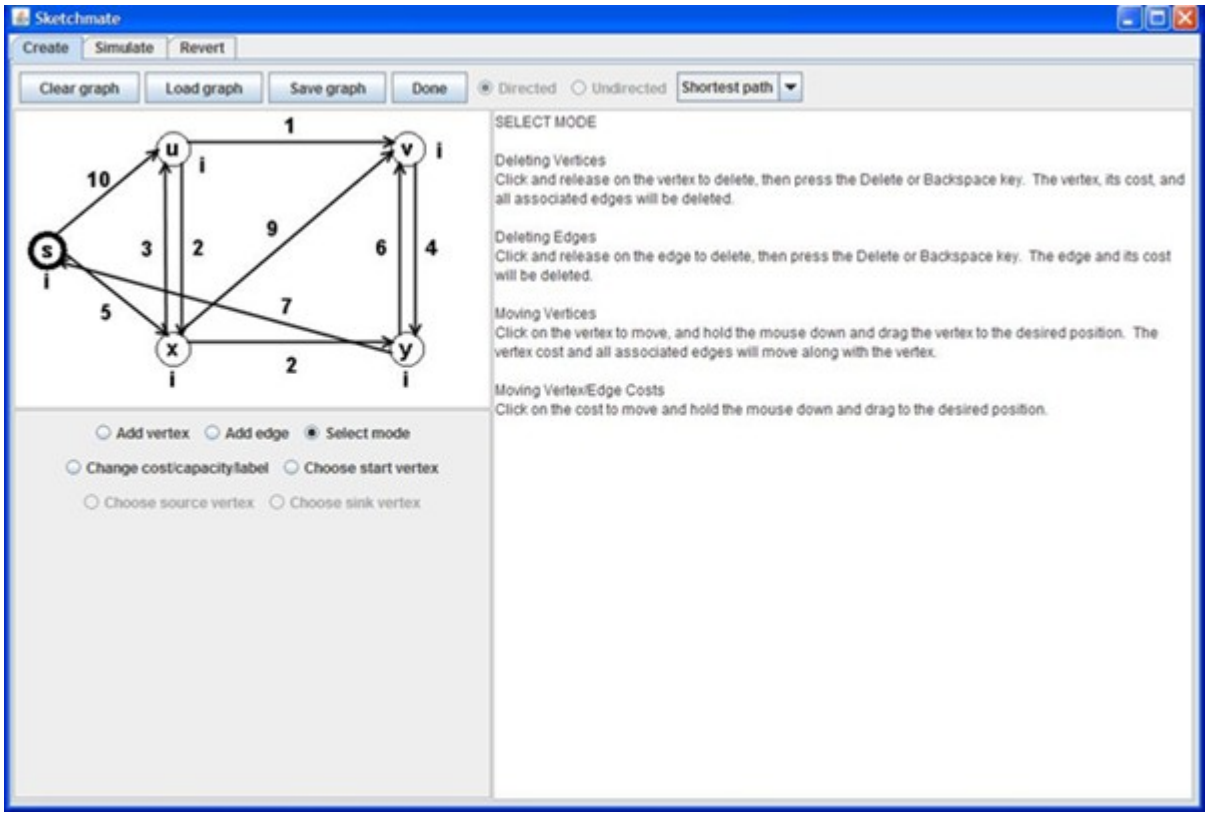


Figure 5.1: Create pane for both the instructor and student tool

along with the moved vertex. The individual cost and capacity textboxes for the vertices and edges can be moved similarly. Allowing users to move the various graph objects will enable them to improve the readability of the graph.

There are also modes for assigning a start vertex in the case of shortest path, and a source vertex and sink vertex in the case of network flow. To assign a start, source, or sink vertex, the user can click on the corresponding radio button operation, and then click on the desired vertex. The chosen start or source vertex will then be drawn with a thickened solid border, and the chosen sink vertex will be drawn with a thickened dashed border.

### 5.1.2 General Operations of the Create Pane

Clicking on "Clear graph" clears the graph drawing area and allows the user to draw a new graph from scratch. When the user is finished creating a graph, clicking on "Done" will transfer the newly created graph to the Simulate and Revert panes. The user also has the ability to load a previously created graph and save a graph for later use. These features enable an instructor to create graphs prior to lecture, as well as continue a simulation with a particular graph in subsequent lectures. It also allows students to reuse graph examples for any number of study sessions.

Other operations in the toolbar include selecting the type of graph (directed or undirected), and selecting the algorithm to simulate from the choices "Shortest path" and "Network flow".

## 5.2 Simulate Pane

The instructor tool Simulate pane allows the instructor to manually manipulate the graph objects to step through the chosen algorithm, thus functioning as an "enhanced whiteboard". The tool has specific built-in knowledge for shortest path and network flow that allows it to check the correctness of a solution after each step of the algorithm has been completed.

### 5.2.1 Simulate Pane for Shortest Path

A screen capture of the Simulate pane using the shortest path algorithm appears in Figure 5.2. The upper left area below the toolbar contains the graph that is being worked on. The darkened edges denote edges along the shortest path found so far from the start vertex to each of the other vertices. An edge can be darkened by clicking on it, and undarkened by clicking on it again. The start vertex appears with a thickened border. The vertices are colored based on their current state: white for a vertex that has not yet been visited or seen, light blue for a vertex that is currently being visited, green for a vertex that has been seen but not yet visited, and gray for a vertex that has been visited. The state of a vertex can be changed by clicking on the appropriate button below the graph to select the desired state, and then clicking on the vertex to change its state. A vertex cost of infinity is indicated with

an "i" underneath or next to the vertex. The user can change the costs of the vertices by clicking on and editing the associated textboxes.

The area to the right of the current graph contains the graph as it appeared at the beginning of the current step so that a student can easily observe the changes that occurred from the previous step to the current step. A step involves the visitation of a single vertex. Visiting a vertex includes sub-steps such as examining each neighboring vertex, changing the costs of the neighboring vertices, and shading an edge if it is on the new shortest path from the start vertex, along with unshading an edge that is no longer on the shortest path from the start vertex. The right-hand region under the toolbar contains a notepad-like object where the instructor can interactively type in notes as the algorithm progresses. The two blank areas on the bottom of the interface are used for network flow, and therefore are not used in this example.

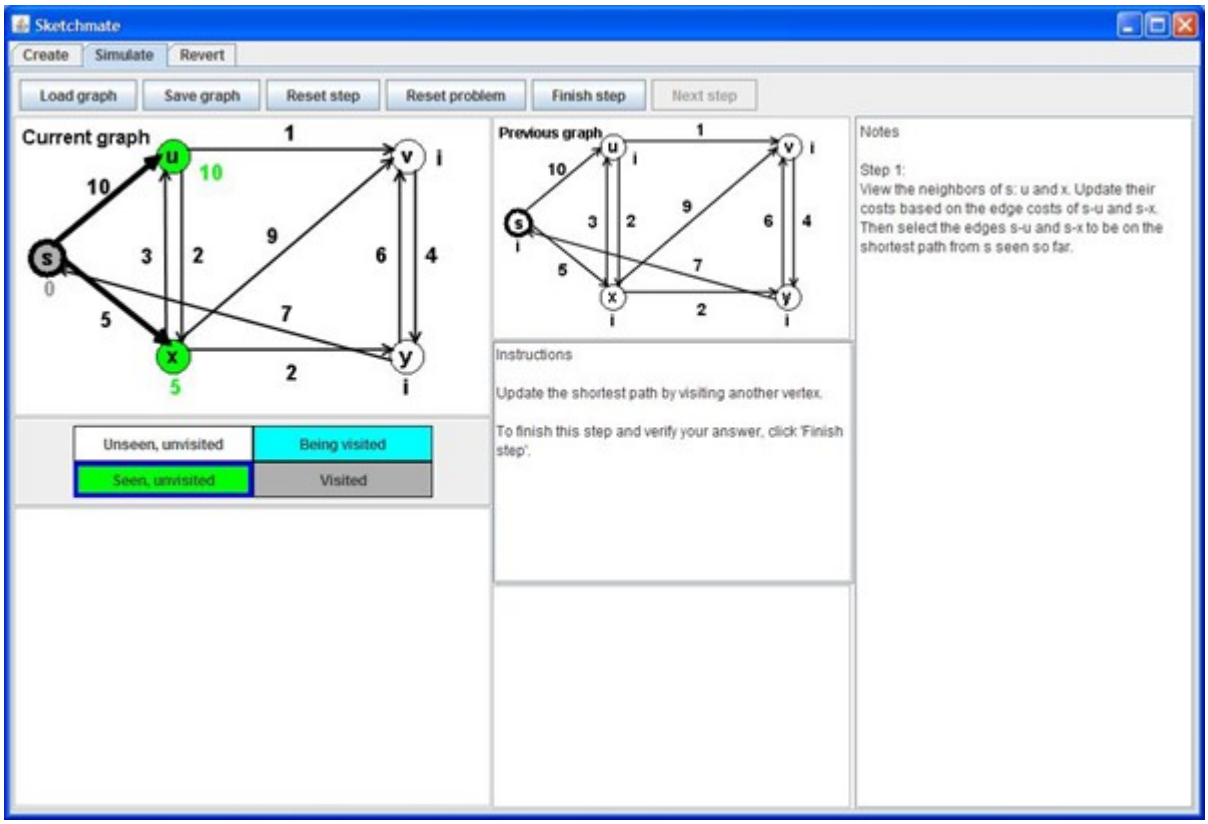


Figure 5.2: Instructor simulate pane for shortest path

### 5.2.2 Simulate Pane for Network Flow

This section describes examples of an instructor working through a network flow problem. Each step of the algorithm includes two substeps: the first substep involves finding the

maximal augmenting path, and the second substep involves updating the graph components along the augmenting path in the flow and residual graphs. Finding the augmenting path involves shading the appropriate edges in the residual graph. Updating the flow and residual graphs involves changing the flows in the flow graph, adding or removing edges in the residual graph, and changing edge capacities in the residual graph.

Figure 5.3 shows an example of how the interface might look after finding the first augmenting path in the residual graph. The upper left region contains the current flow graph, and the lower left region contains the residual graph. The source vertex appears with a thickened solid border and the sink vertex appears with a thickened dashed border. Marking the augmenting path involves selecting edges in the residual graph, as shown by the thickened edges in the previous residual graph. At this point, the instructor is about to update the flow and residual graphs. In both the current flow and current residual graphs, all graph components along the augmenting path, as well as the augmenting path itself, are highlighted in magenta to denote that these are the components that need to be updated.

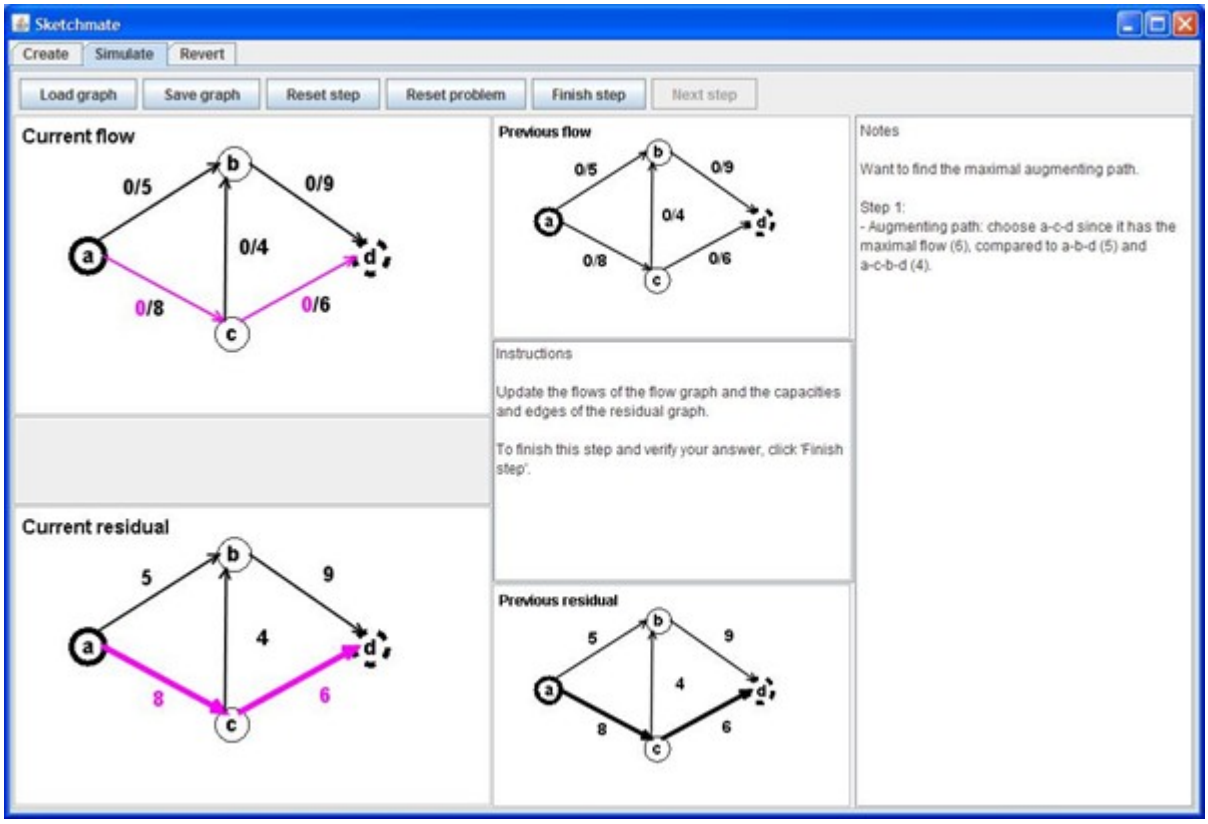


Figure 5.3: Instructor simulate pane after finding the first augmenting path for network flow

An example of the Simulate pane for partially updated flow and residual graphs is given in Figure 5.4. The graph components that still need to be modified are highlighted in magenta. In the flow graph, the left hand numbers along the edges are the edge flows, and the right

hand numbers are the edge capacities. The numbers along the edges in the residual graph are the edge capacities. The previous flow and residual graphs are displayed to the right of their respective current graphs. The middle panel contains instructions to remind the user which substep is currently being worked on. The rightmost pane contains a notepad where the instructor can type notes explaining the steps for solving the problem.

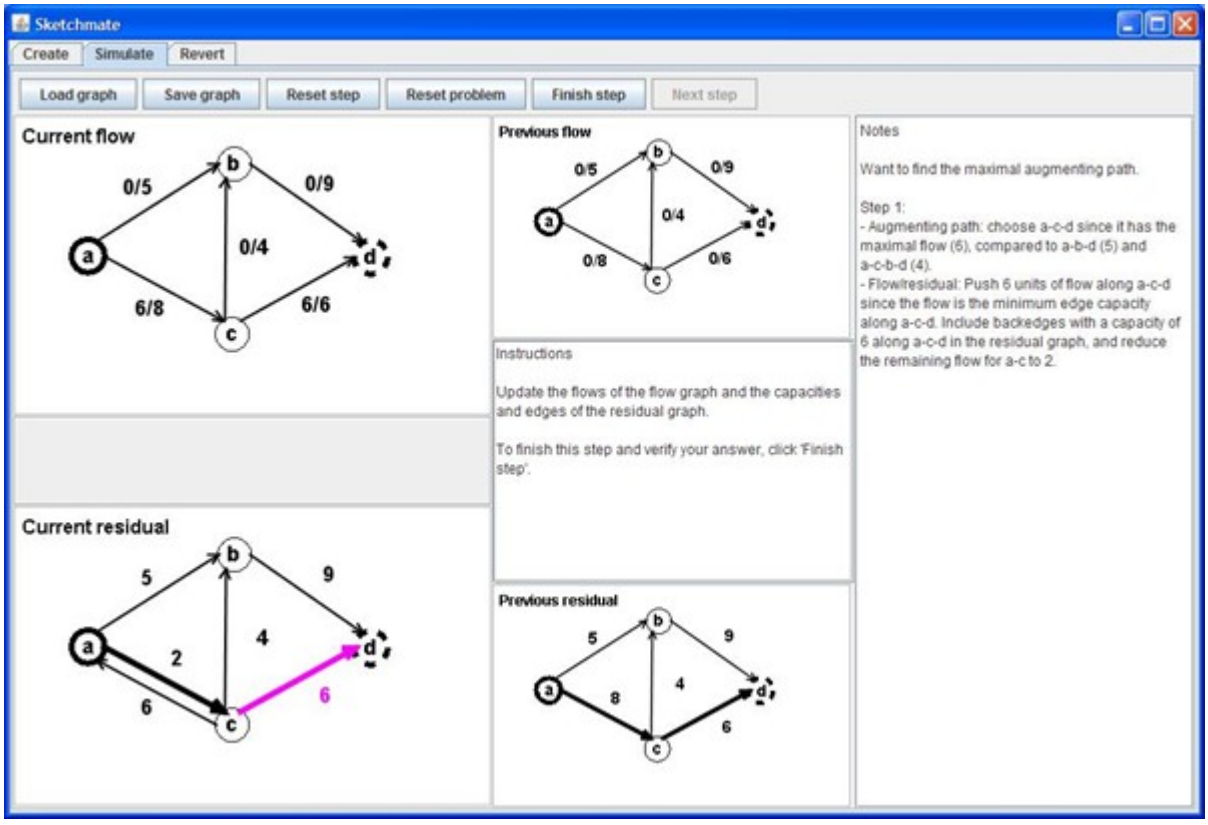


Figure 5.4: Instructor simulate pane during updating the residual graph for the first augmenting path for network flow

### 5.2.3 General Features of the Simulate Pane

Sketchmate has built-in knowledge of how to simulate a shortest path or a network flow problem. The automated solution is created behind the scenes and it can be used to verify the instructor's work.

When the instructor wishes to move to the next step, the "Finish step" button can be clicked to first check the instructor's solution for that step by comparing it to the corresponding step in the solution that is automatically computed in the background. If any components of the graph are incorrect, those components will be highlighted in red. The instructor is free to correct any mistakes at this point. Clicking "Next step" updates the

graph to its correct state and allows the instructor to begin the next step of the algorithm's simulation.

Behind the scenes, the state of the graph after each finished step is recorded. These recordings build a history list that can be accessed through the Revert pane. Clicking on the "Reset step" button will restore the graph to the last recorded step. This allows instructors to undo their last few operations in case they make a mistake. Clicking on "Reset problem" will start the problem over again from the beginning.

## 5.3 Revert Pane

The Revert pane, as shown in Figure 5.5 for a network flow problem, allows users to revert back to any previous step of the algorithm's execution. Reversion can allow the instructor to review previous steps by successively stepping through the radio buttons, or answer a student's question. A listing of the steps of the algorithm is displayed and the user can choose the desired step from the list. The history list displays the order in which the steps are completed. When a step is chosen, the graph corresponding to that step will be displayed in the graph viewing area. After the user clicks on "Restore graph", the associated graph will be restored to the Simulate pane. The list of steps is generated from the solution that is automatically computed behind the scenes in the Simulate pane. Note that the list of steps in this case alternates between finding an augmenting path and updating the flow and residual graphs.

## 5.4 Student Tool

Like the instructor tool, the student tool contains a Create pane and a Simulate pane, but in the student tool the Simulate pane also provides feedback for the submitted solution. Another difference from the instructor tool is that the student tool does not contain a Revert pane.

### 5.4.1 Student Simulate Pane for Shortest Path

A screen capture of the student tool for shortest path is given in Figure 5.6. The user is able to manually manipulate the graph objects in the graph area in the upper left region below the toolbar. The graph at the beginning of the current step is shown at the far right of the interface to provide a reference for the student. Orsega (Orsega, 2009) found that displaying the previous state of the data structure helped prevent a student from getting "lost" when updating the data structure. By being able to check back to the previous state of the data structure, the student could see what changes had already been made and what changes still needed to be made. The correct graph is shown next to the current graph when the student submits a solution to the given step. In this case, it is the same as the current graph since the user correctly updated the graph for this step. The center panel of the right-hand side

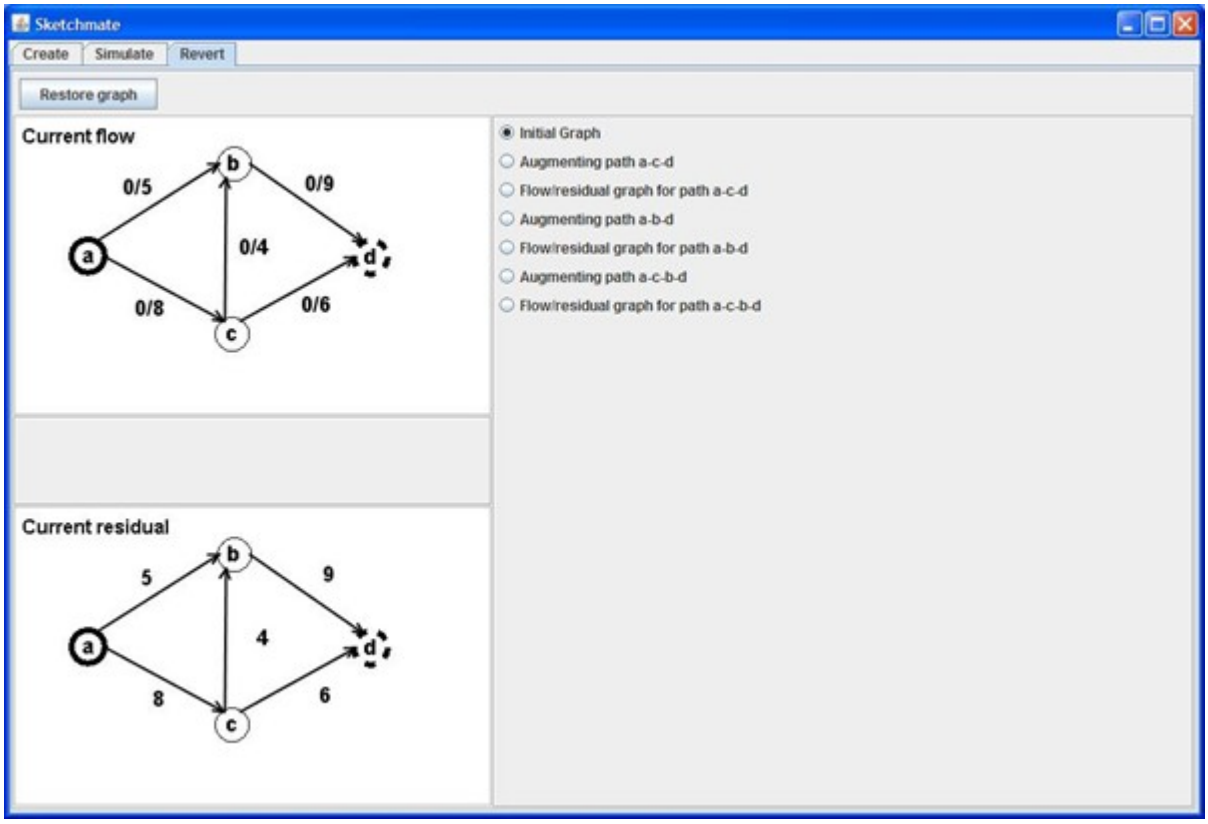


Figure 5.5: Revert pane (instructor tool only)

of the interface contains either instructions or feedback. If the user is working through the current step, then instructions of how to manipulate the graph are displayed. If the user has submitted a solution for the current step, then feedback on the submitted solution is presented. The three bottom panels of the interface are used for network flow, so they can be ignored for this example.

