11-2012

BeSocratic: An Intelligent Tutoring System for the Recognition, Evaluation, and Analysis of Free-form Student Input

Samuel Bryfczynski
Clemson University, sbryfcz@gmail.com

BeSocratic: An Intelligent Tutoring System for the Recognition, Evaluation, and Analysis of Free-form Student Input

A Dissertation
Presented to
the Graduate School of
Clemson University

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy
Computer Science

by
Samuel Paul Bryfczynski
November 2012

Accepted by:
Dr. Roy P. Pargas, Committee Chair
Dr. Brian Dean
Dr. Murali Sitaraman
Dr. Melanie Cooper

Abstract

This dissertation describes a novel intelligent tutoring system, BeSocratic, which aims to help fill the gap between simple multiple-choice systems and free-response systems. BeSocratic focuses on targeting questions that are free-form in nature yet defined to the point which allows for automatic evaluation and analysis. The system includes a set of modules which provide instructors with tools to assess student performance. Beyond text boxes and multiple-choice questions, BeSocratic contains several modules that recognize, evaluate, provide feedback, and analyze student-drawn structures, including Euclidean graphs, chemistry molecules, computer science graphs, and simple drawings. Our system uses a visual, rule-based authoring system which enables the creation of activities for use within science, technology, engineering, and mathematics classrooms.

BeSocratic records each action that students make within the system. Using a set of post-analysis tools, teachers have the ability to examine both individual and group performances. We accomplish this using hidden Markov model-based clustering techniques and visualizations. These visualizations can help teachers quickly identify common strategies and errors for large groups of students. Furthermore, analysis results can be used directly to improve activities through advanced detection of student errors and refined feedback.

BeSocratic activities have been created and tested at several universities. We report specific results from several activities, and discuss how BeSocratic's analysis tools are being used with data from other systems. We specifically detail two chemistry activities and one computer science activity: (1) an activity focused on improving mechanism use, (2) an activity which assesses student understanding of Gibbs energy, and (3) an activity which teaches students the fundamentals of splay trees. In addition to analyzing data collected from students within BeSocratic, we share our visualizations and results from analyzing data gathered with another educational system, PhET.

Dedication

I dedicate my dissertation work to all of my family and friends who have supported me throughtout my research. Foremost, I wish to thank my parents, Paul and Sheila, for always supporting me. From an early age, I was interested in building things and working with technology. My parents played a crucial role in developing and shaping my life's ambition.

A special note of dedication goes to my high school computer science teacher, Richard Gesick. Mr. Gesick was the first one to inspire me to become a computer scientist. From the first time I saw the words "Hello World" appear on the screen in his introductory programming class, I knew my desired career path. He also was there to push and challenge me to become a better developer.

I would also like to thank all of my friends who for 5 years asked me "Are you ever going to graduate?" That alone motivated me to continue and finish my research, so I could answer with a resounding "YES!"

Thank You All.

Acknowledgments

I am very thankful for the help and support of so many people during my time at Clemson. First and foremost, I would like to thank my advisor, Dr. Roy Pargas. From the first days in his undergraduate class, Dr. Pargas was always willing to lend me a helping hand. He has taught me a tremendous amount on how to conduct research and collaborate with others. Next, I would like to acknowledge and thank Dr. Melanie M. Cooper who has shaped me into a well-rounded researcher and academic. Third, I wish to thank my committee members, Dr. Brian Dean and Dr. Murali Sitaramin for the help they provided as I conducted research. Both were always more than willing to open their doors, sit down, and discuss research throughout my studies.

I would like to thank all of my fellow group members. I feel a great deal of gratitude to Nathan, Sonia, Evy, Leah, Barbara, Kaleb, and Josiah. You all made this research a much more enjoyable experience.

I would like to thank all of our collaborators. I'd especially like to acknowledge Mike Klymkowsky from the University of Colorado for all the support and motivation. I'd also like to acknowledge the PhET research group at the University of Colorado, especially Julia Chamberlain and Kelly Lancaster. Their openness to share data and work together led to many improvements and insights that I could not have come to alone.

Table of Contents

List of Tables

List of Figures

Listings

Chapter 1

Introduction

Since the early days of personal computing, software has been developed for educational purposes. The number of such applications continues to increase, and the sophistication of the systems is constantly evolving. There exists a wide spectrum of educational software ranging from Pre-K to industry.

1.1 Problem Statement

Because of the importance of education, many funding agencies support research efforts to develop useful educational software for both teachers and students. One of the primary agencies, the National Science Foundation, is a part of the United States government and funds a large number of these efforts [1]. Furthermore, there is an ever-increasing number of journals and conferences aimed at the exploration and feasibility of using pedagogical software at various learning levels and for a diverse set of disciplines [2]. While this field has been around for decades, new pedagogy techniques and discoveries fuel continuing research.

Today, most higher education institutions use broad learning management systems such as Blackboard, Moodle, or Instructure Canvas to aid in assessment. Additionally, specialized systems (such as the Mastering software series, OWL, etc.) exist for individual disciples and courses. A subset of these systems called intelligent tutoring systems exist (e.g., MathTutor, CogTutor) to provide students with step-by-step guidance during problem-solving. While these systems have been shown to enhance student learning in a range of domains[44, 49, 47], they tend to be difficult

to author, and the majority of questions they can ask fall into one of two categories: free-response text-based questions or multiple-choice/matching questions. Free-response systems allow teachers to ask meaningful questions that require students to have a deep understanding of the subject in order to answer correctly. Multiple choice and matching questions are more restrictive by nature, and research has suggested that these questions cannot be used to properly assess deep knowledge on a subject since the exercises often only involve memorization [33, 32, 64, 60].

Which system a teacher uses depends on the amount of time available to evaluate student solutions. Free response systems require teachers to manually check each submission; this is too time-consuming for teachers to perform on a regular basis. Instead, teachers rely on using multiple-choice or matching questions, which can be quickly or automatically evaluated. The ideal system combines the best parts of both question types. A teacher would be able pose questions that require students to have a deep understanding of the material and reply in an intuitive free-form manner. Moreover, student responses would be automatically evaluated and analyzed.

These challenges lead us to ask the following questions: Can we recognize and give feedback for free-form drawings? Can we provide teachers with an intuitive authoring tool for creating intelligent tutors? Can we record large sets of student data while they complete activities? Can we automatically analyze and visualize large sets of student data so instructors may quickly gain insights into student strategies? And, can we use analysis results to refine and improve activities? These are the questions that our work aims to address.

1.2 Research Approach and Expected Contributions

Our approach to handling these problems is divided into two parts. First, we developed an intuitive interface that allows the authoring of intelligent tutors. These tutors are capable of recognizing various types of free-form student input and provide students with multi-tiered feedback based on their solutions. Secondly, we developed an analysis tool, which allows teachers and researchers to analyze recordings of student actions generated within our system or imported from external systems. Upon analysis, reports and visualizations are generated, which may be used to understand the cognitive processes of students while they complete their work. Furthermore, the analysis results may be used to improve the tutoring activities for future students.

1.2.1 Contributions

This research makes the following contributions:

Contribution 1 - *Free-Form Tutor Authoring Tool.* We believe the majority of current intelligent tutors rely on overly simplistic forms of input from students (i.e., multiple choice questions). This is the case because the evaluation of such input is trivial; however, it is also common for students to find ways to cheat these systems or simply guess their way to the right answer [37]. In addition, a student may become bored or fatigued over the course of a session and deliberately enter incorrect answers in order to elicit the correct answer from an intelligent tutoring system[8]. There is also a growing interest in developing systems with more natural interfaces for students [28, 23, 58, 69]; however, we believe using these systems to author tutors is often too complicated. Furthermore, many of these systems require students to provide extra data, besides their drawings, to the system in order for their work to be analyzed. Adding this additional information increases the student's cognitive load thus making it more difficult for teachers to evaluate a student's understanding of a topic.

We developed the first online intelligent tutoring system that automatically recognizes, evaluates, and provides feedback to graph-based questions. We chose to focus on axis-based graphs because (1) graphs frequently appear throughout science, technology, engineering, and mathematics (STEM) curricuculums and because (2) their free-form nature makes drawing graphs difficult for students to guess the correct answer. Our authoring tool allows teachers to quickly create intelligent tutors in an intuitive and visual way. Teachers set constraints for the desired characteristics they seek from a *correct* graph as well as feedback to present students with when these constraints are not met. Additionally, feedback may be in the form of follow-up questions, links to outside web pages with additional information, or other activities entirely. In addition to mathematical graphs, our system is able to evaluate and analyze student-drawn chemistry molecules, computer science graphs, and simple free-form drawings. Activities created with our system can be used in a variety of disciplines and preliminary results show learning gains for certain activities.

The research questions associated with this contribution include: Is it possible to create an intelligent tutoring system to recognize and evaluate graph drawings? Can we build the system with minimal interface overhead for students to answer the questions in an intuitive manner? Can we build an intuitive authoring tool in order to quickly create constraint-based intelligent tutors?

Contribution 2 - *Student Data Analysis Tools.* One of the main advantages of an intelligent tutoring system (ITS) is the removal of a human teacher. This is especially significant when the ITS is used with large group sizes and individual attention from teachers is unrealistic due to time-constraints. Once a tutor is authored, ITSs are able to provide automatic and personalized instruction to each student without a need for a teacher. However, we feel that there is a deficiency in the types of analysis these systems provide teachers and educational researchers. The majority of intelligent tutoring systems can only generate summative reports with charts and tables based on the frequencies and percentages of student actions. While these reports are useful for determining a general picture of student activity, they fail to provide a sufficient method for in-depth analysis of student reasoning. A more robust tool would allow teachers to identify groups of students who performed similar actions within the system. There have been several research efforts in this area, but they have not yet been integrated into intelligent tutoring systems. Moreover, some of these analysis efforts require the inclusion of external data about students (e.g., test grades) [66, 41].

Our tools are designed to address these challenges. There are two sets of tools provided by our system. One set provides the instructor with the ability analyze a single student's performance. A second set provides the instructor with the ability to analyze the performance of an entire group or class of students.

Our intelligent tutoring system records the actions of students in the system. This generates a large set of student data, especially when used to record the actions of large classrooms of students (>50 students). The tools described use a new set of analysis techniques, which enable the analysis of these data sets. Using our system to collect and replay student work from various activities within a semester, teachers are able to track individual students and understand the source of their errors. Furthermore, our system contains the ability to analyze groups of students using cluster analysis to quickly give teachers a summary of a class' performance and discover common student strategies (i.e., identify groups of students with similar solution processes). With these two abilities, teachers have the ability to provide further instruction to students who are struggling with concepts, and the results of the analysis can be used to improve the tutor for further use with students. Our system is also capable of importing and analyzing student sequence data from other learning systems.

The research questions targeted with this contribution include: Can a free-form system record student work with high enough fidelity to accurately replay them? Is it possible for a system to analyze and visualize sequential student data in such a way as to assist teachers and researchers

4

in identifying student strategies? How can clustering results be used to improve feedback for future groups of students? And, can these techniques be abstracted and used with student data from other systems?

1.3 Thesis Statement

This dissertation defends the following 3-part statement. It is possible to devise a system that

1. enables teachers to create intelligent tutors, which are able to recognize, evaluate, and provide feedback to student drawings, including Euclidean graphs, computer science graphs, chemistry molecules, and simple free-form drawings.

2. allows teachers and researchers to track individual students and identify concepts in which students needs help.

3. allows teachers and researchers to analyze large groups of student sequence data and generate informative reports and visualizations that can be used to gain insights into a class' knowledge and improve future activities.

1.4 Dissertation Organization

The remainder of the dissertation is organized as follows. Chapter 2 presents background material related to intelligent tutoring systems and analysis techniques. Chapter 3 describes the related work completed to date. Chapter 4 discusses our intelligent tutoring system, BeSocratic, with a focus on how it can be used to collect student work. Chapter 5 describes how BeSocratic's post-analysis tools may be used to analyze and visualize student data. Chapter 6 discusses the results obtained using our system including its classroom use, learning gains achieved with the system, analysis results from select activities, and analysis of data outside of the system. Chapter 7 shares our ideas for future extensions and work with BeSocratic. Finally, Chapter 8 concludes with a summary of our thesis and contributions.

Chapter 2

Background

What is an intelligent tutoring system?Intelligent tutoring system (ITS) is a broad term that encompasses any computer system that contains a level of intelligence and can be used in learning environments [25]. ITSs were developed as an extension to earlier computer-aided instruction systems, which usually refers to frame-based systems with hard-coded links, i.e. hypertext with an instructional purpose [29].

Traditional ITSs contain four components: the domain model, the student model, the teaching model, and a learning environment or user interface. ITS projects usually involve the creation of an authoring tool that may focus on these components in a variety of different ways [53]. For example, an ITS that focuses on a specific domain may be able to generate an endless supply of a complex and novel problems in order for students to continuously practice; however, a different system that concentrates on multiple or novel ways to teach a particular topic might find a less sophisticated representation of that content sufficient.

ITSs can also be classified by their underlying evaluation method. One well-known category is model-tracing tutors [13], which track students progress and keeps them within a specified tolerance of an acceptable solution path. Another category is the constraint-based systems which instead of modeling the students thought process, simply require the author to specify feedback for various student actions [52]. Although building tutors using constraints makes the authoring process easier [51, 56], building a knowledge base for a constraint-based tutor still remains a major challenge and some have estimated that creating one hour of instructional content takes 200-1000 hours [72, 53]. Because of this, ITSs are not commonly integrated within classrooms despite showing impressive

learning outcomes [44, 49, 47]. We plan to address this challenge in several ways. Our constraint-based system provides many visual cues during authoring to ease the development of tutors. In addition, our system contains a model-tracing component that teachers may take advantage of to detect student strategies and respond appropriately.

What is Levenstein distance? The Levenstein distance is a string metric for measuring the amount of difference between two sequences [48]. Usually, this metric is defined as the minimum cost of "edits" needed to transform one string into another. The edits can come in a variety of forms including: insertions, deletions, or substitutions of characters. Associated with each type of edit is a cost value. Using dynamic programming, the edit distance can be computed relatively efficiently [63]. Formally stated:

Input: Strings $A[1..n]$ and $B[1..m]$

Costs: Insertion Cost (Ci), Deletion Cost (Cd), Replacement Cost(Cr)

$D[i, j]$: minimum cost to transform $A[1..i]$ into $B[1..j]$.

$$D[i,j] = \begin{cases} D[i-1, j-1] & \text{if } A[i] = B[f] \\ min \begin{cases} D[i-1, j-1] + Cr & \text{for a replacement} \\ D[i-1, j] + Cd & \text{for a deletion} \\ D[i, j-1] + Ci & \text{for an insertion} \end{cases} & \text{otherwise} \end{cases}$$

The Levenstein distance and the equivalent NeedlemanWunsch algorithm are often used in bioinformatics to compare protein and DNA sequences [55]. By finding similarities and differences within these structures, biology researchers can identify relationships within and among species and well as understand interactions between proteins. We have explored the possibility of using a modified Levenstein distance to find similarities within sequences of timestamped student actions recorded within an intelligent tutoring system.

What is a Markov model? A Markov model is a stochastic model that assumes the Markov property (i.e., the conditional probability distribution of future states of the process depends only upon the present state, not on the sequence of events that preceded it) [35]. Markov models are often described in terms of states and transitions. States are often represented as nodes in a graph, and transitions are represented with edges between nodes. Because a Markov model is a stochastic
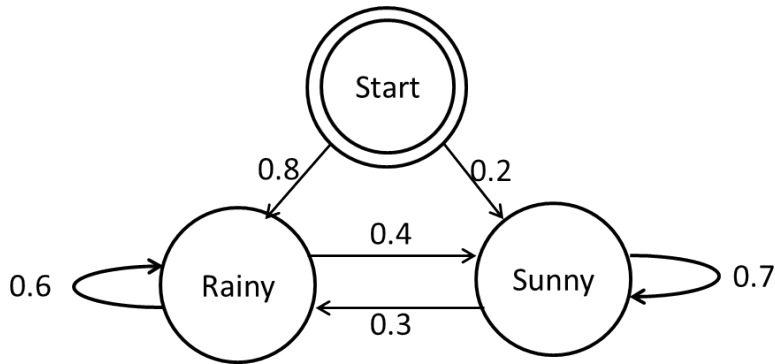
Figure 2.1: An example Markov model of day to day weather patterns. Edges represent transition probabilities between weather types.

model, transition edges are weighted with the probability of moving from one state to another and the sum of transitions from each individual state always equals 1. An example of a Markov model is shown in Figure 2.1. This is a simple example of a model of day-to-day weather patterns (i.e., sunny or rainy). In the example, the model contains a starting state with the overall probability of the day being sunny or rainy. The transitions among state represent the probability of the weather changing from one day to the next. Notice that the outgoing edges (probabilities) sum to 1.

What is a hidden Markov model? Hidden Markov models (HMMs) are an extension of Markov models developed by L. E. Baum and collaborators in the late 1960s [10, 9, 11]. Hidden Markov models differ from Markov models with respect to the visibility of the states. Whereas in Markov models the state is directly visible to the observer, the states are not directly observable within a hidden Markov model. That is to say, the states are hidden from the observer. Instead, a set of outputs are visible and are dependent on the states. Hidden Markov models have a variety of uses in a wide range of disciples such as speech recognition, handwriting recognition, and bioinformatics to name a few [59]. An example of a hidden Markov model for clothing patterns based on weather is shown in Figure 2.2. In this example, notice how the model not only contains the states and transitions present in a Markov model but also three observations: t-shirt, sweatshirt, and jacket. The observation probabilities of this example reflect the weather of the day (e.g., on sunny days you are more likely to wear a t-shirt than a sweatshirt or jacket).

Hidden Markov models are commonly used to analyze sequence data. By sequence data, we mean to say data that is represented by a sequence of actions or items where the order of the actions is important. Examples of sequence data include audio recordings and DNA sequences.

Hidden Markov models are often used to cluster and/or classify these types of sequences. In these scenarios, the sequence data is treated as an observation sequence and a hidden Markov model is used where the number of states, meaning of the states, state transitions, and emission probabilities are unknown. The only parameters that are usually known are the sequences of observations. Using these sequences alone and making some estimation of the number of states, it is common to desire the answers to the following three questions:

1. What is the probability of the observation sequence given the model (i.e., $P(observation sequence|model)$)?

2. What is the single most likely state sequence that best explains the observation sequence?

3. What is the set of model parameters that maximize $P(observation sequence|model)$?

Rabiner describes different approaches to answering these three fundamental questions [59]. The first question can be answered efficiently using the forward algorithm. The second question about finding the most likely sequence to generate the observation sequence can be solved efficiently using dynamic programming and the Viterbi algorithm. Finally and perhaps most interesting is the third question, which asks for a way to optimally set the parameters of a hidden Markov model (i.e., initial probabilities, state transitions and emission probabilities) to maximize the probability of the observation sequence being generated by the model. Perhaps unsurprisingly, finding a globally optimal set of parameters is NP hard. While typical optimization techniques could be used (e.g., simulated annealing or genetic algorithms), an instance of the expectation maximization algorithm known as the Baum-Welch algorithm is typically used to quickly find locally optimal solutions for hidden Markov models. Because of how quickly this algorithm tends to converge, it is often restarted multiple times with the same observation sequence and the best set of parameters is used as the solution.

What is spectral clustering? Spectral clustering is a family of algorithms that can be used to perform cluster analysis on sets of observations. Compared to more "traditional" clustering algorithms such as *k-means*, spectral clustering has many fundamental advantages [71]. At the core of spectral clustering algorithms is a similarity matrix. The similarity matrix contians pair-wise similarity values between data points. Typical similarity measures include Euclidean distance, cosine similarity, Manhattan distance, Mahalanobis distance, and Levenstein distance. Spectral clustering algorithms make use of the spectrum of the similarity matrix to perform dimensionality reduction
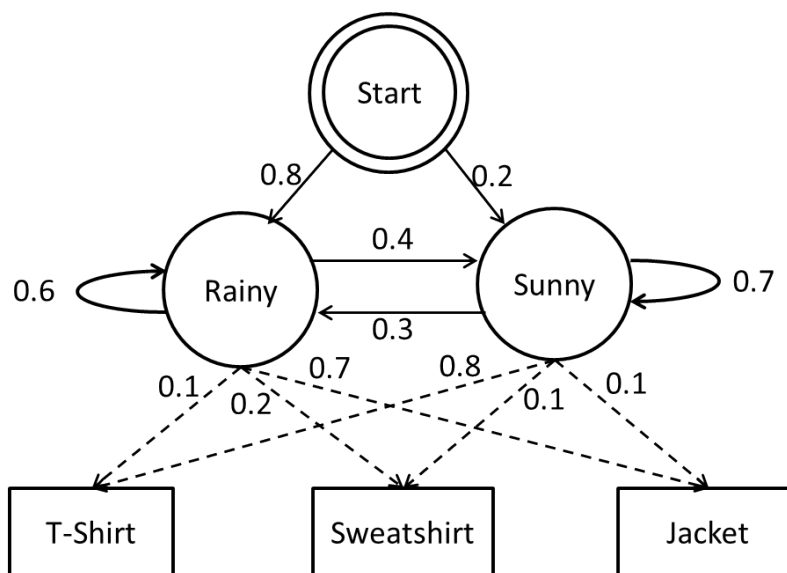
Figure 2.2: An example hidden Markov model of day to day weather patterns and the clothes one would wear based on the weather. Notice that both the state transitions and observations probabilities sum to 1.

for clustering in fewer dimensions. This process can be thought of in terms of graph partitioning, i.e., we would like to partition a graph of our observations (represented by the similarity matrix) such that the edges between different groups of nodes have a very low weight (which means that points in different clusters are dissimilar from each other) and the edges within a group have higher weight (which means that points within the same cluster are similar to each other). This could be solved if we could solve the *mincut* problem; however, the *mincut* problem can be difficult to compute for multiple clusters and has a tendency to simply separate a very small group of nodes from the graph. Instead, spectral methods typically relax the *mincut* problem and approximate the solution using the related *ncut* [65] or *ratiocut* [38] problems. These problems may be relaxed and approximated efficiently using the eigenvector corresponding to the second smallest eigenvalue of the similarity matrix's Laplacian matrix. This separates the graph and thus the data into two groups. Using this process of separating the data with eigenvector/eigenvalue pairs (e.g., second smallest, third smallest, fourth smallest, etc.), the data can be reduced in dimensionality. After which, *k-means* or hierarchical methods are typically used to cluster the resulting points into the desired number of clusters.

Chapter 3

Related Work

3.1 Intelligent Tutoring Systems

We have chosen to focus our research on free-form input types for several reasons. Part of the reason is that we feel, as others do, that it is important for students to construct structures and ideas instead of simply memorizing them. Currently, there is a division between supporters of constructivism and those of direct instruction [62]. Most researchers in education and cognitive science believe that knowledge is ultimately constructed by students. However, there is disagreement among researchers as to the best method to promote this process. Researchers in support of constructivism believe that knowledge is generated from the interaction between experience and thought, and thus students should be taught with personalized inquiry-based instruction. Direct instruction supporters believe that the best way to promote student learning is by simply telling students what to do and how to do it. And while most educators support constructivism, data in favor of constructivism is difficult to acquire [50]. This difficulty comes partly from the fact that constructivism activities and knowledge are assessed using non-constructivist methods. That is, current assessments simply ask students to retrieve facts and complete problems that the students have seen previously. It would be better if assessments could evaluate students on their ability to use learned skills in creative and novel ways, which test a student's true understanding of the material.

Ideally, according to constructivism, each student would have individualized Socratic learning and assessment activities that probe the student with questions and feedback. Practical limitations make these types of assessments difficult to conduct. Personalized instruction and assessment

at the levels proposed by constructivists are not feasible in large classrooms with hundreds of students.

New technologies, however, are starting to bridge this gap and allow what was previously considered to be infeasible to now possibly be practical. As previously stated, most higher education institutions today use broad learning management systems such as Blackboard, Moodle, or Instructure Canvas to aid in assessment. Additionally, specialized systems (such as the Mastering software series, OWL, etc.) exist for individual disciples and courses. While these systems have shown improvements in student learning, the majority of questions they may ask fall into one of two categories: free response text-based questions or multiple-choice/matching questions. Free-response systems allow teachers to ask meaningful questions that require students to have a deep understanding of the subject in order to answer correctly. More restrictive questions use simple multiple choice or matching questions. Research has suggested that multiple choice or matching questions cannot be used to properly assess deep knowledge on a subject since the exercises often only involve memorization.

There have been attempts at creating software that blends these two models and evaluate free-form drawings, but they are usually nothing more than multiple-choice assessments with decorative covers. Students are often clever enough to recognize patterns in the system and ultimately are able to "game" the system. For example, there are reports [37] showing that systems that generate exercises using random number generators are often worked around by students who quickly realize they can program the formula into their calculators (in the same way the teacher programmed it into the testing system).

It seems that we must change the types of questions we ask students. Instead of (randomized) multiple choice or free-response questions, we believe that visualizations may hold the key to assessment. Here, we use the term visualization the way Tufte uses it, that is, as the systematic and focused display of information in the form of tables, graphs and diagrams [70]. Many STEM (Science, Technology, Engineering, and Mathematics) disciplines use visualizations throughout their courses. Examples include drawing graphs in mathematics, free-body diagrams in physics, and energy curves in chemistry.

Teachers often need to draw graphs, diagrams, and other visualizations to express ideas. It has been argued that visualization is central to student learning in science classrooms [30, 31]. It seems that students need to watch scientists and mathematicians explain topics using visualizations

12

in order for the students to gain an understanding of material and processes. Research suggests that many students can improve their problem solving ability by learning to switch between these different visual representations of material [37]. Moreover, there is evidence that when students actually create these visualizations their problem solving skills improve. Research suggests that using visualizations to complete problems shows that students are becoming experienced problem solvers [50].

Motivated by all of this research, we have developed a novel pedagogical system, BeSocratic, which focuses on evaluating visual representations to give students individualized and meaningful instruction and assessment. BeSocratic recognizes free-form student input and provides students with meaningful feedback to improve their problem solving ability. Furthermore, BeSocratic analyzes collected student data in a variety of ways allowing teachers to better manage and track progress within their classrooms. There have been other software applications designed for similar uses, and in this section, we will describe the most similar ones as well as how BeSocratic differs from them.

3.1.1 CogSketch

CogSketch is a Tablet PC application that attempts to recognize users' free-form drawings and interpret them in a meaningful way [28]. To understand the user's drawings, CogSketch uses the visual and spatial properties of the drawing along with artificial intelligence algorithms. CogSketch is being developed by the Spatial Intelligence and Learning Center, a National Science Foundation Sciences of Learning Center. (grant # SBE0541957)

In CogSketch, students draw sketches composed of glyphs. These glyphs are composed of one of more ink strokes and content. The content is a symbolic representation of the glyph, which can either be selected from a concept list or entered directly by the student. Furthermore, students can provide additional information to the system to indicate relationships between sketches. By analyzing the visual relationships (e.g., disconnected, edge-connected, partially-overlapping) along with the content of the glyphs, CogSketch is able to infer the spatial relationships between the contents of those glyphs.