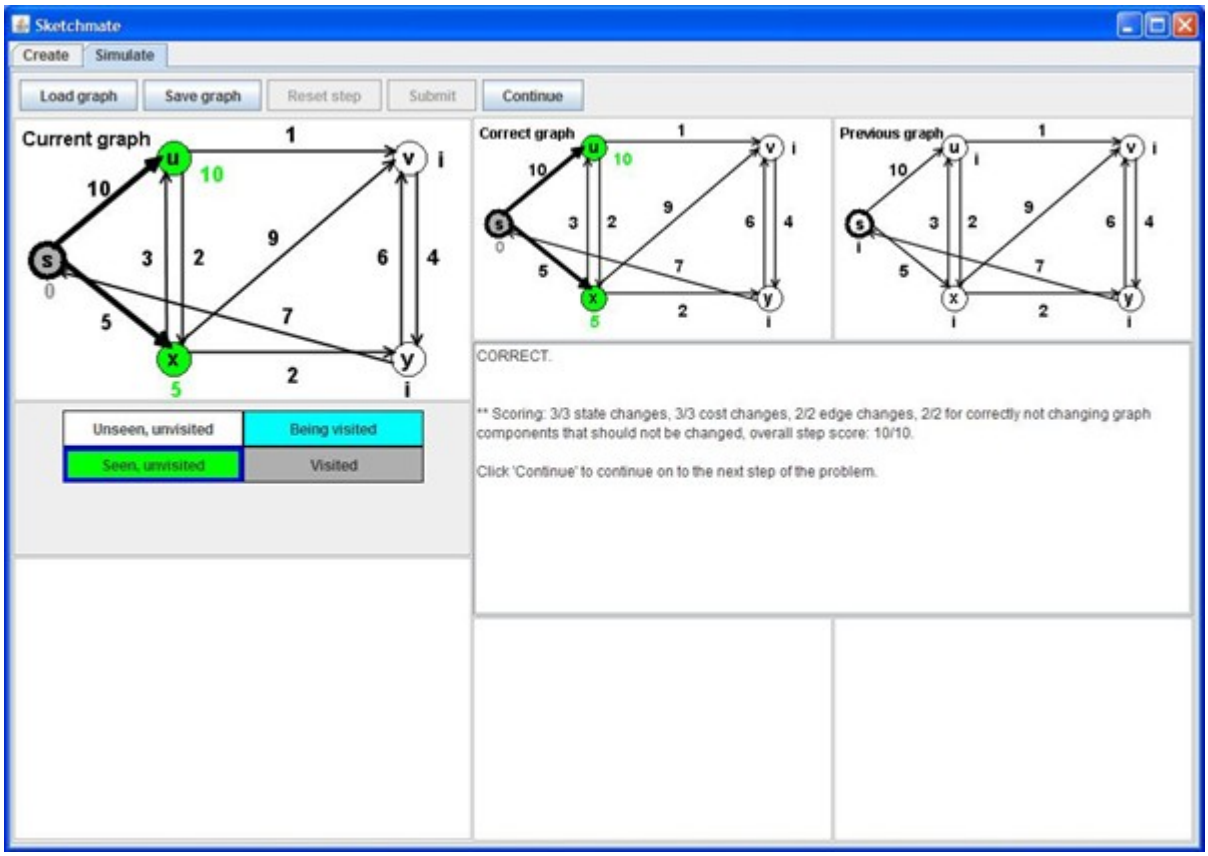


Figure 5.6: Student simulate pane for shortest path

### 5.4.2 Student Simulate Pane for Network Flow

Figure 5.7 shows the student simulate pane for the augmenting path step of network flow. The top set of graphs displays the current, correct, and previous graphs for the flow graph, and the bottom set of graphs displays the same respective graphs for the residual graph. All graph components that are incorrect are highlighted in red in the current residual graph and the corresponding correct components are highlighted in magenta in the correct residual graph for easier comparison. In this case, the user selected an incorrect path a-c-b-d, while the correct path is a-c-d. Note that the flow graph is unmarked for this example since it is not used in finding the augmenting path. The feedback box explains in detail why the

student's answer is incorrect for each incorrect component of the graph. The student is also given scoring for the step, which is explained in Section 5.4.3.

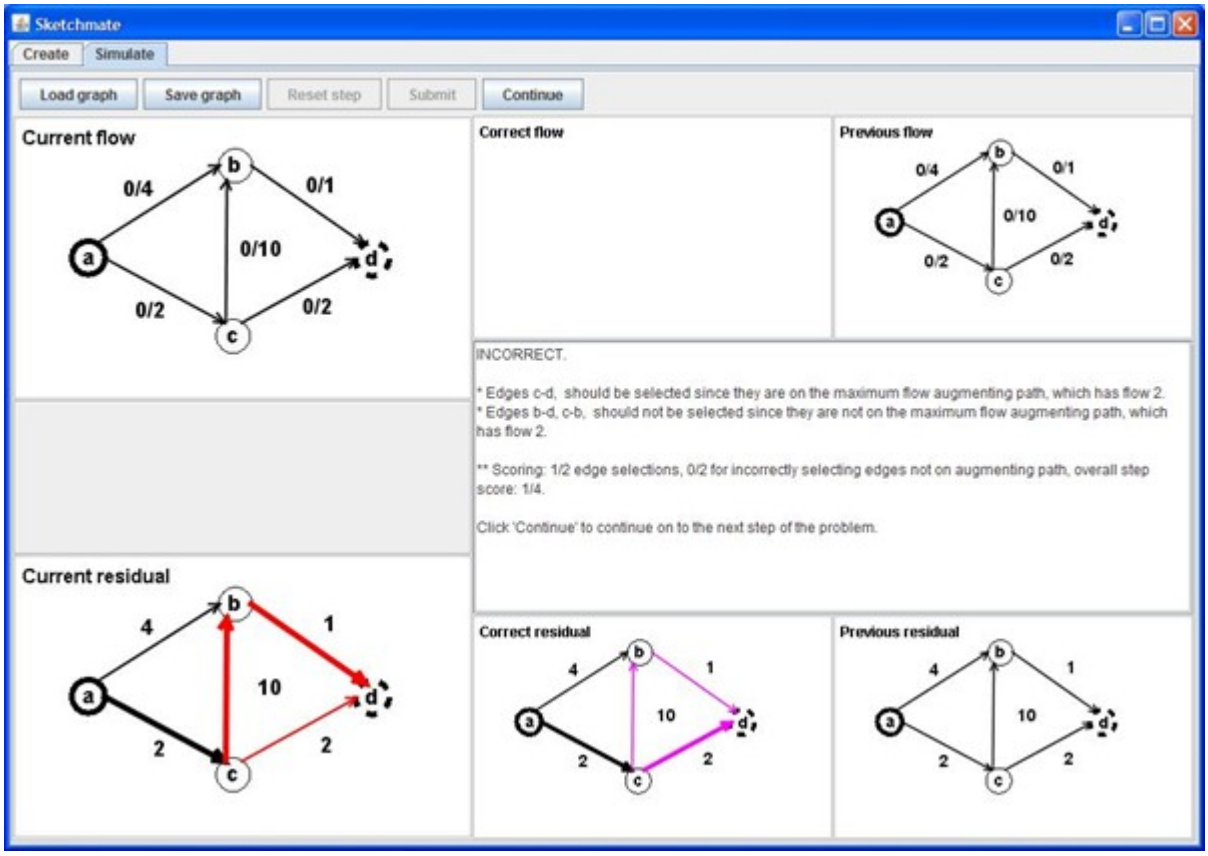


Figure 5.7: Student simulate pane for the augmenting path step of network flow

In Figure 5.8, an example of the student simulate pane for updating the flow and residual graphs is given. As in the previous example, all incorrect flow or residual graph components are highlighted in red, while the corresponding correct graph components are highlighted in magenta in the correct graphs. In the residual graph, the solid edge that is highlighted should have been removed, and the highlighted dashed edge denotes a backedge that should have been added. In other words, the user should have replaced edge c-d with its backedge d-c. The capacity for edge c-a was updated incorrectly, and the capacity for edge b-d should not have been changed, since the edge b-d is not on the augmenting path for this step. The user is also given detailed feedback in this step, for both the flow and residual graphs, as well as scoring for both graphs.

### 5.4.3 General Features of the Student Simulate Pane

If the user makes a mistake in manipulating the graph vertices and edges and has not yet submitted the solution for grading, then clicking the "Reset step" button will restore the

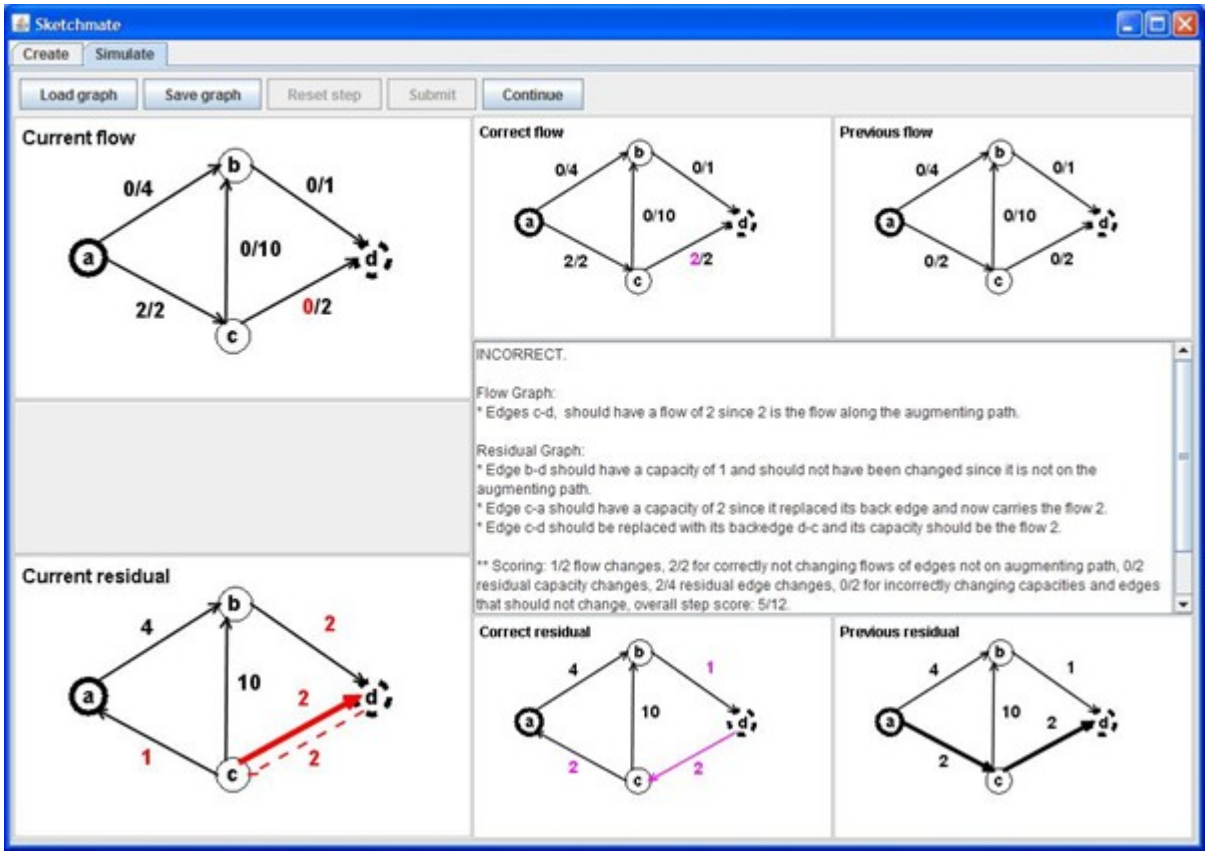


Figure 5.8: Student simulate pane for the flow and residual graph step of network flow

graph to the beginning of that step. This feature compares favorably with students needing to erase or scratch out their work in a paper and pencil based homework.

When the user is finished with the current step, the "Submit" button can be clicked to obtain feedback. The user will be told whether the answer is correct or not, and will also be given an explanation of any graph components that are incorrect. The explanation is generated from the computer automatically solving the problem in the background and comparing the student's solution to the internally computed solution. A full list of possible feedback messages can be found in the Appendix.

The feedback will also give a score for each component of the algorithm, as well as the overall score for that step. For shortest path, the score consists of one point for each correct vertex state change, one point for each correct vertex cost change, one point for each correct edge state change, and an "other" category for deducting points for any changes made to graph components that should not be changed in that step. This "other" score is worth 2 points; if the student changes at least one graph component that should not change in that step, the "other" score is 0 points, and if the student does not change any graph component that should not change in that step, the "other" score is 2 points. For the augmenting path step of network flow, the score consists of one point for each correct edge selection, and an "other" category to deduct 2 points for edges that are incorrectly selected. For the step involving updating the flow and residual graphs, the score consists of one point for each correct flow change, a two point "other" category for the flow graph, one point for each correct residual edge capacity change, one point for each correct residual edge change (i.e., adding or removing edges), and a two point "other" category for the residual graph. The "other" score is awarded in the same manner as it is for shortest path.

Users can click "Continue" to move on to the next step after they are through with reading the feedback. After the problem has been completed, the student will be given a final score, which is the sum of the points earned for each of the steps of the problem.

## 5.5 Sample Sketchmate Sessions

In this section, two sample Sketchmate sessions are presented in a step-by-step manner. First is a session of the student tool working with a shortest path problem, and second is a session of the instructor tool working with a network flow problem.

### 5.5.1 Sample Session of the Student Tool for Shortest Path

The following series of figures (Figures 5.9-5.16) gives a walk-through from beginning to end of how the student tool interface might look while simulating Dijkstra's shortest path algorithm on an example graph. The step where the vertex x is visited is further divided into three substeps, with the updating of each neighboring vertex being one of the substeps, to show in more detail a sample of the student's thought process as the step is being worked through. The steps where vertices y and u are visited contain some incorrect graph components to illustrate the feedback messages given in these cases.

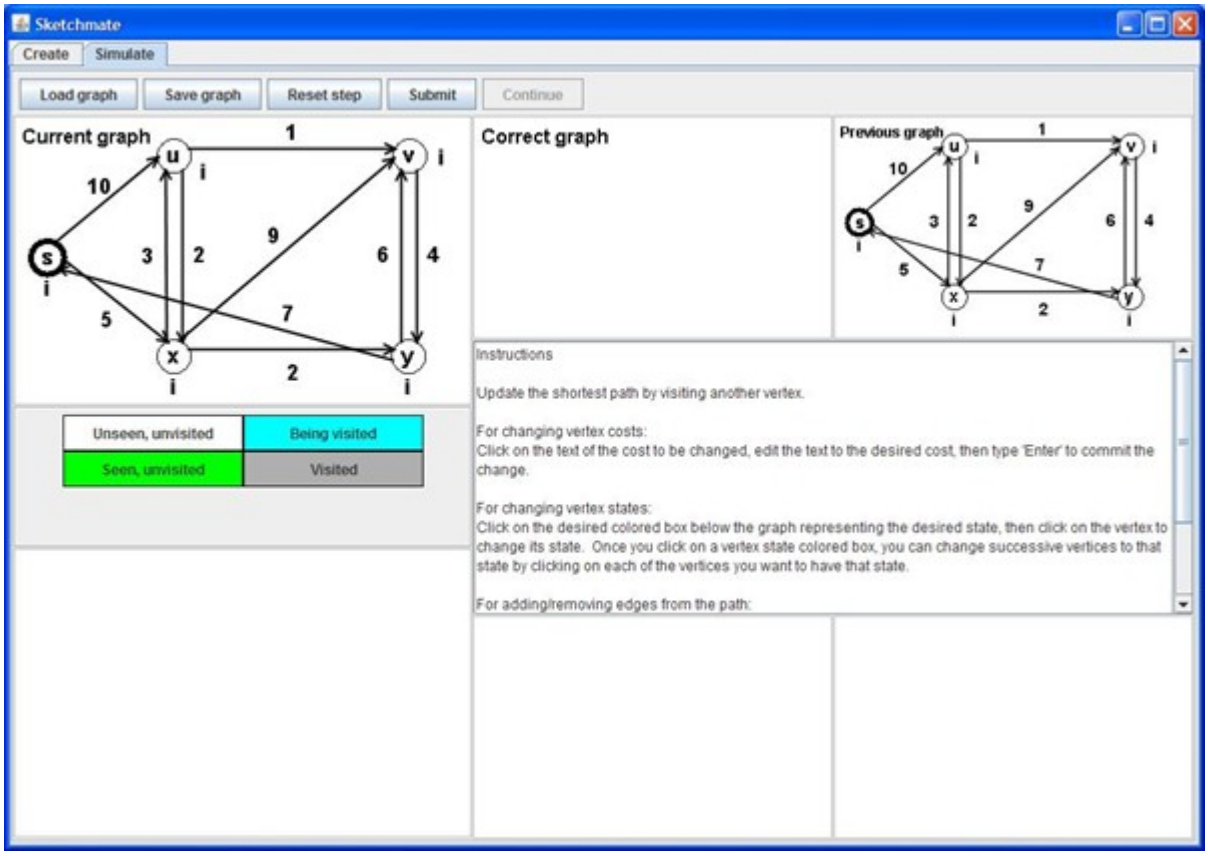


Figure 5.9: The initial screen for the student walk-through example. s is the start vertex.

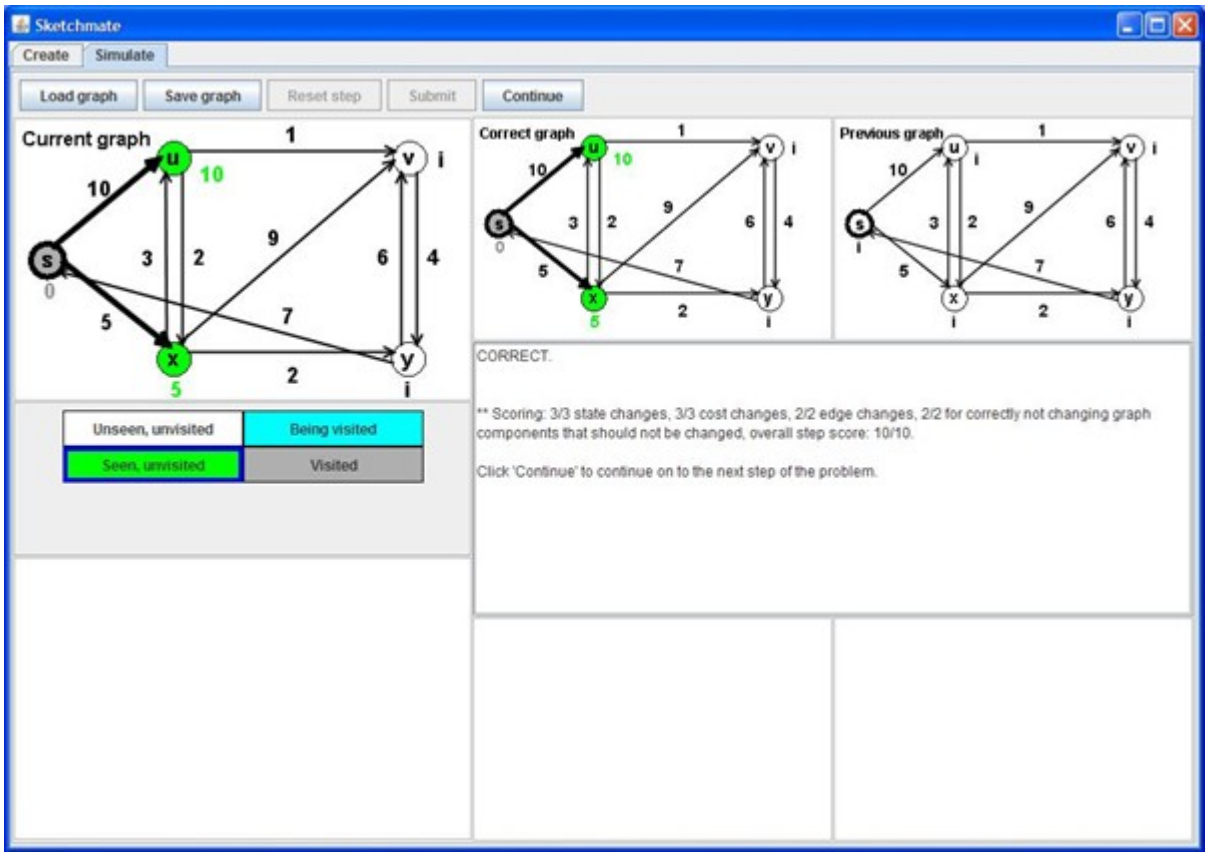


Figure 5.10: Walk-through after visiting s. Vertices u and x have been marked as seen but not visited. The cost of start vertex s is marked as 0. The costs of vertices u and x have been updated to indicate the cost from the start vertex s to each of u and x. Edges s-u and s-x have been shaded to indicate they are added to the shortest path.

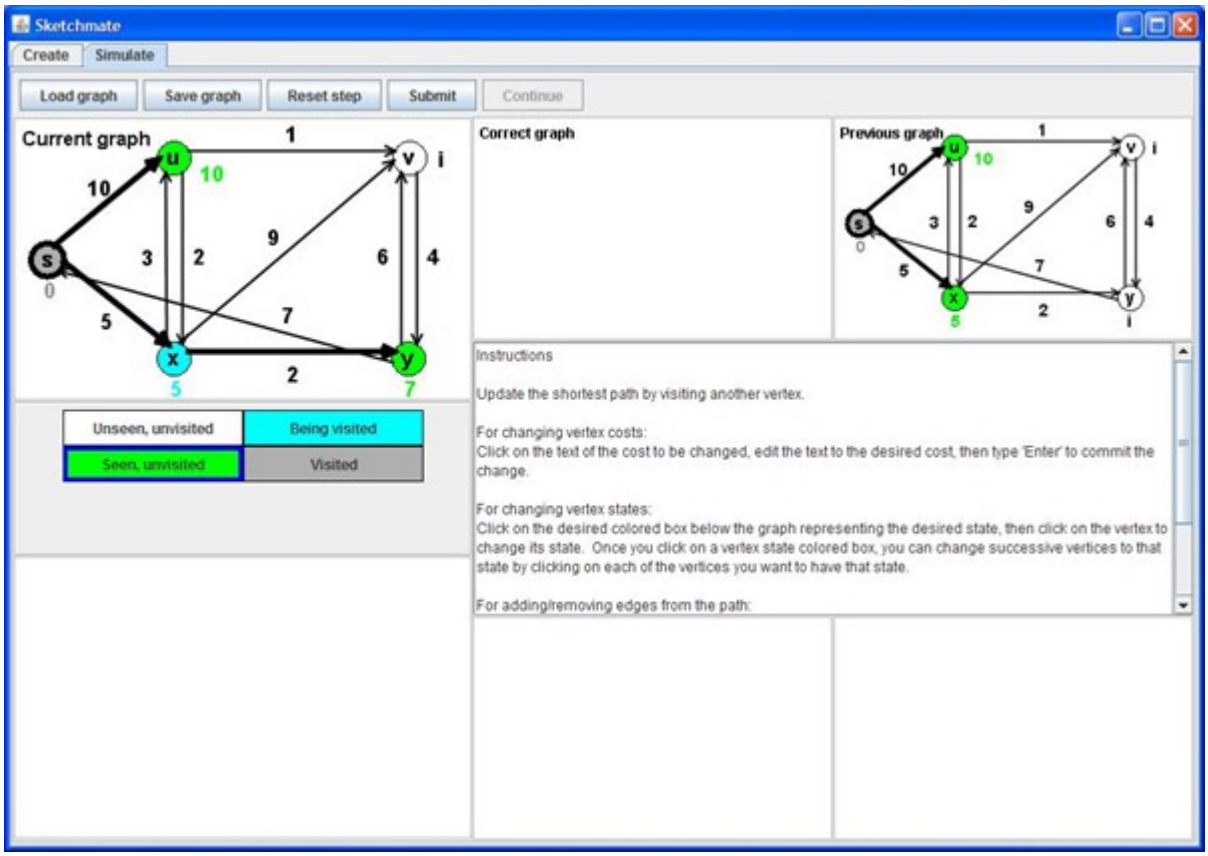


Figure 5.11: Walk-through while visiting x and updating y. Vertex y has been marked as seen but not visited. The cost of vertex y becomes 7 since it is 5 units from s to x and another 2 units from x to y. Edge x-y has been shaded to indicate it is added to the shortest path.

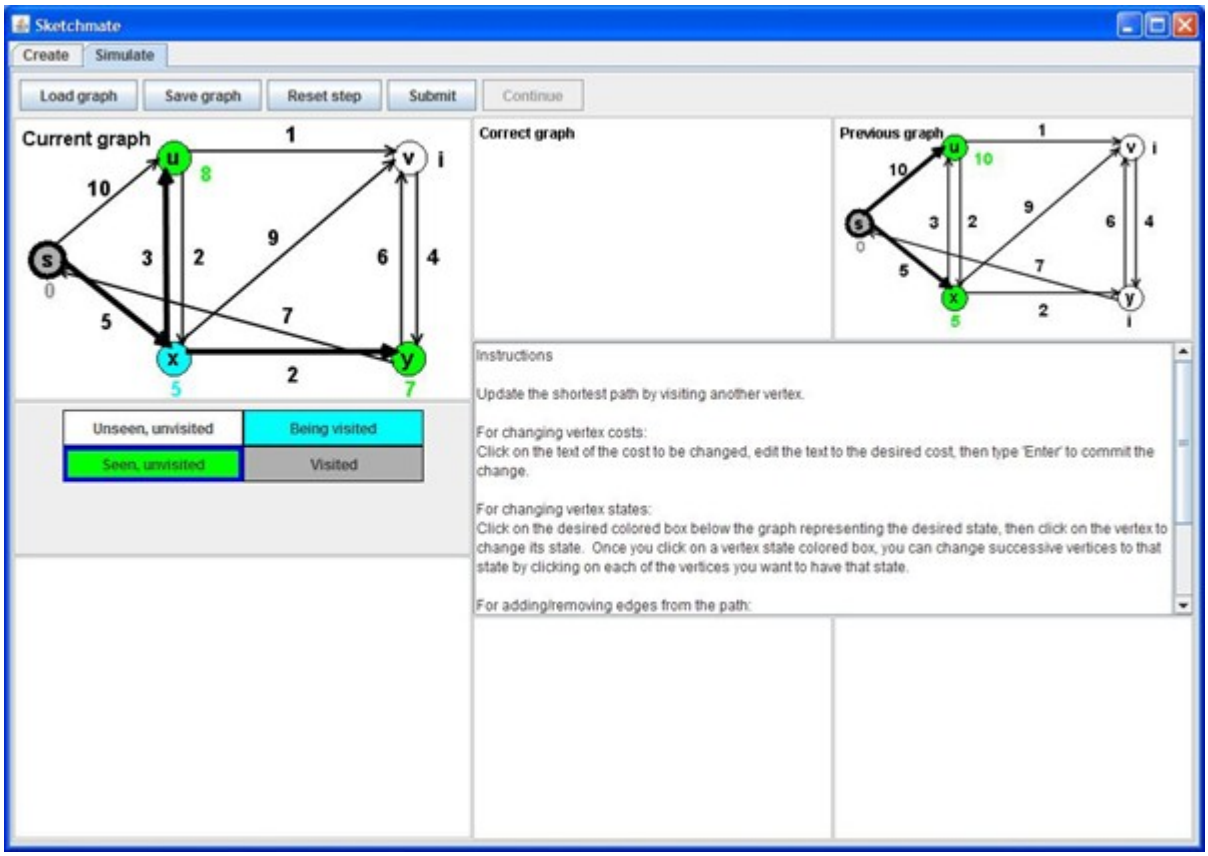


Figure 5.12: Walk-through while visiting x and updating u. The cost of u has been changed to 8 since it is shorter to go from s to u through x than it is to go from s to u directly. To reflect this change, edge x-u has been added to the shortest path and edge s-u has been removed from the shortest path.

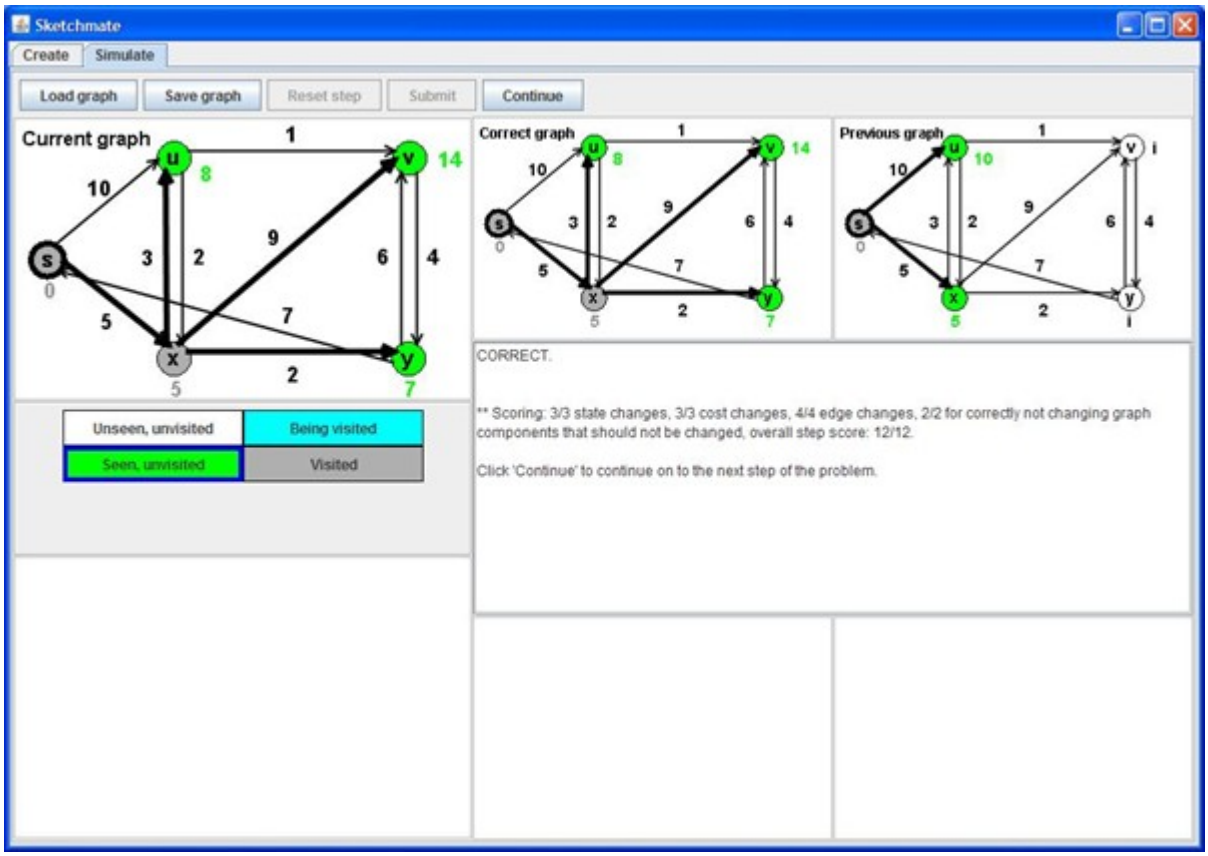


Figure 5.13: Walk-through while visiting x and updating v. Vertex v has been marked as seen but not visited. The cost of vertex v becomes 14 since it is 5 units from s to x and another 9 units from x to v. Edge x-v has been shaded to indicate it is added to the shortest path.

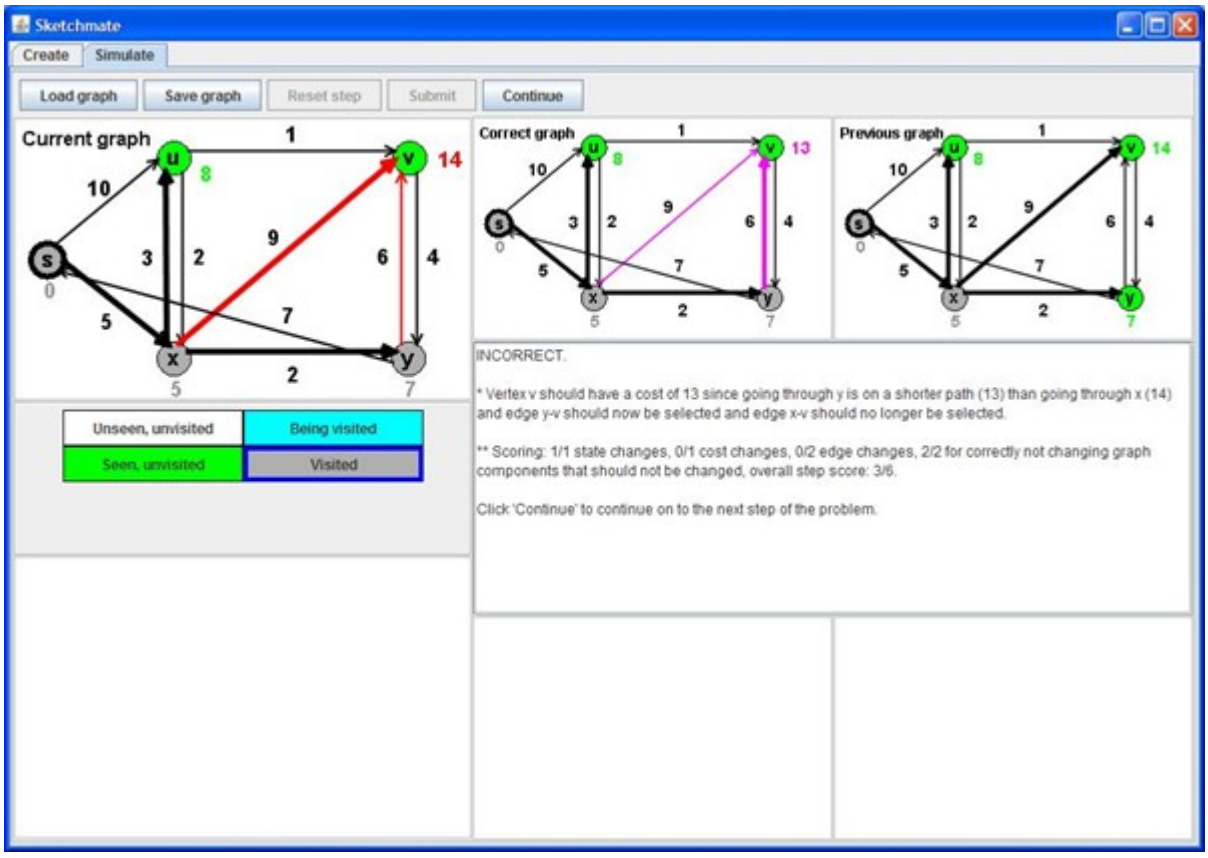


Figure 5.14: Walk-through after visiting y. Vertex y has been marked as visited. The student misunderstands that the cost of vertex v should be reduced to 13 since s-x-y-v is a shorter path than the previous path of s-x-v. In addition to the incorrect cost for vertex v, edge y-v is colored red to indicate it should have been added to the shortest path and edge x-v is colored red to indicate it should have been removed from the shortest path.

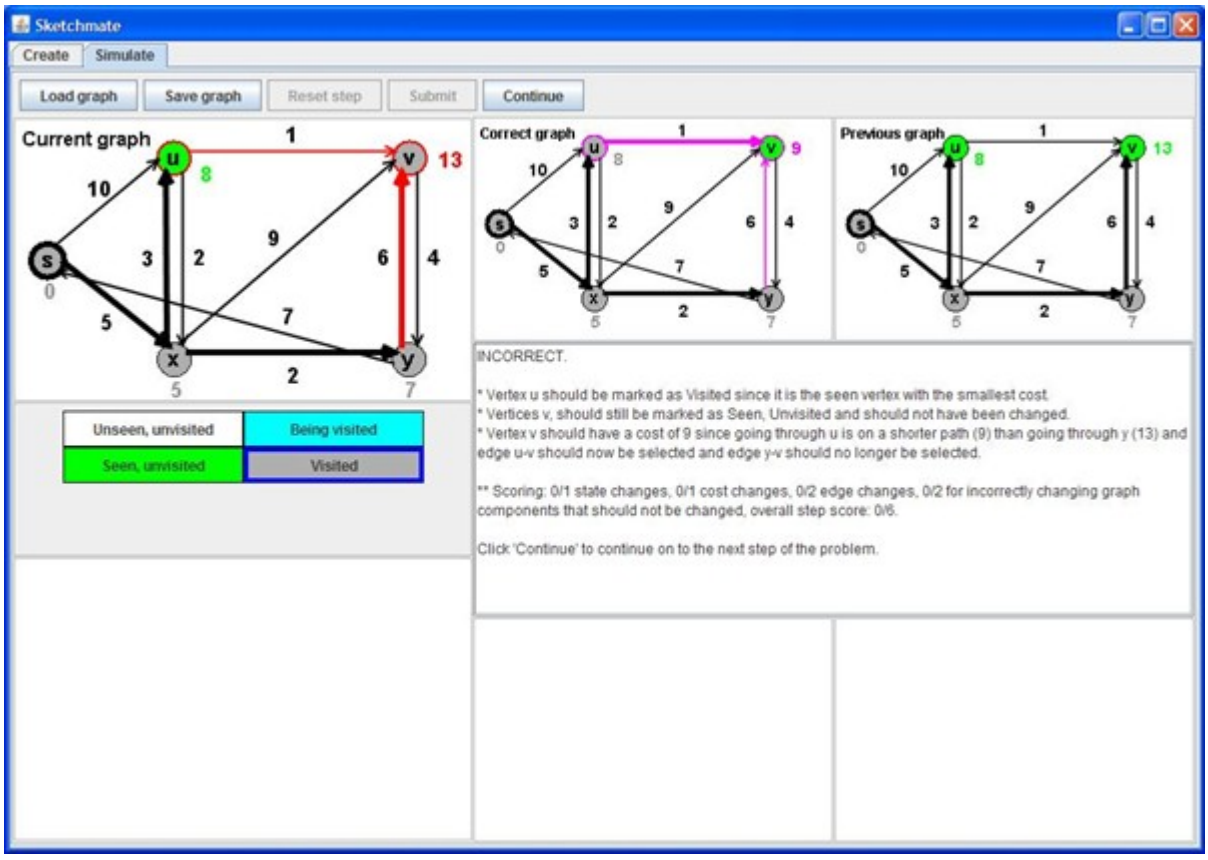


Figure 5.15: Walk-through after visiting u. Note that the student's mistakes in the previous step have been corrected. In this step, the student mistakenly visited vertex v instead of vertex u. Vertex u should have been visited instead since it has a smaller cost than vertex v. Additionally, the cost of vertex v should have been reduced to 9 since s-x-u-v is a shorter path than the previous path s-x-y-v. As indicated by the red edges, edge u-v should have been added to the shortest path and edge y-v should have been removed from the shortest path.

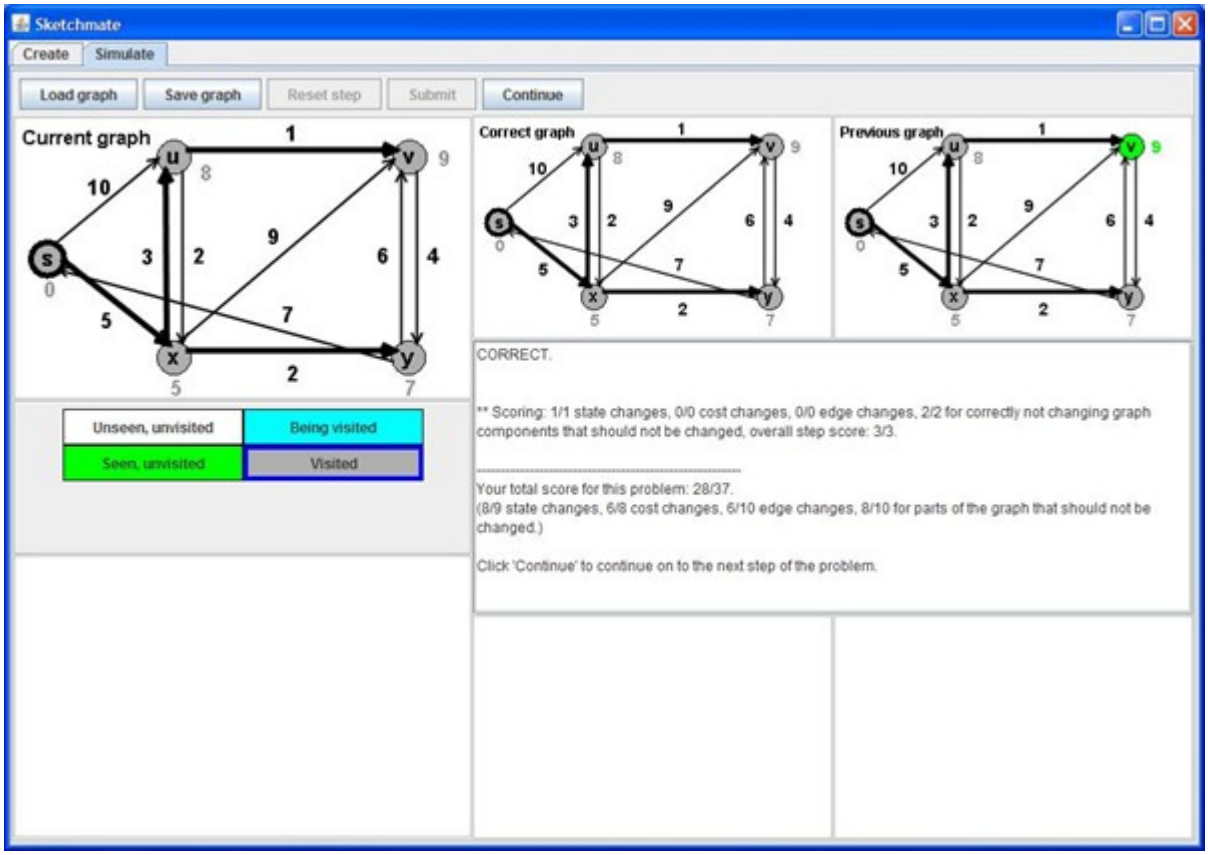


Figure 5.16: Walk-through after visiting v. Note that the student's mistakes in the previous step have been corrected. In this step, vertex v has been marked as visited, and now the problem has been completed.

### 5.5.2 Sample Session of the Instructor Tool for Network Flow

The next series of figures (Figures 5.17-5.27) shows a sample walk-through from beginning to end of a session with the instructor tool working on a network flow problem. There are three sets of finding the augmenting path and updating the flow and residual graphs. The first updating of the flow and residual graphs is divided into substeps to give an idea of what the process of demonstrating that step might be. In the second update of the flow and residual graphs, the instructor only partially updates the graphs, thus the remaining components of the graphs to be updated are highlighted in red upon clicking "Finish step". They are corrected when the user clicks "Next step". Notice that the instructor adds explanations to the notepad at each step and substep to explain what is occurring in the graphs as the solution is being worked through.

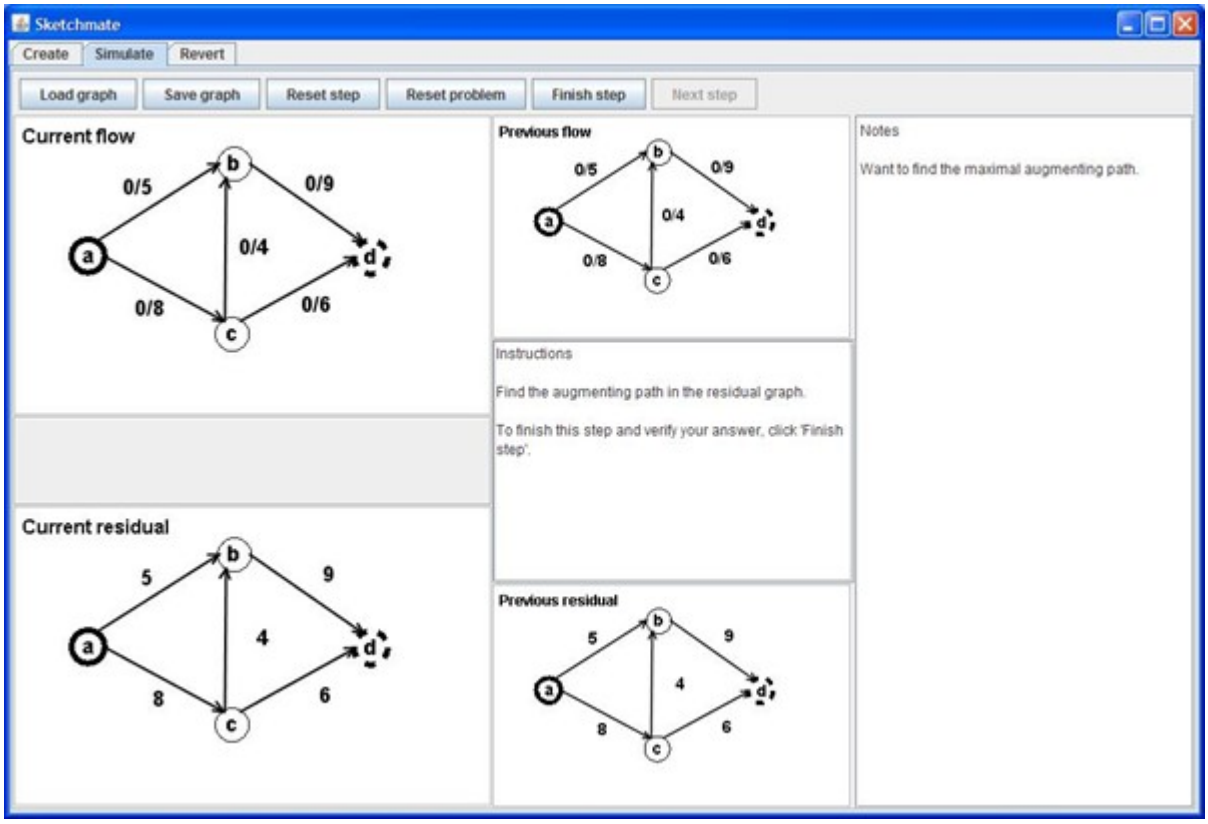


Figure 5.17: Walk-through of a network flow problem in its initial state. The instructor is about to mark the augmenting path in the residual graph.