CogSketch has been tested on a variety of simulation data. The simulation data includes geometric analogies and spatial language learning [27, 45]. In both of these examples, CogSketch generates simulated student data based on research in the cognitive sciences.

In its current state, CogSketch is admittedly built more as an artificial intelligence tool than
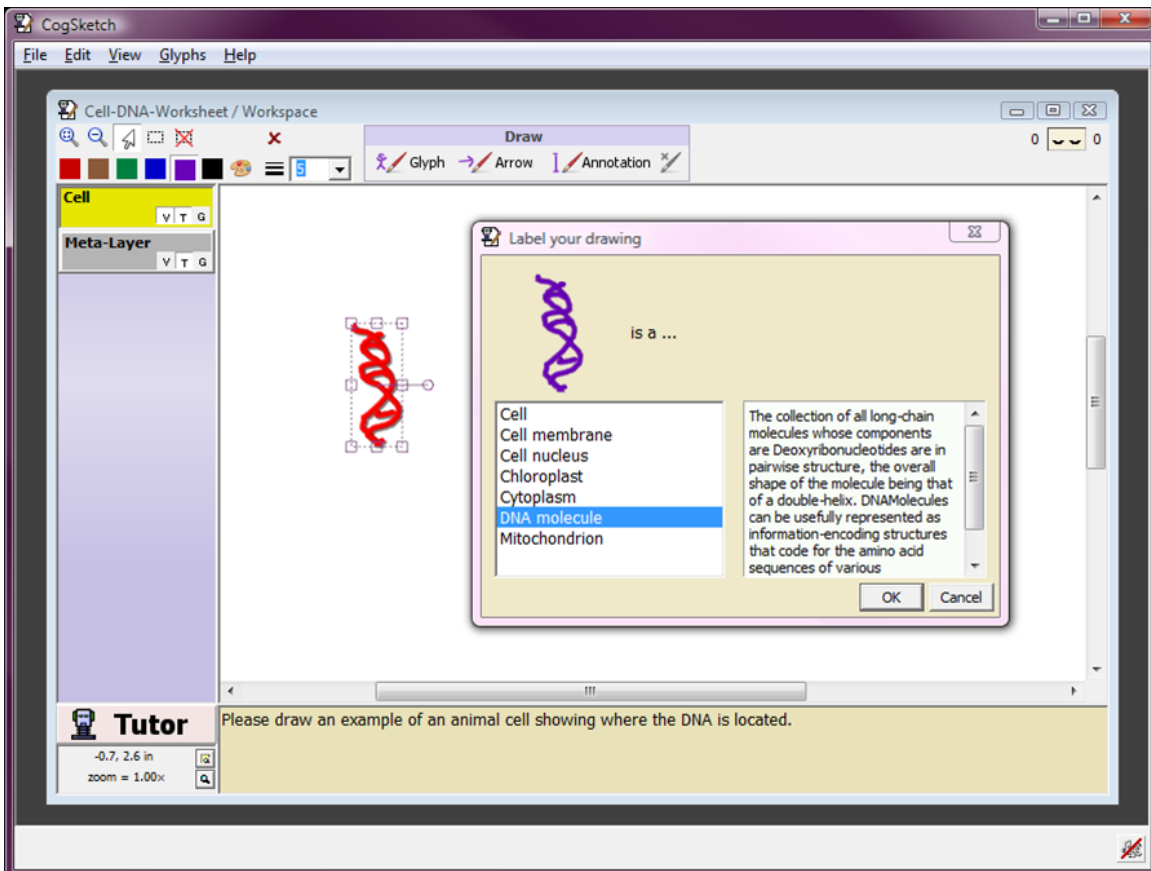
Figure 3.1: Student labeling a Glyph in CogSketch.

as an educational piece of software. However, the researchers behind CogSketch are beginning to build worksheets in which students are tasked with sketching a concept. The promise is that, with the student sketches, CogSketch will be able to make inferences about the relationships within the sketches and provide feedback depending on the variations with the teacher's solution. Entering the correct solution is daunting for most instructors since it requires an understanding of the conventions within their concept list.

BeSocratic differs from CogSketch in several ways. For one, one of the features of BeSocratic focuses on evaluating the characteristics of axis-based graphs and structured node-based graphs instead of the spatial relationships between generic sketches. This allows BeSocratic to use a small set of well-defined properties such as slope, area, minimums, and maximums to compare and evaluate user-drawn axis-based graphs. Using relatively small sets of characteristics such as this, we believe that it is easier for teachers to create tutors than the contrastingly large concept database and relationships that need to be understood before instructors may create an activity within CogSketch. Furthermore, we believe the comparatively simpler interface for students allows us to gain a clearer picture of students as they complete tasks since they are not burdened with a complex interface.

3.1.2 CTAT

Cognitive Tutor Authoring Tools (CTAT) is authoring tool for intelligent tutors that focused on building graphs that represent the problem space [6]. The creation of cognitive tutors usually involves the matching of actual student behavior during problem solving with the desired behavior represented by rules within the model and provide feedback based on deviation from the rules. This is often difficult in open-ended environments because of the numerous ways a student may compete a problem. CTAT aims to circumvent this problem by leveraging the principle of example-tracing tutors, which are developed by demonstration rather than by writing complex rules [6, 39]. That is, example-tracing tutors are developed using "programming by demonstration", an approach that allows authors with no programming skills to build tutors.

This process starts with the creation of a CTAT problem using a set of CTAT widgets. These widgets include textboxs and multiple choice questions that have been augmented for use with a discipline-specific tutoring system. For example, CTAT contains a textbox widget that is able to recognize, parse, and evaluate mathematic formulas. Next, the tutor author demonstrates correct and incorrect actions for the problem. CTAT records and analyzes these action paths to solutions.

A graph is then built from these paths in which nodes represent the state of the interface after an action and edges correspond to changes within the interface. After the behavior graph is created, the author can annotate the graph by selecting erroneous nodes, edges, or paths with feedback messages. Now when students are completing the problem, CTAT is able to compare a student's path with the underlying behavior graph and provide feedback if similarities and annotations are present. Furthermore, CTAT is able to feed the student data back into the system where it may be annotated and used in future classes with improved results. While CTAT contains its own widgets, it can also be used as the foundation for additional tutoring systems.

We have explored the concept of building graphs based on student responses with previous research. In our experience, these graphs became excessively large when used to model even relatively simple students responses. The graphs could easily contain over 200 nodes. Because of this, we feel that annotating the graph with feedback information is too time-consuming and difficult for teachers and researchers who are authoring the tutors. Instead, BeSocratic focuses on using constraint-based rule systems, which make it relatively easy for teachers to create activities.

3.1.3 CogTutor

CogTutor is one such a tutoring system developed by CMU, which leverages the CTAT system [39, 67]. CogTutor seeks to determine whether a cognitive tutoring approach can support and improve knowledge within collaboration environments. This includes the discovery of the extent to which cognitive tutors can evaluate groups of students in collaboration, the problem types that are amenable to a cognitive tutor, and the variety and frequency in which feedback should be given.

CogTutor relies heavily, if not exclusively, on student generated data to create the tutor. Using CTAT, CogTutor records student solutions to a problem and identifies them manually or, in some cases, automatically as correct or incorrect. Using CTAT, CogSketch composes a graph of the problem space that students used in creating their answer. In addition, frequencies are placed along the nodes and edges of the graph to indicate the commonality among the answers. The tutor author manually updates the graph by marking erroneous paths and adding feedback for the paths for future student use.

Of note, the use of novice data in this manner can help avoid the so-called "expert blind spot" problem in which experienced problem-solvers and teachers fail to identify the common errors of novice students [54]. These blind spots lead to students receiving inappropriate feedback or not
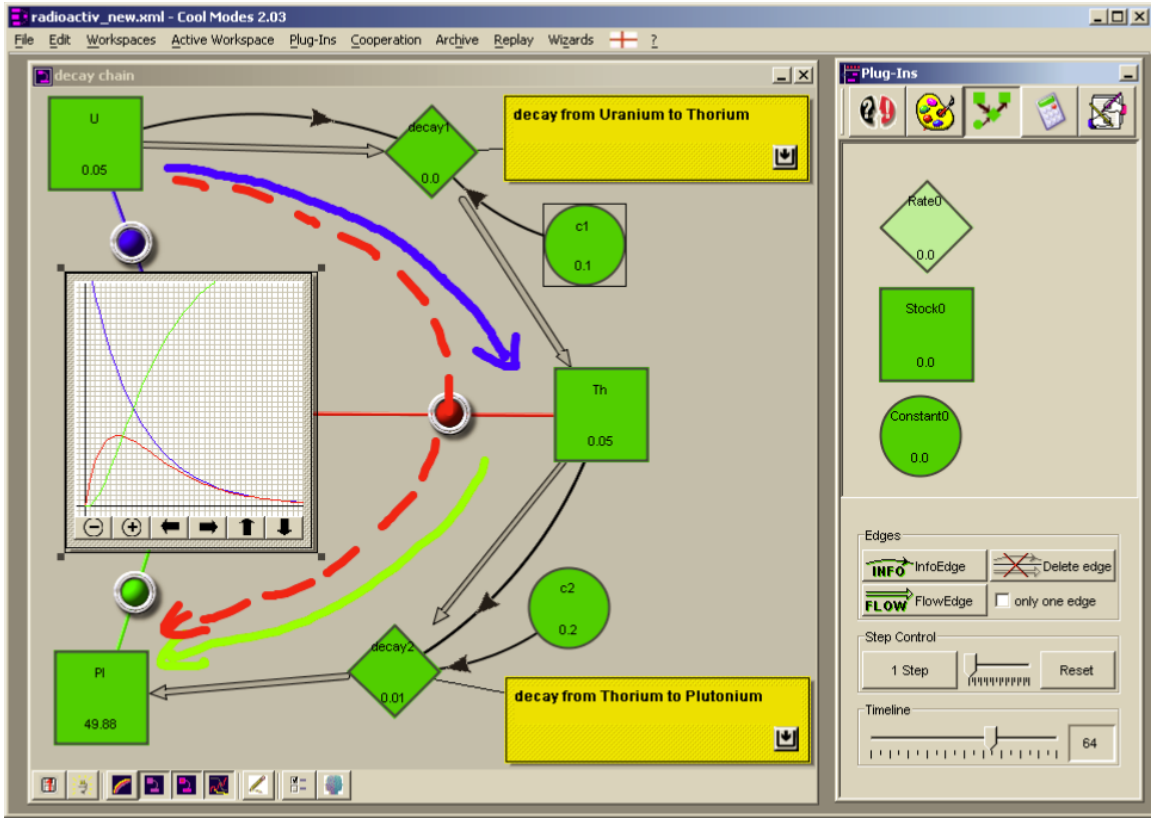
Figure 3.2: Student collaborating with others to answer a question in CogTutor.

receiving feedback when appropriate. By using actual student data, teachers are more likely to identify all of the areas where a student may go astray within the system.

In addition, CogTutor has been used to analyze sequences of student data. In one study, CogTutor was used to record students completing a computer science modeling activity. This activity required students to model traffic lights with cars and pedestrians interacting. CogTutor recorded various actions that the groups of students made while completing the activity (e.g., Chatting, Moving objects, Creating objects, Deleting objects, Running simulations). After the students had completed the activity, instructors manually analyzed and scored the replays for a variety of factors. They then used these scores to evaluate the conceptual understanding of the students.

3.1.4 Other Prominent Intelligent Tutoring Systems

MathTutor is an open-access online tutoring system where middle school students can solve math problems with step-by-step guidance [5]. Part of the research involved with MathTutor is

17

the investigation of using CTAT to help make the authoring of such tutors using "programming by demonstration". MathTutor has been used in large-scale online tests and was shown to effectively attract and sustain the teaching of mathematical concepts to middle schoolers. BeSocratic differs from MathTutor in its input method. Activities created with MathTutor ask students to click on items and enter text into text boxes. BeSocratic instead asks students to construct structure, such as graphs, which we believe would increase student engagement and understanding in similar activities.

PAT is another constraint-based ITS for introductory algebra [61]. The system addresses mathematical modeling of problem situations that can be described by linear equations. Teachers set rules that describe how the system should behave given student inputs. In addition, PAT includes a curriculum authoring tool. Using this tool, teachers can set rules that describe the curriculum they would like to teach and PAT will combine individual tutor activities in a way as to facilitate the curriculum goals. While this tool helped create a coherent classroom curriculum, it required the authoring of curriculum goals in addition to creating many tutors to fill the curriculum.

Authoring Software Platform for Intelligent Resources in Education (ASPIRE) is an authoring and deployment environment for constraint-based ITSs [52]. ASPIRE is composed of two parts: (1) an authoring server that enables experts to develop new constraint-based authoring tutors and (2) a tutoring server that deploys the developed systems. Using the authoring server, experts create high-level descriptions of the domain, as well as samples of problems and solutions. From this, ASPIRE tries to create constraints to teach the targeted domain. With the domain model generated, ASPIREs deployment environment is able to host the tutoring system for students to access it. BeSocratic differs from ASPIRE in the way tutors are created and the types of questions that may be asked. Authoring tutors in ASPIRE is done through a combination in selecting the types of input expected (e.g., Boolean, float, integer) and setting up rules within an overarching relationship graph. We believe BeSocratic's authoring tool is more approachable through its graphical interface where teachers select rule options through drop-down boxes and sliders as well as the ability to quickly test possible student submissions within a preview window. Furthermore, BeSocratic's ability to recognize graphical input differentiates it from ASIPRE, which relies on students entering text.

REDEEM was an early authoring environment that allowed teacher to create simple ITSs from pre-existing computer based training (CBT) applications by imposing their pedagogical preferences about how different groups of students should best be taught [3]. It was shown that using this approach can significantly improve student learning in certain situations where students had high

interactivity with the system. BeSocratic uses the same philosophy and takes advantage of current technology (i.e., Tablet PCs) to provide interactive experiences previously unavailable.

Several researchers are also exploring the feasibility and significance of using crowd source techniques to create personalized tutors for students [4]. In order for the system to provide a personalized tutor, the system relies on student profiles. These profiles are filled out with information pertaining to a student's learning preferences. In the authoring stage of the tutor creation, the system prompts participating authors with a random student profile and a question. The authors then generate instructions, feedback, and drawings, which they believe should be given to the student based on their randomized profile. This process is crowd sourced in that many authors (> 500) are used to create instructions for various student profiles for a specific question. The system then uses a combination of automatic and manual filtering to remove inappropriate feedback (i.e., feedback too short, too long, or gave away the answer). With this filtered set of feedback, real students can fill out profiles and complete the problem. The system gives personalized feedback to the student based on the profile and the feedback gathering during the authoring process. While this approach is interesting, we feel that the overhead of generating and filtering large numbers of tutors for a single task is too time-consuming. BeSocratic instead aims to provide an easy to use authoring tool, which a single instructor could use to create a tutor.

3.2 Analysis Techniques

Recent years have seen an increase in the number of publications on the analysis of student work instead of just the collection of it. Some recent research from CTAT and EDM Vis use visualizations as a large component of analyzing student data. As mentioned previously, CTAT creates a graph of a tutor's various interface states. In a similar manner, EDM Vis integrates into other ITSs and can be used to better understand student learning and augment tutors with additional feedback for students at specific steps in their problem solving processes [42]. We have explored this idea in previous research [16], and concluded that this method is often too time consuming for problems with large state spaces. In these cases, the state graphs become too large, and it becomes increasingly difficult to identify locations to place feedback and unclear if the students really need help or should be left alone.

As we described previously, BeSocratic models the performance of students and classes with

hidden Markov models. Although HMMs have been applied extensively in speech processing [59], the application of HMMs to ITSs is relatively new[68]. In more recent years, further research has been conducted on the uses of HMMs to cluster and categorize student data.

Stevens et al. were among the first to explore the application of machine learning techniques to student data in order to discover student strategies during problem solving [18, 22, 68]. In their research, students were tasked with solving various problems within IMMEX (Interactive Multi-Media Exercises). This system is a Web-based problem set platform that enables the online delivery of complex, multimedia simulations, the collection of student performance data. While this system is very flexible in that it allows students to choose their own sequence of actions, students are still restricted to mouse clicks or drag and drop actions. The feature that really sets IMMEX apart is the modeling of student input data. In one study, IMMEX used artificial neural network clustering to provide individual performance measures, and then sequences of repeated performances were probabilistically modeled by hidden Markov modeling to provide measures of student progress over time. Using the results from this analysis, they suggested instructor interventions based on early student performances with these simulations may assist students to recognize effective and efficient problem-solving strategies and enhance learning. While this method appeared promising, subsequent research using this method did not prove as revealing as the initial study.

In 2009, Jeong et al. explored using hidden Markov models with student data generated from their teachable agent applications [41]. In this application, students were tasked with creating a concept map for a river ecosystem. The application also provided a virtual agent that students could query for help and clarifications. After recording students using this system, each student's action sequence was transformed into a sequence of 6 activities base on the actions taken. Using hidden Markov models, the researchers were able to identify patterns within the responses. They were then able to compare which patterns result in higher success rates for various questions. The results indicate clear differences between different interventions, and links between students learning performance and their interactions with the system.

Beal used a similar method to model the actions of high school students using a mathematics ITS with hidden Markov models [12]. The student data was first coded into a sequence of global action descriptors (e.g., Guess, Skip, Attempt). The sequences are then fit to a HMM with 3 hypothesized hidden states corresponding to engagement levels (i.e., Low, Medium, High). Having fit a HMM to students' action pattern data, the transition matrices helped them determine the

	Attempt	Help Request
Slow	Guess	Drill
Fast	Try	Reason

Table 3.1: Shih's Mapping from <Action,Duration> to one variable

probabilities of students moving from between levels of engagement as well as staying within the same level of engagement. They were then able to cluster groups of student with similar engagement levels using the Kullback-Leibler (KL) distance between individual transition matrices. In addition, they described additional work designed to predict a students likely engagement trajectory, with the goal of diagnosing the student's behavior early enough that interventions could be deployed to increase or sustain engagement. They found that the prediction capabilities of HMMs is significantly better than a simpler Markov model. They admit that "the primary limitation of the study is that we do not have direct evidence that the hidden state in the HMMs actually corresponds to any of the processes that are known to reect 'engagement' in the learner, including transient shifts in the learner's attention, emotions and cognitive effort."

Most recently, research from Shih, Koedinger, and Scheines used hidden Markov models to discover student strategies within a typical intelligent tutoring system [66]. In a similar manner as BeSocratic's system, their research tried to find commonalities within sequences of student actions within their system. To simplify the analysis, they first reduced their actions to Attempt or Hint-Requests. To further discretize the data, these actions were classified into 4 categories based on whether a short or long amount of time preceded them. Table 3.1 shows their mapping from <Action, Duration> tuples to a single variable.

Discretizing the data in this fashion allows HMMs to efficiently analyze the data while still keeping some of the temporal data that is crucial to student sequence data. Their research included developing an algorithm to create HMMs for clusters of student data. Their algorithm used biases to generate a low number of HMMs with as few states as possible so that they were easier to interpret after training. In our opinion, the only down-side to this method was that their algorithm used external data about each student (e.g., test scores) to try and improve its clustering ability. Our project's goals also differ from theirs in that we would ideally not require teachers to be familiar with HMMs in order to analyze their students' work. Furthermore, we have tried to expand the simple 4 state representation used and instead use more context-specific states.

Chapter 4

BeSocratic

With our goals defined, we built BeSocratic, an online intelligent tutoring system that contains a variety of question types that are free-form in nature yet well-defined to the point in which they may be automatically evaluated and analyzed. This chapter describes BeSocratic and how it is used to collect student data. Section 4.1 explains the structure of BeSocratic activities and the various modules that may be used when creating tutors. Section 4.2 describes the 4 main tools of BeSocratic: Authoring, Completion, Analysis, and uRespond. Section 4.3 details the devices that BeSocratic currently supports along with our plans for supporting future devices.

4.1 Activity Structure and Modules

BeSocratic's interactive tutors are referred to as *activities*. Each activity is made up of one or more *activity steps*. This relationship between activity steps and activities is analogous to slides in a slide-show. And just as slides in a slide-show have various items within them (e.g., text, images, videos), activity steps contain one or more *modules*, which equip each step with its core functionality.

BeSocratic modules are divided into two general categories: *non-interactive* and *interactive*. Non-interactive modules, which include text boxes, images, videos, and ink canvases, do not provide students with feedback. These are generally used to convey information and/or instructions to students or are used to gather information for manual analysis. Interactive modules, on the other hand, allow BeSocratic to pose free-form questions, provide automatic feedback to students, and enable automatic analysis for teachers. These modules currently include SocraticGraphs, OrganicPad, and

If a container with He solid in it is heated (for example by placing the container on a heated block), the solid will melt and then evaporate. Draw a diagram in the black outlined box showing how the energy from the container is transferred to the He atoms and write your explaination of your picture in the blue outlined box.

The energy from the flames causes the elctrons to excite and thus move apart from eachother.

fire

Draw   Erase ▾   ✗Reset

Figure 4.1: A BeSocratic slide containing a Display Text module along the top, a Chalkboard module on the left, and a Text Input module on the right.

GraphPad. All of these modules may be mixed and matched within slides to build rich, interactive activities. We describe these modules in greater detail in the following subsections. Furthermore, we explain how to build and integrate additional modules into BeSocratic.
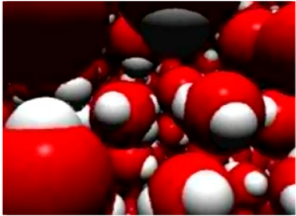
4.1.1 Display Text

Perhaps the simplest BeSocratic module is the Display Text module. This module allows teachers to add rich text within a slide. This text can be formatted with a variety of attributes, including alignment, font, size, bold, italics, underline, and color. In addition, these modules may embed hyperlinks to send students to outside websites for additional instruction or material. This module is frequently used to direct students in completing a task.

4.1.2 Images and Video

BeSocratic contains two modules for showing media to the user: Image and Video. Image modules allow teachers to display static images to students. Video modules allow students to watch short videos within a slide. BeSocratic stores these modules by first uploading the media to our
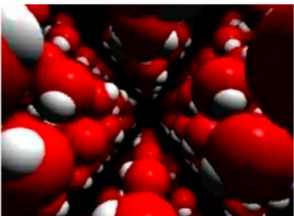
23

Figure 4.2: A BeSocratic slide containing two Video modules.

server. Then, the module only needs to store a hyperlink to the media file. This allows for faster upload times on subsequent saving. It also facilitates faster download times for teachers during group analysis when the media needs to be downloaded for each student. Because modern browsers cache web page files, each media item only needs to be downloaded once using the hyperlink and then is automatically reused for each subsequent reference to the same link.

4.1.3 Text Input

The Text Input module records text input from students. This module provides a place for teachers to ask open ended questions to probe into a student's thought process and reasoning. Furthermore, Text Input modules have the ability to load text answers from previous questions so that students may re-edit their work. In such an activity, the starting slides first ask students to answer a text-based question. This is typically followed by taking students through a tutorial using other slides. Finally, these activities ask students the same starting question and allow them to edit their previous answer. Using our analysis tools, the teachers can then compare the differences between the students' first and final answers to track changes in their reasoning. This is an example of one

24

the ways in which BeSocratic can assist teachers in manually analyzing text responses. Additional examples are described in Chapter 5.

4.1.4 Chalkboard

The Chalkboard module allows students to create ink drawings using their mouse, stylus, or finger. Besides drawing objects or ideas, this module can also be effective in places where students need a place to show their work. We also regularly place Chalkboard modules on top of images and ask students to perform tasks such as circling various features of the underlying image.

4.1.5 3D Model

The 3D Model module allows students to view and manipulate 3D models of molecules. During the activity's creation, teachers simply enter a chemical name. BeSocratic then queries the ChemPub chemical database to find a match. If a match is found, the molecule's structural properties including atoms locations and bonds are downloaded and rendered. Students may then use gestures to rotate and scale the 3D molecules. This module is typically used in chemistry or biology activities.
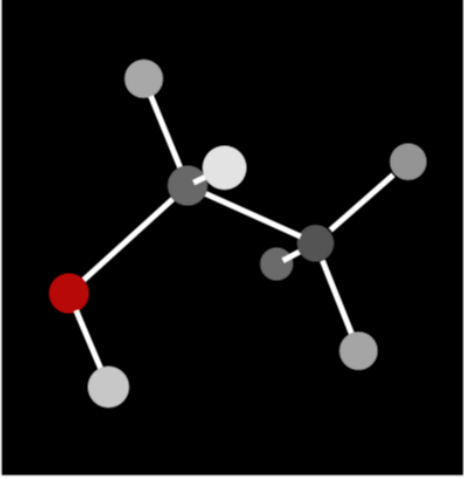
4.1.6 Multiple Choice

The Multiple Choice module allows teachers to ask simple multiple choice questions. The multiple choice questions can be set to accept either "Only 1 Answer" or "Multiple Answers". This setting simply changes whether the control next to each answer is a radio button or check box thus allowing one or more answers to be selected at a time. Later in Section 4.2.2, we describe BeSocratic's ability to summarize and visualize student responses to Multiple Choice questions.

4.1.7 Fill-in-the-Blank

BeSocratic contains a module to evaluate simple fill-in-the-blank questions where students are only asked to enter a few words. Teachers provide a set of correct answers along with multiple levels of feedback for when the students provide a correct or incorrect answer. In addition to each correct string, teachers may also set options such as "character tolerance". The character tolerance value is a positive integer value indicating the maximum number of "edits" that can be made to a

Figure 4.3: A BeSocratic slide containing a 3D Model of an ethanol molecule.

student's answer to transform it into the teacher's solution. This value is calculated using the same Levenshtein distance measure that was previously discussed in Chapter 2 and used in BeSocratic's analysis tools. If the number of "edits" is less than or equal to the tolerance value, the module treats the comparison as a match. We have found this tolerance value useful when students slightly misspell the correct answer. In that case, students are not frustrated by simple spelling mistakes and are able to continue with the activity.

4.1.8 Image Plotter

The Image Plotter module lets students drag a red dot on top of an image. This allows teachers to specify questions in the form "Tap on the location in the image in which...". In Section 4.2.3, we describe how this module fits into the uRespond system and allows for interesting heat map visualizations of the locations that students tapped.

Apply this information

Use what you know to calculate what the value of ΔG° would be if the reaction were carried out at room temperature (298K). Enter your answer in the space below.　Information you may need:

ΔG° = ΔH° - TΔS°
ΔH° = -1536.4 kJ/mol
ΔS° = 44.74 J/K
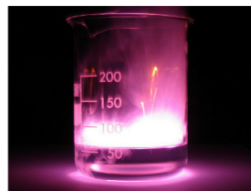
ΔG° (in kJ)= -1550　　✓ Check

Is the reaction spontaneous or nonspontaneous at 298 K?
nonspontaneous　✓ Check

Figure 4.4: A BeSocratic slide containing two Fill-in-the-Blank modules. One asks the students to enter a number. The other asks students a text response question.

The E_{red} of Fe/Fe2+ is -0.44 V and is +0.80 V for Ag/Ag+. Tap on the portion of the voltaic cell below that represents the cathode.
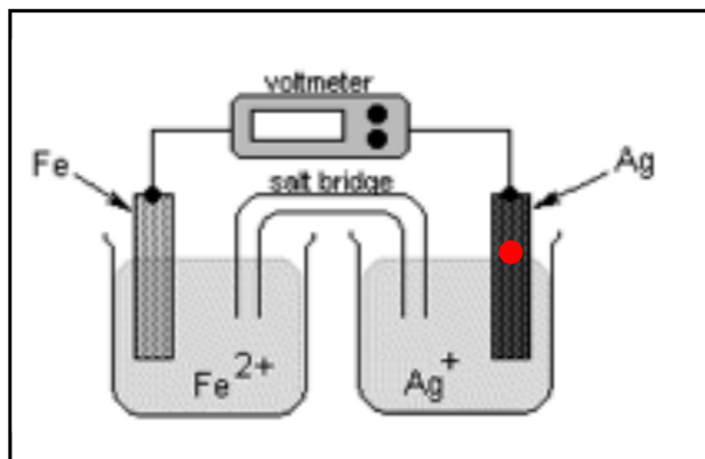
Figure 4.5: A BeSocratic slide containing an Image Plotter module.
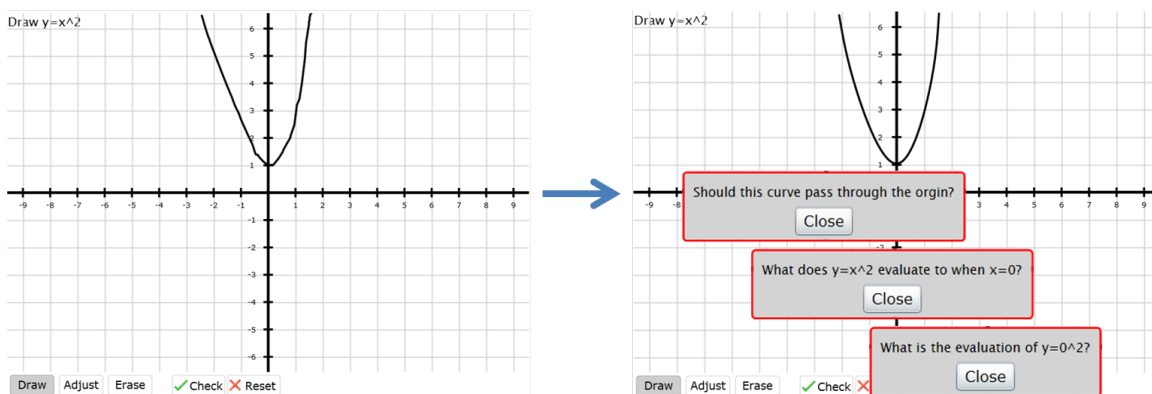
Figure 4.6: A student drawing a raw stroke onto the SocraticGraphs module (left), and the "clean" curve that was generated along with 3 levels of feedback for the student's error (right)

4.1.9   SocraticGraphs

SocraticGraphs is an interactive module that allows teachers to pose graph-based questions, that is, questions that require students to respond by drawing 2D mathematical graphs or simple drawings. SocraticGraphs evaluates the graphs based on rules that have been pre-specified by the teacher. Moreover, SocraticGraphs displays multi-tiered feedback to students based on which rules are not currently satisfied by the student's solution. This functionality in SocraticGraphs is based on an underlying context-free grammar that provides teachers with a robust, flexible, and expandable constraint-based authoring tool. Example productions of the grammar are shown in Listing 4.1. Examples of such rules include: the number of maxima/minima, area under the curve, slope, and intersections with designated areas. By combining such rules, activities designed with SocraticGraph modules may be used in a many STEM disciplines, including mathematics, chemistry, biology, and engineering.

Figure 4.6 shows an example of a student completing an activity step that contains a SocraticGraphs module. In this example, the student is tasked with drawing a curve for the equation: $y = x^2$. The left side of the figure shows the student's raw input stroke. The raw stroke is converted into a smooth curve with adjustable handles that are shown as orange squares. Once the student feels that the answer is correct, the student can click the *Check* button and the curve is evaluated against the teacher's rules. Depending on the results of the evaluation, multiple levels of feedback are given to students. The right side of the figure displays an example of several tiers of feedback that could be given for this problem.

28

In order to evaluate a user-drawn stroke, SocraticGraphs must first convert the raw stroke into a "clean" curve. The first step of this process is the removal of noise introduced by accidental hand movements or from the hardware. To accomplish this task, SocraticGraphs first re-samples the stroke to pixel resolution using the Bresenham line algorithm, which takes 2 points and interpolates a line between them at the pixel level [14]. This step is necessary since the various input methods (i.e., mouse, stylus, or touch) often have different or variable sampling rates, which makes filtering difficult. Once the stroke has been transformed into a consistent sampling, we use a series of sliding-window low-pass filters to remove various levels of high frequency noise and thus smooth the curve. Next, because students should have the ability to adjust their curve after it is drawn, it is important that SocraticGraphs extract the "key" points along the curve. These points become handles for students to drag and make adjustments. Finding these key points is accomplished using the Douglas-Peuker reduction method [26]. This method reduces the number of points in a curve through an approximation of the series of points within the curve. In short, the Douglas-Peuker method uses a tolerance field to find points of maximum dissimilarity. As the tolerance increases, the method selects fewer points and thus makes a coarse approximation. Conversely as the tolerance becomes lower, a larger number of points are extracted, and the approximation becomes more refined. Finding one ideal value, which balances detail and smoothness for all problem types, is not possible. Instead, we have hand-tuned three tolerance values to support the recognition of curves with low, medium, and high detail. Tutor authors select the level of detail required for the question. Once the Douglas-Peuker method has returned the points making up the approximation to the stroke, SocraticGraphs renders the curve. Because the number of points returned is often small, simply drawing lines between points may not appear smooth to the user. Instead, SocraticGraphs generates a cubic Bezier spline using the points as knots in the spline and generating control points from the derivatives of the curve at each knot [43]. All of these steps combined allow hand-drawn strokes to be converted into smooth, manipulatable curves.

As previously mentioned, the SocraticGraphs module's ability to evaluate curves is based on a context-free grammar. The productions of this grammar compose the constraints that are used to evaluate a student's drawing. Listing 4.1 shows an example of some of these productions and a full listing is shown in Apprendix A. Expanding these productions gives a natural way for the system to formulate rules for the desired characteristics of the curves. For example, the rules in the figure could be expanded to form "maxima in curve 1 > minima in curve 2". This is a well-

```
Max → MaxValue MaxConditional IntValue
MaxValue → maxima in CurveLocation
MaxConditional → = | != | < | > | <= | >=

Min → MinValue MinConditional IntValue
MinValue → maxima in CurveLocation
MinConditional → = | != | < | > | <= | >=

IntValue → int | MaxValue | MinValue
CurveLocation → curve double | CurveSegment
CurveSegment → curve segment in curve int from CurveSegmentLocation to CurveSegmentLocation
CurveSegmentLocation → Relative Position double | X = double | Y = double
```

Listing 4.1: Sample SocraticGraphs productions for the number of local maximums and minimums in a curve.

defined rule that is clear to both the author and the system. The characteristics currently available include: number of local maximums and minimums, the area under the curve, the slope, the shape of the curve (i.e., linear, exponential, logarithmic, sigmoidal, normal), and the number of curves. In addition, SocraticGraphs allows authors to specify areas where the curve should be contained within, pass through, or have specific points be contained within (e.g., the mean of the curve should equal the origin (0,0)). Each rule also has an optional tolerance value associated with it as well. This tolerance gives authors flexibility when setting the rules, so their students do not have to draw curves perfectly correct. Expecting such precision would be an unrealistic; since for example, it is nearly impossible for students to make a pixel perfect drawing of the curve $y = x^2$. Instead, a tolerance is set for students to draw the curve within an acceptable level of accuracy.

Condition Editor

Identifier: Condition ☑ Correct

Feedback for Response Following All Rules on First Attempt:

Edit [A] Good job.

Feedback for Response Following All Rules After Failed Attempt:

Edit [A] That looks much better.

Match [All ▼] of the following:

1 Rule:

Minimums ▼ in Curve ▼ 1 ▼ = ▼ 1 ▼

Tolerance:

Feedback for Incorrect Rule:

1. Edit [A] How many minimums should be present within the curve?
2. Edit [A] Are you sure that is the correct number of minimums for the...

2 Rule:

Intersection ▼ in which Curve ▼ 1 ▼ intersects ▼ 0,0 ▼

Tolerance:

Feedback for Incorrect Rule:

1. Edit [A] What does y=0^2 equal?
2. Edit [A] What does y=0*0 equal?
3. Edit [A] Try again.

3 Rule:

Intersection ▼ in which Curve ▼ 1 ▼ equals ▼ Math Equation ▼ y=x^2

Tolerance:

Feedback for Incorrect Rule:

1. Edit [A] Have you checked to ensure you curve goes through the correct...

✚ Add Rule   ▬ Remove Rule   Copy Rule

Step 1 of 1

Preview:

Please draw the graph for the equation y=x^2.

Draw Adjust Erase ✓ Check ✗ Reset

72%   Keep Changes   Cancel   100%

Figure 4.7: This is the rule editor for a SocraticGraph module. The left pane contains the rules for the graph along with feedback to use when the answer is wrong. The right pane provides a place for teachers to test out different graphs and understand how the system will respond to student drawings.

Developing a constraint-based tutor, like any other intelligent tutoring system, can be a labor-intensive process. To address this concern and alleviate the need for teachers to have knowledge of programming or context-free grammars, we have developed a graphical interface on top of the grammar to facilitate creating rules. Figure 4.7 shows an example of our rule editor. The left side of the figure displays a number of rules that have been added for the question. In this example, rules have been set to ensure each student's curve contains one minimum, intersects the origin within a tolerance, and is contained within the equation $y = x^2$. Notice that associated with each rule is one or more tiers of feedback. This feedback is displayed to students when their curves do not satisfy the corresponding rule. The top left panels of the editor provides a place for tutor authors to specify feedback for instances when a student's curve meets all of the rules on their first attempt and after an unsuccessful attempt. It is also important to note that the right side of the editor provides a live preview area for experimenting with the various rules and drawing. When an author draws a curve in this area, each rule becomes shaded with red or green depending on the results of the evaluation with the current curve. In the example shown, the origin and math formula rules are shaded red because the curve does not intersect the origin nor is it contained within the area defined by the formula. Conversely, the one-minimum rule is shaded green because the curve only contains one local minimum. Using this visual interface, tutor authors can quickly create, prototype, and experiment with various rule configurations.

Rules are assessed in sequential order. If all of the rules evaluate to true on the student's first attempt, the correct feedback will be displayed. If an incorrect attempt was already made, the "Correct After Incorrect" feedback will be displayed. And if the student's drawing does not fulfill all of the rules, the feedback for the first incorrect rule in the list will be shown to the student. The feedback given can come in a variety of forms such as simple text, text and a text box (for asking text-based follow-up questions), text and an ink canvas (for student drawings), an external website (that students must visit before closing the feedback), or an outside BeSocratic activity (which students must complete before continuing). We find that these types of feedback discourage students from guessing, and we feel that this is leading to higher levels of student engagement.

4.1.10 OrganicPad

OrganicPad is a module that focuses on drawing Lewis structures. The OrganicPad module allows users to draw Lewis structures intuitively using using handwriting recognition and gestures.

32
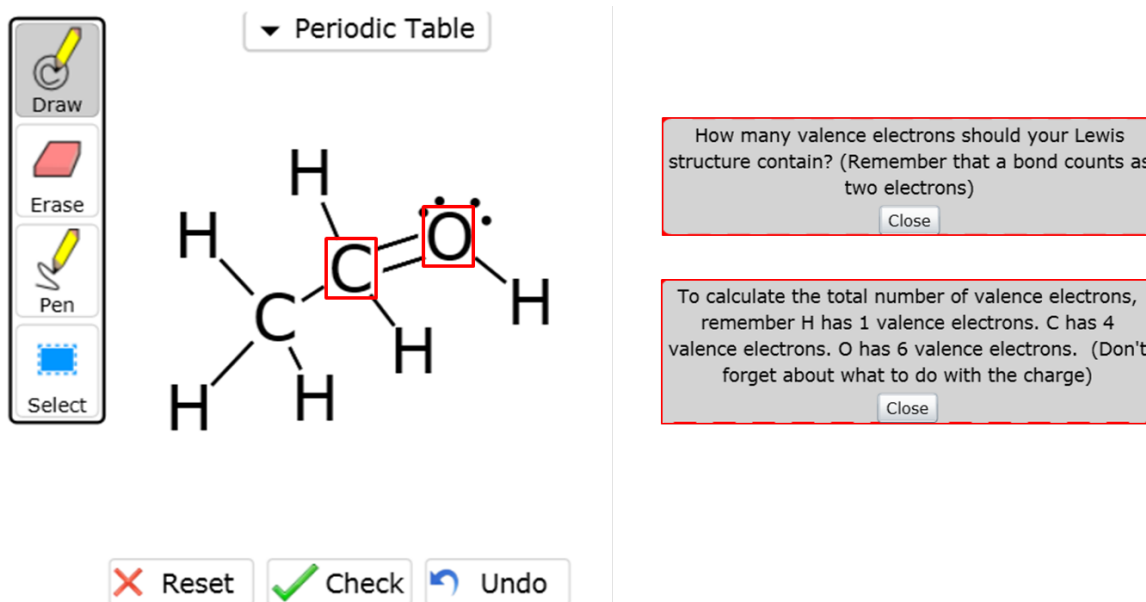
Figure 4.8: An OrganicPad module with a student-drawn structure (left). The structure contains an error , which is highlighted with boxes. Multiple-tiers of feedback is also shown (right)

Each student's writing is converted into formatted text that is used to create an interactive molecular structure on the canvas. OrganicPad can assess a student's structure against the teacher-provided solution and give feedback based on how the two compare. This feedback is multi-tiered and Socratic in nature; the feedback starts with very general hints and becomes more specific if the student continues to struggle. In addition, OrganicPad can highlight the parts of the structure that are causing errors using graph isomorphism algorithms that compare the structures for similarities and differences.

Using OrganicPad, we were able to evaluate and analyze student work and gain insights into the students' thought process [17, 23, 24, 57]. OrganicPad was originally designed as a standalone desktop application but has now been added into the set of BeSocratic modules to be used within browsers.

Figure 4.8 shows an example of a student completing a problem using an OrganicPad module. In this example problem, the student was tasked with drawing the chemical structure for ethanol. In the example, the student has drawn an extra bond between the oxygen and carbon atoms. In this case, the OrganicPad module highlights the incorrect part of the structure and gives multi-tiered feedback to the student.

The development of OrganicPad required several key insights to be made. Namely, we have created an interface that is intuitive to use. By taking advantage of handwriting recognition, we have created an intuitive interface for student to construct molecules. In addition, we represent the molecules are graphs. This insight allowed us to develop graph isomorphism algorithms that compare the student's submission with solutions supplied for the activity creator and provide feedback based on those comparisions.

4.1.11 GraphPad

A third interactive module in BeSocratic is the GraphPad module. GraphPad is designed to help computer science students learn to construct a variety of data structures. This module uses ink recognition to convert student drawings into interactive nodes and edges that may be analyzed. GraphPad recognizes nodes, undirected and directed edges, and labels for nodes and edges. This simple design allows the construction of practically any graph structure that may be found in a computer science data structures curriculum including: lists, stacks, trees, generic graphs, and finite state automata.

In order to evaluate student-drawn structures, teachers must provide data structures, which they mark as correct or incorrect. Furthermore, one or more tiers of feedback are associated with each structure and are displayed to students when their structures match.

GraphPad was originally created for desktop computers where it showed positive results in computer science data structure classrooms [15, 58]. GraphPad has been ported to BeSocratic where it may be used on a wider variety of devices. Figure 4.9 shows a student who has drawn a binary search tree using the GraphPad module.

4.1.12 Building Additional Modules

BeSocratic supports the addition of new modules through the use of a base framework. Each module in BeSocratic inherits from a base SocraticModule object, which provides default functions for saving and loading the module. New modules can be added to the system through the following steps:

1. Download the BeSocraticLib.dll base library from http://besocratic.clemson.edu.

2. Start a new Silverlight 4.0 project in Microsoft Visual Studio 2010 and add besocraticlib.dll as
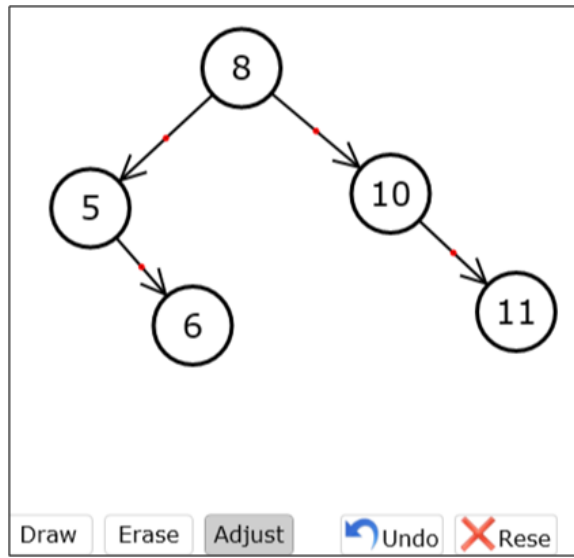
Figure 4.9: A GraphPad module with a binary search tree drawn

a project reference. besocraticlib provides the developer with base classes and functions that can be used to create and integrate new modules for BeSocratic.

3. Create a Silverlight UserControl in Visual Studio 2010 and implement your module. In this step, the developer creates the interface and programs the functionality of the module (e.g., drawing and recognizing curves for SocraticGraphs).

4. Change the module's base class from UserControl to BeSocraticModule. As part of this step, the user should override several of the methods of the BeSocraticModule class. These methods include: SaveModule() which returns a string serialization of the module, LoadModule(String data) which loads the string serialization of the module, and SaveRecording() which returns a serialized list of timestamped actions since the last SaveRecording() method call. These methods allow BeSocratic to easily integrate the module into its system where it provides a unified authoring, viewing, and analysis system for the module.

5. Lastly, the module needs to be sent to the BeSocratic developers using the forms found on the besocratic.clemson.edu website. Because BeSocratic is tightly integrated with a database and webservices, we are not able to offer all of it as open source. This makes it impossible for developers to integrate their modules without contacting us. To help expedite the process, we offer a submission page where new modules may be submitted for approval and integration
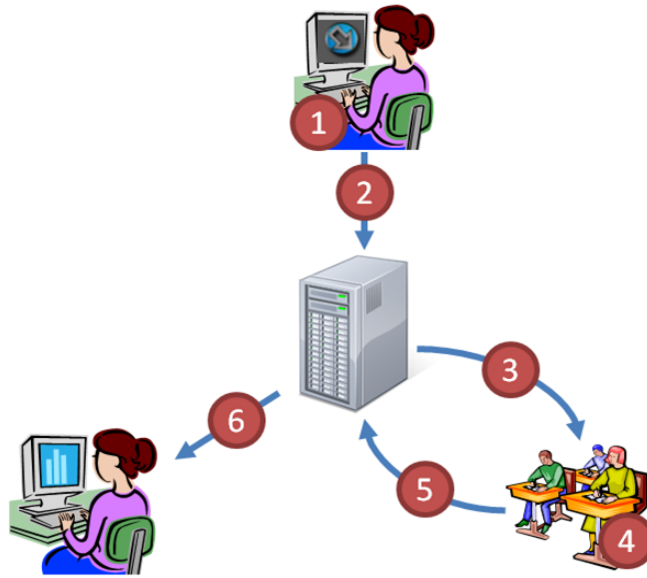
Figure 4.10: A diagram of BeSocratic. Activity creation, assessment and analysis can be achieved through communication between teacher and students, as outlined in steps 1 through 6

with BeSocratic at http://besocratic.clemson.edu.

Following those steps, new modules may be created and added to BeSocratic. Examples of possible future modules include educational games (e.g., a matching game), math tutors using equation handwriting recognition, and circuit diagram construction.

4.2 Tools

Figure 4.10 shows the general flow of a BeSocratic activity's cycle. (1) The teacher creates and develops an activity using the activity authoring tool. (2) The teacher uploads the activity to the BeSocratic database. At this point, students (3) download the activity and (4) complete it. Once the activity is finished, the BeSocratic (5) uploads their final work along with a replayable version of their work to the BeSocratic database. Finally, (6) teachers may replay and analyze the student submissions.

BeSocratic accomplishes this set of actions using three main tools: Authoring, Completion, and Analysis. The Activity Authoring Tool enables teachers to create BeSocratic activities and make them available for students to complete. The Activity Completion Tool lets students complete activities at their own pace. The uRespond system provides a real-time interactivity component for

activity completion similar to Clicker systems. The Activity Analysis Tool facilitates the replay and analysis of student work. Together, these tools allow teachers to assess and analyze the knowledge levels of their students. The following sections describe these tools in greater detail.

4.2.1 Activity Authoring Tool

Since it has been shown that tutor authoring is often a difficult and time-consuming task, BeSocratic's Activity Authoring Tool has been designed to resemble Microsoft PowerPoint using the Microsoft Ribbon interface. This provides teachers with a familiar interface to create activities. Figure 4.11 shows an example of an activity being created with the Activity Authoring Tool. The left sidebar (B) displays a thumbnail list of the activity steps. The Activity Authoring Tool provides a variety of functions and options along the top of the screen (A). Modules are added to the current activity step by selecting the module from the list in (A) and dragging your cursor on the activity step canvas (C). Selecting a module in the activity step canvas (C) brings up a new tab along the top (A) with more options for controlling the module. The Authoring Tool also contains a preview function for the teacher to preview how the activity will appear to the students. This allows teachers to quickly prototype various configurations for the activity.

Once the activity is created, teachers specify a roster along with start and end timestamps so that students in the roster may log into BeSocratic and complete the activities within the start and end dates specified. Activities are saved to and loaded from the BeSocratic database so the teacher may access them from any computer browser connected to the Internet. A complete diagram of BeSocratic's database is shown in Apprendix B. We have built a file system into BeSocratic so teachers may organize the locations where they save their activities. In addition, the file system allows activities and folders to be shared for collaboration among groups.

Activity Completion Tool   Once an activity has been created, students may download and complete the activity. When students log into BeSocratic, they are prompted with a list of available activities. After an activity is selected, the Activity Completion Tool (Figure 4.12) loads the activity for completion. The Activity Completion Tool is designed to present students with a minimalist and intuitive interface. This allows the students to focus on completing the task at hand instead of negotiating through a clumsy interface. As students complete activities, their actions are stored in a database along with a snapshot of each slide. The tool saves each student's work after each slide,

37

Figure 4.11: The BeSocratic Activity Authoring tool
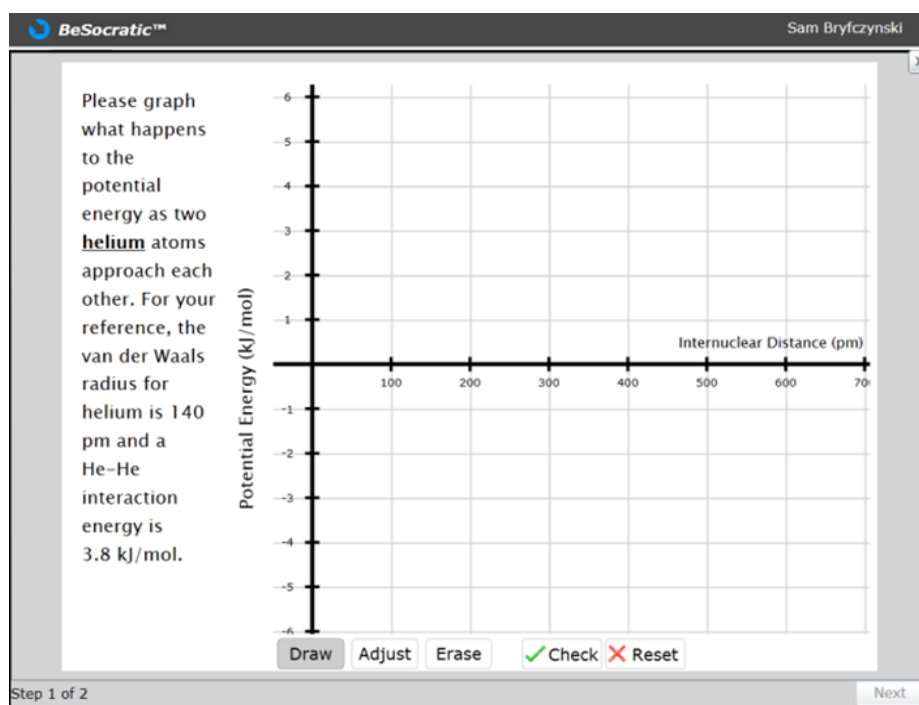
Figure 4.12: The BeSocratic Activity Completion Tool. Students use this tool when completing an activity

which permits students to leave the activity and return to their work at a later time.

4.2.2 In-Class Analysis Tool

The In-Class Analysis Tool provides teachers with replay and analysis functionality. This tool provides several views for teachers to quickly look at student submissions: List, Grid, Visualization, and Spreadsheet. It is important to note that this tool is not simply for looking at submissions outside of class. We use this tool during lectures to identify and address problematic concepts. We believe that showing students examples from their own piers leads to better engagement.

List View   Perhaps the simplest way to view student submissions is through the list view where a list of the students who have submitted an answer appear on the left hand side of the screen. Selecting a name from the list will load the student's submission in the center on the interface as seen in Figure 4.13. At this point, teachers may also replay the selected student's work to isolate areas of difficulty.

BeSocratic's ability to replay submissions helps teachers potentially gain deeper insight into
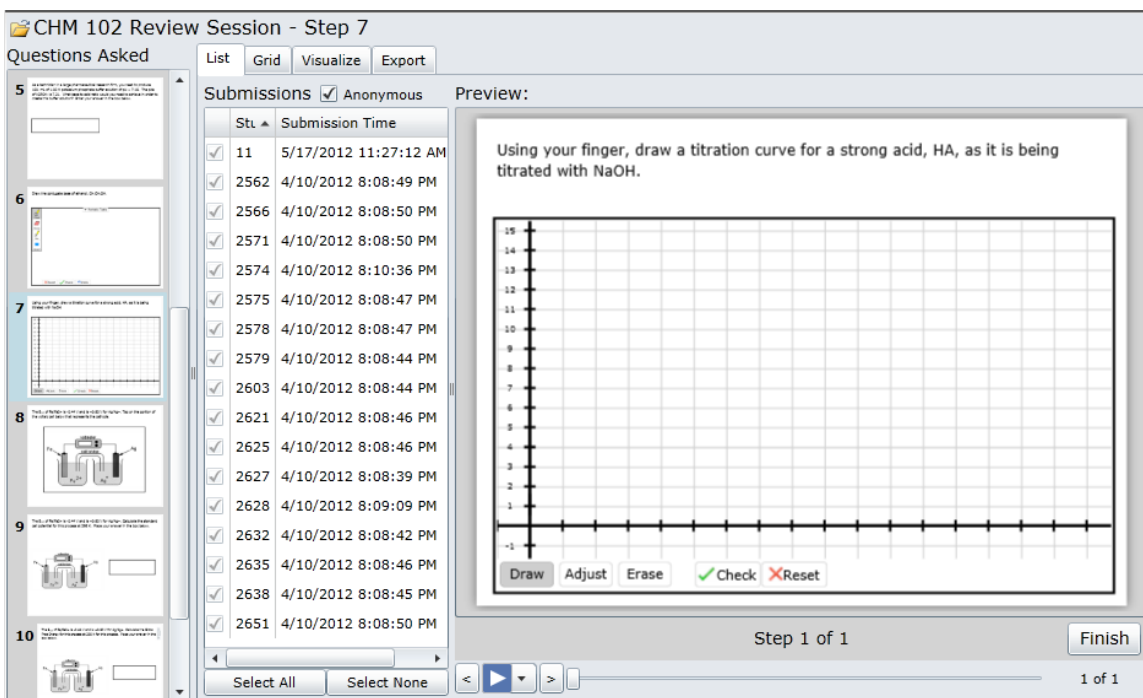
Figure 4.13: List view of a set of submissions.

the cognitive processes of students as they work on their assignments. Using the replays, teachers can identify locations in the student work where they made mistakes. By identifying errors in this fashion, teachers can refine their instruction to address and fix the root of the problems. In addition, teachers have the option to project submissions, anonymously, to the class and use it as a part of a lecture.