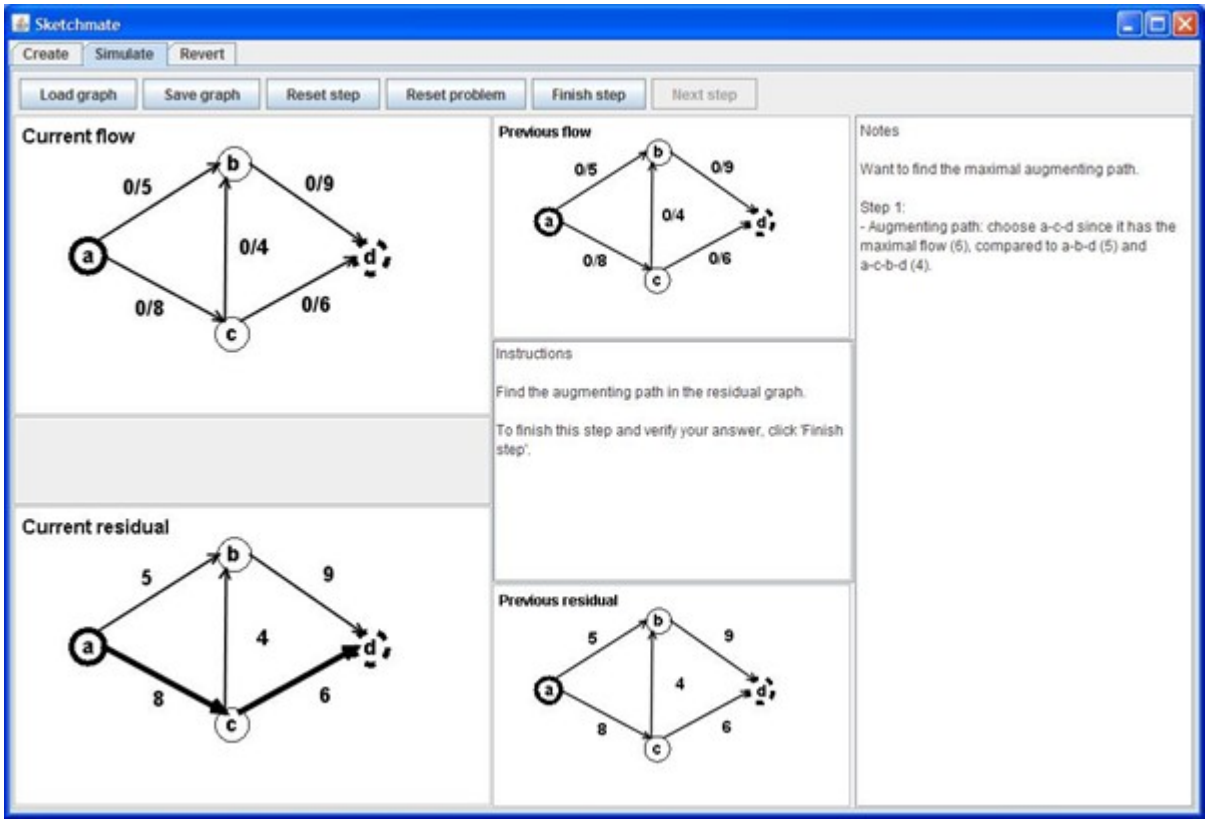


Figure 5.18: Walk-through after the instructor marks the first augmenting path, but before the instructor clicks "Finish step". In the residual graph, edges a-c and c-d are marked to be on the augmenting path since that path has the maximal flow of 6.

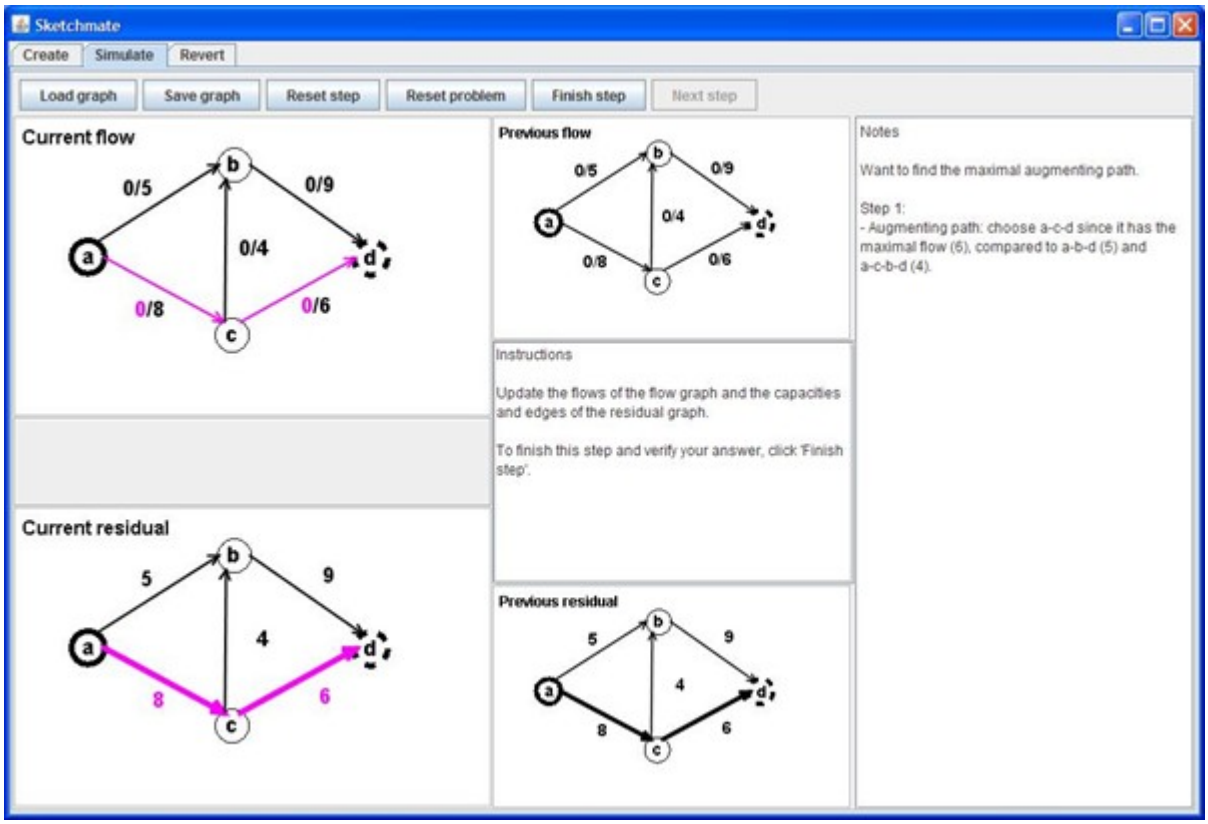


Figure 5.19: Walk-through after finding the first augmenting path and after clicking the "Finish step" and "Next step" buttons. In the residual graph, edges a-c and c-d are marked to be on the augmenting path since that path has the maximal flow of 6. All graph components that need to be updated, as well as the augmenting path itself, are highlighted in magenta in the flow and residual graphs. All of these graph components are along the augmenting path.

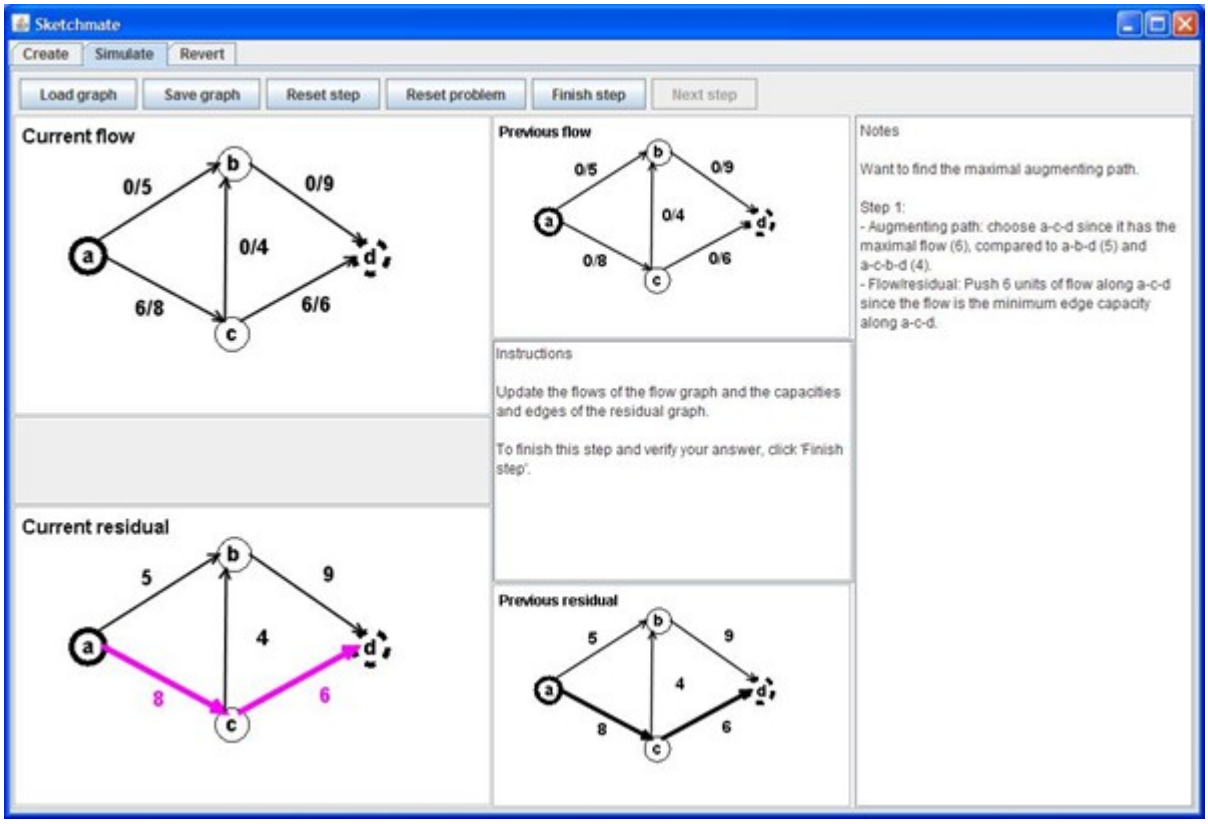


Figure 5.20: Walk-through after updating the flow graph for the first augmenting path. The flow of the augmenting path is 6, so the flow for edges a-c and c-d in the flow graph have been updated to 6.

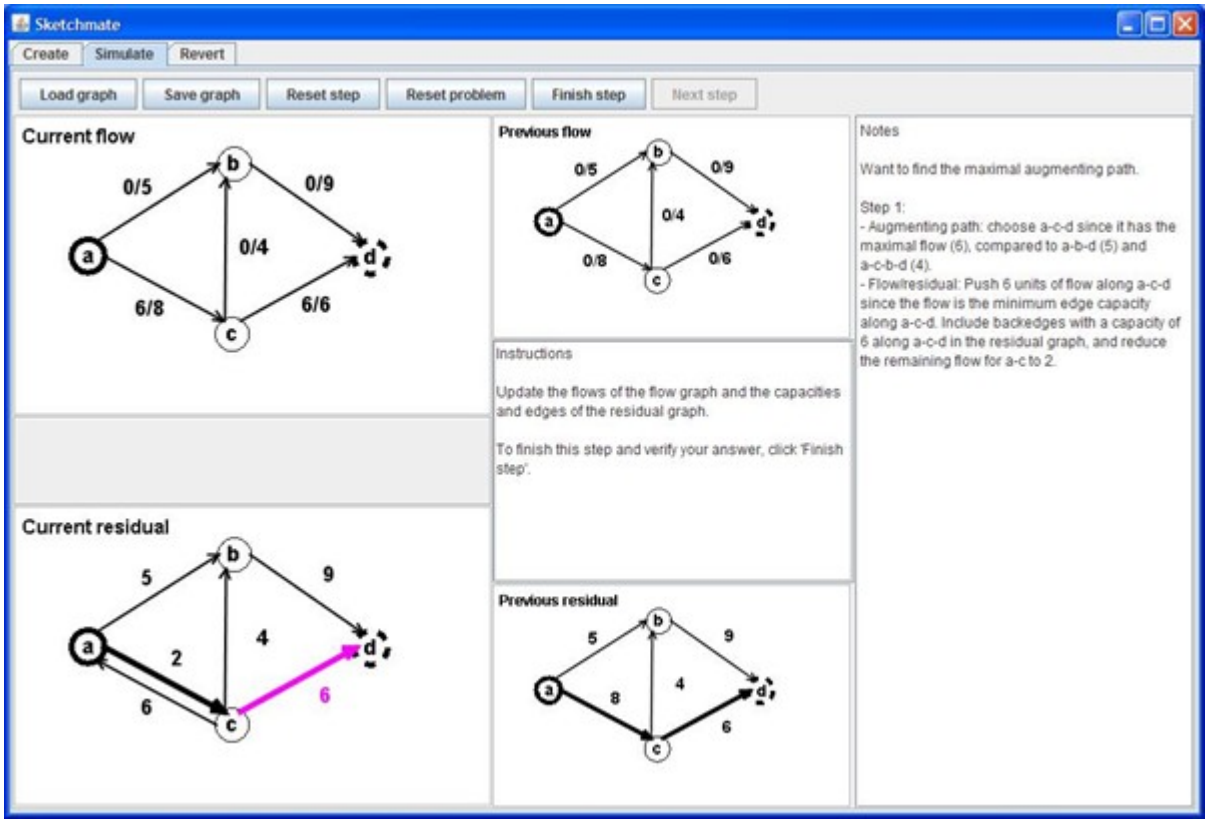


Figure 5.21: Walk-through during updating the residual graph for the first augmenting path. Backedge c-a has been added to the residual graph and assigned capacity 6 since 6 is the flow along the augmenting path. The capacity of a-c is reduced by the flow of 6 units to 2.

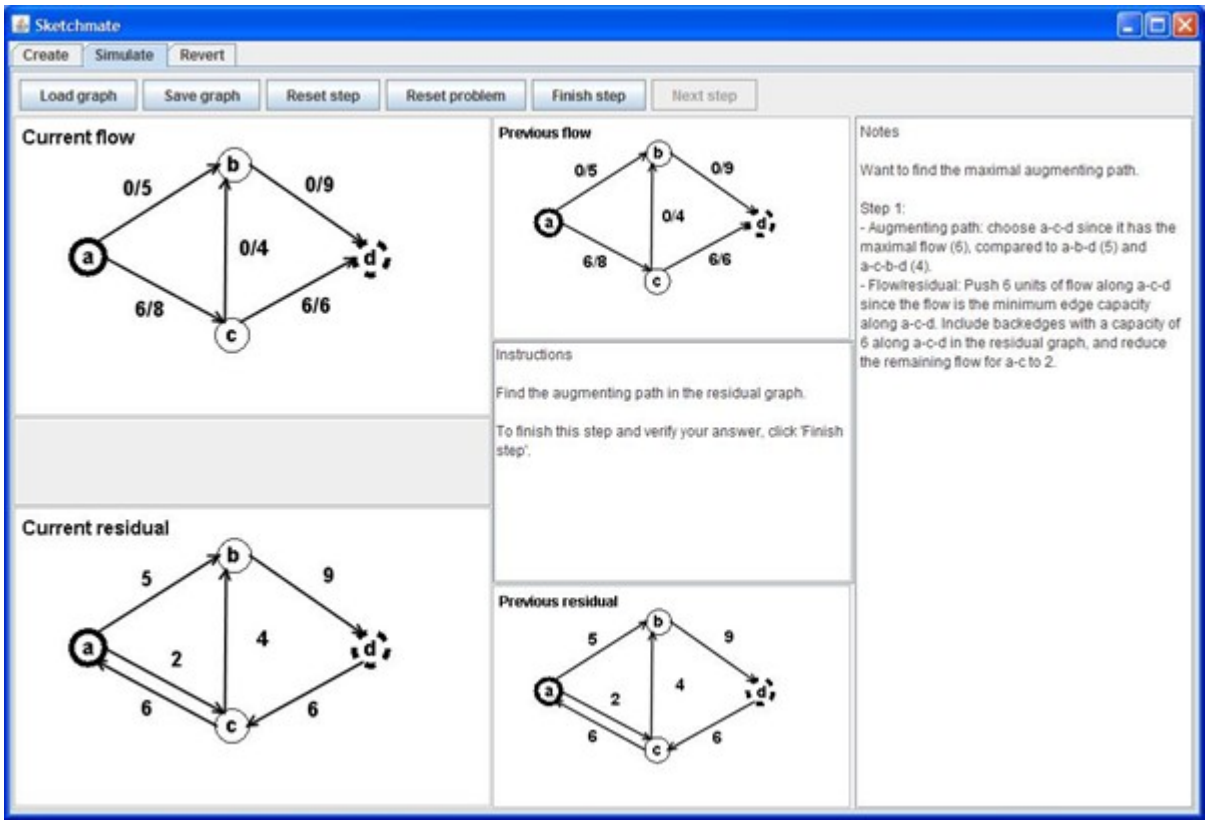


Figure 5.22: Walk-through after updating the residual graph for the first augmenting path. Edge c-d has been flipped to become edge d-c since all of the flow has been used up and is now a back flow.

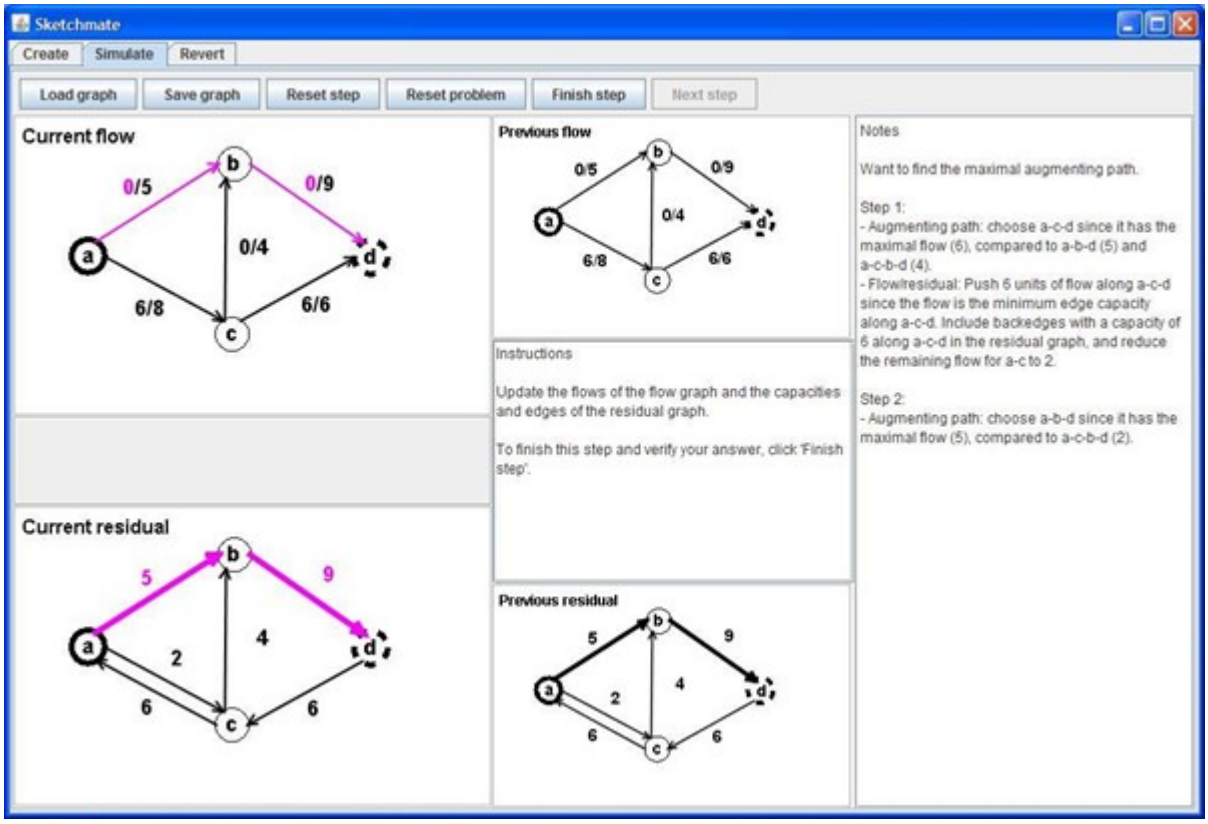


Figure 5.23: Walk-through after finding the second augmenting path. The instructor has marked edges a-b and b-d to be on the augmenting path. The instructor is about to update the flow and residual graphs.

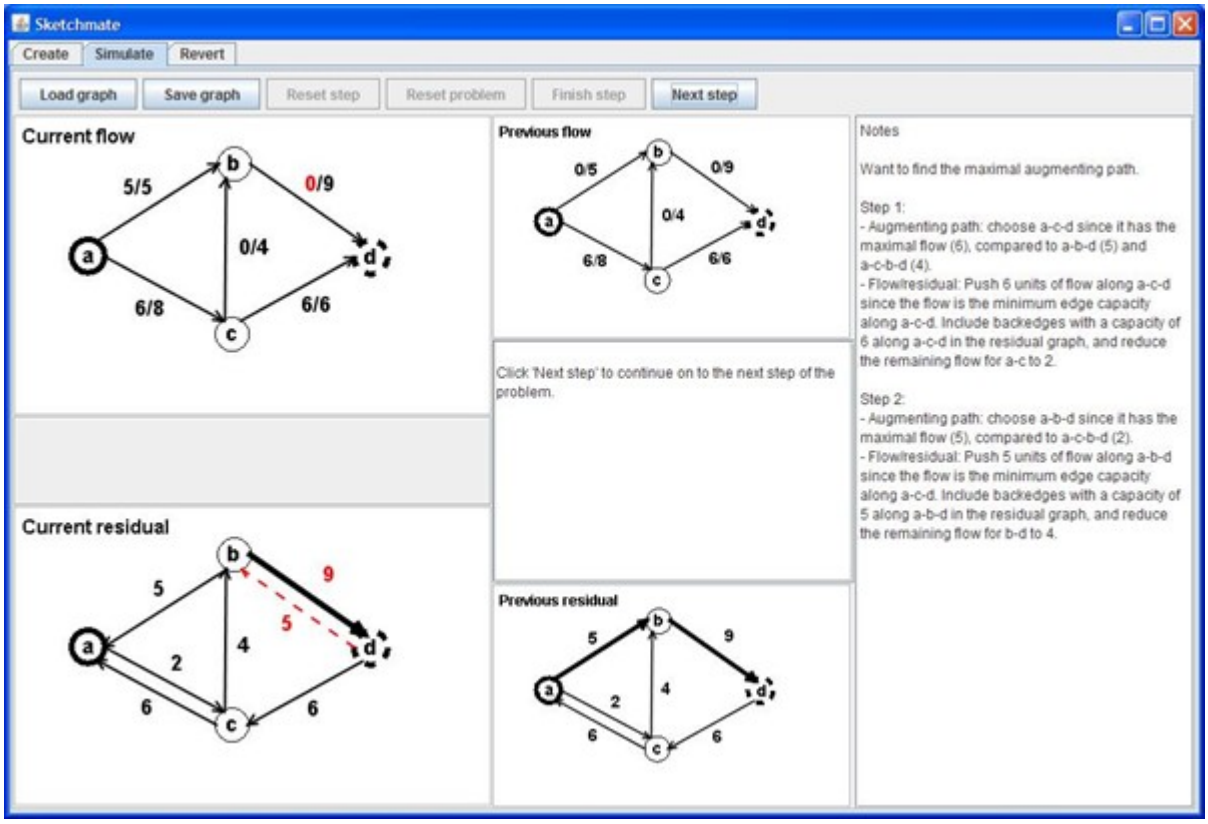


Figure 5.24: Walk-through after only partially updating the flow and residual graphs for the second augmenting path. The instructor only updated the flow and residual graphs for the edge a-b before clicking "Finish step". The graph components along edge b-d that should have been updated are marked in red as incorrect. The dashed edge d-b in the residual graph denotes a backedge that should have been added.