Grid View   Another similar way to visualize submissions is through a grid view as seen in Figure 4.14. This displays all of the students in a grid with a thumbnail of each student's final submission for the selected slide. Selecting a submission enlarges the thumbnail and provides controls to replay the submission. By placing the submissions in a grid, we find that teachers are able to quickly scan the images for interesting submissions that may require further investigation.

Visualization View   Although viewing students responses individually or in a grid may be appropriate for instructors with small enrollments, for instructors who teach larger enrollment courses, reviewing each response one at a time is untenable. As such, we are currently developing a series of visualizations to organize and present the data in as streamlined a fashion as possible and can be
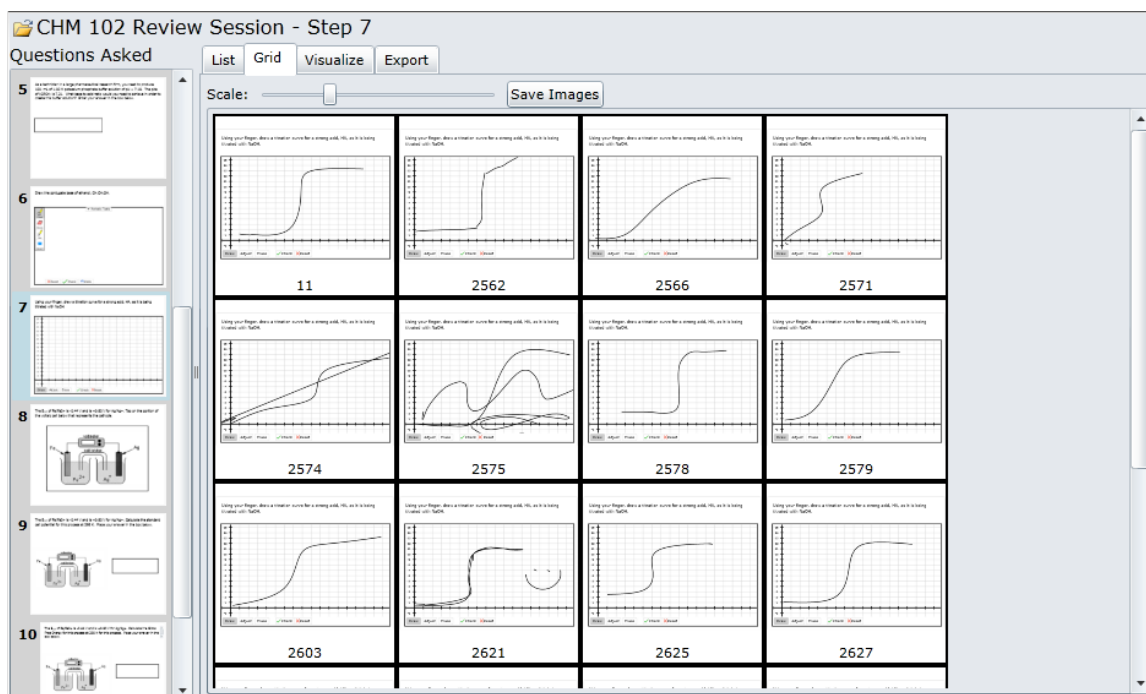
Figure 4.14: Grid view of a set of submissions.

accessed by switching to the "Visualize" tab. The visualization tab is contextual in that it changes depending on the type of question that is being asked. The screen shots in Figures 4.15-4.19 show our work with these visualizations.

Both the multiple-choice questions and the OrganicPad questions utilize representations that rely primarily on pie charts (Figures 4.15 and 4.16). In the case of a OrganicPad question, however, the charts are interactive, with several layers of analysis available. On the top layer, the students responses are initially organized by formula (i.e. H_2O, O_2H). By clicking on any wedge, a second pie chart is loaded to the right of the first that organizes all structures with the same formula by equality using graph isomorphism. When a wedge within this chart is selected, screen shots of all student submissions for that particular graph structure are loaded below it.

Figure 4.17 includes a typical visualization of an image plotter question. As it currently functions, the system records an (x, y)-coordinate pair for each student submission and overlays each on the original image. BeSocratic can also generate heat maps based on the common locations that were tapped by students.

An early version of the text visualization is shown in Figure 4.18. In this case, students were

41
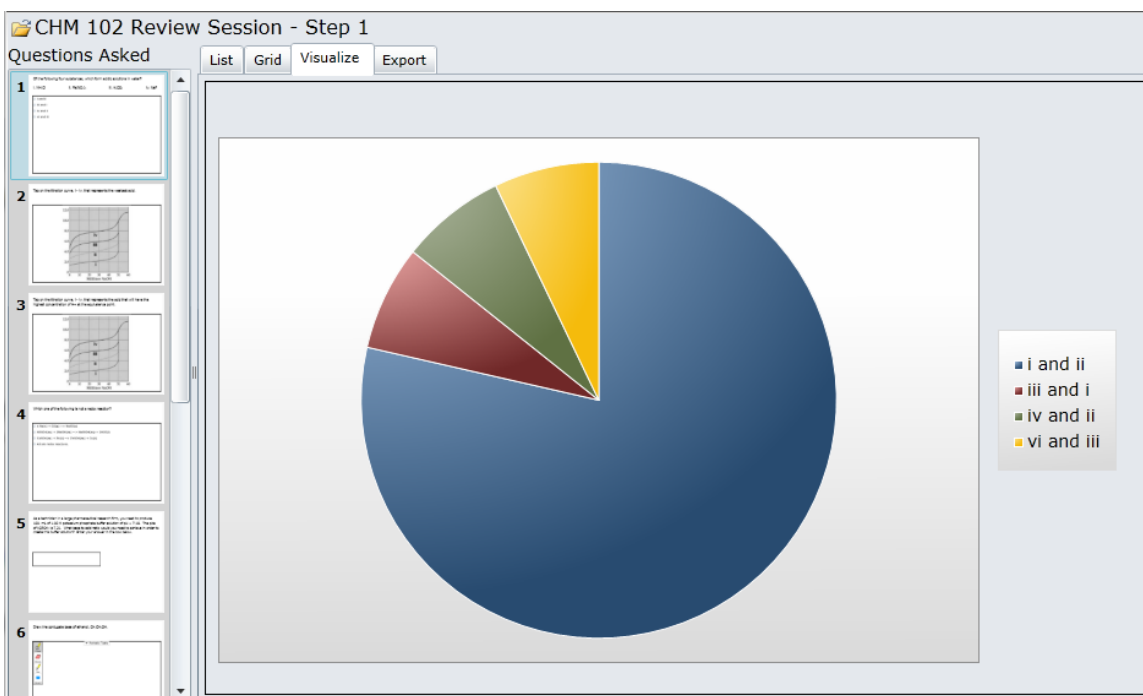
Figure 4.15: Multiple Choice Pie Chart Visualization
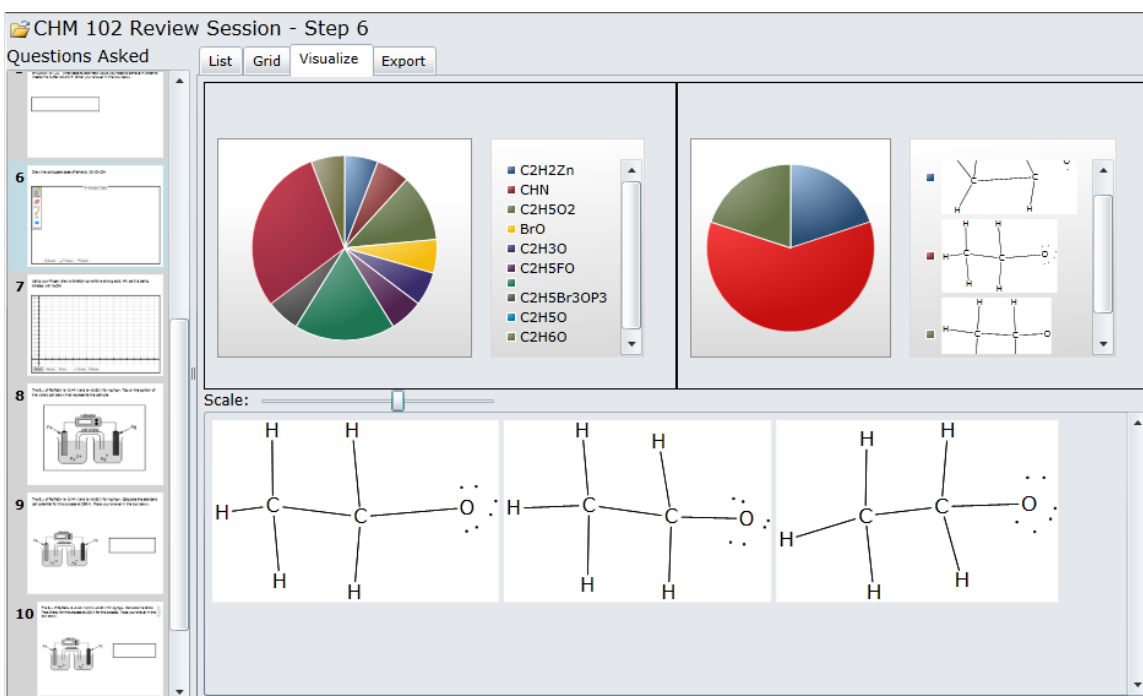
Figure 4.16: Visualization for OrganicPad. The molecules are first grouped by chemical formula in the first pie chart (top left). Then the molecules are grouped by structure using graph isomorphism (top right).

Figure 4.17: Visualization for Image Plotter. In this example, all of the locations tapped by students are overlaid on the background image.
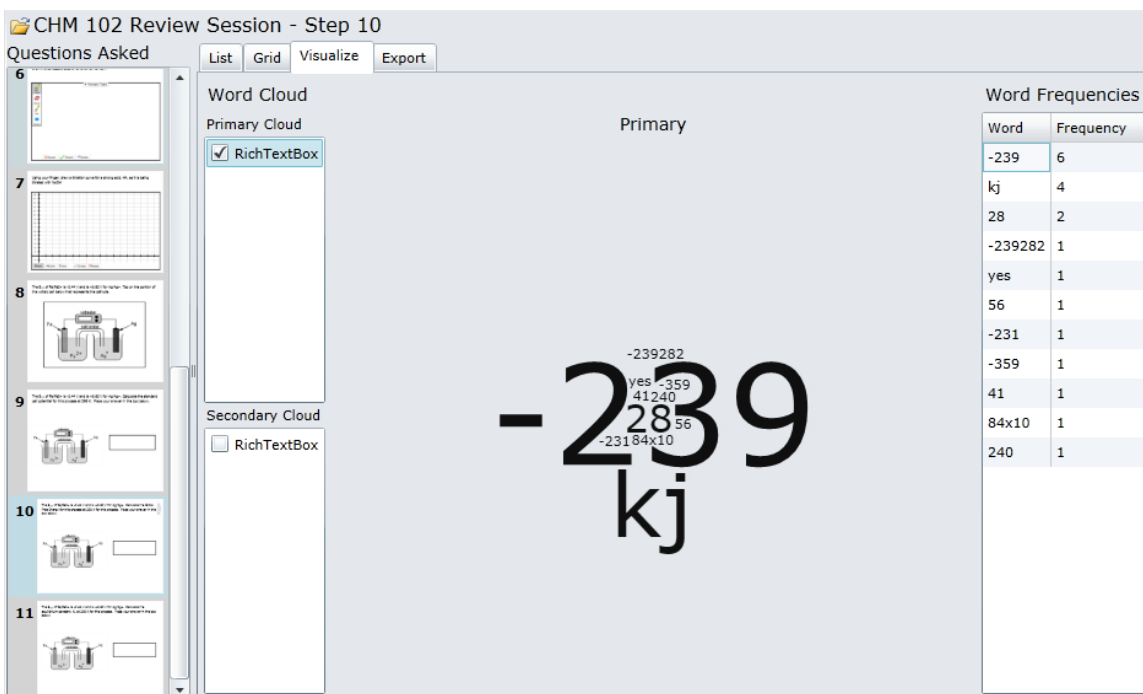
Figure 4.18: Visualization for Text. A word cloud is generated from the student responses.

asked to calculate the standard cell potential for a specific voltaic cell. The current visualization makes use of a word cloud representation where the system analyzes the prevalence of word usage in the answers and portrays frequently used words in larger fonts than less frequently used words.

Finally, Figure 4.19 shows the visualization available for graphing questions. In this example, the various titration curves drawn by the students are overlaid one on top of the other. Although this may not be the cleanest way of presenting the data, the instructor can nonetheless quickly determine that most of the students drew something that looks like the characteristic "S" shaped titration curve.

We believe that these visualizations can help teachers quickly identify common problems that students are having and allow teachers to provide further instruction to address the issues.

Spreadsheet View   In addition to these visualizations, we recognize that it is sometimes easier to see information in a spreadsheet format. By switching to the spreadsheet view, teachers can select a variety of options to indicate the information they would like to include in a spreadsheet. The results are shown to the teacher. Furthermore, they may be saved as a Microsoft Excel file for further analysis within other software.
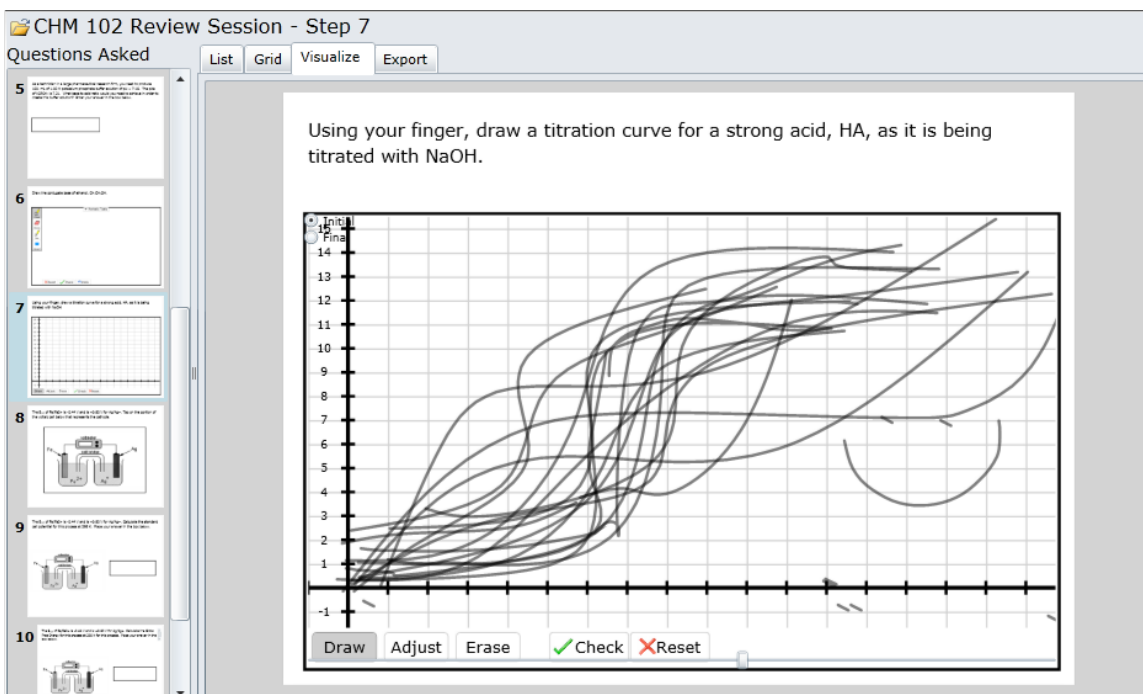
Figure 4.19: Visualization for Graphs. Each graph is rendered semi-transparently on top of each other.

4.2.3 uRespond

The functionality described so far has been asynchronous in nature where teachers create activities, student complete activities and teachers analyze student submissions; however, BeSocratic also contains a system called uRespond that provides a way for teachers to interact with students in real-time. This work is part of a collaboration with the University of North Carolina Wilmington. In a similar manner to Clicker systems, uRespond's goal is to allow teachers to ask BeSocratic questions to students and receive real-time responses. This allows teachers to identify misunderstandings and address them in class instead of having to wait until tests or assignments to identify and fix problems.

Internally, the uRespond system works in a similar fashion to the standard BeSocratic activity model in Figure 4.10. Teachers start by initializing a uRespond session with a name, save location, and roster. This starts an initially empty activity. Once students have loaded the blank activity associated with the uRespond activity, the teacher may either send students previously created activity steps from other activities or send predefined template questions. Figure 4.20 displays an example of the teacher's view during a uRespond session. Template questions and previously-created activity steps are shown along the bottom of the figure. Clicking one will send

Figure 4.20: BeSocratic's uRespond system. uRespond provides a real-time component to BeSocratic where teachers can send students questions from a template bank (bottom left) or from other activities (bottom right). Students who have submitted answers are displayed on the left, and selecting a name loads the student's work.

the question to all the students.

Students respond using the same activity completion tool that was previously described. When a student responds, a new entry into the submission table is added with the student's user key, replay and final submission. The teacher's uRespond client periodically polls the database for any new submissions for the current question. If new submissions are found, the teacher's view is updated with the new submissions as seen along the left side of Figure 4.20. The student's activity also polls the database for additional questions added by the teacher. If a new question is found, it is downloaded and loaded for the student to complete.

We feel that part of the power behind uRespond is the flexibility it provides teachers with. By that, we mean that questions during a uRespond session can be mixed and matched depending on the feedback received from students. For example, teachers may have a series of planned questions that they create in the authoring tool ahead of class. As they send students these questions and start seeing the answers that students respond with, the teacher may want to change the plan and ask a follow-up question that had not initially been planned on. This is where the teachers may select from a predefined question such as a question with text and an ink canvas where they ask student to draw something. Alternatively, teachers can load an appropriate question from another activity or quickly create a new question altogether with the authoring tool. Using uRespond in this manner, teachers can control the flow of class by recognizing difficult concepts and immediately addressing them.

4.3   Supported Devices

BeSocratic currently runs on a variety of devices with varying levels of functionality. BeSocratic requires the Microsoft Silverlight plug-in to run inside of the most common browsers (e.g., Internet Explorer, Chrome, Firefox, and Safari) on Windows and Mac computers. Silverlight was chosen for several reasons. For one, there was already a large code base developed from previous projects for the .NET framework. This allowed for rapid prototyping and deployment of BeSocratic. Furthermore at the time when this project was started, HTML 5 and JavaScript's future was not as well defined as it is currently. JavaScript interpreters at the time ran too slow for the types of visualizations we had envisioned. In recent years, browsers have made great stride in optimizing and hardware-accelerating JavaScript interpreters, which vastly improved the speed of web applications.
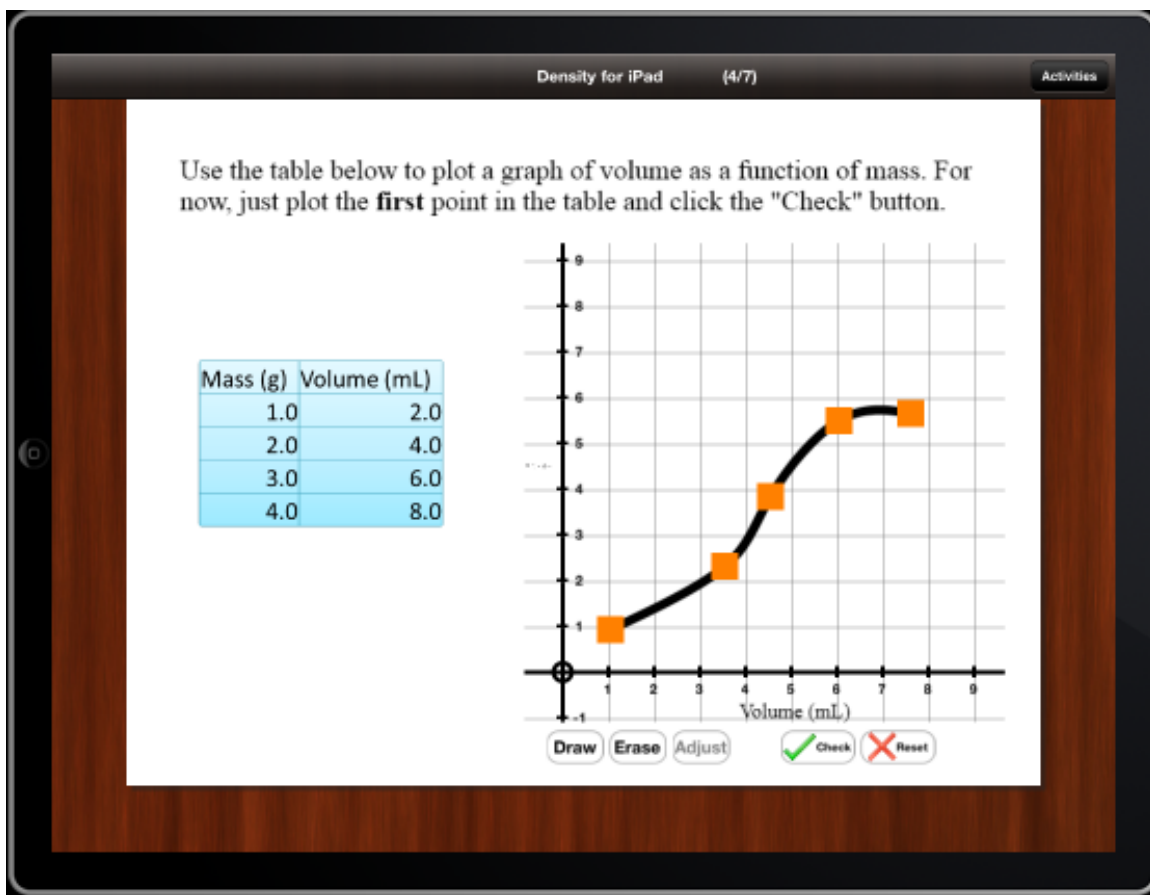
Figure 4.21: The iOS version of BeSocratic with an activity including a SocraticGraph module.

So while current JavaScript interpreters may be able to efficiently complete the types of task we required, through the majority of this research that was not the case, and we chose to develop using Silverlight since its hardware accelerated functions and rendering allowed us to smoothly display the visualizations that we desired.

In its current form, BeSocratic is targeted for use with Tablet PCs and touchscreens; however, it is by no means necessary to use either. We believe that a Tablet PC's digital stylus provides a natural interface that limits the cognitive load sometimes associated with complex mouse and keyboard interfaces. Furthermore, our early results seem to indicate that student learning increases when students make gestures with their fingers using a touch interface.

Based on those early results, we have also developed a prototype iOS application. A screenshot of the app is shown in Figure 4.21. This application only contains the student's viewing tool. We feel that because of the fine control needed to author a BeSocratic activity, iOS devices are not

yet appropriate for tutor authoring. Our app focuses on loading activities created in BeSocratic's authoring tool and uRespond. Students are able to complete activities within the app just as they would using a personal computer. In addition, the app records a replay of the student's work that is fully interoperable with BeSocratic's analysis tools. We are also pursuing the possibility of creating an application for the Android platform as new devices become more widely used.

These applications are possible because of a web API that has been built around BeSocratic's core functionality. This API exposes many database methods, including registering and authenticating users, finding and retrieving active activities for a user, and inserting and updating submissions. The web API also contains methods for automatically performing some of the more complex tasks within specific modules. We decided to provide these so other developers would not have to replicate our work and could deploy apps much faster. These methods include: the graph isomorphism and sub-graph isomorphism algorithms used within OrganicPad and GraphPad, the curve recognition algorithms in SocraticGraphs for converting a raw list of points to a spline, and the curve evaluation algorithms in SocraticGraphs for checking curves. Finally because the iOS devices make adding rich text (i.e., text of various colors, fonts, and sizes) difficult, the web API provides a text-to-image methods, which render XAML or HTML formatted text to a PNG image for use within applications. We are continuing to add some of our complex algorithms to the API in hopes of shortening the development time of new BeSocratic applications for emerging devices.

Chapter 5

Advanced Analysis Tools

BeSocratic provides a variety of ways for teachers to analyze student work. Previously in Section 4.2.2, we described some simpler analysis methods using BeSocratic's In-Class Analysis Tool, which allowed teachers to view quick summaries of student answers. In this section, we describe another set of techniques and tools that allow teachers and researchers to take a closer look at their students' submissions. These analysis techniques allow teachers to identify student strategies and problems. By identifying these problems, teachers may quickly address the problems to avoid further issues from arising. Section 5.1 describes how BeSocratic records student actions within the system so they may be replayed and analyzed. Section 5.2 describes the coding tools available to teachers who want to manually analyze their submissions. Then, Section 5.3 and Section 5.4 detail the methods BeSocratic uses to cluster and visualize student replays and feature vectors. Section 5.5 describes how groups of students can be tracked over time to identify trends. Section 5.6 explains methods to reuse their analysis results to provide more targeted feedback for future submissions. Lastly, Section 5.7 describes how BeSocratic is able to import and analyze data from our system.