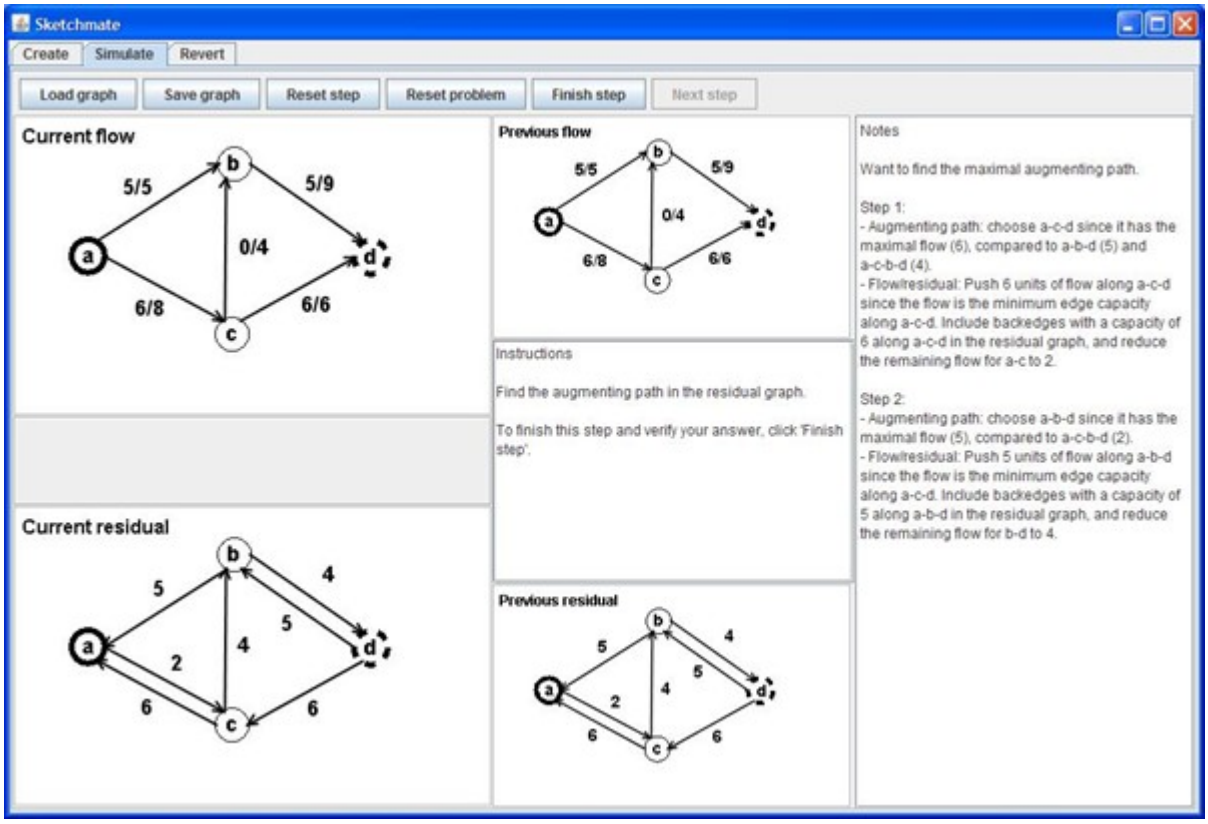


Figure 5.25: Walk-through after the instructor clicked the "Next step" button and the updates to the flow and residual graphs have been corrected by the computer.

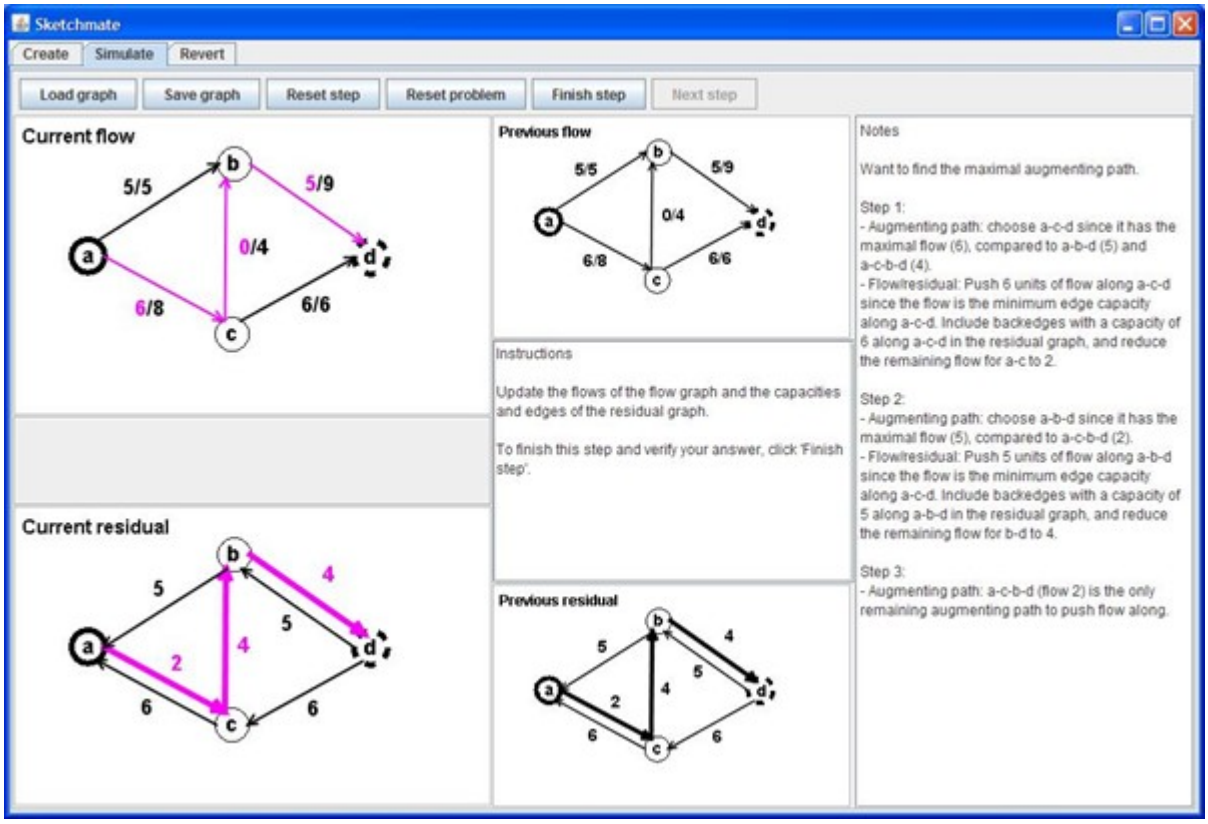


Figure 5.26: Walk-through after finding the third augmenting path. The instructor has selected edges a-c, c-b, and b-d to be on the augmenting path. All graph components along this path that need to be updated are highlighted.

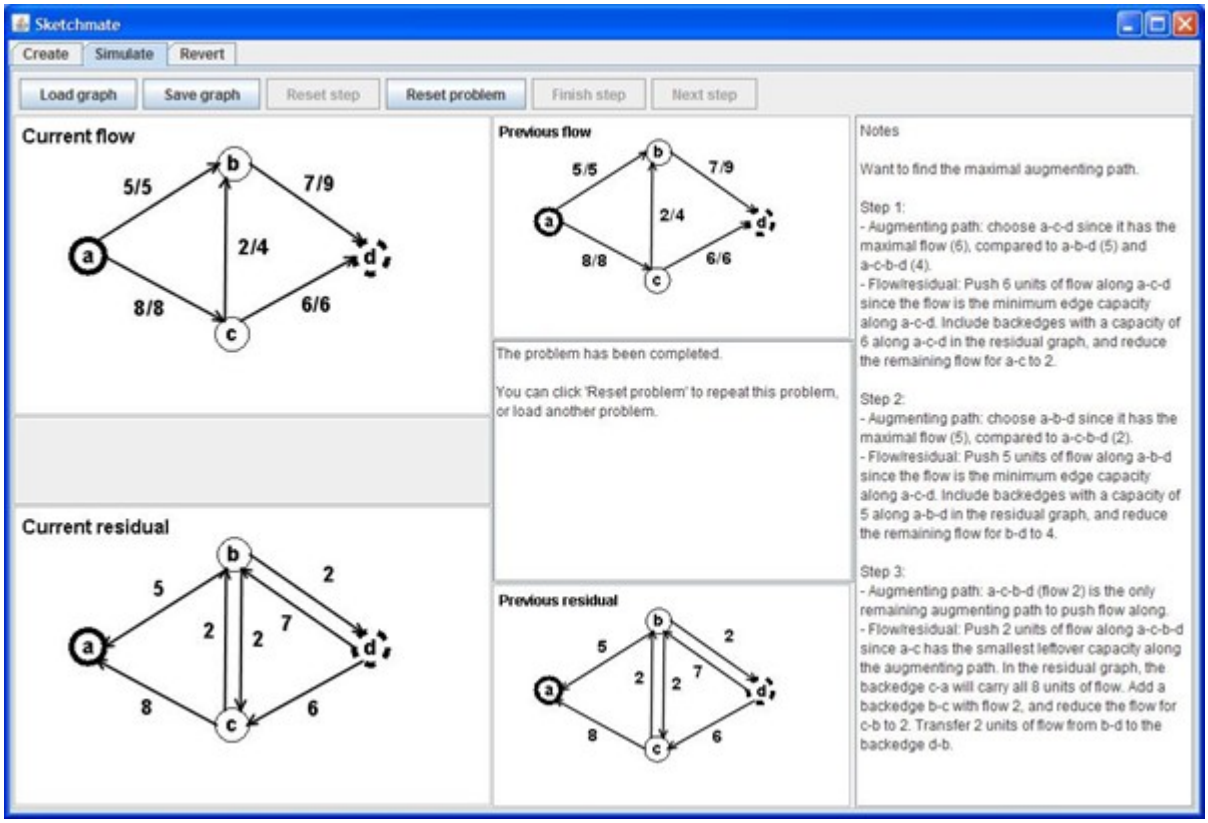


Figure 5.27: Walk-through after updating the flow and residual graphs for the third augmenting path. The instructor has updated the flow and residual graph as explained in the notepad text under Step 3. The problem is complete at this point since there cannot be any more outflow from the source vertex a.

# Chapter 6

# Analysis and Evaluation of the Student Tool Experiment

The focus of the development of the student tool was on creating an interactive learning environment and providing detailed immediate feedback on the student's progress as they work through exercise problems. The only existing tool to provide such feedback to students as applied to graph problems is PILOT (Baker, 2000). While the experimenters of PILOT formally evaluated learning outcomes, they did not study learning rate. The goal of our experiment with the student tool was to measure whether student learning rate and learning outcome improved through using Sketchmate as compared to using a traditional paper and pencil technique. The hope was that the feedback Sketchmate provides students would help students learn how to complete graph problems more effectively than with a paper and pencil method which provides no feedback.

The experiment was a between-subjects, pre-test and post-test design. Students were given practice exercises on both shortest path and network flow to work through during the class period using either Sketchmate or paper and pencil. Students were allotted a fixed amount of time to complete the pre-tests, practice exercises, and post-tests. The tests and the exercises were both designed to be open-ended, that is, they contained more problems than the students could complete in the allotted time. This design allowed us to measure learning rate, given by the amount learned within a fixed period of time. The experiment was designed to test several hypotheses.

The first hypothesis was that there would be no difference in learning for shortest path, as measured by the difference between pre- and post-tests, when using either method. The shortest path algorithm is a relatively simple algorithm, and we presumed that the method of learning would be inmaterial in terms of learning outcomes. Results of the PILOT study (Baker, 2000) support this presumption. The hope was that this hypothesis would be rejected so that the conclusion would be that using Sketchmate for shortest path exercises would lead to better learning.

The second hypothesis was that the students would learn better for network flow problems using Sketchmate than paper and pencil. The reasoning behind this hypothesis is that we presumed that Sketchmate would have more of an impact on learning for a more difficult

algorithm such as network flow. In addition, since the students were given more problems than they could complete in the allotted time, we presumed that the Sketchmate students would complete more problems and thus earn more points than students who used paper and pencil.

We also tested for significant differences in student performance on the practice exercises. The hypothesis was that Sketchmate students would perform the same as paper and pencil students for shortest path problems, and that Sketchmate students would perform better than paper and pencil students on network flow problems, for the same reasons as explained above.

Finally, we collected data of how students perceived ease of use and effectiveness with either of the methods. The hope was that students would enjoy using the Sketchmate tool at least as much, and possibly more than, using paper and pencil.

## 6.1 Subjects, Setting, and Materials

Approximately 40 students from the Fall 2011 undergraduate Algorithms course in the Electrical Engineering and Computer Science department at the University of Tennessee-Knoxville participated in the study. The experiment was included as a lab activity for the course. The study was conducted in a computer science lab on campus. The students were provided with a means of accessing Sketchmate through the internet. The experimenters developed nine different shortest path problems and six different network flow problems of varying difficulty. Both paper and Sketchmate versions of the problems were produced. The students needed to show all of the steps in solving the problems.

## 6.2 Design

The experimenters used a between-subjects, pre-test and post-test model for testing differences in learning rate and learning outcome on both the practice exercises and on the difference between pre- and post-test scores for students using Sketchmate (experimental condition) and students using paper and pencil (control condition). For the pre- and post-tests, the dependent variables were the difference between pre- and post-test scores in terms of number of points (a measure of learning rate) and percent accuracies (a measure of learning outcome). For the practice exercises, the dependent variables were the number of points (a measure of learning rate), number of correct steps (a measure of learning rate), and percent accuracies for each (measures of learning outcome). For the pre- and post-tests, significant differences across dependent variables were tested for using repeated measures analysis of variance (ANOVA), and for the practice exercises, significant differences across dependent variables were tested for using one-way analysis of variance (ANOVA). For both the tests and the exercises, differences are considered significant at the $\rho < 0.05$ level for the learning rate measures, and the $\rho < 0.025$ level for the learning outcome measures. These differences in significance levels were necessary because the learning rate and learning outcome measures

were largely based on the same data set. In order to obtain valid tests of significance for the learning outcome measures, the confidence interval needed to be halved.

## 6.3 Procedures

The Algorithms course consisted of twice weekly lectures and a weekly three hour lab. There were two lab sections and the experiment took place in each lab section in the computer lab. Students were first given an 8-minute written pre-test on shortest path graph problems and a 12-minute written pre-test on network flow graph problems. These two algorithms were already covered in lecture prior to the experiment. Hence, the pre-tests should have measured the students' ability to work these problems after having seen a lecture on these problems, but before they had done any homework with these problems.

After the pre-tests, the students were randomly assigned to two different groups. One group completed shortest path problems with paper and pencil, and the other group completed the same exercises with Sketchmate. The set of problems was designed so that there were more problems than the students could complete in the allotted time. Students were given 20 minutes to complete as many problems as possible. After this part of the experiment, the students using paper and pencil switched to using Sketchmate, and the students using Sketchmate switched to using paper and pencil, and both groups completed the same exercises on network flow. For this activity students were allotted 25 minutes. The students were given a 5-10 minute demonstration of how to use the computer tool for each problem type. For the students who worked with Sketchmate, their scores were recorded to a log file each time they completed a problem. After each exercise portion, print-outs of these log files were collected along with the other paper materials.

After the exercise sessions, all students were given about 10 minutes to review the solutions to the exercises. Then all students were given a written post-test for both shortest path and network flow problems. The structure of these post-tests was identical to that of the respective pre-tests. The post-tests were given in the same class period as the exercises to avoid the possibility of students studying for them and therefore distorting their scores. Sections 4.2 and 4.3 give an example of a shortest path problem and a network flow problem similar to the problems used in this experiment.

The participants were also given a written survey to obtain feedback on how well the students liked each of the two methods, and how well the students felt those methods helped them learn the material. There were several Likert scale questions with a rating between 1 (unfavorable) to 7 (favorable), as well as free response questions.

Only students who attempted the pre-tests and post-tests and who followed directions properly were used in the study. The primary experimenter scored all attempted pre- and post-test problems and in-class exercises for accuracy. The grading scheme was identical to the automatic grading scheme used by Sketchmate, but it is quickly reviewed here. For each step of shortest path, there were four components to the score: one point for each correct vertex state change, one point for each correct vertex cost change, one point for each edge properly added to or subtracted from the shortest path, and an "other" score for deducting

for any changes in areas of the graph that should not be changed in that step. This "other" score was either 2 points for all unchanged components being correctly left unchanged, or 0 points for one or more unchanged components being incorrectly changed. For each step of network flow, there were components of the score for the augmenting path, flow graph, and residual graph. For the augmenting path, the score consisted of one point for each correct edge selection and an "other" score. For the flow graph, the score consisted of one point for each correct flow change and an "other" score. For the residual graph, the score consisted of one point for each correct edge capacity change, one point for each correct edge addition/deletion, and an "other" score.

A second experimenter independently scored all of the attempted problems on both the shortest path and network flow post-tests to determine interscorer agreement. For shortest path the experimenters achieved 98.79% agreement, and for network flow the experimenters achieved 99.71% agreement, which suggests that the post-test problems were scored consistently. Most of the differences resulted from illegible handwriting of the students, or in one case, a student being unclear about where he or she began the problem.

## 6.4 Results and Discussion

In this section we present our results for the shortest path tests and exercises, followed by our results for the network flow tests and exercises. We then discuss our results for the survey questions.

### 6.4.1 Shortest Path

The results of the pre- and post-tests of the shortest path portion of the experiment are shown in Table 6.1. The mean, standard deviation, and median for the pre-test, post-test, and difference between pre- and post-test are given for both the Sketchmate and paper and pencil groups. The top three rows give the statistics for the total number of points earned on the pre- and post-test. These numbers do not take into account whether the student finished the attempted problems. The bottom three rows give the statistics for the percent accuracy of completed problems. These numbers only consider the scores for problems that were completed by the student.

While there was a statistically significant difference between pre- and post-test scores for all students ($\rho = 0.000$), there was not a statistically significant difference in learning rate or learning outcome between the Sketchmate and paper and pencil groups. For the total points (learning rate) measure, the paper and pencil group actually improved by 5 points more on average than the computer group, though statistically, the two groups are essentially equivalent ($\rho = 0.974$). Interestingly, the median measure indicates that the computer group actually scored 2 points more than the paper and pencil group. This suggests that it is inconclusive which group achieved a higher learning rate. Note that the statistical significance values are for the mean rather than the median (this is true for all results reported in this dissertation). For the percent accuracy measure, the Sketchmate students

had a mean 8 percentage point gain and a median 11 percentage point gain over the paper and pencil group. While these results are statistically equivalent with $\rho = 0.896$ for the mean, they do show that Sketchmate students scored roughly one letter grade higher than paper and pencil students.

Table 6.1: Student tool points/accuracy for shortest path pre-test and post-test (SD = standard deviation). 197 possible points for both pre- and post-test.

| | **Sketchmate** | | | **Paper/Pencil** | | |
|---|---|---|---|---|---|---|
| **Measure** | **Mean** | **SD** | **Median** | **Mean** | **SD** | **Median** |
| **Pre points** | 48 | 22 | 42 | 46 | 23 | 42 |
| **Post points** | 92 | 31 | 99 | 94 | 40 | 92 |
| **Diff. points** | 43 | 23 | 46 | 48 | 36 | 44 |
| **Pre accuracy** | 63.5% | 21.9% | 58.3% | 66.4% | 19.1% | 72.4% |
| **Post accuracy** | 83.6% | 17.9% | 91.0% | 79.2% | 16.3% | 85.1% |
| **Diff. accuracy** | 20.1% | 18.0% | 18.6% | 12.7% | 23.3% | 7.2% |

Figure 6.1 shows a plot of the improvement from pre- to post-test for the total number of points for both groups. The lines are nearly on top of one another, so it is clear that the performance of the two groups is essentially equivalent. Figure 6.2 illustrates a plot of the difference between pre- and post-test for the percent accuracy for both groups. While this plot clearly shows that the Sketchmate group had a higher improvement in scores, it was still not enough to rise to the level of statistical significance.

Tables 6.2 and 6.3 give the results for the shortest path exercise problems. The mean, standard deviation, and median for the total number of points and the percent accuracy of points are given in Table 6.2. The first two rows give the results for the case where the grading was cut off once a student received less than 50 percent of the points in a step of the problem. This cutoff applied to both the Sketchmate and the paper and pencil groups. The last two rows of the table show the results for the case in which there was no cutoff in the grading and all of the attempted steps of the problems were graded, regardless of the student's score for any given step. This grading without cutoff scheme was only applied to Sketchmate students; the paper and pencil students were always graded with a cutoff. The reason for this grading decision is that once the paper and pencil students went off track and scored lower than 50 percent on a step, the rest of their steps for that problem would be based on a largely incorrect step, thus making it difficult to determine which portions of their answer were actually correct or not, or if they were correct just by coincidence. However, when students completed homework problems using Sketchmate, then if the students answered a step incorrectly, Sketchmate corrected their answer before moving on to the next step. Thus, each student answer for a step was based on beginning that step with the correct graph.

Each of the four measures in the table resulted in enough difference between Sketchmate and paper and pencil that the results were statistically significant, as shown in the last column of the table. With a grading cutoff applied, Sketchmate students scored 85% on

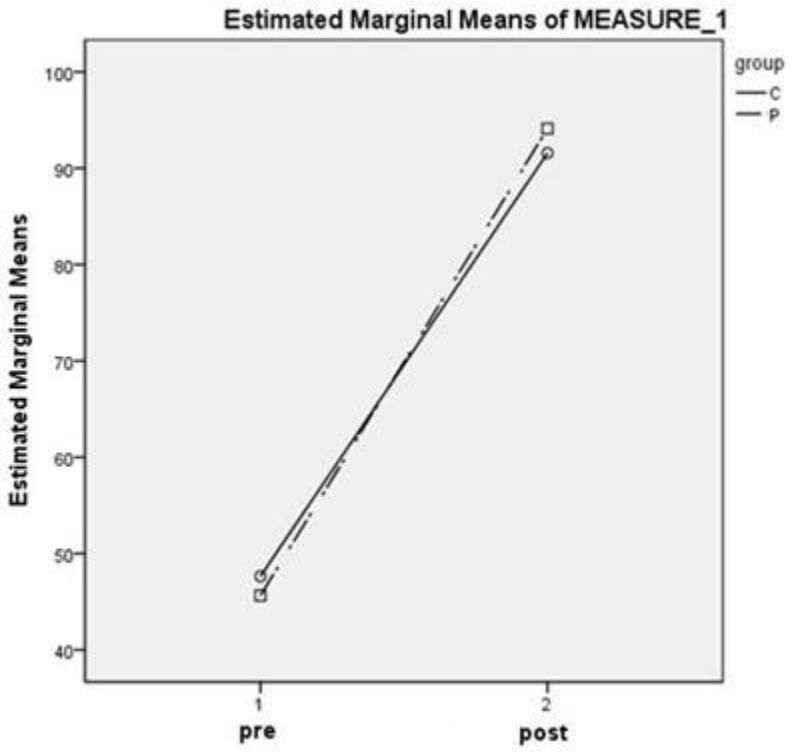


Figure 6.1: Improvement of total number of points for shortest path pre-test and post-test. C = Computer (solid line), P = Paper/Pencil (dashed line)

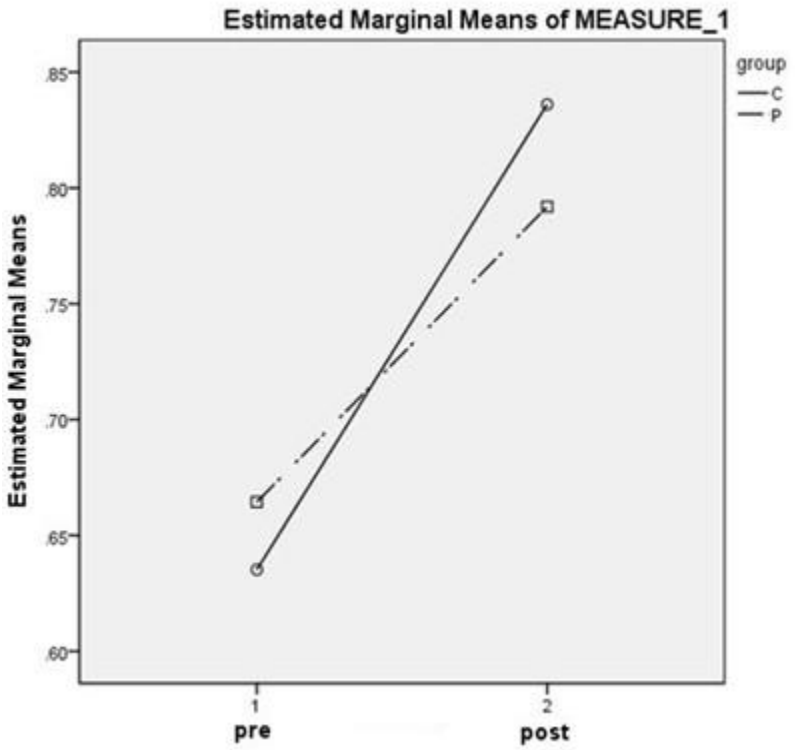


Figure 6.2: Improvement of percent accuracy for shortest path pre-test and post-test. C = Computer (solid line), P = Paper/Pencil (dashed line)

average, and without a grading cutoff, they were able to score 89% on average, as opposed to the paper and pencil group only scoring 70% on average. Again, for statistical significance, the necessary $\rho$ value for the total points measure was 0.05 and the necessary $\rho$ value for the percent accuracy measure was 0.025. It should be noted that because Sketchmate only recorded the students' scores for a problem after they completed that problem, only the problems that were completed are included in the data for the exercise problems in order to treat the two groups equally. Also note that the results for the paper and pencil group are the same for the cutoff and no-cutoff cases, since the grading without cutoff scheme only applied to Sketchmate students. Another encouraging result is that Sketchmate students scored an average of roughly one and half letter grades higher than paper and pencil students in the case of cutoff, and roughly two letter grades higher in the case of no-cutoff than paper and pencil students.