5.1 Recording

BeSocratic's ability to analyze data is possible because it records a list of all the actions that a student performs within the system. These actions are recorded at the module level, i.e., each module only records the actions of a student within itself. Examples of such actions include: strokes being added in an ink canvas, text changed within a text box, drawing and checking curves in a
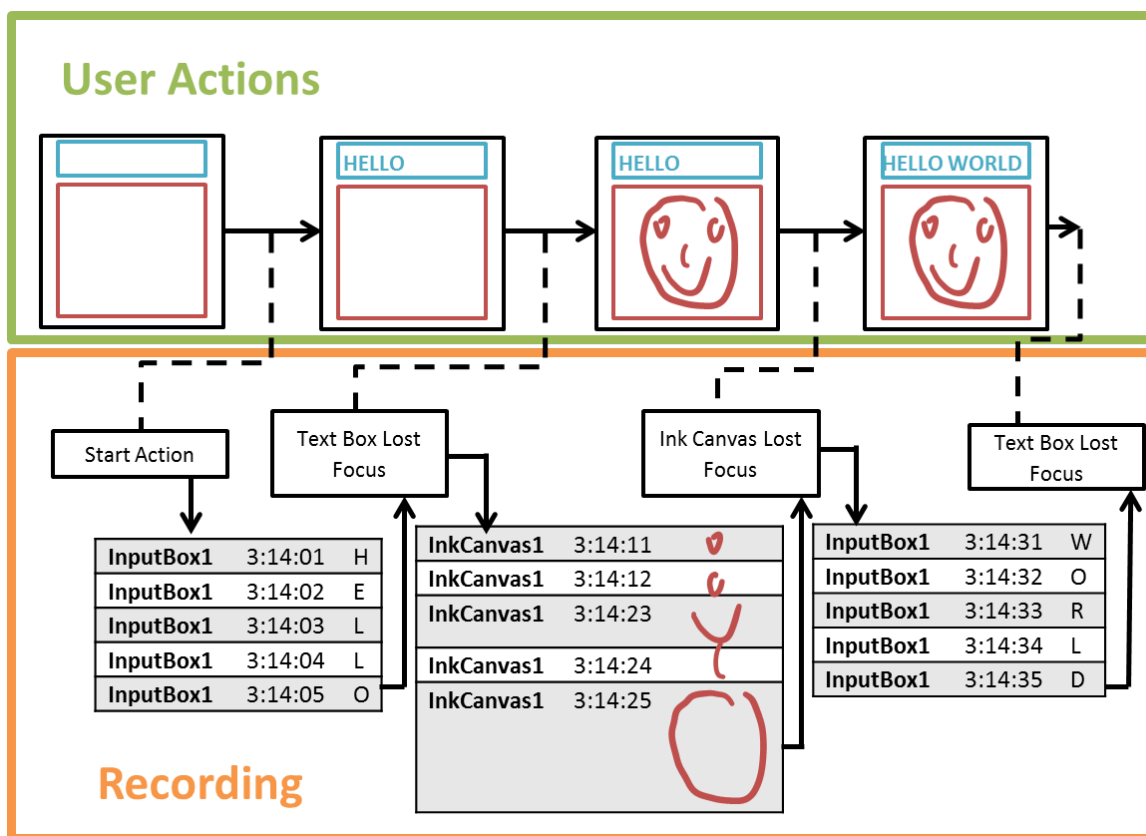
Figure 5.1: An example of the recording process in BeSocratic. The top section shows an example of user actions where a student enters text into a text box, then makes a drawing in an ink canvas, and then makes additional changes to the text box. The lower section shows an example of the data structure that would be used to record these actions.

SocraticGraphs module, and adding atoms and bonds in an OrganicPad module. It is also important to note that each action is timestamped, which allows for an accurate replay of the student work. While a student completes an activity, each action that is performed bubbles up from the module to a higher level observer that concatenates each action into one unified recording. Figure 5.1 shows an example of this process. In the example, a student is entering text into a text box module and switches to drawing inside of an ink canvas module. Each action is concatenated thus creating an accurate recording of all the actions of the student.

In addition, the current BeSocratic modules are designed to record only the changes in state. For example, only the changes to a text box (i.e., text added or removed) are saved. This is in contrast to making copies of the text box after every action. By only recording the differences between actions, the recordings require relatively fewer bytes. This makes transmitting submissions

faster for both the students uploading their solutions and for teachers downloading and analyzing submissions.

5.2 Coding

BeSocratic's advanced analysis tools are possible because each action in a student's recording provides a basic form of coding. As described in Section 5.1, each action is recorded as a serialized timestamped string (e.g., ink stroke drawn, text added, curve checked). Using these codes, BeSocratic is able to perform a variety of analysis techniques. While the codes are created automatically (i.e., without input from the teacher), we recognize that the automatic codes only go so far. They often fall short in understanding the context of the actions students are performing, and teachers often desire the ability to replay their students' work and mark the submissions with context specific codes. BeSocratic provides two tools to enable this functionality, Text Coder and Replay Coder, for coding text data and generic replays respectively. These tools allow teachers to manually code their data and use the resulting coded submissions with BeSocratic's automatic analysis tools. This section describes these two coding tools. We will discuss some of the data that we have coded with these tools later in the Results section.

5.2.1 Coding Text

Following BeSocratic's focus on the Socratic method, our activities in BeSocratic often involve asking the students open-ended questions that students respond to by typing simple text responses. Since analyzing and coding free text responses is difficult to automate, researchers often manually code text-based responses. BeSocratic's Text Coder allows teachers to do just that. The tool enables teachers to load text responses generated by students in BeSocratic or load text responses from other sources via a comma separated value or Microsoft Excel file. Figure 5.2 shows an example of text being coded with BeSocratic's Text Coder. On the left side, teachers select the question they would like to code. BeSocratic then presents the teacher with a list of all the students who have answered that question along with a customizable list of available codes on the right. When a teacher selects a student's id, the student's answer is presented in the center of the tool. At this point, teachers can code the answer by highlighting parts of the student's text with their mouse cursor and clicking on a code from the right panel. When a selection is coded, an instance of that

52
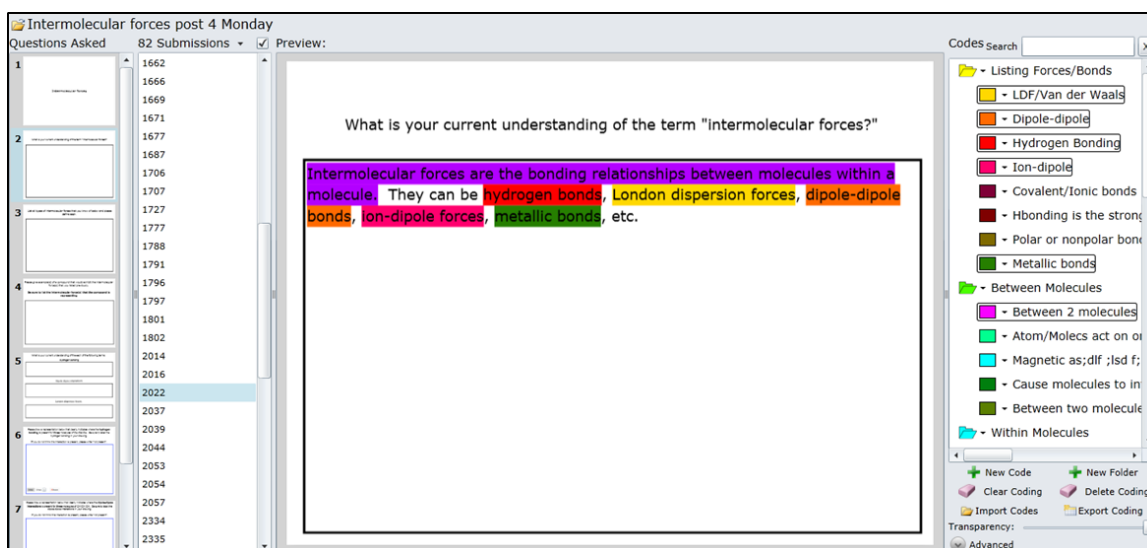
Figure 5.2: Screenshot of a text submission being coded.

code is created with the code key, coder key, and starting and ending positions of the selection. This entire item is added to a dictionary and saved into the database. This allows for teachers to load their codes from any computer and view how other teachers have coded the submissions without having to manually share files. The codes themselves can be renamed, given a color, and put into folders.

Once the teacher has coded their responses, the tool can export the results so they may be further analyzed by BeSocratic or within other tools such as Microsoft Excel or SPSS. The exported data can be generated with a variety of features. This includes being able to export the used codes, each codes frequency, and the text associated with each code. Furthermore, BeSocratic allows multiple teachers to code the answers, and BeSocratic can compare the coding in order to determine inter-rater reliability.

5.2.2 Coding Replays

BeSocratic's Replay Coder allows teachers to place codes throughout a submission replay. The Replay Coder uses an interface very similar to the previously described Text Coder. Figure 5.3 shows an example of manually added codes for the replay of a graph being drawn. The Replay Coder uses the standard BeSocratic replay method of deserializing each action in a recording and displaying the actions to the teacher during replay. If the teacher would like to add a manual code

53
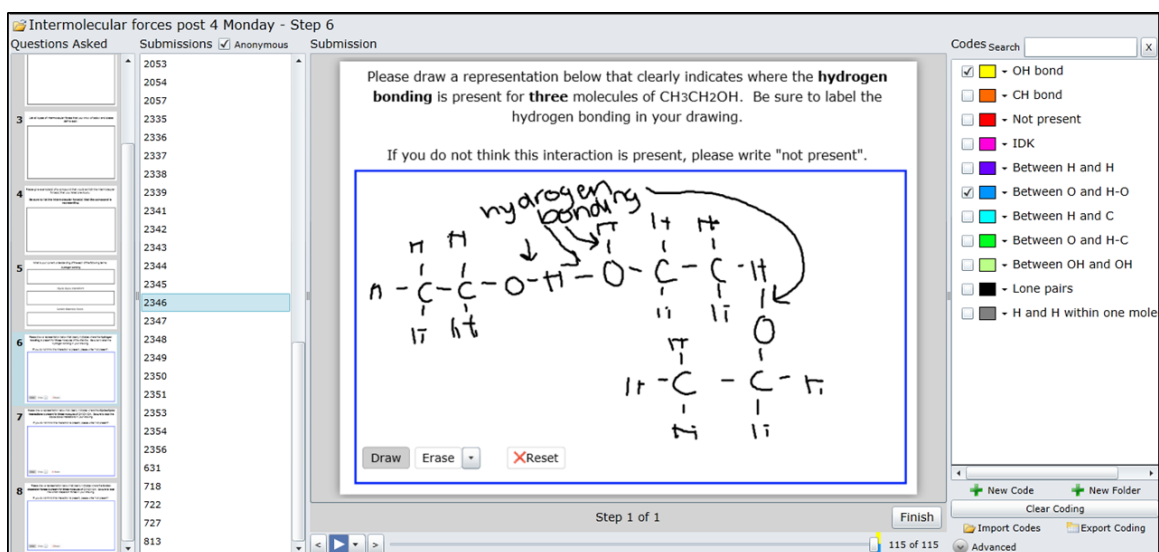
Figure 5.3: Screenshot of a replay being manually coded.
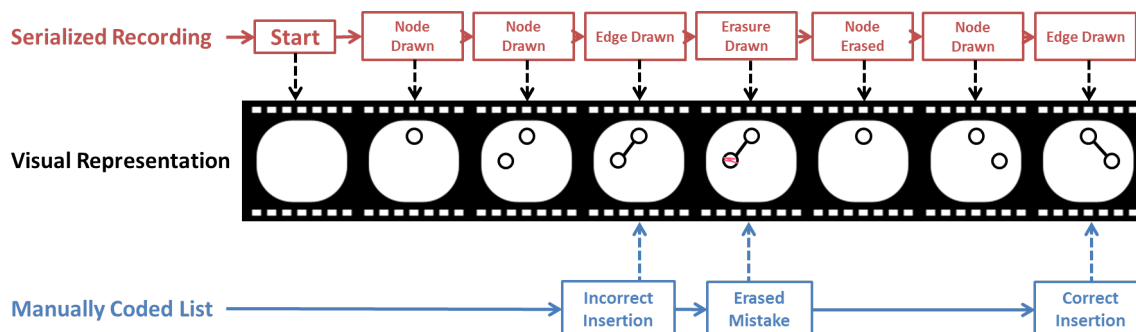
Figure 5.4: An example of a replay sequence using GraphPad in which additional codes where manually.

during replay, the teacher clicks the pause button and checks one or more of the codes from a list along the right side of the screen. When the teacher marks the code, a reference to the code is placed within an ordered list keyed with the current replay step as seen in Figure 5.4. This list is then saved into the database. Just like the Text Coder, codes can be given names, colors, and can be grouped into folders. In addition, each added code is displayed along the bottom of the screen behind the replay slider. This allows teachers to quickly identify locations within the replay where they added codes.

Once submissions have been coded, they may be exported in several formats. Notably, teachers can export the codes into sequences that may be used in BeSocratic's sequence analysis tools described in Section 5.3. Furthermore just as with the Text Coder, the Replay Coder allows

multiple teachers to independently code submissions. Then the Replay Coder can compare each teacher's coding with those of the other teachers in order to establish inter-rater reliability.

5.3   Sequence Clustering

One of BeSocratic's primary goals is to explore clustering sequences of student actions in order to discover student problem-solving strategies. To research this, we created the Sequence Clustering tool. The tool works in three logical steps: data selection, filtering, and clustering.

The data selection step involves loading the data to be analyzed. This data may either come from BeSocratic itself or from outside data sources. BeSocratic data can either come from the automatically generated recording or from manually coded sequences. Outside sources must conform to the XML format shown in Listing 5.4 and discussed further in Section 5.7. Either way, the data is imported as lists of timestamped student actions.

Once the data is loaded, teachers can filter and categorize the individual action types. For example, it may be beneficial to group two positive feedback message actions under one category labeled "Correct Answer".

After the categories have been set, the sequences are ready to be visualized and clustered. In order to visualize a large amount of student sequence data at one time, BeSocratic displays each sequence in a compact form as shown in Figure 5.5. When creating these sequence visualizations, BeSocratic first creates a list of the unique codes used in all of the selected submission sequences. Then, BeSocratic maps a unique color to each unique code so that the codes can be represented by colored bars displayed in a list (Figure 5.5). These colors can also be manually assigned if desired. Tool-tips are implemented such that when the mouse cursor hovers over a particular code (i.e., colored bar), a small popup appears with the code's description.

At their core, clustering algorithms rely on the ability to accurately determine similarity between observations. For our purposes, the observations are the student replays themselves, so we needed a way to measure the similarity between 2 replays. We explored the use of two such measures: edit distance and hidden Markov model likelihood.
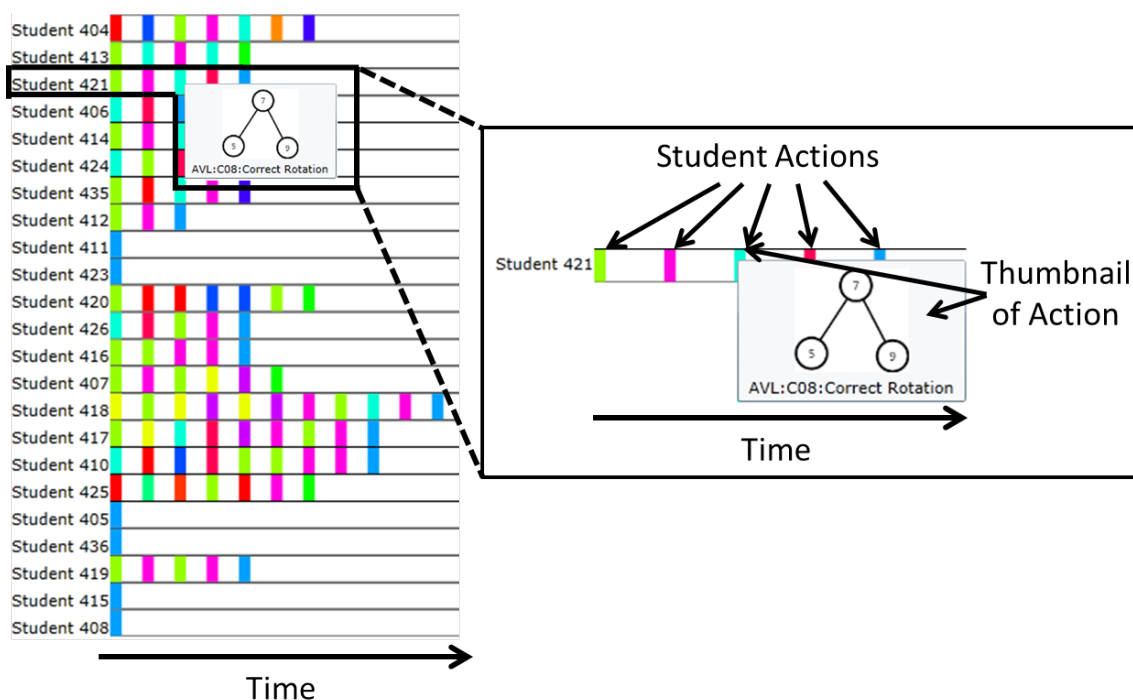
Figure 5.5: Compact visual representation of sequences. Actions are color coded by type. Hovering over an action gives a thumbnail of the actual action in addition to a text description.

5.3.1 Edit Distance Based Clustering

As described in Chapter 2, the edit distance, or Levenshtein distance, is a metric for measuring the amount of difference between two string sequences. We apply this technique to our replay sequence such that we determine how many student actions would need to be edited to convert one student's replay into another student's replay. Our Levenshtein distance measure is modified slightly from the original in that it includes the time of the actions into account. For example if two sequences have the same actions but one sequence took longer, the distance would be greater than two sequences with the same time characteristics. By performing this operation pairwise over each pair of replays, we arrive at a matrix of pairwise distances between replays. Inverting each value results in a similarity matrix where entries in the matrix correspond to the degree of similarity between replays. Using this matrix of values, BeSocratic can cluster and visualize the sequences several ways including spectral ordering, hierarchical clustering, spectral clustering, heat maps, and force-directed graphs. These tools were developed in an iterative process based on needs that we saw when analyzing student data. Each one of these tools was designed reveal different aspects of

56
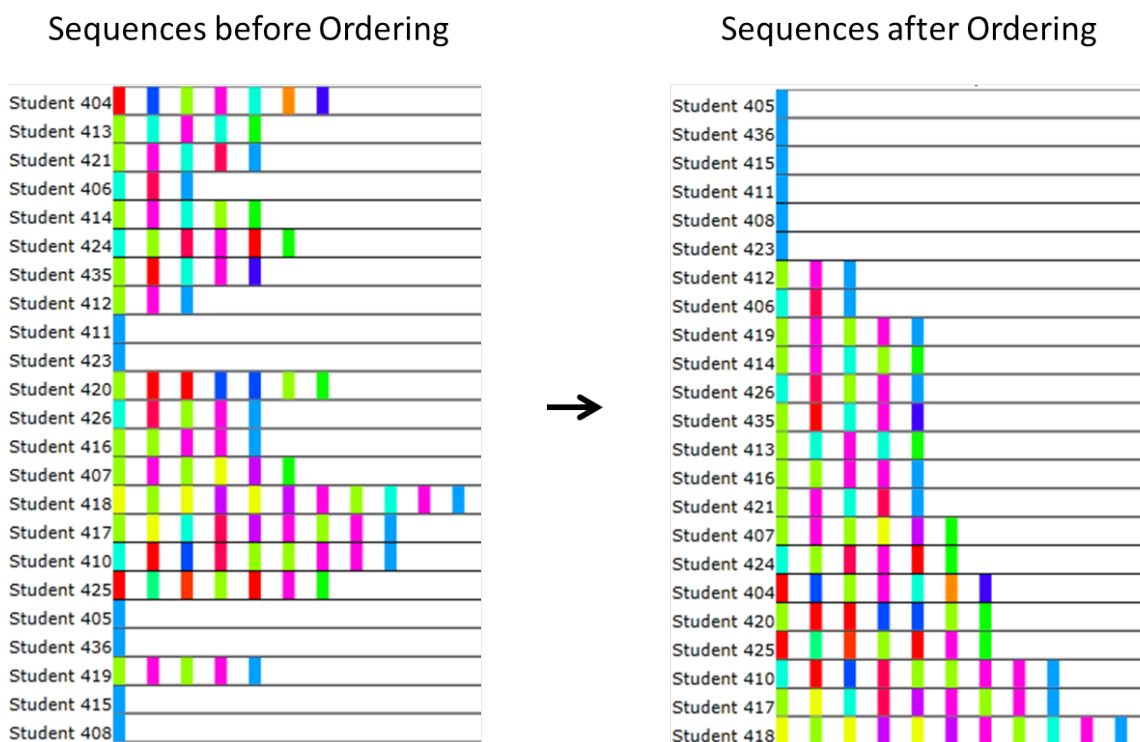
Figure 5.6: Screenshots of student sequences before and after spectral ordering.

the students' submissions.

Spectral Ordering   With our compact sequence representation, BeSocratic displays multiple student sequences in a list. However, visually identifying similarities and differences within the sequences can be difficult, especially with large group sizes. Ideally, BeSocratic should assist teachers in identifying the similarities and differences. For example, it would be desirable to order the sequences so that similar sequences are placed closer together than dissimilar sequences. With such an ordering, we would expect to see groups of sequences that resemble each other grouped together. To perform this task, we use findings from spectral analysis. As described in Chapter 2, spectral clustering is a set of techniques that use a similarity matrix to perform a dimensionality reduction for clustering in fewer dimensions than the input. To accomplish this ordering, we use spectral analysis to reduce the dimensionality of our sequences to one dimension, which we use to embed the sequences into a list. The first step is the creation of a similarity matrix between all of our sequences. This is accomplished by creating an nxn adjacency matrix for our n sequences. Each matrix value is the similarity between each pair of sequences based on their Levenshtein distance.

57
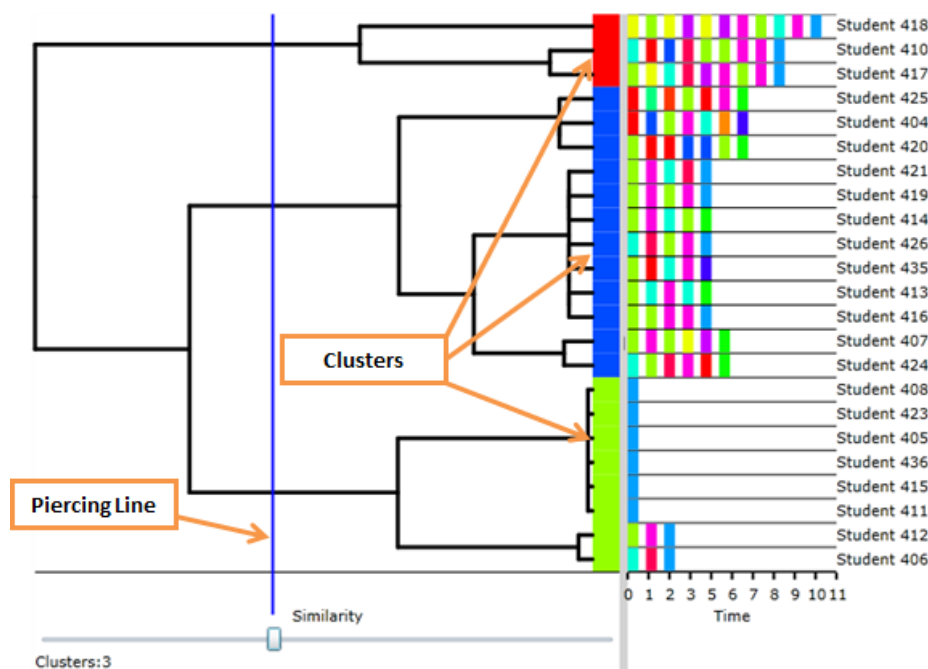
Figure 5.7: Hierarchical visualization of clusterings using a dendrogram. Sliding the piercing line changes the clustering of the sequences. In this example, the piercing line intersects 3 branches of the tree and thus separates the sequences into 3 clusters.

BeSocratic then uses spectral methods to reduce the dimensionality of the sequence to one dimension. This one-dimensional ordering is then used to reorder the sequences into a list. Figure 5.6 gives an example of student data before and after ordering.

Dendrograms Dendrograms are tree diagrams used to illustrate the arrangement of the clusters produced using hierarchical clustering. Dendrograms are commonly used in evolutionary biology to visualize the similarity between various species. We can use the same visualization technique to show the results of our own hierarchical clustering. We generate our dendrograms using a "bottom-up" or agglomerative algorithm that initially starts with each sequence placed in its own set. The algorithm repeatedly finds the two most similar sets using the matrix of similarity values and groups them inside their own set. This process continues until there is one set left consisting of all of the sequences. Once finished, the algorithm has built a tree of sets that range from a root with all sequences to the bottom leaves with only one sequence each. A dendrogram of such a tree can be seen in Figure 5.7. Each student sequence is connected to each other through the tree. The height of the line from the bottom level of the tree corresponds to the dissimilarity between the two sets
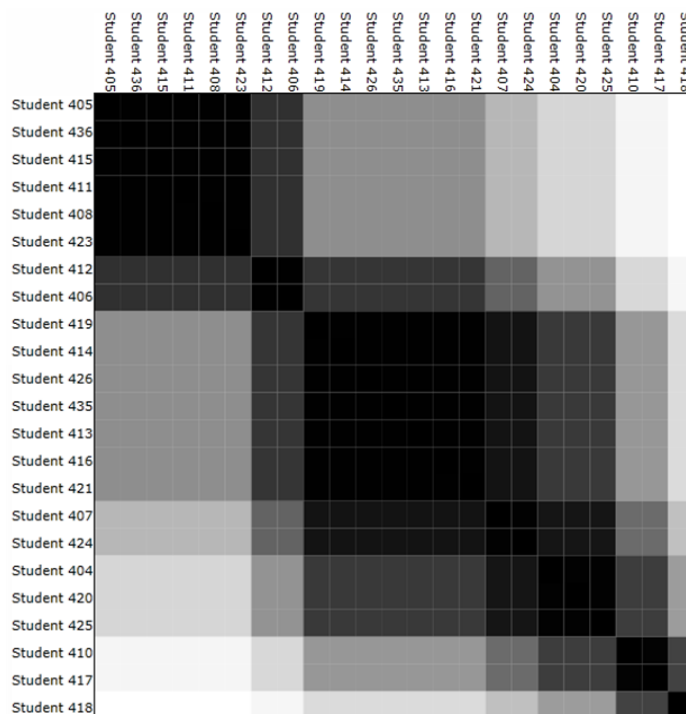
Figure 5.8: Heat map generated from 23 student sequences. Dark cell shadings correspond to high similarity between sequences. Note that the large dark square areas may indicate an underlying cluster in the data.

it joins. For example, notice that the root level is the furthest from the bottom level since the root connects the two most dissimilar sets. With our tool, users may move a line to 'pierce' the dendrograms. Clusters of sequences corresponding to the dendrograms branches that were pierced are shaded with the same color. Notice the three branches pierced and the resulting three clusters in Figure 5.7. Also note that building a dendrogram also orders the sequences such that similar sequences are grouped closer together.

Heat Maps Another visualization BeSocratic generates for student data is a heat map. Our heat maps are created by first ordering the sequences using either the spectral or hierarchical methods described above. Using this ordered set of sequences, we can rearrange the rows and columns of our similarity matrix and render the matrix where each cell is shaded based upon the similarity value present. An example is shown in Figure 5.8. Darker cells correspond to higher similarity. We can expect to see dark blocks along the diagonal of groups where sequences were similar amongst themselves. This can be used to identify groups of students with similar sequences and thus similar
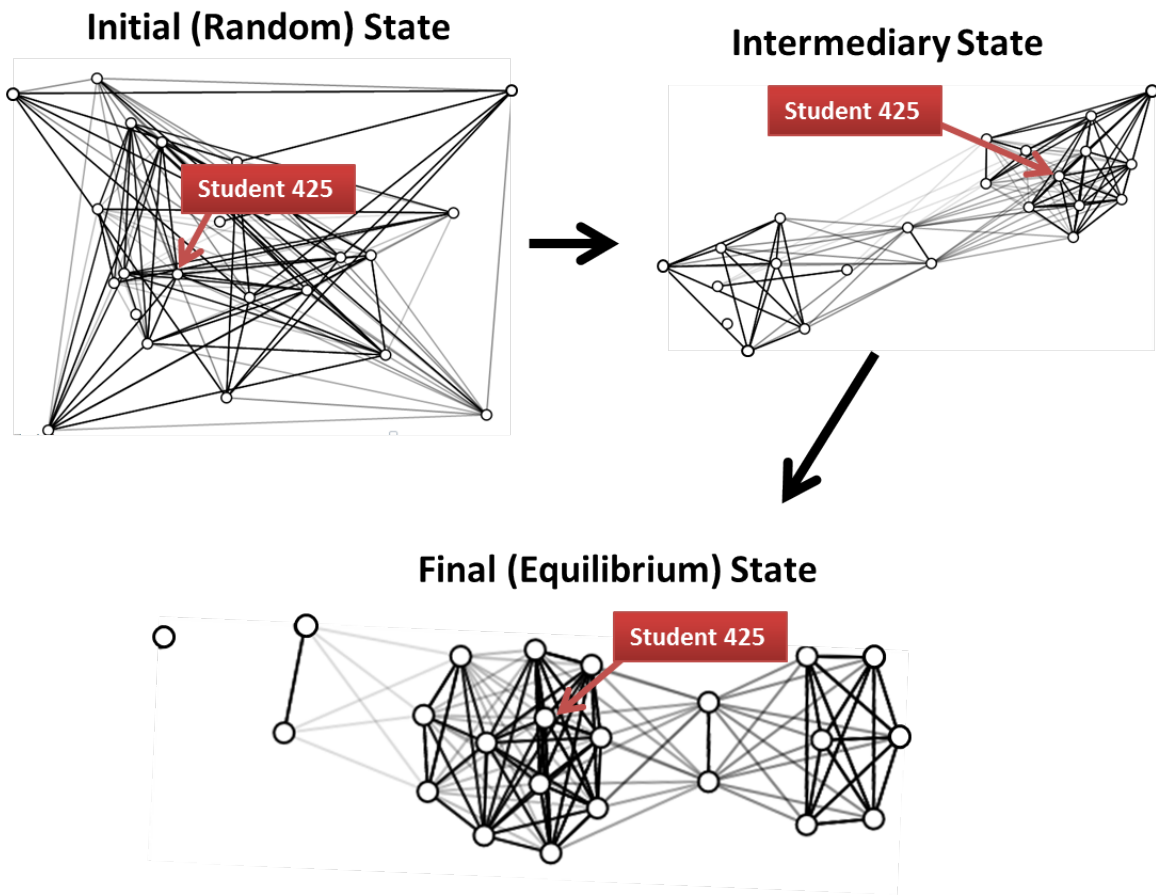
Figure 5.9: 3 stages (initial, intermediate, final) within a force-directed layout of student sequences.

strategies.

Force-directed Layout In addition to the previously described visualizations, BeSocratic contains a force-directed layout algorithm, which may help identify groups of similar students. Similar to the other layout algorithms, our system uses the similarity matrix described above as a graph's adjacency matrix, i.e., a complete graph with edges corresponding to the similarity value between the connected nodes. If we treat the graph as a large spring and charge system where edges are springs (whose strength corresponds to the similarity) and nodes are charged particles, we can run a physical simulation on the system and expect to see nodes (i.e., sequences) that are similar to be pulled together in groups by the springs. Conversely, nodes (or groups or nodes) that are not similar will be separated by the charge forces. Once the simulation has reached equilibrium, a graph displaying relative sequence similarity will have formed. A threshold can also be applied to remove

some of the weaker edges. This can help spread the graph into better defined clusters. An example of such a layout being applied to student sequences is shown in Figure 5.9.

Difficulties   After using these edit distance based methods to cluster different student data sets, a few difficulties became clear. For one, the edit distance metric tends to be mainly influenced by the insertion and deletion costs. Because the sizes of the sequences are often different, the distance scores become large when there is a size difference despite the fact that the types of actions may be very similar. To demonstrate, take three students (S1, S2, and S3) who performed sequences of two actions (A and B). S1 performs the actions ABABAB. S2 performs the actions ABABABABAB. S3 performs the actions ABABABBBBB. Because S2 performed an extra series of ABAB actions, the edit distance between S1 and S2 is the cost of 4 insertion operations. This is the same for the cost between S1 and S3 who performed an extra series of BBBB actions. So S2 and S3 would be considered the same "distance" from S1 even though S2 simply made a few more repeats of the AB action pattern. This is problematic. As we study the data using edit distance values, we find that the edit distance could not determine when similar patterns of actions were being performed by a student. Students are being separated primarily by the length of their sequences instead of the action types being performed. Attempts to tune the edit distance operations and costs to account for this are difficult as questions types can vary greatly. This leads us to question the value of using edit distances to measure similarity between student sequences. Instead, we switched our focus to using hidden Markov models (HMMs) to analyze the student sequences. By nature, HMMs do not fall into this trap since the edges within the model can loop to previous nodes and thus represent a repeating pattern of this nature.

5.3.1.1   Hidden Markov Model Based Clustering

Compared to edit distance, Hidden Markov models use a fundamentally different method for determining the similarity between sequences and sets of sequences. As others have done [12, 41, 66], we will form the analogy that students have hidden mental states and the only things that are visible to the teacher are the students' actions within BeSocratic. By creating hidden Markov models to fit students' actions, we are hopefully able to gain insights into the underlying hidden states of the model and understand what students are thinking while they answer questions.

We first tried to accomplish this process using a variation of the *k-means* clustering algo-

```
Randomly Initialize K HMMs
While(stopping criteria not met)
{
        foreach(Student Sequence)
                Find 'closest' HMM based on Likelihood Calculation

        foreach(HMM)
                Optimize with Baum-Welch and the Closest Student Sequences
}
```

Listing 5.1: Pseudo-code for BeSocratic's K-Means HMM clustering algorithm.

rithms (pseudo-code shown in Listing 5.1). BeSocratic first generates k randomly initialized HMMs. Next, the algorithm proceeds in a manner similar to the standard *k-means* algorithms where the algorithm alternates between fitting the sequences to the "closest" HMM using the likelihood calculation and then readjusting each HMM's parameters using the Baum-Welch algorithm. As with the traditional *k-means* algorithm, the outcome of the clustering is highly dependent on the starting parameters of the the models; so to find "good" clusters, the tool runs the algorithm many times and uses internal evaluation measures such as the Davies-Bouldin index and Dunn index to find the "best" clustering. The end result of the clustering is groups of one or more replays each of which correspond to a hidden Markov model. Of note, BeSocratic runs this algorithm in parallel. This is possible since each run of the *k-means* algorithm is independent of the other and don't rely on the results of other runs.

After using this method with several sets of student replays, some issues become clear. Namely, the nondeterministic nature of HMM modeling makes automatically generating k clusters difficult as multiple runs often do not results in the same models, especially when k is high. Furthermore, teachers rarely know k ahead of time. So instead, we use a divisive hierarchical clustering method, which repeatedly split sets of sequences using the algorithm described above and a k value of 2. While this method works better, several ideas emerge when researchers use the tool. Watching researchers analyze the hierarchical results, we find that they often reach branches in the tree where they do not want to split the sequences, yet they want to split other branches further. Furthermore, we find that researchers often desire the ability to merge branches that were separated by the algorithm. We conclude that discovery of strategies needs to be done in a semi-supervised fashion. By this, we mean one that the user can control the hierarchical clustering as it proceeded. Our tool allows researchers to "split" and "merge" clusters based on their insights into how the sequences are
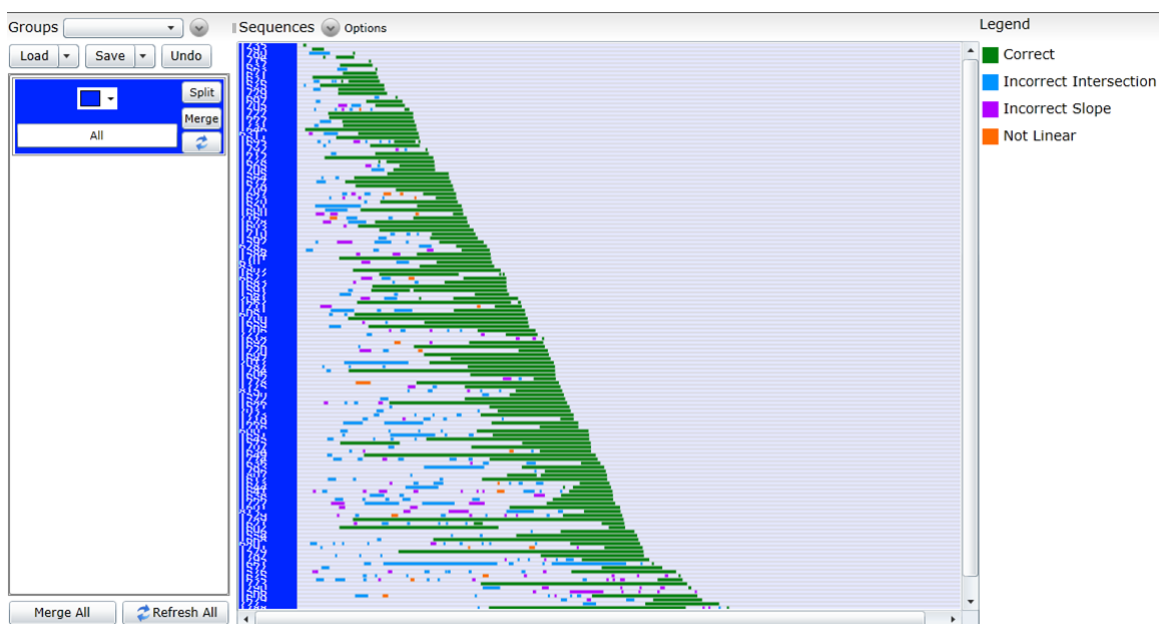
Figure 5.10: Initial state of BeSocratic's Sequence Clustering Tool. Sequences are shown in the center. Each row represents a student's replay. A legend of colors for each event is shown on the right.

being clustered.

Figure 5.10 shows BeSocratic's Sequence Clustering tool. As seen in the figure, the tool starts with all the sequences in one large set. In the center, sequences are displayed in the previously described list. Along the left side, each cluster is represented by a box containing a color picker, a text box for labeling, and split, merge, and refresh buttons. Clicking the "Split" button, splits the cluster into two groups using the *k-means* algorithm previously described. After many splits, the example data is grouped as seen in figure 5.11. At this point, the teacher decided that none of the groups needed to be split into finer clusters. Instead, they noticed that the data seemed to divide into 3 broad groups and used the "Merge" button to combine groups into the 3 groups seen in figure 5.12. Using this simple method of splitting and merging clusters, teachers can explore, identify, and summarize student strategies. In Section 6, we will show how this tool is being used to identify strategies with data collected within BeSocratic.

This tool relies on teachers to identify and label clusters. This is done using a combination of directly looking at the sequences in each group and looking at the hidden Markov models themselves. Using Figure 5.11 as an example, teachers split the sequences to finer and finer clusters. Eventually, patterns emerged within the clusters. Perhaps the first and easiest group to identify is the "Correct"
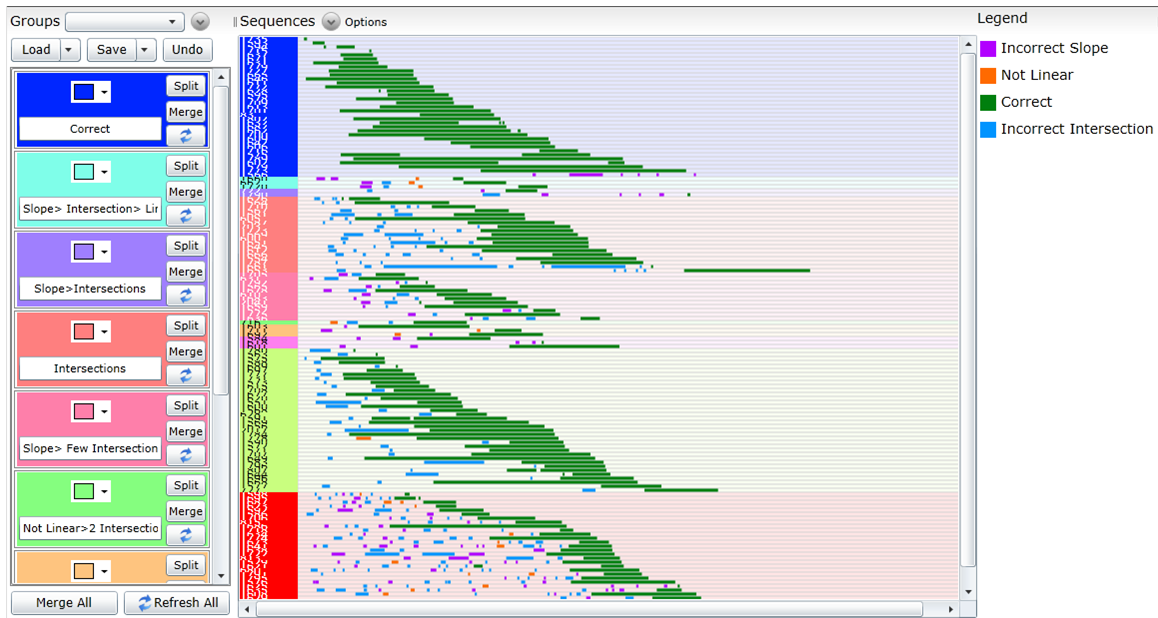
Figure 5.11: Sequences that have been fully clustered. No further splits were desired. By looking at the differences between these clusters, it is possible to identify groups of students who are at risk for various reasons.
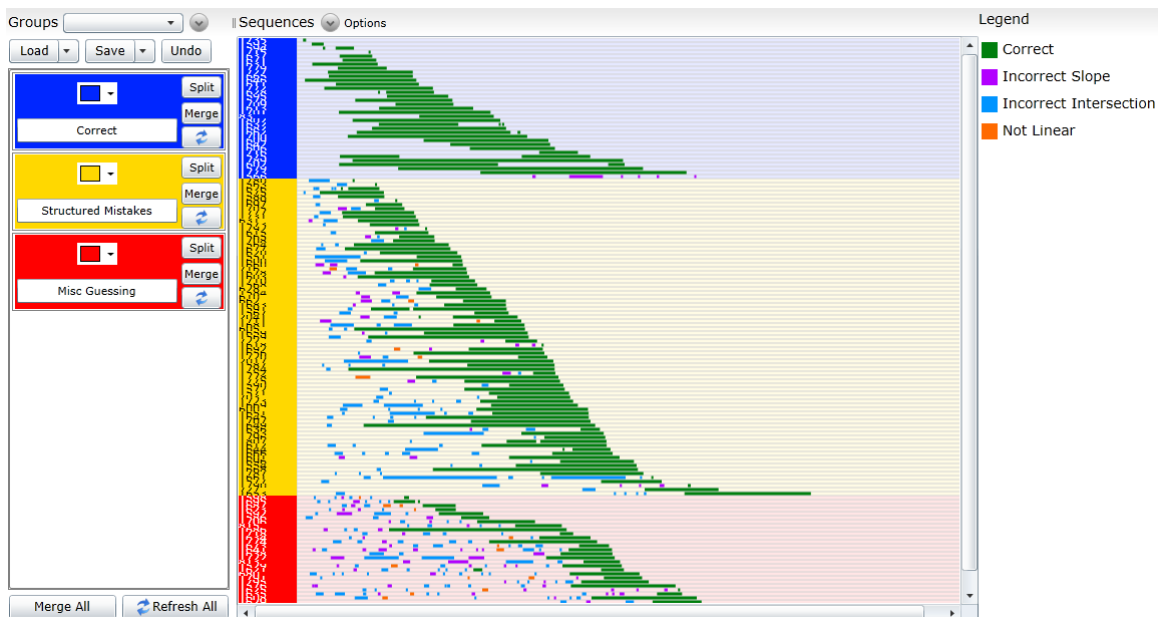
Figure 5.12: Sequences merged into 3 groups: Correct (blue), Structured Mistakes (yellow), Misc Guessing (red)
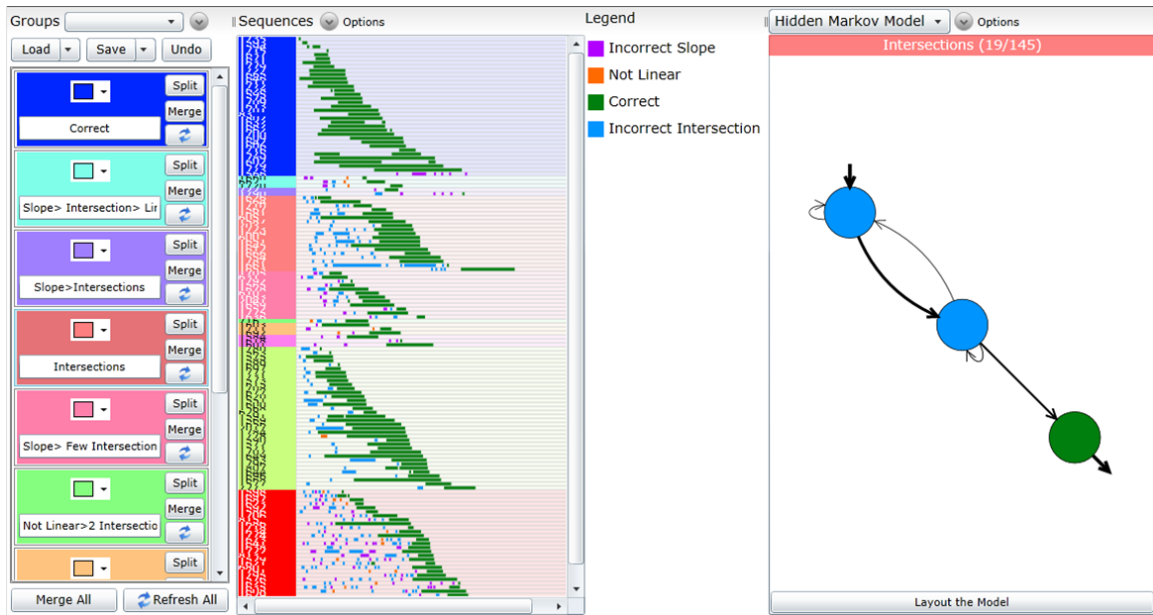
Figure 5.13: 3 State Hidden Markov Model for the Intersection Group

group (shown in blue). All of the sequences within that (blue) group has only 1 (green) "correct" action. In addition, none of the other clusters contain a sequence with only a "correct" action. Therefore, the teacher label is able to label cluster with "Correct" and determine that no further splitting is necessary for that group. Another example of this process is seen in the "Intersections" (light red) group. In this group, the teacher recognizes that each sequence contained 2 or more (blue) "Incorrect Intersection" actions followed by a "Correct" (green) action. None of the other groups contain sequence with this pattern, so the cluster was labeled with "Intersections".

While this method can be effective, sometimes it is difficult to see the patterns within a group. In this case, we find it useful to look at the hidden Markov model directly. After all, each HMM models the patterns within a group of sequences so that it maximizes the likelihood of the sequences being generated from the model. Figure 5.13 shows an example of an HMM being displayed for the previously described "Intersections" group. This hidden Markov model was generated by optimizing a model for all of the sequences in the "Intersections" group using the Baum-Welch algorithm. This figure shows (using a starting arrow) that all of the students start at the top left (blue) node, which represents the student making an "Incorrect Intersection" action. Then the student appears to transition the other blue node at which point they either draw the correct answer or make further incorrect intersection actions. It is also noticeable that all of the sequences end with

65
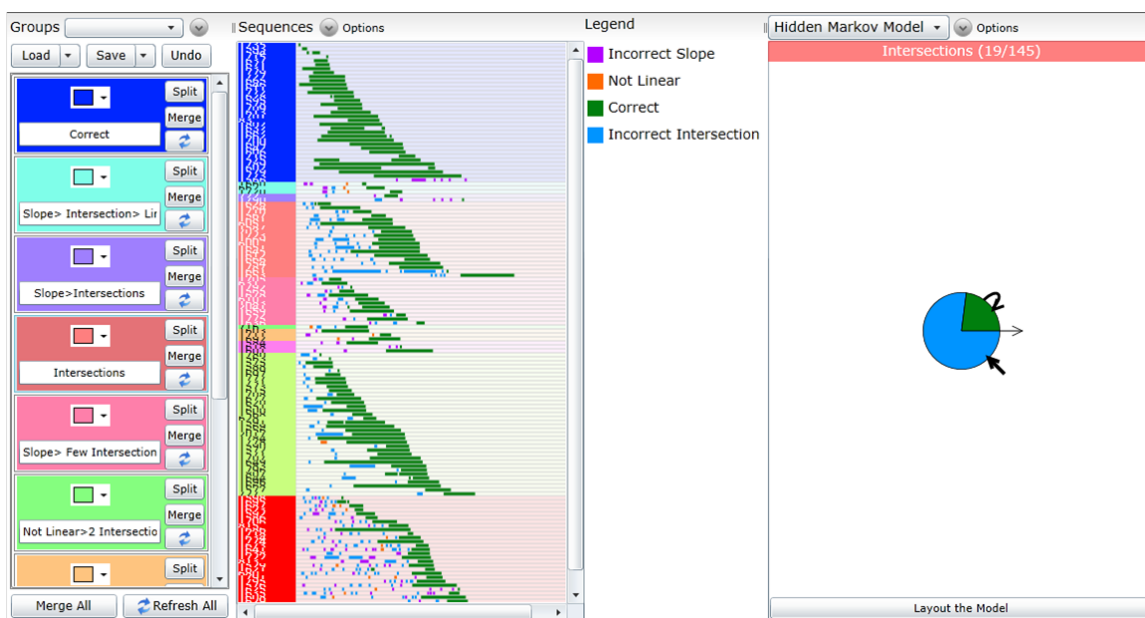
Figure 5.14: Hidden Markov model with too few states.

a green correct action. In this visualization, each state is represented by a circle. The circles are filled with a pie chart showing the emission probabilities. In our example, each state simply has 1 emission with a probability of 100%. State transitions are displayed as arrows between states. Thicker arrows represent transitions with higher probabilities. Arrows pointing directly into a state represent the initial probabilities of each state. Arrows pointing directly out of a state represent the probability of the sequence ending at that state. Teachers may also choose to display the actual probabilities, but they have been disabled for simplicity in these figures.

Choosing the Number of States This process brings up a question previously left out of the discussion, "How many states should each hidden Markov model contain?" Modeling with too few states results in a model with states that are needlessly combined as seen in Figure 5.14. In this scenario, it becomes difficult, if not impossible, to unambiguously determine the strategy of a group. Furthermore, the sequences may be modeled with too many states as seen in Figure 5.15. In this case, the model over-fits the sequences, and determining strategies becomes more difficult.

Of course, teachers can manually adjust the number of states and "dial in" to a reasonable number, but we wanted to provide an automatic way to find that number. Common approaches to solve this problem use measures such as the Akaike information criterion (AIC) or the Bayesian
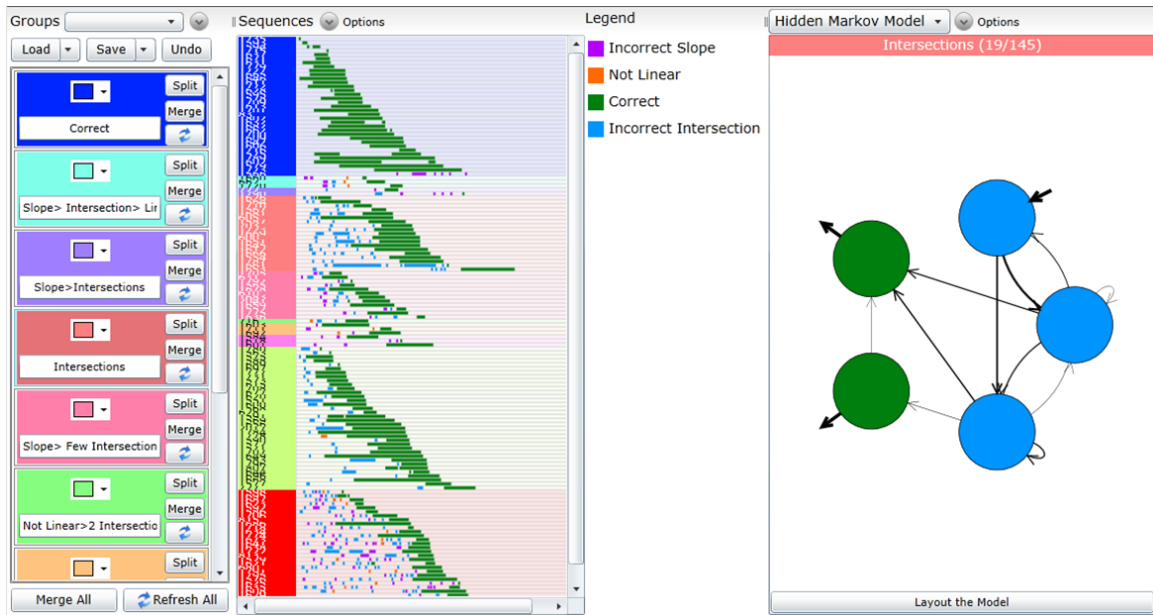
Figure 5.15: Hidden Markov model with too many states.

information criterion (BIC) to determine the number of states. In short, these measures weigh the balance between the goodness of fit for a model against the number of variables in the model. This seems reasonable since adding variables to a model will likely increase the goodness of fit. So by accounting and biasing the goodness of fit based on the number of variables, these measures can be used to select better models. Unfortunately, these measures did not prove to be effective for our needs. They tended to be inconsistent in the number of states to use and often used too few states. Instead, we find a simple elbow metric to be most effective. As the number of states and thus variables increase, the overall likelihood that the sequences were generated by the model increases. However, the likelihood eventually plateaus and increases a negligible amount as more variables are added to the model. By using an elbow metric, we select the model whose likelihood gain is maximized. Figure 5.1 shows an example of the AIC and likelihood gap values for the example in Figure 5.11. Notice that if we were to use the AIC criterion, the model would only have 1 state. If we used the elbow criteria we would select the model with 3 states. We have found empirically that this elbow metric performs better with our data.

Table 5.1: Example State Selection Data

States	Variables	Likelihood	Gap	AIC
1	12	0.88	0.88	24.11
2	21	2.45	1.57	41.22
3	32	4.17	1.72	62.75
4	45	4.18	0.01	88.76
5	60	4.18	0	118.75
6	77	4.18	0	152.75
7	96	4.18	0	190.75

5.4   Feature Vector Clustering

While BeSocratic's primary focus was to explore the possibilities of clustering sequences of student actions, we recognized that we would like to use more traditional feature vector clustering in some situations. To round out BeSocratic's analysis features, we decided to build a feature vector clustering tool into the system. This tool allows teachers to import their feature vector data from BeSocratic or from outside sources. The data can either be in the form of an Excel file, Comma Separated Value file, or our Feature Vector XML format (example shown in Listing 1 and described further in Section 5.7).