Table 6.2: Student tool points/accuracy for shortest path exercise problems (SD = standard deviation, C = cutoff, NC = no-cutoff). 312 possible points.

| | **Sketchmate** | | | **Paper/Pencil** | | | |
|---|---|---|---|---|---|---|---|
| **Measure** | **Mean** | **SD** | **Median** | **Mean** | **SD** | **Median** | **Sig.** |
| **Points (C)** | 191 | 80 | 194 | 135 | 76 | 153 | 0.042 |
| **Accuracy (C)** | 84.9% | 11.1% | 84.3% | 69.9% | 22.9% | 72.5% | 0.020 |
| **Points (NC)** | 199 | 79 | 199 | 135 | 76 | 153 | 0.021 |
| **Accuracy (NC)** | 88.7% | 8.5% | 89.7% | 69.9% | 22.9% | 72.5% | 0.003 |

Table 6.3 presents the analogous results in terms of the number of correct steps, as opposed to earned points. A step is considered correct if the student receives at least 50 percent of the points for that step. The percent accuracy measure in the cutoff case indicates how much of the problem a student successfully completed before the grading was cut off or before the student ran out of time (if applicable), averaged over all of the problems that were attempted by the student. For example, suppose a student attempted two problems, the first of which contained 5 steps and the second of which contained 6 steps. The student attempted all 5 steps in the first problem and all 6 steps in the second problem. However, the student was cut off after 4 steps in the first problem and after 5 steps in the second problem. That means that the student successfully completed 3/5 steps (60%) in the first problem and 4/6 steps (66.7%) in the second problem. Averaging over both problems, the student's percentage accuracy would therefore be 63.4%. In the no-cutoff case, the percent accuracy measure indicates the percentage of successful steps throughout the entire problem, averaged over all of the problems that were attempted by the student.

Both measures in the case of no-cutoff achieved the level of statistical significance. Both measures in the case of cutoff approached statistical significance with a $\rho$ value of 0.064 for number of steps ($\rho = 0.05$ is significant), and a $\rho$ value of 0.039 for percent accuracy of steps ($\rho = 0.025$ is significant). On average, Sketchmate students improved from 84% to 91% when the grading cutoff was not used, as compared to 68% accuracy for the paper and pencil

students. It is interesting that the mean and median scores for Sketchmate are very close to one another, while the mean and median scores for paper and pencil are 9 percentage points apart. This is possibly explained by the higher standard deviations for paper and pencil than for Sketchmate. As with the number of points result, Sketchmate students achieved roughly one and a half letter grades higher on average for the cutoff case, and roughly two letter grades higher on average for the no-cutoff case than paper and pencil students.

Table 6.3: Student tool steps/accuracy for shortest path exercise problems (SD = standard deviation, C = cutoff, NC = no-cutoff). 41 possible steps.

| | Sketchmate | | | Paper/Pencil | | | |
|---|---|---|---|---|---|---|---|
| **Measure** | **Mean** | **SD** | **Median** | **Mean** | **SD** | **Median** | **Sig.** |
| **Steps (C)** | 25 | 10 | 26 | 18 | 11 | 19 | 0.064 |
| **Accuracy (C)** | 84.3% | 13.8% | 84.4% | 68.4% | 27.4% | 77.4% | 0.039 |
| **Steps (NC)** | 27 | 10 | 28 | 18 | 11 | 19 | 0.019 |
| **Accuracy (NC)** | 91.0% | 8.9% | 93.6% | 68.4% | 27.4% | 77.4% | 0.003 |

### 6.4.2 Network Flow

Table 6.4 gives the results of the pre- and post-tests of the network flow portion of the experiment. Similar to the shortest path case, the mean, standard deviation, and median for the pre-test, post-test, and difference between pre- and post-test are given for both the Sketchmate and paper and pencil groups. Again, the top three rows give the statistics for the total number of points earned, and the bottom three rows give the statistics for the percent accuracy of completed problems. As with shortest path, there was statistical significance ($\rho = 0.000$) for the difference between pre- and post-test for both groups, but no statistical significance for the difference between the two groups. For the total points (learning rate) measure, the Sketchmate group improved by 9 points more on average, and 21 points more using the median measure, and while the result looks favorable, the result is not statistically significant ($\rho = 0.583$ for the mean). For the percent accuracy measure, the Sketchmate group scored an average of a 12 percentage points gain more and a median of 6 percentage points gain more than the paper and pencil group, but this difference was not enough for statistical significance ($\rho = 0.274$ for the mean). However, even though this result is not statistically significant, the difference between the two groups amounts to roughly one letter grade higher for Sketchmate students.

Figure 6.3 shows a plot of the difference between pre- and post-test for the total number of points for both groups. Even though the Sketchmate group improves slightly more, the lines are nearly parallel, and therefore do not show statistically significant improvement. Figure 6.4 gives a plot of the improvement from pre- to post-test for the percent accuracy for both groups. This graph shows a much larger difference in score improvements, although it was still not enough to be statistically significant. However, as mentioned previously, the

Table 6.4: Student tool points/accuracy for network flow pre-test and post-test (SD = standard deviation). 218 possible points for both pre- and post-test.

| | Sketchmate | | | Paper/Pencil | | |
|---|---|---|---|---|---|---|
| **Measure** | **Mean** | **SD** | **Median** | **Mean** | **SD** | **Median** |
| **Pre points** | 74 | 61 | 72 | 68 | 56 | 69 |
| **Post points** | 133 | 57 | 145 | 118 | 67 | 124 |
| **Diff. points** | 59 | 46 | 72 | 50 | 46 | 51 |
| **Pre accuracy** | 40.6% | 37.6% | 31.7% | 36.4% | 36.1% | 41.2% |
| **Post accuracy** | 78.1% | 26.3% | 88.7% | 61.3% | 35.3% | 72.5% |
| **Diff. accuracy** | 37.6% | 39.4% | 29.5% | 25.0% | 37.4% | 23.0% |

Sketchmate group improved by roughly one letter grade higher compared to the paper and pencil group.

In Tables 6.5 and 6.6, results of the network flow exercise problems are shown. Table 6.5 gives results for total number and percent accuracy of points, and Table 6.6 gives analogous results for total number and percent accuracy of correct steps, just as in the shortest path exercise problems. Note that finding the augmenting path and updating the flow and residual graphs are considered two different steps in these calculations. The Sketchmate and paper and pencil groups are much closer in the scoring for these exercises than with the shortest path exercises, as is evident from the significance values in the last column of the tables. The only measures that achieved statistical significance for either the points or the steps measures were the percent accuracy measures in the no-cutoff case. The total points measure in the no-cutoff case was close to statistical significance, with a $\rho$ value of 0.075 ($\rho = 0.05$ is significant).

Table 6.5: Student tool points/accuracy for network flow exercise problems (SD = standard deviation, C = cutoff, NC = no-cutoff). 354 possible points.

| | Sketchmate | | | Paper/Pencil | | | |
|---|---|---|---|---|---|---|---|
| **Measure** | **Mean** | **SD** | **Median** | **Mean** | **SD** | **Median** | **Sig.** |
| **Points (C)** | 197 | 113 | 211 | 182 | 101 | 194 | 0.696 |
| **Accuracy (C)** | 68.6% | 26.1% | 74.9% | 60.3% | 31.2% | 64.7% | 0.407 |
| **Points (NC)** | 247 | 106 | 282 | 182 | 101 | 194 | 0.075 |
| **Accuracy (NC)** | 88.2% | 9.2% | 89.8% | 60.3% | 31.2% | 64.7% | 0.002 |

The most interesting results here are the differences between using cutoff and no-cutoff with the Sketchmate students, both for number of points and for number of steps, in terms of percent accuracy. In the case of number of points, Sketchmate students scored nearly 20 percentage points higher on average when the grading was allowed to continue instead of being cut off. When the grading cutoff was applied, Sketchmate students scored roughly one

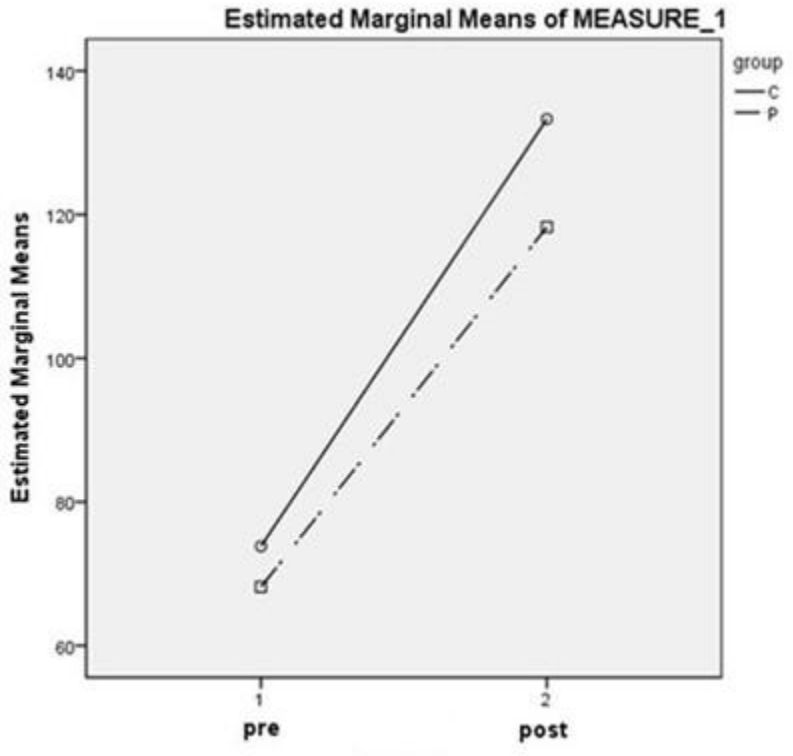


Figure 6.3: Improvement of total number of points for network flow pre-test and post-test. C = Computer (solid line), P = Paper/Pencil (dashed line)

Table 6.6: Student tool steps/accuracy for network flow exercise problems (SD = standard deviation, C = cutoff, NC = no-cutoff). 51 possible steps.

| | **Sketchmate** | | | **Paper/Pencil** | | | |
|---|---|---|---|---|---|---|---|
| **Measure** | **Mean** | **SD** | **Median** | **Mean** | **SD** | **Median** | **Sig.** |
| **Steps (C)** | 29 | 17 | 34 | 28 | 15 | 30 | 0.812 |
| **Accuracy (C)** | 69.5% | 27.5% | 75.7% | 63.1% | 31.8% | 70% | 0.542 |
| **Steps (NC)** | 36 | 15 | 41 | 28 | 15 | 30 | 0.129 |
| **Accuracy (NC)** | 87.7% | 12.4% | 91.6% | 63.1% | 31.8% | 70% | 0.007 |

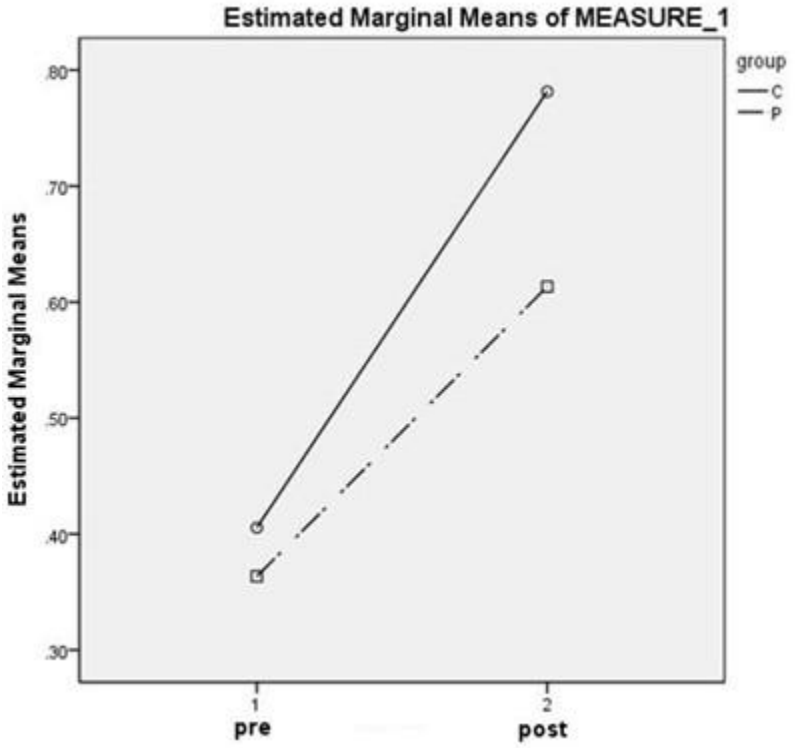


Figure 6.4: Improvement of percent accuracy for network flow pre-test and post-test. C = Computer (solid line), P = Paper/Pencil (dashed line)

letter grade higher for both mean and median than the paper and pencil students. However, when no grading cutoff was applied, Sketchmate students scored roughly three letter grades higher using the mean measure, and roughly two and a half letter grades higher using the median measure, than the paper and pencil students. Sketchmate students scored more than 18 percentage points higher in the case of number of steps when the grading was not cut off, as opposed to when the grading cutoff was applied. When the grading cutoff was applied, Sketchmate students achieved roughly half a letter grade higher for both mean and median than the paper and pencil students. However, when no grading cutoff was applied, Sketchmate students scored roughly two and a half letter grades higher using the mean measure, and roughly two letter grades higher using the median measure, than the paper and pencil students. Not using a grading cutoff allows the Sketchmate students to achieve statistically significant higher scores than the paper and pencil students. Because Sketchmate gets students back on track after making serious errors, the students are able to regroup and build up many more points beyond the step at which grading would have otherwise stopped.

### 6.4.3 Survey Results

The numerical results from the survey are given in Table 6.7 for the shortest path questions and Table 6.8 for the network flow questions. The first three columns of numbers give the mean, standard deviation, and median for the students who used Sketchmate for that algorithm, and the last three columns show the same statistics for the students who used paper and pencil for that algorithm. The students gave an assessment of their comfort level with the algorithm before and after working through the experimental activities. It is clear that all students felt more comfortable with the algorithms after the activities, regardless of whether they used Sketchmate or paper and pencil for the algorithms. They were also asked to provide an assessment of their experience with the various operations for updating the graphs in each step of the algorithm. Shortest path involves marking the edges in the shortest path from the start vertex to the other vertices in the graph, updating vertex states, and updating vertex costs. Network flow involves marking the augmenting path, updating the flows and capacities along the edges, and adding or removing edges to/from the residual graph. The students rated their experience for each of these operations for both Sketchmate and paper and pencil on a scale of 1 to 7, with 1 being the most unfavorable and 7 being the most favorable. All students could give a rating for the paper and pencil portions since all students worked through the pre- and post-tests with paper and pencil. However, the students who only used paper and pencil for the given algorithm could not give an assessment for the Sketchmate portions of the questions, hence the blank entries in the tables.

An interesting result is that on average, for each operation of both algorithms, the Sketchmate students favored the Sketchmate version over the paper and pencil version. For example, the Sketchmate students rated marking the shortest path with paper and pencil as 4.2, which is lower than their rating of 5.5 for the Sketchmate version of marking the shortest path. This result suggests that in general, students prefer Sketchmate to paper and pencil. Table 6.9 shows the significance levels and effect sizes for each of the operations ($\rho = 0.05$

Table 6.7: Student tool survey results for shortest path questions (SD = standard deviation, P = Paper/Pencil, S = Sketchmate). The Sketchmate column gives the ratings that Sketchmate users assigned to Sketchmate and to Paper/Pencil, while the Paper/Pencil column gives the ratings that the Paper/Pencil group assigned to Paper/Pencil. For Sketchmate users, the Paper/Pencil ratings refer to their experience with Paper/Pencil on the pre- and post-tests.

| | **Sketchmate** | | | **Paper/Pencil** | | |
|---|---|---|---|---|---|---|
| **Question** | **Mean** | **SD** | **Median** | **Mean** | **SD** | **Median** |
| **Comfort before** | 4.2 | 1.6 | 4 | 4.8 | 1.4 | 5 |
| **Comfort after** | 5.8 | 1.2 | 6 | 5.9 | 1.2 | 6 |
| **Shortest path (P)** | 4.2 | 1.9 | 5 | 4.3 | 1.5 | 4 |
| **Shortest path (S)** | 5.5 | 1.5 | 6 | - | - | - |
| **Vertex states (P)** | 5 | 1.2 | 5 | 4.9 | 1.5 | 5 |
| **Vertex states (S)** | 5.4 | 1.5 | 5.5 | - | - | - |
| **Vertex costs (P)** | 5.4 | 1.7 | 6 | 4.6 | 1.2 | 5 |
| **Vertex costs (S)** | 5.8 | 1.1 | 6 | - | - | - |

Table 6.8: Student tool survey results for network flow questions (SD = standard deviation, P = Paper/Pencil, S = Sketchmate). The Sketchmate column gives the ratings that Sketchmate users assigned to Sketchmate and to Paper/Pencil, while the Paper/Pencil column gives the ratings that the Paper/Pencil group assigned to Paper/Pencil. For Sketchmate users, the Paper/Pencil ratings refer to their experience with Paper/Pencil on the pre- and post-tests.

| | **Sketchmate** | | | **Paper/Pencil** | | |
|---|---|---|---|---|---|---|
| **Question** | **Mean** | **SD** | **Median** | **Mean** | **SD** | **Median** |
| **Comfort before** | 3.2 | 2.1 | 3 | 3.6 | 1.9 | 3 |
| **Comfort after** | 5.1 | 1.6 | 5 | 5.6 | 1.3 | 6 |
| **Aug. path (P)** | 4.3 | 1.5 | 4 | 4.2 | 1.9 | 5 |
| **Aug. path (S)** | 5.5 | 1.5 | 6 | - | - | - |
| **Flows/capacities (P)** | 4.6 | 1.2 | 5 | 5.4 | 1.7 | 6 |
| **Flows/capacities (S)** | 5.5 | 1.2 | 6 | - | - | - |
| **Edge add/remove (P)** | 4.3 | 1.5 | 4 | 4.9 | 1.8 | 5 |
| **Edge add/remove (S)** | 5.6 | 1.4 | 6 | - | - | - |

is significant). The difference in rating for the augmenting path was statistically significant with $\rho = 0.030$. The differences in rating for the shortest path and adding/removing edges were near statistical significance with $\rho = 0.060$ for shortest path and $\rho = 0.058$ for adding/removing edges. The effect sizes for vertex states and augmenting path were between medium and large, while the effect sizes for shortest path and adding/removing edges were medium. The effect sizes for updating the vertex costs and edge flows and capacities were between small and medium, which is consistent with the fact that a number of students commented that these operations were somewhat cumbersome.

Table 6.9: Student tool survey results for the difference between Sketchmate students' ratings for Sketchmate and Paper/Pencil for the shortest path and network flow operations. Statistical significance level is 0.05. A small effect size is 0.1, a medium effect size is 0.25, and a large effect size is 0.4. Effect sizes larger than 0.4 are generally unrealistic.

| **Operation** | **Significance** | **Effect size** |
|---|---|---|
| **Shortest path** | 0.060 | 0.286 |
| **Vertex states** | 0.108 | 0.373 |
| **Vertex costs** | 0.195 | 0.2 |
| **Augmenting path** | 0.030 | 0.337 |
| **Edge flows/capacities** | 0.144 | 0.146 |
| **Edge add/remove** | 0.058 | 0.234 |

Table 6.10 shows a summary of the results of the free response portion of the survey. Students commented on aspects of the computer tool that they liked and that they did not like, as well as suggestions for improvement, as indicated in the Positive and Negative columns of the table. Only the aspects of the tool for which two or more students commented are given in the table. A large number of students commented that they liked the immediate feedback and that it helped them learn the algorithm. In general, students also seemed to like the interface look and feel and the ease of use of the tool. The most frequent negatives commented by students were that it was somewhat cumbersome to change vertex costs and edge flows and capacities, and that the application would occasionally crash or freeze. A separate survey question asked the students whether they prefer using Sketchmate or paper and pencil. Out of the 34 responses to this question, 27 students preferred Sketchmate, 3 students preferred paper and pencil, and 4 students either had no preference or they preferred using a combination of both Sketchmate and paper and pencil. A chi-square test showed that this result was statistically significant with $\rho = 0.000$. Overall, the students enjoyed using Sketchmate, and upon improving the functionality and user-friendliness of the tool, future studies may show even more preference for Sketchmate.

Table 6.10: Student tool survey free response results. Numbers in parentheses are the number of students who commented on that particular aspect of the tool. 37 students filled out a survey.

| **Positive** | **Negative** |
|---|---|
| Feedback/explanation (24) | Changing costs/capacities (23) |
| Well-organized/nice overall (10) | Crashes/bugs (10) |
| Ease of use (10) | Tedious (6) |
| Helpful/useful (8) | Include more explanation for feedback (4) |
| Efficient/fast (8) | Residual graph messy (edges) (3) |
| Clean/neat (7) | Should split flow/residual into 2 steps (2) |
| Look & feel/layout (6) | Should highlight areas to be changed (2) |
| Intuitive (4) | Would like to undo last operation (2) |
| Corrects the working graph (2) | Would like to show file name of graph (2) |
| Prefer to paper/pencil (2) | Would like to work with random graph (2) |

## 6.5 Limitations

In this section we discuss limitations of both the experimental design and the Sketchmate software.

### 6.5.1 Limitations of the Experiment

One major limitation in this experiment was the small number of participants, as it was restricted by the number of students enrolled in the course, which was approximately 50 students. A much larger sample, perhaps a couple hundred students, was needed in order to reduce the variation in the data and allow for the possibility of producing more results with statistical significance. Class sizes of that magnitude were simply not available for this study. It may be difficult to ever achieve statistical significance because algorithms classes will almost never be large enough to reduce the variability to the point where it is possible to obtain statistically significant conclusions. Attempting to pool data from different classes is not a viable option, even if taught by the same instructor, because variations in aspects such as the sequencing of topics, the manner in which the topics are taught, and student abilities would introduce additional independent variables that would be almost impossible to control for.