Once data is loaded into the tool, the features are displayed in a series of rows and columns as seen in Figure reffvs. Each row represents the feature vector for an individual student. Each column represents an individual feature (e.g., Button 1 Clicked, Incorrectly Drawn Graph, etc.). Cells are shaded darker when the feature has a higher frequency (e.g., Button 1 was clicked many times). Often times, the data sources analyzed have hierarchical data where features are logically grouped together into categories (e.g., Button 2 and Button 3 perform similar actions). We wanted a way to visualize this hierarchy between features. The tool performs this by first generating K spatially separated colors for the K top level categories. Then for each category, the tool generates a monotone color palette based on the category's color. Each subcategory and feature is then assigned one of these monotone colors. The result is a more visually appeasing structure where groups (categories) of features are more easily recognized.

This tool contains other options for teachers to choose in order to customize the visualization. These include changing the width and height of each row, showing and hiding features and categories, collapsing and expanding categories, re-categorizing features, and assigning custom colors. Teachers also have the ability to switch between viewing the data as variable or boolean (i.e., whether the

68
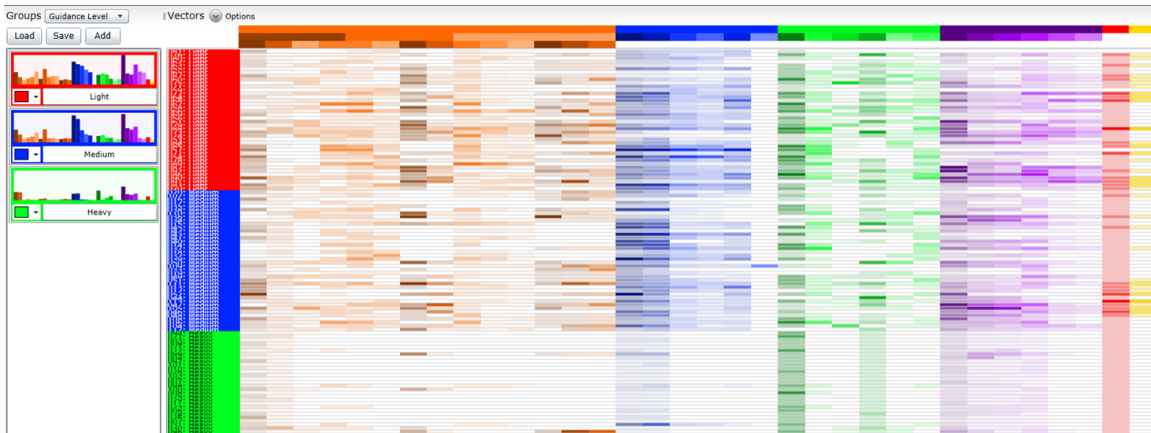
Figure 5.16: 3 groups of feature vectors visualized in a grid. Each row represents a student. Each column represents a feature.

feature is present or not).

In addition to simply visualizing the feature vectors, we implemented a set of clustering algorithms with aims to groups similarly performing students. Since often times teachers will not know the number of clusters to group their students into, we focused our tool on hierarchical clustering. Figure 5.17 shows an example of data that has undergone agglomerative clustering using a Manhattan distance metric and complete linkage criteria. The right side of the screen displays the dendrogram generated from the clustering. By dragging the bottom slider, teachers set the number of clusters in which to group the data. In the example, the teacher has chosen to split the feature vectors into 3 clusters: red, blue and green. In additions, the left column contains a bar chart for each cluster. This bar chart shows the average frequency of each feature. By comparing bar charts between clusters, teachers can identify the differences between clusters.

The Feature Vector Clustering tool contains several methods and options for clustering. Teachers can choose to cluster using agglomerative hierarchical clustering, divisive hierarchical clustering, or k-means clustering. For agglomerative clustering, teachers choose the metric to use (i.e., Euclidean distance, Squared Euclidean distance, Manhattan distance, cosine similarity, or Jaccard distance) and the criteria to use (i.e., maximum, minimum, or mean). Teachers can also choose to first perform a spectral transform on the data and then perform clustering of those feature vectors. For divisive clustering, we employ spectral techniques to repeatedly split the data into 2 groups. Finally, we implemented the k-means algorithm to try and cluster the data if k is known. Teachers also have the option to first transform the data using spectral methods and then use k-means.
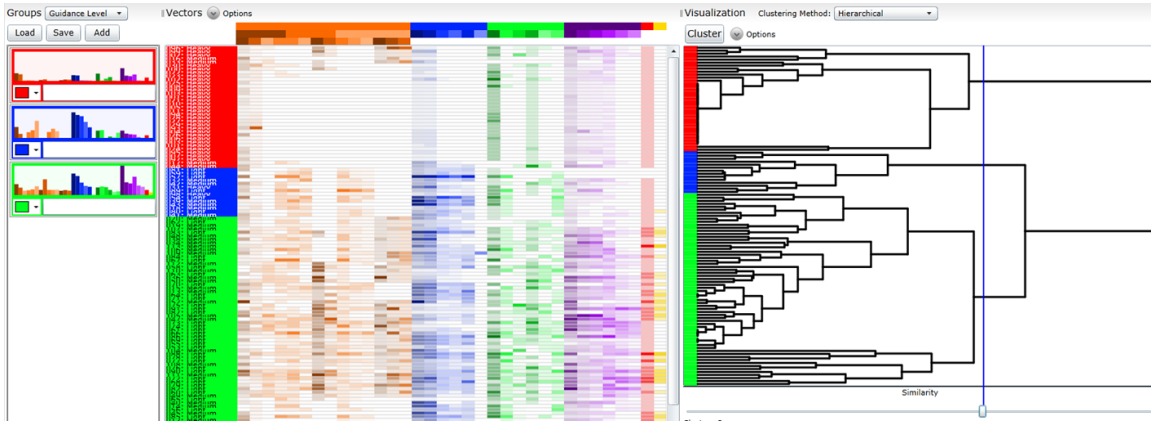
Figure 5.17: A set of feature vectors clustered into 3 groups: red, blue, and green.

We have used this tool to analyze and discover interesting characteristics from a variety of data sources. It has appeared that using agglomerative clustering with a maximum criteria and cos similarity metric works well for most of these sources. Some of the other methods tended to perform poorly. For example, using a minimum criteria often resulted in poor clustering because of the so called "chaining phenomenon" in which clusters may be forced together due to single elements being close to each other, even though many of the elements in each cluster may be very distant to each other. We have therefore set the defaults for the tool to use cos similarity and maximum criteria. In Section 6, we will share in detail how this tool is being used within our research as well as the research of others.

5.5 Tracking Students and Groups of Students

Because we envision BeSocratic activities being used over many courses, semesters, and years with students, it is important that we provide a way to track and visualize student performances over time. We started by simply appending multiple student sequences together and attempting to use the same clustering methods described previously. Clustering these potentially long and complicated sequences proved to be difficult. The number of actions per sequence and the number of action types quickly became large. This required us to increase the number of states needed for each hidden Markov model, which both slowed down the optimization step and made interpreting the models difficult.

Instead, we found it better to first cluster each individual performance into groups and then

re-cluster those groups themselves. In other words, teachers start by creating labeled groups of students for various questions and activities using either sequence clustering, feature vector clustering, or through manual assignment. BeSocratic takes these sets of groups/clusters and generates a stacked column chart for each set as seen in Figure 5.18. This visualizes the proportion of each group in each set. Furthermore, BeSocratic uses the student IDs associated with each student in each group to draw an arrow from each group to the following group. These arrows are weighted by the probability of students making the transition from a group into another group. Using these visualizations, teachers may be able to see how clusters between questions or activities are correlated and gain deeper understanding into the relationships between multiple activity steps or activities. As with the rest of BeSocratic's tools, the visualizations are customizable in that the way probabilities, colors, labels, and edge thickness are displayed can all be adjusted.

This tool also serves another purpose. Teachers can export the results of the tracking for further analysis. It may be exported as feature vectors for use and analysis in Microsoft Excel or within BeSocratic's feature vector analysis tools. The tracking data may also be exported as sequence data, which has each student transition from one group to another. BeSocratic can then analyze this data with its Sequence Clustering tools that have been previously described. By using the tracking and clustering tools repeatedly, teachers can view student work at multiple levels. At the lowest level, teachers have the ability to view individual student replays. As the teachers zoom farther out, they can see performances of students over multiple questions, activities, courses, and semesters.

5.6   High Level Feedback

As we described previously, clustering can identify groups of students who are potentially at risk. For example by comparing the actions between the red, yellow, and blue groups in Figure 5.12, one can notice that the green and yellow groups either finished the problem correctly or seemed to have gotten stuck on a few errors. The red group, on the other hand, made many mistakes before arriving at the correct solution. Furthermore, they seemed to make the same types of mistakes many times. This could indicate a few different things. For one, it could indicate that the system is having trouble recognizing their answers. Usually, inspecting the actual replay of the student's work can confirm if this is the case. If it turns out that the system is having trouble recognizing a student's
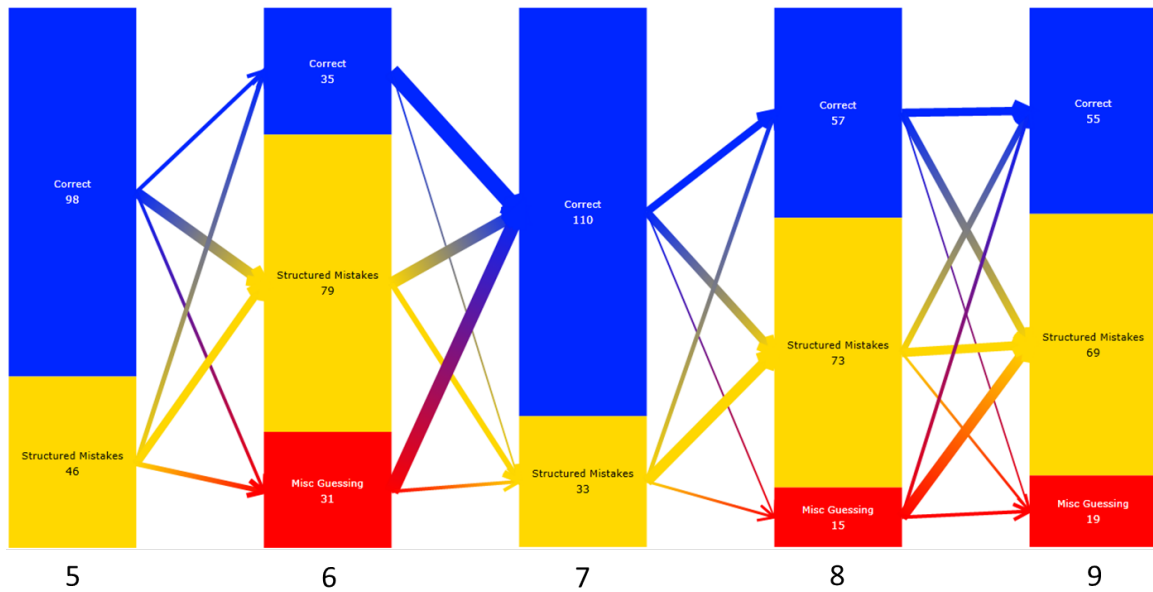
Figure 5.18: Clustered student replays. In this example, the clustering resulted in 4 groups of students highlighted on either side by green, red, blue, and yellow. By looking at the differences between these clusters, it is possible to identify groups of students who are at risk for various reasons.

answer, adjustments should be made to the question's rules in order to minimize this happening for future students. Alternatively if it appears the system was correctly interpreting the students' drawings, a replay with many mistakes could indicate that the student lacks sufficient knowledge on the subject to answer the question correctly and may simply be trying to guess their way to the correct answer. This seems to indicate that the students either need to engage with the system or they need more instruction. While the former can be difficult to overcome, we felt the latter could be addressed.

We came to the conclusion that we would like to detect when a student was falling into one of these wandering/guessing states; and if we detect this, provide further feedback or information to hopefully help them get back on the correct track. To accomplish this goal, BeSocratic's Sequence Clustering tool is able to create hidden Markov model-based classifiers that enable activities to detect troublesome actions and provide feedback to students. This section will detail this process.

To create a classifier to use for feedback, teachers first cluster the student submissions into finely detailed clusters as described in previous sections. Then, the teacher would select the group of submissions in which they would have liked to give feedback. We will call this group the positive group. All of the other clusters will be referred to as the negative groups. At this point BeSocratic
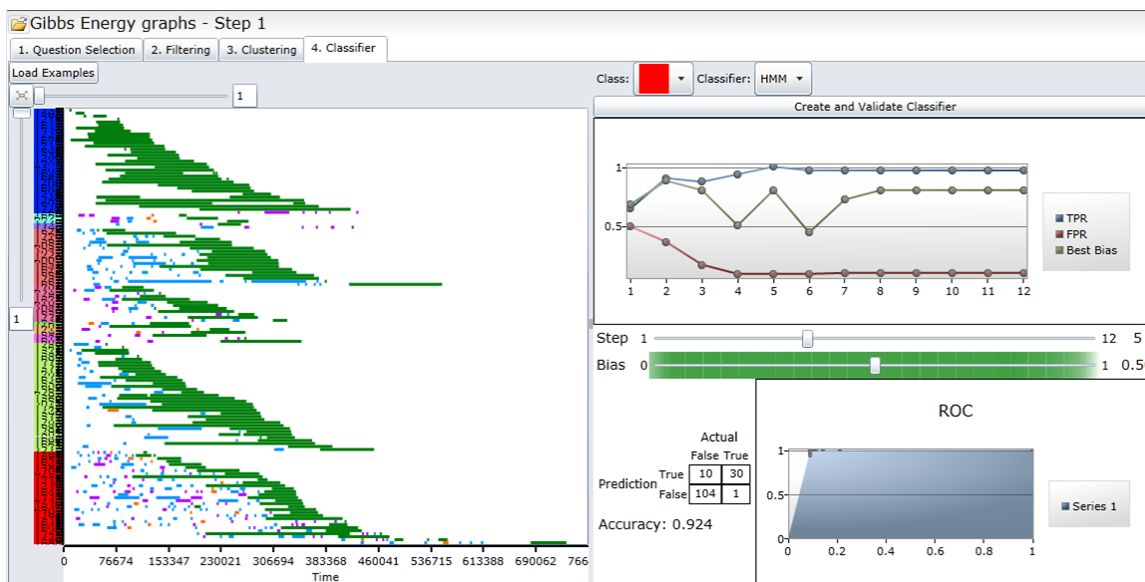
Figure 5.19: Classifier Creator being used to generate a classifier for the red (Misc. Guessing) group. The charts help determine how the classifier would perform with new student action sequences.

performs an analysis of the positive group to determine if it can accurately classify new submissions as belonging to the positive group or not. In order to achieve this, BeSocratic separates the data into training and testing sets following the 10-fold cross validation method. Using the training data (i.e. 9 of the 10 folds), the algorithm builds and optimizes a HMM for each cluster (both positive and negatives). Then using each sequence in the testing data (i.e., the 1 remaining fold), our algorithm calculates the likelihood that the sequence was generated from each of the HMMs (i.e., the model for our positive cluster as well as the models for the other negative cluster). This likelihood value is calculated using the same forward algorithm that was described in Section 2. After the likelihoods have been calculated, BeSocratic compares the likelihood value for the positive cluster against the likelihood value of the most likely negative cluster. If the positive likelihood is greater than the highest negative likelihood, the sequence is classified as belonging to the positive cluster. Otherwise, the sequence is classified as not belonging to the positive cluster. Using this method to classify each sequence, BeSocratic calculates various classifier measures such as true-positives, false-negatives, true-negatives, false-positives, and accuracy.

In addition to simply using a greater than/less than evaluation of the two likelihood, BeSocratic varies a bias value to determine if a bias would improve the classifier's performance. The bias affects the predicted classification of a sequence according to the formula

73

$$classification = positiveLikelihood * bias > negativeLikelihood * (1 - bias).$$

Using this formula, it is clear to see that if the bias is 0.5 the formula will classify the sequence positively if targetLikelihood is greater than otherLikelihood and negatively otherwise. If the bias is set to either extreme (i.e., 0 or 1), the formula will classify the sequence as negative or positive for 0 or 1 respectively. By varying this bias and performing hypothesis testing (i.e., calculating the true-positives, true-negatives, false-positives, and false-negatives), the classifier creation tool generates a receiver operating characteristic (ROC) curve. This curve along with statistics such as the false positive rate can help predict how effective the classifier will be with new student data.

Perhaps unsurprisingly, the performance of this type of hidden Markov model-based classifier varies depending on the sequence length. And since these classifiers will be evaluated after each student action, it became clear that the evaluation should take into consideration not just full-length sequences but also partial sequences. For example, we would expect the classifiers to perform relatively poorly after only a single student action since many of the underlying HMMs may be likely to start with a particular action. As the sequences continue to grow, one would expect the classifier accuracy to improve. To visualize this trend, the classifier creator tool generates line charts for the true positive rate and false positive rate as they vary with the number of actions to consider.

All of this functionality enables teachers to create classifiers that can be used to identify trends in new student sequences and intervene (if necessary) with feedback. Once the teacher has created a classifier that they feel will be effective, the classifier is saved to an external file. This classifier contains the filters for transforming a raw sequence, the positive HMM, the negative HMMs, the bias value, and the minimum action count to begin classifying student sequences. The teacher can then add the classifier back into the source activity step as a rule. In addition, the teacher sets one or more tiers of feedback. An algorithm (outlined in Listing 5.3) determines if the feedback should be displayed to students based on the actions they are making in the system. Then as new students complete the activity, each action registered within the system is first filtered using the classifier's filters. Next, the algorithm checks to see if the length of the transformed sequence is greater than the minimum length of the classifier. If the length is greater than or equal to the minimum length, the likelihoods for the sequence being generated by the positive HMM and negative HMMs are calculated. The likelihoods are then weighted with the classifier's bias and compared to determine if the sequence was most likely to be generated by the positive HMM or one of the negative HMMs. If the result is a positive classification, the next level of feedback for the classifier

```
List<Sequence> testing;//testing fold
List<Sequence> training;//training folds

Cluster positive; //cluster we wish to detect
HMM positiveHMM = trained with training sequences within positive group

List<Cluster> negativeClusters; //other clusters
List<HMM> negativeHMMs = trained with training sequences within negativeClusters

//evaluate the testing data
foreach(seq in testing)
{
   double positiveLikelihood=positiveHMM.Likelihood(seq)
   double negativeLikelihood=negativeHMMs.Max(hmm=>hmm.Likelihood(seq))

   double bias=0.5;//this can vary

   bool predictedClassification=
         positiveLikelihood * bias > negativeLikelihood * (1-bias);
   bool actualClassification=target.Contains(seq)

   if(predictedClassification==true && actualClassification==true)
      //true positive
   if(predictedClassification==false && actualClassification==false)
      //true negative
   if(predictedClassification==true && actualClassification==false)
      //false positive
   if(predictedClassification==false && actualClassification==true)
      //false negative
}
```

Listing 5.2: Pseudo-code for Building And Evaluating Classifiers

```
//called after student makes an action in the system
public void ActionAdded(Classifier classifier, Sequence rawPartialSequence)
{
   //transform the raw sequence using the filters from the classifier
   Sequence sequence=classifier.Filters.Filter(rawPartialSequence);

   //determine if the sequence is of minimum acceptable length
   if(sequence.Length<classifier.MinimumActionCount)
      return; //do nothing

   //calculate the likelihoods for the target and other groups
   double positiveLikelihood=positiveHMM.Likelihood(seq)
   double negativeLikelihood=negativeHMMs.Max(hmm=>hmm.Likelihood(seq))

   //predict the classification
   bool predictedClassification =
         positiveLikelihood * classifier.Bias > negativeLikelihood * (1-classifier.Bias);

   //show feedback if classification is positive
   if(predictedClassification==true)
   {
      //show the next feedback tier
   }

}
```

Listing 5.3: Pseudo-code for Classifying New Sequences

is displayed to the student. Otherwise, the student continues as normal.

Using this method, we believe we can create activities with more effective feedback that ultimately helps student learn in more meaningful ways. We have begun testing this functionality with students and will describe our results in Section 6.

5.7 Analyzing External Data

As all of these tools were being created, we recognized that teachers and researchers using other software that records their student's work would want to use BeSocratic's analysis tools. To allow for external data sources to be analyzed, we developed XML specifications for importing such data. We have 2 different XML specifications for importing sequence data and feature vector data.

```xml
<?xml version="1.0" encoding="UTF-8"?>

<sequences>

  <sequence key="Sam Bryfczynski">

    <attributes>

       <attribute key="Email" value="sbryfcz@clemson.edu" />
```

```
    </attributes>
    <events>
        <event key="Button 1 Clicked" startTime="100" timespan="0" />
        <event key="Button 2 Clicked" startTime="230" timespan="0" />
        <event key="Button 1 Clicked" startTime="270" timespan="0" />
    </events>
  </sequence>
</sequences>
```

Listing 5.4: Example of sequence data encoded in our XML Specification

An example of sequence data in our XML format is shown in Listing 5.4. In this specification, each sequence is composed of two main components: attributes and events. Attributes provide a way to set additional information about a sequence. These attributes typically include things such as student ID, name, sex, ethnicity, and major. There is no set list of attributes. They are simply made up of key/value pairs of strings or numbers. Once loaded into BeSocratic, these attributes can be used as clusters themselves. For example by using the sex attribute, teachers can visualize the differences between male and female students. The other major component of the sequence tag is events. Events represent either actions performed by the student or actions performed by the system. Each event is composed of a key (string indicating the type of event), a start time (either a number or timestamp), and either a timespan or stop time to indicate duration. Events can also take an optional color attribute to dictate how they are colored within the visualizations. It is important to note that events can occur various times throughout a sequence. In this case, their keys will be identical but their start and end times will be different.

An example of feature vector data in our XML format is shown in Listing 1. It is very similar to the XML specification we designed for sequences. Just like the sequence XML specification, each featureVector can contain attributes to describe the featureVector. In addition, featureVectors contain feature elements. Each feature is simply a key/value pair with a string key and a numeric value. Unlike the sequence XML specification, feature keys are unique and can only be used once per featureVector. Features can also be grouped into a hierarchy through the use of featureFolder elements. This allows categories and subcategories of features to be exported.

```
<?xml version="1.0" encoding="UTF-8"?>
```

```xml
<featureVectors>

   <featureVector key="Sam Bryfczynski">

      <attributes>

         <attribute key="Email" value="sbryfcz@clemson.edu" />

      </attributes>

      <features>

         <feature key="Button 1 Clicked" value="2" />

      </features>

      <featureFolders>

         <featureFolder key="Button 2 and 3">

            <feature key="Button 2 Clicked" value="1" />

            <feature key="Button 3 Clicked" value="0" />

         </featureFolder>

      </featureFolders>

   </featureVector>

</featureVectors>
```

Listing 5.5: Example of a feature vector encoded in our XML Specification

We have used these specifications to export data from other systems into BeSocratic. We started by exporting data collected in previous research from OrganicPad and GraphPad. Since we had access to the source code, simple export functions were created to convert replays from those programs into our XML specification and import that data into BeSocratic for further analysis. Furthermore, researchers from other institutions have formatted their data in our format and been using our analysis tools to visualize their data in new and meaningful ways. We will describe an example of such collaboration in the next chapter.

Chapter 6

Results

BeSocratic has been implemented in chemistry, molecular biology, and computer science courses in three different universities (Clemson University, University of Colorado Boulder, and University of North Carolina Wilmington) and has collected over 200,000 student submissions.

Instructors are using BeSocratic inside the classroom with lecture sizes sometimes exceeding 100 students. In addition, activities are being utilized outside the classroom as homework assignments, which often require students to create drawings via the chalkboard module. Then during lecture, instructors find it useful to view the submissions in a grid to identify and project interesting submissions. One advantage of this method is the paper savings, since drawing exercises require teachers to distribute, collect, and store large amounts of paper. BeSocratic handles recording, storing, and retrieving electronic submissions for the teacher, which may be accessed from anywhere with an internet connection.

The intent of this chapter is to describe the typical activity creation process, report specific results from several BeSocratic activities, and discuss how BeSocratic's analysis tools are being used with data from other systems. We specifically detail two chemistry activities and one computer science activity: (1) an activity focused on improving mechanism use, (2) an activity targeting Gibbs energy, and (3) an activity directed at teaching splay tree operations. In addition to analyzing data collected from students within BeSocratic, we share our visualizations and results from analyzing data gathered with another educational system, PhET.

Figure 6.1: Image taken during a trial of a BeSocratic activity in a chemistry class. This trial included 93 students in general chemistry.
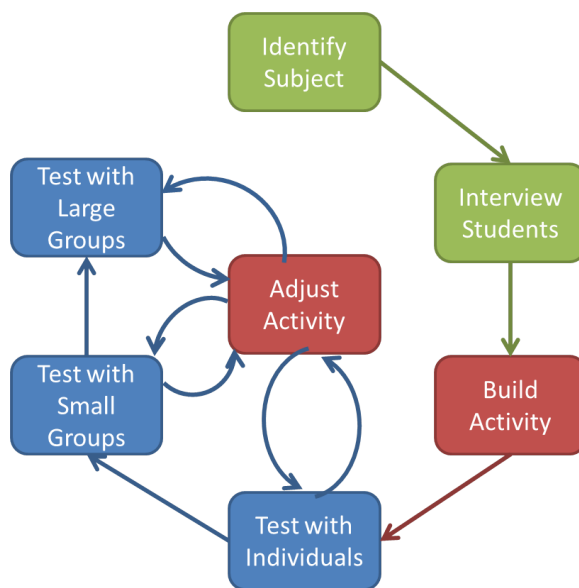
Figure 6.2: Typical BeSocratic Activity Creation Process

6.1 Activity Development Process

Most BeSocratic activities are created with a process similar to that shown in Figure 6.2. To start, a topic is selected with the intent to improve or assess student understanding. Once the topic for the activity has been chosen, the first step in the construction process is to interview students in order to determine specific problems they may have. The information obtained from these interviews is used to build the initial version of the activity. This preliminary version is then tested using one-on-one interviews with students to observe how they interact and navigate through the activity. With these results, further refinements are made to address the students comments and feedback. The revised version of the activity is then pilot-tested with approximately 20-40 students to evaluate how a larger group of students progress through the activity. This is an attempt to determine if there are any issues that have not been previously addressed. Once any problematic areas have been worked out, the activity is administered on a large scale with approximately 100-200 students. Following this process, we have developed activities for introductory chemistry, molecular biology, and computer science courses.

6.2   Gestures

One unique vision of the program was to evaluate the possibility of using "embodied cognition" to enhance student learning. "Embodied cognition" suggests that the mind links together ideas and actions, and some recent research indicates that gestures are a form of embodied cognition [40, 7, 34]. By gestures, we mean various movements using the body including ones we use every day such as pointing and waving. Namely, we focused on the familiar types of hand and finger gestures that are becoming prevalent in emerging devices such as the iPhone, iPad, and Android devices. Research in this area has shown that learners who are allowed to gesture while performing a new task show improved learning and retention over those who do not [21]. Furthermore, Cook and Goldin-Meadow found that requiring learners to gesture while learning improves retention of new knowledge by children acquiring new mathematical techniques [20].