Another issue was with the design of the pre- and post-tests. The problems increased in difficulty for the first two or three problems, and then became easier for the problems at the end of the tests. This design decision ensured that most students would have time to complete at least two of the more difficult problems, which was a much better test of students' knowledge than testing students on easy problems. We did not expect most students to have time to work the easy problems towards the end of the test, and while few students were able to get to the later problems, those who did scored high on these problems since they were

much easier. Thus, students were not tested with enough difficult problems, and therefore some post-test scores may have been inflated. For the shortest path post-test, the only student in the experiment who completed the last two problems was a paper and pencil student. A better test design would have been to order the problems in terms of difficulty so that each problem is of the same or greater difficulty, but never of less difficulty, than the previous problems. This test design flaw was corrected for the instructor tool experiment.

Another possible issue is the difference of format for the pre-test, exercises, and post-test. For the students in the paper and pencil group, paper and pencil was the format of all of the problems they worked through, whether it was pre-test, exercises, or post-test. However, for the Sketchmate group, the pre- and post-tests were in paper and pencil format, while the exercises were in computer format. Thus, students needed to become accustomed to a different format between the pre-test and the exercises, and again between the exercises and the post-test. This "context switching" possibly may have detracted from their performance.

An additional concern was that many of the students in both groups were able to complete all the problems in the network flow practice problem set. The experimenters overestimated the time that students would need to complete the problems, resulting in a shortage of problems, which could have led to distorted results since the experiment was designed to test students on the basis of an open-ended problem set. There were not enough practice problems available to properly test if students using Sketchmate would complete more problems than students using paper and pencil.

### 6.5.2 Limitations of Sketchmate

One of the limitations of the Sketchmate software is that it contained a few small programming bugs, as is common for all new and developing software packages. In several cases, the program would crash while the students were working through the exercises. This would cause the students to have to repeat the exercise they were on at the time of the crash, thus slowing down their progress and resulting in students not getting through as many problems as they could with bug-free software. While this was not a major factor, it did detract from some students achieving higher scores on the exercises.

Related to the programming bugs in the software is the fact that Sketchmate was not as user friendly as it could have been. Many students commented on the survey that it was somewhat difficult and unwieldy to edit the vertex costs and edge capacities and flows. This shortcoming slowed the students' progress on working through the exercises.

## 6.6 General Discussion

The results of this study were encouraging overall. While it was somewhat of a disappointment that learning rates did not improve significantly for a more complicated problem like network flow, it was a rather pleasant surprise that for a simple problem like shortest path, the improvement in percent accuracy for Sketchmate users was roughly one letter grade higher (8% for mean and 11% for median) than for paper and pencil students. It was also

pleasing, but more expected, to see a mean improvement in percent accuracy of at least one letter grade higher for network flow.

It was somewhat surprising that the Sketchmate results for the shortest path practice exercises were actually better than the Sketchmate results for the network flow practice exercises, in terms of levels of significance. Since the network flow algorithm is more complicated, it is possible that the more complex computer interface could have presented a greater challenge for students, as compared with shortest path. It would be of interest to see if improving the efficiency and user-friendliness of the interface would result in any difference in student performance. The measures of total points in both the cutoff and no-cutoff cases showed a significant improvement for Sketchmate users for the shortest path exercises. In terms of percent accuracy, Sketchmate students achieved two to three letter grades higher on the shortest path exercises. The percent accuracy measure for the no-cutoff case showed a significant improvement for Sketchmate students for the network flow exercises. This result amounts to nearly three letter grades higher for Sketchmate students in terms of total points.

The most encouraging results are the differences between the measures when grading cutoff is used as opposed to using no grading cutoff, especially in the case of network flow. With network flow, eliminating the grading cutoff allowed the differences in the percent accuracies for points and for steps to rise to the level of statistical significance. This allowed Sketchmate students to score two and a half to three letter grades higher on the exercises than the paper and pencil students, whereas with cutoff applied, Sketchmate students would only score up to one letter grade higher than paper and pencil students. In many cases, students would choose an incorrect augmenting path in the first step, resulting in grading being cut off immediately. When students are then shown the correct path with Sketchmate, they are able to learn from this feedback and get back on track to correctly solve much, if not all, of the remainder of the problem. An argument could be made to also apply no grading cutoff to paper and pencil solutions, and attempt to find correct portions of the graph for the rest of the student's problem solution. However, attempting this technique is risky and becomes a very subjective process in that the grader would need to somehow guess at the student's thought process. With Sketchmate correcting the graph at each step, the students are starting with a correct set of graphs each time, resulting in there being no doubt in grading the student's answer for the next step.

According to student feedback from the survey, in general, students prefer using Sketchmate to using paper and pencil to work through the problems. Students commented that the immediate feedback and explanation was very helpful for them, and that they enjoyed the overall interface experience and ease of use of the tool. The students made suggestions for improvement in the areas of user-friendliness and functionality, and with these improvements, Sketchmate may become more effective as a learning tool in the future.

# Chapter 7

# Analysis and Evaluation of the Instructor Tool Experiment

The focus of the development of the instructor tool was on providing an interface in which the instructor can manually manipulate the components of a graph one by one in an interactive environment, annotate their lecture with the use of a notepad object, and revert to previous steps of the algorithm for review. The only existing tool that supports manual simulation of graph algorithms for the instructor is MatrixPro (Karavirta et al., 2004), however, it only allows the instructor to invoke a pre-defined library of high-level operations, rather than allowing the instructor to manipulate the graphs at the level of vertices and edges. No existing tool designed for simulating graph problems provides the Revert pane or notepad features. The goal of the experiment with our instructor tool was to measure student learning rate and learning outcome applied to graph problems resulting from observing a lecture based on using Sketchmate compared to observing a lecture using the traditional whiteboard. The hope was that a Sketchmate lecture would lead to equivalent learning rates and learning outcomes as a whiteboard lecture, and that this dynamic environment can help use class time more effectively than a static whiteboard environment using hand-drawing and erasing.

The experiment was a between-subjects, pre-test and post-test design. Students observed a lecture on network flow based on using either Sketchmate or the whiteboard. Students were allotted a fixed amount of time to complete the pre-test and post-test. The tests were designed to be open-ended, that is, they contained more problems than the students could complete in the allotted time. This design allowed us to measure learning rate, given by the amount learned within a fixed period of time.

The experiment was designed to test the hypothesis that there would be no difference in learning for network flow problems with a Sketchmate-based lecture as compared with a whiteboard lecture. In other words, the hypothesis was that a Sketchmate lecture would not detract from learning as compared with a whiteboard lecture. The literature [(Hundhausen et al., 2002), (Narayanan and Hegarty, 2002)] suggests that the use of computer simulators paired with passive learning does not increase learning outcomes. The main reason for developing the tool is that it allowed for a cleaner and smoother presentation compared to the whiteboard, and that an interactive manual simulation tool would make lectures more

efficient and seamless.

We also collected data of how students perceived the presentation efficiency and effectiveness with either of the methods. The hope was that students would enjoy observing the Sketchmate lecture at least as much, and possibly more than, observing the whiteboard lecture.

## 7.1 Subjects, Setting, and Materials

Approximately 40 students from the Spring 2012 undergraduate Programming Languages and Systems course in the Electrical Engineering and Computer Science department of the University of Tennessee-Knoxville participated in the study. The experiment was included as a class activity for the course. The study was conducted in a department lecture hall on campus. The experimenters developed whiteboard and Sketchmate versions of two different network flow problems of varying difficulty. The students needed to observe a lecture on both problems.

## 7.2 Design

The experimenters used a between-subjects, pre-test and post-test model for testing differences in learning rate and learning outcome as measured by the difference between pre- and post-test scores for students observing a Sketchmate-based lecture (experimental condition) and students observing a whiteboard lecture (control condition). The dependent variables were the difference between pre- and post-test scores in terms of number of points (a measure of learning rate) and percent accuracies (a measure of learning outcome). Significant differences across dependent variables were tested for using repeated measures analysis of variance (ANOVA), with differences being considered significant at the $\rho < 0.05$ level for the learning rate measure, and the $\rho < 0.025$ level for the learning outcome measure. These differences in significance levels were necessary because the learning rate and learning outcome measures were largely based on the same data set. In order to obtain valid tests of significance for the learning outcome measure, the confidence interval needed to be halved.

## 7.3 Procedures

The experiment took place during a regular lecture period of the Programming Languages and Systems course. Students were first given a 12-minute written pre-test on network flow graph problems. The students should have previously seen network flow in an algorithms course, but the algorithm is sufficiently complex that we presumed that students would not have retained much knowledge about the algorithm. Comparing the pre-test scores in Tables 6.4 and 7.1, this seems like a reasonable assumption since the students in this experiment did not perform better on the pre-test than the students in the algorithms course in the student tool experiment.

After the pre-test, the students were randomly assigned to two different groups. The first group observed the instructor give an approximately 20 minute lecture on two network flow problems using Sketchmate. The second group observed the instructor lecturing on the same two network flow problems using the traditional whiteboard for approximately 20 minutes, immediately after his Sketchmate presentation. One problem involved a straight-forward network flow graph and the other involved a graph in which an augmenting path followed a backedge. Section 4.3 gives an example of a network flow problem similar to the problems used in this experiment. The time taken to lecture on the two problems was noted for each technique.

After the lecture sessions, the students were given a written post-test on network flow problems. The structure of the post-test was identical to that of the pre-test. The post-test was given in the same class period as the lectures to avoid the possibility of students studying for it and therefore distorting their scores.

The students who observed the computer tool lecture were also given a survey that asked them questions about how they liked the computer tool compared to the whiteboard lecture on the same material that they received in a previous semester. There were several Likert scale questions with a rating between 1 (unfavorable) to 7 (favorable), as well as free response questions. The survey included a question that asked students if they would prefer observing a future network flow lecture presented with the whiteboard or the computer tool, and they were asked to explain their preference.

Only students who attempted the pre-test and post-test and who followed directions properly were included in the study. The primary experimenter scored all attempted pre- and post-test problems for accuracy using the same scoring scheme described in Chapter 6.

An interscorer agreement test was not performed for this study, since the method for grading the pre- and post-tests was exactly the same as the method used in the student tool study.

## 7.4 Results and Discussion

Table 7.1 shows the results of the experiment with the instructor tool used for network flow. The mean, standard deviation, and median for the pre-test, post-test, and difference between pre- and post-test are given for both the Sketchmate and whiteboard groups. The top three rows give the statistics for the total number of points earned (learning rate), and the bottom three rows give the statistics for the percent accuracy of completed problems (learning outcome). There was statistical significance ($\rho = 0.000$) for the difference between pre- and post-test for both groups, but no statistical significance for the difference between the two groups. For the total points measure, the Sketchmate group scored 28 points higher on average and 30 points higher using the median measure, which seems promising, but was not enough to be statistically significant ($\rho = 0.147$). For the percent accuracy measure, the Sketchmate group only scored an average of roughly 6 percentage points higher than the whiteboard group, which is consistent with the high $\rho$ value ($\rho = 0.502$). Note that the statistical significance values are for the mean rather than the median.

An interesting result for percent accuracy is that using the mean measure, the Sketchmate group scored roughly 6 percentage points higher than the whiteboard group, while using the median measure, the whiteboard group scored roughly 2 percentage points higher than the Sketchmate group. This indicates that the results of this experiment are ambiguous for the percent accuracy measure in that it is unclear which method resulted in higher learning outcomes. This odd result is most likely explained by the very high variability in the test scores. A future experiment would perhaps produce a clearer indication of which method is more effective in terms of learning.

Table 7.1: Instructor tool experiment results for network flow pre-test and post-test (SD = standard deviation). 257 possible points for both pre- and post-test.

| | **Sketchmate** | | | **Whiteboard** | | |
|---|---|---|---|---|---|---|
| **Measure** | **Mean** | **SD** | **Median** | **Mean** | **SD** | **Median** |
| **Pre points** | 80 | 64 | 61 | 64 | 74 | 13 |
| **Post points** | 180 | 63 | 195 | 136 | 79 | 151 |
| **Diff. points** | 100 | 69 | 85 | 72 | 64 | 55 |
| **Pre accuracy** | 44.7% | 32.8% | 36.3% | 41.5% | 39.1% | 36.1% |
| **Post accuracy** | 76.4% | 25.6% | 83.9% | 67.5% | 33.5% | 82.4% |
| **Diff. accuracy** | 31.7% | 34.1% | 22.9% | 26.0% | 43.0% | 25.3% |

Figure 7.1 shows a plot of the difference between pre- and post-test for the total number of points for both groups. There is clearly more improvement for the Sketchmate group, however, not enough for statistical significance. Figure 7.2 gives a plot of the improvement from pre- to post-test for the percent accuracy for both groups. These lines are even closer together than those of the plot of number of points, thus illustrating the lack of statistical significance in the results.

In Table 7.2, the mean, standard deviation, and median of the results for the student survey are given. Only the students who viewed the computer-based lecture completed the survey, as the students who viewed the whiteboard lecture could not comment on any aspects of using Sketchmate as a lecture aid. Students rated each feature on a scale of 1 to 7, with 1 being most unfavorable and 7 being most favorable. As expected, all students felt more comfortable with the network flow algorithm after the lecture than before the lecture. In general, students preferred the Sketchmate lecture, as indicated by a score of 5.2 for Sketchmate preference, and a score of 5.4 for overall experience of the Sketchmate lecture. Individual aspects of updating the graphs were also favorable. Students also felt that the Revert pane and notepad features were quite helpful in their understanding of the algorithm.

The results from the free response questions of the survey are shown in Table 7.3. Students were asked to comment on aspects of Sketchmate that they liked and that they did not like, and to give suggestions for improvement. The comments are only reported if two or more students addressed that particular aspect. A number of students indicated that they liked the fact that the graph items to be changed were highlighted in color, and that using

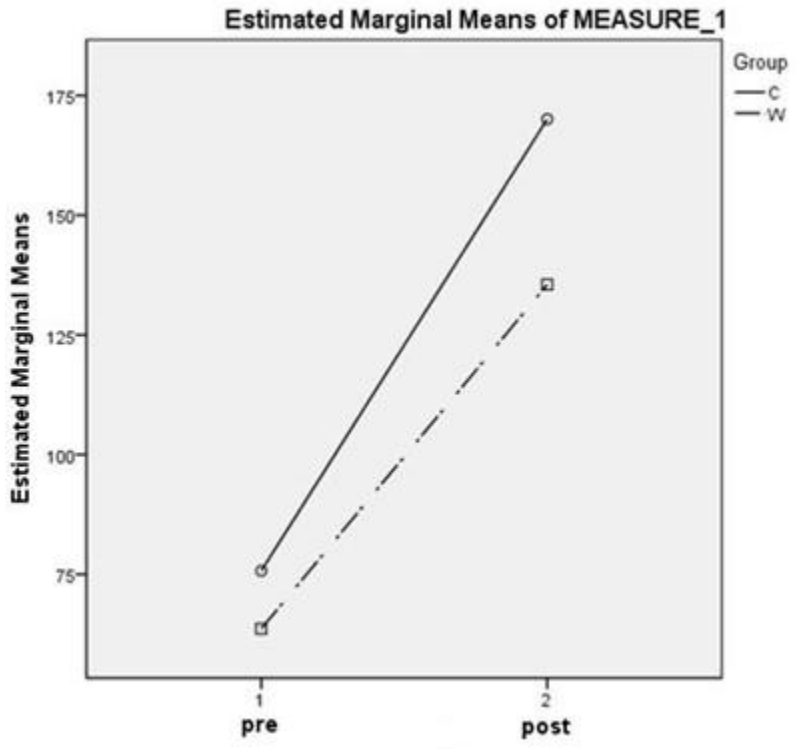


Figure 7.1: Improvement of total number of points for network flow pre-test and post-test. C = Computer (solid line), W = Whiteboard (dashed line)

Table 7.2: Instructor tool survey results for students viewing Sketchmate lecture (SD = standard deviation)

| Question | Mean | SD | Median |
|---|---|---|---|
| **Comfort before** | 4.1 | 1.9 | 5 |
| **Comfort after** | 6.3 | 1.0 | 7 |
| **Preference for Sketchmate** | 5.2 | 1.3 | 5 |
| **Overall experience** | 5.4 | 0.9 | 6 |
| **Augmenting path** | 5.5 | 1.0 | 5 |
| **Edge flows/capacities** | 5.1 | 1.1 | 5 |
| **Edge add/remove** | 5.2 | 1.3 | 5 |
| **Revert pane** | 5.6 | 1.1 | 6 |
| **Notepad** | 5.5 | 1.2 | 5 |

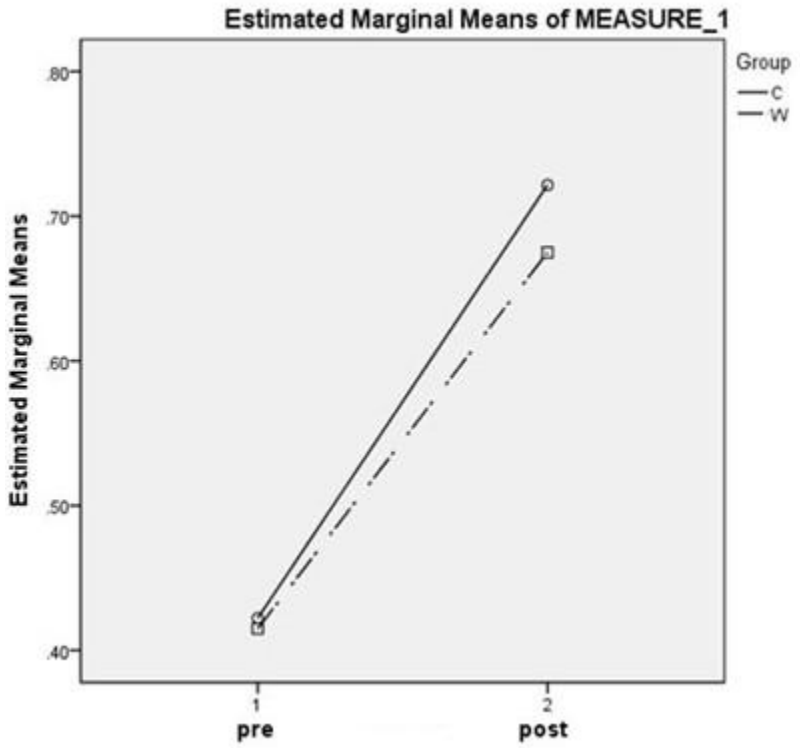


Figure 7.2: Improvement of percent accuracy for network flow pre-test and post-test. C = Computer (solid line), W = Whiteboard (dashed line)

the tool is cleaner and more efficient, especially since the instructor does not need to erase and redraw graphs. The most frequent negative aspects are that the interface is complicated and very busy, and that updating the residual graphs and edge capacities could be made smoother and cleaner. Overall, the students seemed to enjoy the use of Sketchmate in the network flow lecture, and preferred it to an analogous whiteboard lecture.

Table 7.3: Instructor tool survey free response results. Numbers in parentheses are the number of students who commented on that particular aspect of the tool. 17 students filled out a survey.

| **Positive** | **Negative** |
|---|---|
| Efficient/fast (no redrawing graphs) (9) | Cluttered/complicated (5) |
| Clean/neat (no handwriting) (7) | Drawing graphs messy/slow (4) |
| Easy/clear (5) | Updating flows/capacities (2) |
| Highlights items to change (3) | Layout (2) |
| Notepad (3) | Would like more color (2) |
| Revert pane (2) | Would like for students to try it (2) |
| Useful/helpful (2) | Fast-paced lecture (2) |
| Well-organized (2) | Notepad text too small (2) |
| Like better than whiteboard (2) | Would like notes/images distributed (2) |

## 7.5 Limitations

In this section we discuss limitations of both the experimental design and the Sketchmate software.

### 7.5.1 Limitations of the Experiment

One major limitation in this experiment was the small number of participants, as it was restricted by the number of students enrolled in the course, which was approximately 50 students. A much larger sample, perhaps a couple hundred students, was needed in order to reduce the variation in the data and allow for the possibility of producing results with statistical significance. Class sizes of that magnitude were simply not available for this study.

Another issue is that the students only observed a lecture passively; they did not have an opportunity to work with the computer tool hands-on. Had more time been allowed for the lecture, the instructor may have incorporated methods of engaging the students by having them follow along with the lecture by using the tool themselves. This approach may possibly increase student learning outcomes.

### 7.5.2 Limitations of Sketchmate

A number of students commented that the interface of Sketchmate was rather busy and complicated. They would have preferred fewer graphs being shown and splitting the updating of the flow and residual graphs into two separate steps. Both the students and the instructor remarked that it was not necessary to show the previous flow graph. Students also felt that drawing the residual graph could have been cleaner in terms of drawing new edges and updating the edge capacities. Students seemed to find these interface shortcomings distracting and it may have detracted from their learning. The instructor also commented that clicking and typing to edit the edge flows and capacities was somewhat unwieldy and slowed the lecture slightly. He suggested that a tab-key or a click-and-type (without the need to manually delete the old value) approach might help the edge flow and capacity editing process become smoother.

## 7.6 General Discussion

The results of this study were inconclusive. Students observing the lecture using Sketchmate did not learn significantly better than students observing the whiteboard lecture. While Sketchmate students achieved higher learning rates for both the mean and median, results were mixed for the percent accuracy measure. Sketchmate students scored a higher mean but a lower median than the whiteboard students in terms of percent accuracy. This inconsistency may possibly have been a result of the very high amount of variability in the data. These ambiguous results warrant further investigation in a future study.

The instructor reported that he was able to cover more material in less time in the Sketchmate lecture compared to the whiteboard lecture. The Sketchmate lecture took approximately 19 minutes while the whiteboard lecture took 20 minutes. In both lectures, he verbally addressed five important points about flow and residual graphs, but in the whiteboard lecture he only had time to write down three of these points, while in the Sketchmate lecture he had time to type all five of these points. In the Sketchmate lecture he had time to review the second example using the Revert pane, while in the whiteboard lecture he did not have any time to review the second example. Even if he would have had time, there was no easy way to review the steps, as the previous graphs had been erased. In the whiteboard lecture, the instructor did not have time to update the residual graph in the last step of the second problem, whereas in the Sketchmate lecture, he had time to complete both problems in their entirety, as well as review the second problem via the Revert pane.

Another advantage of Sketchmate that the instructor reported is that it is much cleaner to work with the graphs using Sketchmate than by hand-drawing the graphs on the whiteboard. During the whiteboard lecture, he often needed to switch between different markers and erase augmenting paths drawn in different colors, erase flow numbers to write a new number, redraw edges after wiping them out while erasing the augmenting path, and erase edges in the residual graph. It was much cleaner using Sketchmate, since he was able to simply choose the affected item and modify it. He also found it easier in Sketchmate to highlight

alternative augmenting paths since he could simply click and select/deselect edges, rather than needing to draw an augmenting path with a different color marker and then erase it. Finally, the instructor found the ability to load pre-prepared graphs very helpful since hand-drawing a graph on the whiteboard was time consuming and during his lecture he made several mistakes in the graph while drawing it on the whiteboard.

The instructor also commented on several other features of Sketchmate that he found helpful in delivering his lecture. As the students also indicated, he found the fact that Sketchmate highlights in color all graph edges and flows that need to be changed useful. It served as a reminder of what changes he needed to make to the graphs, plus it was much clearer to the students that the changes in the flow and residual graphs only occur along the augmenting path. The instructor also found it helpful that Sketchmate displays the previous residual graph that clearly shows the augmenting path, because the augmenting path in the current residual graph is obscured upon creating backedges. Lastly, the instructor found the notepad to be a valuable feature in that it allowed him to quickly type definitions and important points.

According to the student feedback from the survey, in general, students prefer observing a Sketchmate-based lecture to a whiteboard lecture. Students commented that highlighting changes, the Revert pane, and the notepad were helpful for them, and that they enjoyed the overall interface experience and clean aspect of the tool. The students made suggestions for improvement in the areas of interface complexity and the smoothness of drawing and updating residual graphs, and with these improvements, Sketchmate may become more effective as a learning tool in the future. In general, both the instructor and the students preferred using Sketchmate in lecture as opposed to the whiteboard, and with further development and refinement of the computer tool, it may prove to serve as a useful lecture aid in the future.

# Chapter 8

# Conclusions and Future Work

This research involved developing and testing two versions of a computerized instructional tool for teaching the shortest path and network flow graph algorithms. One version is for students to use in studying practice exercises and completing homework problems, and the other is for instructors to use as an aid in presenting lectures. An experiment was designed and performed for each tool to measure its effectiveness compared with traditional teaching methods such as paper and pencil based homework and a whiteboard lecture. The following sections present a summary of the findings from both studies, as well as possible directions for future work.

## 8.1 Student Tool

The student tool allows students to practice shortest path and network flow graph problems and receive detailed immediate feedback at each step of the algorithm. Results from the experiments showed that students improved by about a letter grade more with Sketchmate than with paper and pencil between pre- and post-test for both shortest path and network flow, although the result was not statistically significant. Sketchmate students achieved higher learning rates and higher percent accuracy than paper and pencil students on the shortest path exercises, and these results were statistically significant. This corresponded to a one and a half to two letter grade difference. The Sketchmate students achieved higher learning rates and higher percent accuracy than paper and pencil students on the network flow exercises, but these results were only statistically significant for the percent accuracy measure when no grading cutoff was applied. The result amounts to a two and a half to three letter grade difference. Overall, the students preferred using Sketchmate to using paper and pencil for working through the exercises, and they felt that the immediate detailed feedback was very helpful. Future goals for the student tool include improving learning rates, reducing the amount of time to complete homework, and increasing, or at least maintaining, accuracy of student homework submissions.

## 8.2 Instructor Tool

The instructor tool provides a manual simulation environment for shortest path and network flow graph problems with the ability to annotate simulations with notes and review previous steps in the algorithm through the use of a Revert pane. Experimental results were somewhat ambiguous and need further study. Sketchmate students achieved higher learning rates than whiteboard students according to both the mean and median scores. However, for the percent accuracy measure, the mean suggested that the Sketchmate tool might provide a slightly better learning outcome, but the median suggested that a whiteboard lecture might provide a slightly better learning outcome, and in any event, neither result was statistically significant. The instructor reported that using Sketchmate allowed him to cover more material in less time, and that it was much cleaner and more efficient to work with than hand-drawn graphs on the whiteboard. In general, the students also preferred the use of Sketchmate compared to the whiteboard because it is cleaner, it clearly indicates where changes need to be made, and it includes additional useful features such as the Revert pane and notepad. Future goals for the instructor tool include finding techniques to allow for easier, faster, and more effective preparation and presentation of lectures involving data structures and algorithms, as well as more efficient use of class time.

## 8.3 Lessons Learned

Several observations resulted from performing the two experiments. These observations are discussed below.

1. Providing immediate feedback after intermediate steps in a homework simulation exercise and correcting a graph so that it starts in a correct state after each step seems to be a significant improvement over the traditional method of completing a homework assignment and receiving some feedback on it a week or two later. Immediate feedback could be one of the biggest wins one gets from computer-aided tutoring systems. The experimental results showed that manual simulation with interactive feedback in a homework assignment can result in significant improvement in homework assignment grades even for simple problems such as shortest path. They also suggested that such interactive simulation systems might result in some improvement in overall learning outcomes, as measured by the improvement from a pre- to a post-test, although unlike the homework exercises, the results for the tests were not statistically significant.

2. Displaying both a "before" and "after" view of graphs, both in the instructor and student tools, provided students with helpful contextual information, since in the traditional single view, the old state of the graph quickly gets destroyed and it can be difficult to remember what it looked like or what changes still need to be applied by the algorithm to complete the next step.

3. Providing a Revert pane is very helpful for quickly reviewing a problem or stepping back to a previous step if a student has a question.

4. Providing an annotation window is very helpful for quickly entering notes that the instructor would like students to copy. The instructor who tested the tool initially thought the window would be superfluous, but then found it quite useful during his actual lecture.

5. Getting statistically significant results in experimental studies with students is extremely difficult because of the limited size of upper division classes at many universities. It took a nearly 2 letter grade difference between the experimental and control groups to get statistical significance in a class of 50 students, which is an unrealistic level of improvement for most tools. It is also difficult to compare studies across different semesters or studies from different universities because additional independent variables get introduced, such as different instructors, different sequences of topics, or different student capabilities.

## 8.4 Future Work

This research offers many possible avenues for future work. Possible future research directions for both the instructor and student tool include:

1. Implementing manual simulation to operate on data structures other than graphs, such as trees, heaps, lists, and hash tables. Applying manual simulation to other data structures would offer insight into its level of success in multiple domains.

2. Working with randomly generated graphs in addition to user supplied graphs. This ability would allow for unlimited practice of graph algorithms since users would not need to design and input a graph themselves.

3. Modifying the means for updating vertex costs and edge flows and capacities through the use of a type-and-tab method or a click-and-type method, to replace the present method that involves deleting the old value and typing Enter to commit the change. Both the students and the instructor found the current approach to be rather cumbersome.

4. Displaying a table next to the graph showing the state information of the vertices such as cost, predecessor, and whether or not the vertex has been visited. This table would allow for an easier transition from the graphical representation to the algorithm implementation in code.

5. Providing a code window where code can be entered and then executed, while changing the appropriate component of a graphical representation of the data structure. This would enable the user to visualize the correspondence between the current line of code and its effect on the data structure and might help with the transition from the graphical representation to the algorithm implementation in code.

6. Collecting experimental data from different courses at various colleges, universities, and community colleges to test the tool's effectiveness across different institutions and levels of instruction, and to help resolve whether the tool is beneficial to learning, detrimental to learning, or neutral to learning.

In addition, the following items are possible future directions for the instructor tool:

1. Allowing the instructor to change the costs or capacities of the edges during the middle of the simulation to address student "what if" questions or to correct a mistake that might have been made when creating an impromptu graph.

2. Simplifying the instructor tool interface layout and reducing its level of complexity to make it less overwhelming and distracting for students.

3. Allowing the instructor to manually simulate a general prioritized graph search algorithm in addition to the built-in shortest path and network flow algorithms.

4. Providing a means for the students to actively work with the tool during class, rather than passively viewing a lecture. If students are actively engaged in the material, it may help students learn the subject better.

# Bibliography

AlgoViz (2010). Data Structures and Algorithms Visualization Wiki. http://web-cat.cs.vt.edu/AlgovizWiki.

Anderson, R., Anderson, R., Simon, B., Wolfman, S., VanDeGrift, T., and Yasuhara, K. (2004). Experiences with a Tablet PC Based Lecture Presentation System in Computer Science Courses. In *SIGCSE '04: Proceedings of the 35th CSE Technical Symposium on Computer Science Education*, pages 56–60. Norfolk, VA, USA, ACM Press.

Athanasios, B. (2009). JAVENGA Java Applet. http://users.uom.gr/~thanasis/JAVENGA.html.

Baloukas, T. (2009). JAVENGA: JAva-Based Visualization Environment for Network and Graph Algorithms. *Computer Applications in Engineering Education*.

Babcock, P. and Marks, M. (2010). The Falling Time Cost of College: Evidence from Half a Century of Time Use Data. *Review of Economics and Statistics*.

Baecker, R. M. (1981). Sorting Out Sorting. *SIGGRAPH Video Review*, 7.

Baker, R. (2000). PILOT: An Interactive Tool for Learning and Grading. Senior Thesis.

Berque, D. (2006). An Evaluation of a Broad Deployment of Dyknow Software to Support Notetaking and Interaction Using Pen-Based Computers. *Journal of Computing Sciences in Colleges*, 21(6):204–216.

Berque, D., Johnson, D., and Jovanovic, L. (2001). Teaching Theory of Computation Using Pen-Based Computers and an Electronic Whiteboard. In *ITiCSE '01: Proceedings of the 6th Annual Conference on Innovation and Technology in Computer Science Education*, pages 169–172. Canterbury, UK, ACM Press.

Bridgeman, S., Goodrich, M., Kobourov, S., and Tamassia, R. (2000). PILOT: An Interactive Tool for Learning and Grading. In *SIGCSE '00: Proceedings of the 31st SIGCSE Technical Symposium on Computer Science Education*, pages 139–143. Austin, TX, USA, ACM Press.

Brodowski, J. (1999). Animal Animation. http://www.animal.ahrgr.de/showAnimationDetails.php3&anim=15.

Brown, M. H. (1988). Exploring Algorithms Using BALSA-II. *Computer*, 21(5):14–36.

Buckalew, C. and Porter, A. (1994). The Lecturer's Assistant. In *SIGCSE '94: Proceedings of the 25th SIGCSE Symposium on Computer Science Education*, pages 193–197. Phoenix, AZ, USA, ACM Press.

Chalidabhongse, T. H. (1996). Network Flow Applet. http://www.cs.pitt.edu/~kirk/cs1501/animations/Network.html.

Cormen, T., Leiserson, C., and Rivest, R. (1990). *Introduction to Algorithms*. MIT Press/McGraw Hill.

Crescenzi, P. (2009). ALVIE Java Application. `http://alvie.algoritmica.org/`.

Galles, D. (2006). Data Structure Visualization Java Application. `http://www.cs.usfca.edu/galles/visualization/download.html`.

Golub, E. (2004). Handwritten Slides on a TabletPC in a Discrete Mathematics Course. In *SIGCSE '04: Proceedings of the 35th SIGCSE Technical Symposium on Computer Science Education*, pages 51–55. Norfolk, VA, USA, ACM Press.

Hoebel, N. and Burrer, M. (2003). GALGO Java Applet. `http://www.informatik.fb2.fh-frankfurt.de/~hoebel/graphen/galgo/en/index.html`.

Holmes, M., Spiker, R., and Goldberg, M. (1999). ProjectLinks Java Applet. `http://links.math.rpi.edu/devmodules/graph_networking/xhtml/page14.xml`.

Hundhausen, C., Douglas, S., and Stasko, J. (2002). A Meta-Study of Algorithm Visualization Effectiveness. *Journal of Visual Languages and Computing*, 13(3):259–290.

Ikeda, K. (2004). Shortest Path Problem Java Applet. `http://www-b2.is.tokushima-u.ac.jp/~ikeda/suuri/dijkstra/Dijkstra.shtml.en`.

Karavirta, V., Korhonen, A., Malmi, L., and Stalnacke, K. (2004). MatrixPro - A Tool for On-the-Fly Demonstration of Data Structures and Algorithms. In *Proceedings of the 3rd Program Visualization Workshop*, pages 26–33. University of Warwick, UK.

Karavirta, V., Korhonen, A., Malmi, L., and Stalnacke, K. (2006). Trakla2 Java Application. `http://www.cs.hut.fi/Research/TRAKLA2/exercises/Dijkstra.html`.

Khuri, S. and Holzapfel, K. (2001). EVEGA: An Educational Visualization Environment for Graph Algorithms. In *ITiCSE '01: Proceedings of the 6th Annual Conference on Innovation and Technology in Computer Science Education*, pages 101–104. Canterbury, UK, ACM Press.

Korhonen, A., Malmi, L., and Silvasti, P. (2003). TRAKLA2: A Framework for Automatically Assessed Visual Algorithm Simulation Exercises. In *Proceedings of Kolin Kolistelut / Koli Calling - 3rd Annual Baltic Conference on Computer Science Education*, pages 48–56. Joensuu, Finland.

Kumar, A. (2004). Using Online Tutors for Learning - What do Students Think? In *Proceedings of 34th Annual Frontiers in Education Conference*. Savannah, GA, USA. Session T3C.

Kumar, A. (2005a). Generation of Problems, Answers, Grade, and Feedback - Case Study of a Fully Automated Tutor. *Journal on Educational Resources in Computing*, 5(3).

Kumar, A. (2005b). Results from the Evaluation of the Effectiveness of an Online Tutor on Expression Evaluation. In *SIGCSE '05: Proceedings of the 36th SIGCSE Technical Symposium*, pages 216–220. St. Louis, MO, USA.

Kumar, A. (2006). Explanation of Step-by-Step Execution as Feedback for Problems on Program Analysis, and its Generation in Model-Based Problem-Solving Tutors. *Journal of Technology, Instruction, Cognition, and Learning*, 4(1):65–107.

Kumar, A. (2008). Female Students Assess Software Tutors More Positively Than Male Students. In *Proceedings of Frontiers in Education Conference*. Saratoga Springs, NY, USA. Session S4F.

Kumar, A. (2009). Need to Consider Variations within Demographic Groups When Evaluating Educational Interventions. In *ITiCSE '09: Proceedings of Innovations and Technology in Computer Science Education*, pages 176–180. Paris, France.

Kumar, A. (2010). Problets website. `http://www.problets.org/about/publications.html`.

Laakso, M. and Salakoski, T. (2004). Automatic Assessment of Exercises for Algorithms and Data Structures - A Case Study with TRAKLA2. In *Proceedings of Kolin Kolistelut / Koli Calling - Fourth Finnish/Baltic Sea Conference on Computer Science Education*, pages 28–36.

Laffra, C. (1996). Dijkstra's Shortest Path Algorithm Java Applet. `http://www.dgp.toronto.edu/people/JamesStewart/270/9798s/Laffra/DijkstraApplet.html`.

Mayer, R. and Anderson, R. (1991). Animations Need Narrations: An Experimental Test of a Dual-Coding Hypothesis. *Journal of Educational Psychology*, 83(4):484–490.

Mayer, R. and Anderson, R. (1992). The Instructive Animation: Helping Students Build Connections Between Words and Pictures in Multimedia Learning. *Journal of Educational Psychology*, 84(4):444–452.

Mayer, R. and Sims, V. (1994). From Whom is a Picture Worth a Thousand Words? Extensions of a Dual-Coding Theory of Multimedia Learning. *Journal of Educational Psychology*, 86(3):389–401.

Myers, B. (2001). Using Handhelds and PCs Together. *Communications of the ACM*, 44(11):34–41.

Myers, B., Stiel, H., and Gargiulo, R. (1998). Collaboration Using Multiple PDAs Connected to a PC. In *CSCW '98: Proceedings of the 1998 ACM Conference on Computer Supported Cooperative Work*, pages 285–294. Seattle, WA, USA, ACM Press.

Myller, N., Laakso, M., and Korhonen, A. (2007). Analyzing Engagement Taxonomy in Collaborative Algorithm Visualization. In *ITiCSE '07: Proceedings of the 12th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education*, pages 251–255. Dundee, Scotland, UK, ACM Press.

Naps, T. (1998). A Java Visualiser Class: Incorporating Algorithm Visualisations into Students' Programs. In *ITiCSE '98 Proceedings of the 6th Annual Conference on the Teaching of Computing and the 3rd Annual Conference on Integrating Technology into Computer Science Education: Changing the Delivery of Computer Science Education*, pages 181–184. Dublin, Ireland, ACM Press.

Naps, T. (2005). JHAVE - Addressing the Need to Support Algorithm Visualization with Tools for Active Engagement. *IEEE Computer Graphics and Applications*, 25(6):49–55.

Narayanan, N. H. and Hegarty, M. (2002). Multimedia Design for Communication of Dynamic Information. *International Journal of Human-Computer Studies*, 57(4):279–315.

Ng, M., Ang, W., and Morris, J. (1998). Auckland Java Applet. `http://www.cs.auckland.ac.nz/software/AlgAnim/dijkstra.html`.

Olsen, D., Taufer, T., and Fails, J. (2004). ScreenCrayons: Annotating Anything. In *UIST '04: Proceedings of the 17th Annual ACM Symposium on User Interface Software and Technology*, pages 165–174. Santa Fe, NM, USA, ACM Press.

Orsega, M. (2009). *Evaluating Sketchmate: A Digital Drawing Tool for the Splay Tree Data Structure*. PhD thesis, University of Tennessee Knoxville.

Orsega, M., Vander Zanden, B., and Skinner, C. (2011). Two Experiments Using Learning Rate to Evaluate an Experimenter Developed Tool for Splay Trees. In *SIGCSE '11: Proceedings of the 42nd ACM Technical Symposium on Computer Science Education*, pages 135–140. Dallas, TX, USA, ACM Press.

Orsega, M., Vander Zanden, B., and Skinner, C. (2012). Experiments with Algorithm Visualization Tool Development. In *SIGCSE '12: Proceedings of the 43nd ACM Technical Symposium on Computer Science Education*, pages 559–564. Raleigh, NC, USA, ACM Press.

Palmiter, S. and Elkerton, J. (1991). An Evaluation of Animated Demonstrations for Learning Computer-Based Tasks. In *CHI '91: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 257–263. New Orleans, LA, USA, ACM Press.

Pane, J., Corbett, A., and John, B. (1996). Assessing Dynamics in Computer-Based Instruction. In *CHI '96: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 194–204. Vancouver, British Columbia, Canada, ACM Press.

Papagelis, A. (1997). Minimum Routes Finder Java Applet. `http://students.ceid.upatras.gr/~papagel/project/kef5_7_1.htm`.

Pierson, W. C. and Rodger, S. H. (1998). Web-Based Animation of Data Structures Using JAWAA. In *SIGCSE '98: Proceedings of the 29th SIGCSE Technical Symposium on Computer Science Education*, pages 267–271. Atlanta, GA, USA, ACM Press.

Robling, G., Schuler, M., and Freisleben, B. (2000). The ANIMAL Algorithm Animation Tool. In *ITiCSE '00: Proceedings of the 5th Annual SIGCSE/SIGCUE ITiCSE Conference on Innovation and Technology in Computer Science Education*, pages 37–40. Helsinki, Finland, ACM Press.

Shaffer, C., Cooper, M., Alon, A. J. D., Akbar, M., Stewart, M., Ponce, S., and Edwards, S. (2010). Algorithm Visualization: The State of the Field. *ACM Transactions on Computing Education*, 10(3):1–22.

SMART-Technologies (2012). SMART Board Interactive Whiteboards. `http://www.smarttech.com/SmartBoard`.

Sniedovich, M. (2000). TutORial Javascript Application. `http://www.ifors.ms.unimelb.edu.au/tutorial/dijkstra/island.html`.

Stasko, J. (1992). Animating Algorithms with XTANGO. *SIGACT News*, 23(2):67–71.

Teviotdale, R. and Naps, T. (2008). JHAVE Java Application. `http://jhave.org/learner/graphs/Dijkstra/dijkstra.php`.

Weiss, M. (2006). *Data Structures and Algorithm Analysis in C++, Third Edition*. Addison Wesley.

Wilkerson, M., Griswold, W., and Simon, B. (2005). Ubiquitous Presenter: Increasing Student Access and Control in a Digital Lecturing Environment. In *SIGCSE '05: Proceedings of the 36th Technical Symposium on Computer Science Education*, pages 116–120. St. Louis, MO, USA, ACM Press.

Yang, J., Nielsen, J., and Shaffer, C. (1996). Swan Network Flow Application. `http://research.cs.vt.edu/AVresearch/Swan/`.

# Appendix

The following list presents all the possible feedback messages a student could receive while working through a shortest path problem. The letters and numbers assigned to the vertices and edges are arbitrary. For purposes of this presentation, each message should be treated as independent of the others. The purpose is to show each possible scenario of incorrectly updating the graphs.

- Vertex v should be marked as Visited since it is the start vertex.
- Vertex v should be marked as Visited since it is the seen vertex with the smallest cost.
- Vertices x,y should be marked as Seen, Unvisited since they are neighbors of the visited vertex v.
- Vertices u,w should still be Unseen, Unvisited since they are not neighbors of a previously visited vertex.
- Vertex v should still be marked as Visited and should not have been changed.
- Vertices x,y should still be marked as Seen, Unvisited and should not have been changed.
- Vertex v should have a cost of 0 since it is the start vertex.
- Vertex v should have a cost of 10 and should not have been changed.
- Vertex v should have a cost of 10 and edge u-v should now be selected.
- Vertex v should have a cost of 8 since going through u is on a shorter path (8) than going through w (10).
- Vertex v should have a cost of 8 since going through u is on a shorter path (8) than going through w (10) and edge u-v should now be selected.
- Vertex v should have a cost of 8 since going through u is on a shorter path (8) than going through w (10) and edge w-v should no longer be selected.
- Vertex v should have a cost of 8 since going through u is on a shorter path (8) than going through w (10) and edge u-v should now be selected and edge w-v should no longer be selected.
- Edge u-v should now be selected since a shorter path to v was found through u.
- Edge u-v should now be selected and edge w-v should no longer be selected since a shorter path to v was found through u.
- Edge w-v should no longer be selected since a shorter path to v was found through u.
- Edge x-u should not have been deselected since it is still on the shortest path.

- Edge y-z should not have been selected since it is not on the shortest path.

The following list presents all the possible feedback messages a student could receive while working through a network flow problem. The letters and numbers assigned to the edges are arbitrary. For purposes of this presentation, each message should be treated as independent of the others. The purpose is to show each possible scenario of incorrectly updating the graphs.

- Edges a-b, b-c should be selected since they are on the maximum flow augmenting path, which has flow 6.
- Edges c-d, d-e should not be selected since they are not on the maximum flow augmenting path, which has flow 6.
- Edge a-b should have a flow of 8 since 6 is the flow along the augmenting path which got added to its previous flow of 2.
- Edge b-c should have a flow of 6 since 6 is the flow along the augmenting path.
- Edges a-b, b-c should have a flow of 2 since the corresponding backedges in the residual graph are on the augmenting path, therefore flow needs to be subtracted.
- Edges c-d, d-e should have a flow of 4 and should not have been changed since they (and their backedges) are not on the augmenting path.
- Edge a-b should have a capacity of 8 since it replaced its backedge and now carries the flow 6.
- Edge b-c should have a capacity of 6 since it replaced its backedge and now carries all the flow.
- Edge c-d should have a capacity of 4 and should not have been changed since it is not on the augmenting path.
- Edge a-b should have capacity 2 since it is reduced by the flow 6 because it is on the augmenting path.
- Edge b-c should have capacity 8 since the flow 6 is added to it because it is a backedge to an edge on the augmenting path.
- Edge a-b should have capacity 6 since 6 is the flow along the augmenting path.
- Edge d-c should not be in the residual graph since it and its backedge are not on the augmenting path.
- Edge a-b should be deleted from the residual graph since it got replaced by its backedge.

- Edge b-c should be deleted from the residual graph since its backedge now carries all of the flow.
- Edge a-b should be replaced with its backedge b-a and its capacity should be the flow 6.
- Edge c-d should not have been reversed and should've been left alone since it is not on the augmenting path and its capacity should be 4.
- Edge b-a should now be in the residual graph since it is a backedge that replaces a-b and its capacity should be the flow 6.
- Edge c-d should not have been deleted since it is not on the augmenting path and its capacity should be 4.
- Edge b-a should be added as a backedge to a-b and its capacity should be the flow 6 and the capacity of a-b should be reduced to 2.
- Edge d-e should not have been removed and its capacity should be 4.

# Vita

Kristy Sue Van Hornweder was born on November 4, 1977 in Superior, Wisconsin. She earned a Bachelor's degree in Computer Science and Mathematics from the University of Minnesota-Duluth in 2000. In 2002, she earned a Master's degree in Computer Science, and in 2004, she earned a Master's degree in Applied and Computational Mathematics, both from the University of Minnesota-Duluth. She worked as an adjunct instructor in Mathematics at the College of St. Scholastica in Duluth, Minnesota in the fall of 2005. In 2006, Kristy began work towards her PhD in Computer Science at the University of Tennessee-Knoxville. She completed this degree in 2012, and has accepted a one-year Visiting Lecturer position in Computer Science at the University of Virginia in Charlottesville. She would like to eventually obtain a position as a tenured professor in Computer Science at a teaching university or college in Western North Carolina or Eastern Tennessee.