One way that we are trying to use embodied cognition is within the subject of organic chemistry. This course is sometimes thought of as a gateway course for large numbers of students wishing to pursue careers in the biological and medical sciences. In organic chemistry, the ability to understand and use representations is key. For example in a process referred to as a *mechanism*, arrows drawn to symbolize the movement of electrons can be used to predict the outcome of novel reaction. Often times, beginning organic chemistry students who do not learn how to properly draw these arrows can become overwhelmed, and instead of learning and understanding the concepts, resort to trying to memorize a large volume of material. As part our previous research regarding arrow use, we found that less than 50% (and in some cases, as few as 20%) of students used arrows to assist their prediction of reaction products. In some of those studies, the students were even told explicitly to draw arrows yet did not. When we asked students to predict the product of an unfamiliar reaction, we found that the mechanism users significantly outperformed the non-mechanism users [36].

We believe that this finding alone is reason to encourage students to properly apply arrows. To facilitate this goal, we created a BeSocratic activity using embodied cognition in order to improve student arrow use and thus learning of mechanisms. Since these activities can utilize a pen-based or gesture-based input system (using fingers on a touchscreen), this enables the examination and comparison of gesture-based learning activities with more traditional pen-based activities. This study used BeSocratic to investigate how gestures might help organic chemistry students develop a

Figure 6.3: A screenshot of the treatment BeSocratic activity in which students are asked to gesture to show the movement of electrons. Each student's actions were captured and contextual feedback was provided.

more meaningful application of the arrow convention to predict the outcome of reactions (Figure 6.3).

To accomplish this, we implemented a pilot-study at a research university (Clemson University) followed by a replication study the following semester at a comprehensive university (University of North Carolina at Wilmington). Both studies included first-semester organic chemistry students and had the same experimental design, which involved a pre-test, post-test, and a transfer task as shown in Figure 6.4. The pre-test was administered in the middle of the semester after students were instructed on mechanisms in the classroom. After one week, the students were given two sequential weeks of one of the three treatment activities. Group A completed a BeSocratic activity that required students to gesture using their fingers to trace the path of the electrons, and students received contextual-feedback. Group B did the activity on paper drawing the arrows and the products, while Group C only drew the products. Each treatment activity lasted about 15-20 minutes and introduced students to the concept of an electron source and electron sink and reinforced the arrow symbolism and the steps in a mechanism. Two weeks after the treatment activities, the students completed a post-test; this was followed three weeks later by a transfer task. The tasks that students completed for evaluation purposes are shown Figure 6.5. For comparison purposes, we also used control data to verify that our treatment groups' pre-test results were among historical averages on mechanism use and success rates.

All tasks were recorded using OrganicPad and analyzed by a faculty member and a graduate student. The submissions were coded to determine if the students' products were drawn correctly and whether they used a mechanistic approach. In order to ensure that the coders were in agreement, 25 submissions were randomly selected and analyzed independently for mechanism use by both

Figure 6.4: Experimental Design for Gesture Studies

Figure 6.5: Post-test and Transfer Task Questions
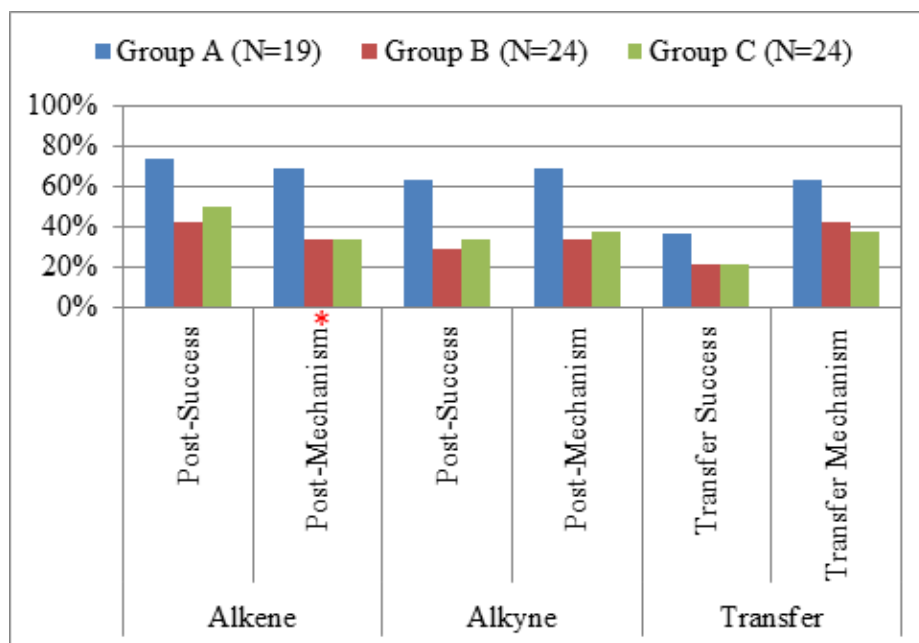
Figure 6.6: Fall 2010 Gesture Results

researchers. The inter-rater reliability (Cohen's kappa) for this analysis was 0.92.

When comparing the students' pre-test results for the pilot-study administered at Clemson University, we found that there were no significant differences among the treatment groups A (N=19), B (N=24), and C (N=24) in the students' pre-test mechanism use and success rates. Figure 6.6 shows the results from the post-test and transfer task for the three groups. The number of asterisks in Figure 6.6 depicts the level of significance: * $p < 0.05$,** $p < 0.01$, and *** $p < 0.001$. Upon analyzing the post-test responses, we found significant differences in the mechanism use of the alkene reaction for the three treatment groups, where the embodied group significantly outperformed the other two treatment groups ($X^2(2, N = 67) = 6.827, p = 0.033, \phi_c = 0.32$). The effect size was found to be 0.32, which is a medium effect size [19]. Although there were no further significant statistical difference found for the other tasks, these results seemed promising and warranted further replications.

The replication study was similar to the pilot-study in that the same instructional materials and treatment groups were utilized; however, this study was conducted at the University of North Carolina at Wilmington. As was the case with the pilot-study, our groups A (N=20), B (N=17), and C (N=20) were statistically equivalent in their initial success rates and mechanism use. Results from this replication study, shown in Figure 6.7, indicated similar results to the pilot study in that
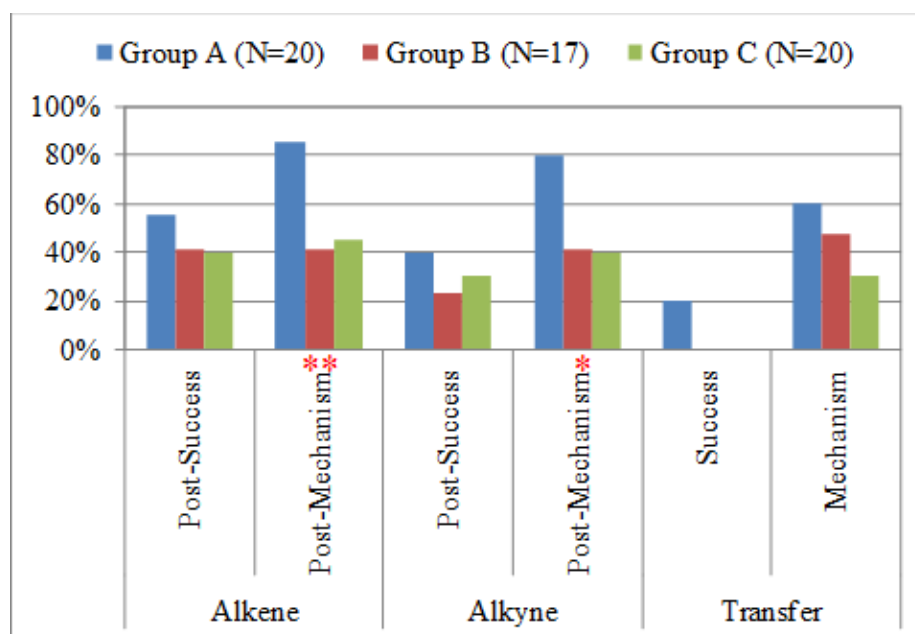
Figure 6.7: Spring 2011 Gesture Results

student mechanism use in the alkene reaction was significantly different among the three groups. The embodied group (group A), again, significantly outperformed the other two treatment groups with a medium effect size ($X^2(2, N = 57) = 9.341, p = 0.009, \phi_c = 0.40$). In addition to this difference, the post mechanism use for the alkyne reaction was also significantly different, where group A outperformed groups B and C ($X^2(2, N = 57) = 8.154, p = 0.017, \phi_c = 0.38$). Furthermore, there was also a significant difference in the student success rate of the transfer task where group A was significantly higher than the other two treatment groups ($p = 0.019$(Fisher's Exact Test)).

The gesture-based intervention seems to show an effect; however, we hypothesize several other factors that could be contributing to or causing these results. For example since groups B and C were using pencil and paper to complete their activities, it might be that the novelty of the computer interface could make the students in group A more willing to pay attention to the activity. Another possibility was that the feedback generated by BeSocratic could be causing this effect. Even with these uncertainties, we feel that these results are encouraging in that student mechanism use and success can be improved after using BeSocratic intervention activities. We are currently conducting additional studies with larger sample sizes to address these concerns. Some changes that are currently being made are (1) all groups are using iPads to verify that the technology is not causing the effect, (2) all assessments and interventions are being conducted within BeSocratic,

86

and (3) the design of the treatment interventions consist of an embodied group using finger-based gesturing, a feedback-only group that is identical to the embodied group except using a stylus instead of fingers to gesture, and a control group that undergoes the same activities but does not receive any feedback.

6.3   Gibbs Energy

In addition to the previously described Mechanisms activity, we have also designed a BeSocratic activity for entry-level general chemistry students to teach the relationships between thermodynamic functions and Gibbs energy. Understanding the entropy changes associated with a relatively simple process, such as a phase change, is not a menial or trivial task. It requires an understanding of relationships between several complex concepts, and these relationships are not something that is explicit or that students could be expected to derive intuitively. A BeSocratic activity was developed as a way to engage students and assess their understanding of these relationships. Of note, the graphs drawn in this activity are more complicated than the simple gesture based graphs that are evaluated in the Mechanisms activity. This section describes the Gibbs activity that was created, the experimental design we conducted to test the effectiveness of the activity, and the results we obtained from using the activity with students.

6.3.1   Activity Description

The equation $\Delta G = \Delta H - T\Delta S$ represents the relationship between Gibbs energy, enthalpy, and entropy changes respectively. While students may remember this equation in subsequent chemistry classes, this equation only presents the basic relationship between these thermodynamic functions. Students must be able to move past the simple terms and understand the underlying and implicit relationships in order to determine if and why chemical processes occur. It is essential for students to understand more than the simple math or how to solve an equation. To accomplish this, we developed an activity asking students to work through graphical representations of the relationships between enthalpy, entropy, and Gibbs energy.

The activity starts by asking students to explain, using text, the relationship between the enthalpy change, the entropy change, and the Gibbs energy change for a simple process. Next, students are given a few assumptions, such as a constant ΔH, and asked to draw the $T\Delta S$ line with
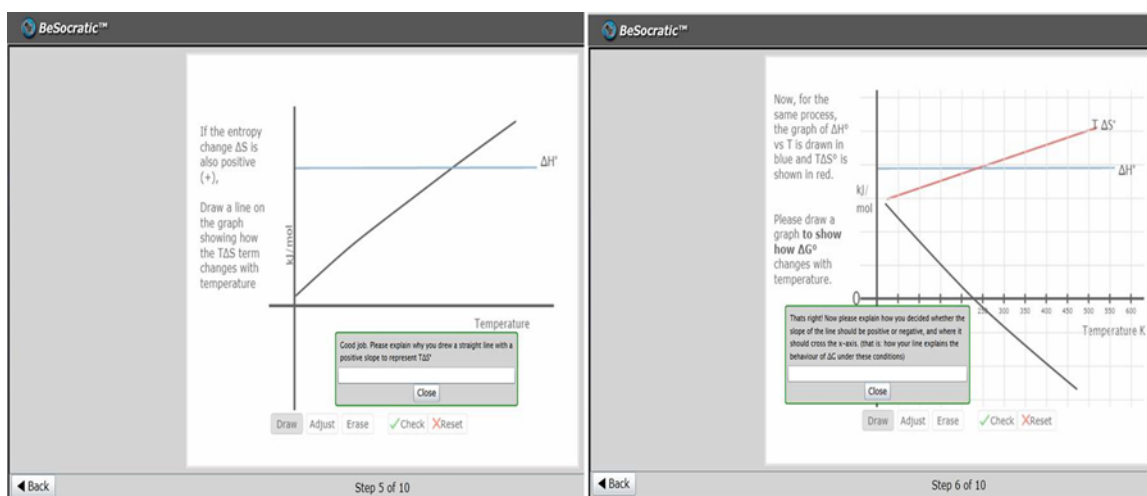
Figure 6.8: Examples of slides and feedback in the Gibbs Energy Graphs activity.

a positive entropy change as temperature increases. This graph should be a line with a positive slope. Subsequently, after establishing that there is a point of intersection between the entropy and enthalpy change students are asked to graph the Gibbs energy change for a specific process. It is imperative that students understand that Gibbs energy equals zero at the point where their enthalpy and entropy graphs intersect.

Using BeSocratic's SocraticGraph module, the activity evaluated and provided feedback to the various graph drawings. This feedback was designed to be Socratic in nature and guide the students to the correct answer using directed feedback. We also used the system to ask students to explain their reasoning each time they checked their work. This allowed us to analyze their responses and gain insight about their reasoning as they worked through the activity. Examples of several slides that students were presented with is shown in Figure 6.8.

6.3.2 Pilot Study Spring 2012

In the spring of 2012, the Gibbs activity was pilot-tested with several groups of general chemistry students along with pre- and post-assessments. These assessment questions were directed to see if students understood the relationship between enthalpy, entropy, and Gibbs energy changes for an unspecified process.

After the pilot-study concluded, we used BeSocratic's sequence analysis tools to cluster the students' submissions for each question using hidden Markov modeling. This process involved con-

tinually "splitting" the groups until eventually there were multiple groups of submissions with clear strategies that were visible and one group of students who all performed relatively different action sequences. At this point, we considered the sequences clustered to fine detail. Using BeSocratic's Tracking Visualization to view the progression students took from one group to another over multiple steps, it became clear that the data was clustered to too-fine of detail. Because of this, we merged many of the finely detailed clusters into one cluster resulting in three overall clusters: *Correct on First Try*, *Structured Mistakes*, and *Miscellaneous Guessing*. The *Correct on First Try* group all drew the graph correct on their first attempt. The *Structure Mistakes* group all drew 1 or more incorrect graphs, yet they seemed to have a structure to their errors. The structure was determined based on the types of errors that were made (e.g. Incorrect Slope, Incorrect Start of Graph, etc.). For example, a group of students received the same sequence of two Incorrect Start of Graph actions followed by three Incorrect Slope actions. The final group, *Miscellaneous Guessing*, was composed of all the submissions in which the types of errors made were not orderly and often students made the same types of mistakes several times. This process was repeated for all 5 of the graph-drawing slides. Using these new groups, we again used BeSocratic to visualize how students moved between groups (see Figure 6.9). From looking at this visualization along with the other visualizations, it became clear that there was not a strong correlation between how students performed from one question to another.

6.3.3 Pilot Study Fall 2012

The results and analysis of the spring 2012 pilot study were used to improve the activity for a subsequent study in the fall of 2012 with off-sequence second semester general chemistry students. Students were randomly selected to be in the control group or in one of three treatment groups. The first treatment group received the prior activity with unchanged feedback. The second treatment group received the activity with revised, multi-tiered feedback. The last treatment group participated in the same prior activity, but it had been modified to recognize when students are randomly guessing using hidden Markov model-based classifiers. All groups received the same instruction from the same professor. Each group was checked for equivalence using demographic factors such as sex and major. Since the results of the spring 2012 pilot study clearly indicated that students came in with little to no knowledge on the subject of Gibbs energy, it was decided that a pre-test was not necessary. However, a post-test was administered 2 weeks after treatment and a delayed
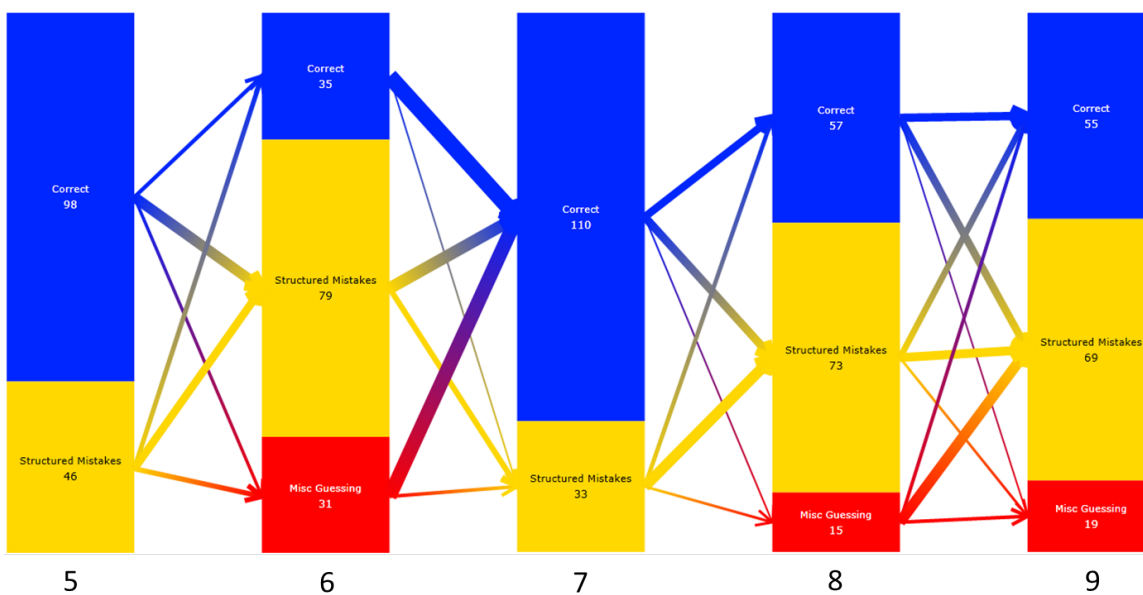
Figure 6.9: Clustering results from Slides 5-9 of the Spring 2012 Gibbs Energy Graphs activity

post-test was administered 2 months later at the end of the semester as a measure of retention. The experimental design and sample sizes for this study are shown in Figure 6.10.

The first post-test was composed of 3 questions that asked the students to construct graphs for the entropy, enthalpy and Gibbs energy change for a specific process. Students were given the exact values for ΔH and ΔS and asked to plot all the values in a graph and explain their reasoning. After student's completed this first post-test, the answers were separately coded by 2 group members for correctness. The average inter-rater reliability (Cohen's kappa) for this analysis was 0.94. Results from the coding are show in Table 6.3.3. The results showed significant differences between several of the treatment groups. In the enthalpy question, the control group performed significantly lower than both the unchanged activity group and the improved feedback group with p-values less than 0.05 and effect sizes of 0.24 and 0.25 respectively. In the Gibbs energy question, the group using hidden Markov model-based feedback performed significantly better than the improved feedback group with a p-value less than of 0.05 and effect size of 0.23.

While the results from this activity showed little or no learning gains, we feel that it reveals several things. For one, the subject (i.e., Gibb energy) is particularly complex and difficult. It may be that students need more time with the material in order to understand the concepts. However, looking at the various graphs that students drew showed some consistency. We believe that a future

90
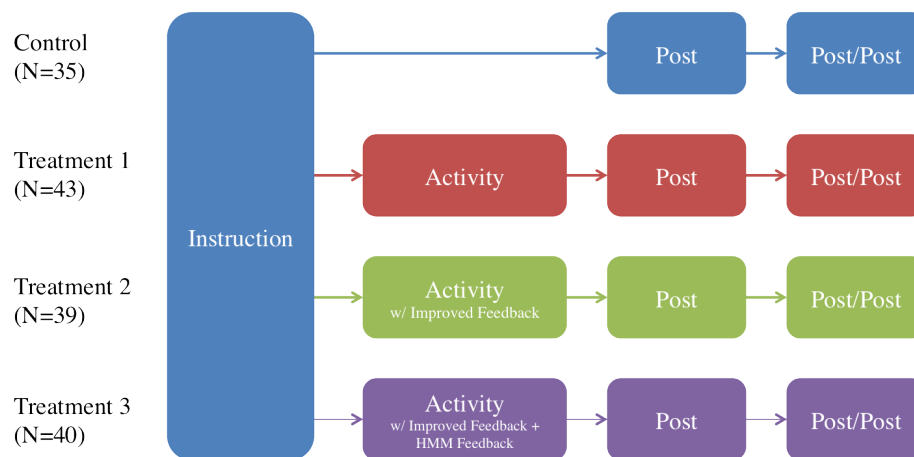
Figure 6.10: Experimental design for the Gibbs activity for Fall 2012
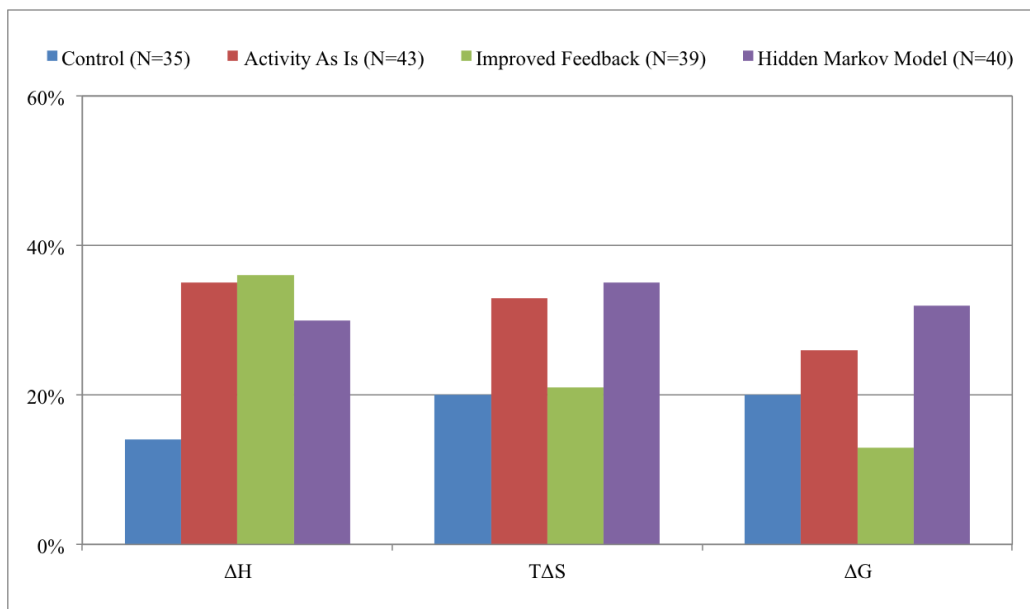
Fall 2012 Post-test		Success	
Question	Groups	p	ϕ
Entropy (ΔH)	C vs. AsIs	0.038	0.24
	C vs. IF	0.034	0.25
Gibbs Energy (ΔG)	IF vs. HMM	0.047	0.23

Table 6.1: Results from the Gibbs Fall 2012 post test. Differences between the two groups were explored for statistical significance using a chi-square (2) analysis. Effect sizes were calculated for all statistically significant comparisons and are reported as Phi (ϕ) values.

activity that targets these commonly drawn graphs with more specific feedback could uproot the underlying issues. This activity also served as a proof of concept for hidden Markov model-based classifiers. While we only see one instance where the classifier based feedback improved student performance significantly, we believe that it warrants further investigation in larger studies.

6.4   Splay Trees

In addition to the chemistry activities described, we are beginning to explore BeSocratic's potential to teach data structures in computer science classrooms. Data structures are fundamental concepts that every computer scientist must learn if they are to succeed as a software developer. As a proof of concept, we created an activity designed to help teach students the fundamental operations of splay trees. Splay trees are self-adjusting binary search trees, which have significantly better amortized performance guarantees over simple (non-self-adjusting) binary search trees. Their deterministic nature makes them a perfect fit for an activity as GraphPad modules can be customized to evaluate and respond to specific student-drawn trees.

Screenshots of several slides from this activity are shown in Figure 6.4. The activity was broken down into 4 sections. In the first section, students proceeded through a GraphPad tutorial where they learned to draw, label, and move nodes and edges. The next section taught students to perform the *splay* operation including in-line, out-of-line, and single rotations. Throughout this section, students were required to practice these rotations by manipulating graphs on the screen. To ensure that students stayed on task, they were asked to check their work and only allowed to proceed once they had successfully answered each question. Multi-tiered feedback was included to assist students who became "stuck". Next, students applied the rotations to insertions and deletions within a splay tree. Once again, students were required to check their work at each step and received multi-tiered feedback to assist them. The activity concluded with a transfer question that asked the students to complete a series of insertions and deletions into an initially empty splay tree.

49 students completed the splay tree activity as part of this pilot study. The students were split across four lab sections but all had the same lecture instructor. After the data was collected, we clustered each submission based on the sequence of correctly-drawn and incorrectly-drawn structures for each student. This revealed that students were able to easily complete many of the activity's questions correctly. However, visualizations of several questions showed students

Figure 6.11: Example slides from the Splay Tree activity

struggling to construct the correct answer. Upon closer inspection of the mistakes, it is clear that students had trouble with the proper order of in-line and out-of-line rotations. These mistakes were often repeated many times by the same students indicating that these students were following the same *incorrect* algorithm to the same *incorrect* answer. Each student who initially encountered difficulty eventually realized their mistakes and was able to apply the proper algorithm to construct the correct splay tree. While feedback was given to students upon checking incorrect answers, we believe the feedback may have been too generic. We believe more targeted feedback based on the results of this pilot study will improve student performance in the future.

The final slide of the activity asked students for any comments/problems they wished to share. The students' comments were mostly positive. Referring to splay tree construction, one student commented, "Helpful to see it in person rather than just an explanation of how it works. Good test to see if you can do it yourself. I feel much more confident in my understanding of splaying". Others noted the difference between the BeSocratic activity and typical programming-based labs, "This was a fun change of pace to our usual lab, and I feel like I've at the very least gotten a much better understanding of rotations and splay trees." A few students did note that the feedback could have been more specific by saying things such as, "There was no way to tell what was done right and what was wrong. I was just told it was wrong." Based on the submissions and difficulties seen in the students' replays, we have added additional feedback to alleviate student frustrations in the future.

While this study only focused on one data structure, we envision a suite of activities to help teach a range of data structures such as lists, queues, stack, trees, heaps, and graphs. Based on the results from the splay tree activity, we are planning follow up studies with the activity as well as creating new activities for other data structures. Our next focus is to evaluate the teaching effectiveness of the activity by running control and treatment studies with students in the spring semester of 2013. We also hope to pilot test several other data structure activities during this time.

6.5   Analysis of External Data

As we describe in Section 5.7, BeSocratic's analysis tools can be used with data generated from other sources. After testing the tools with internally generated data, we wanted to explore its functionality with data from an outside project. We were fortunate to develop a collaboration with
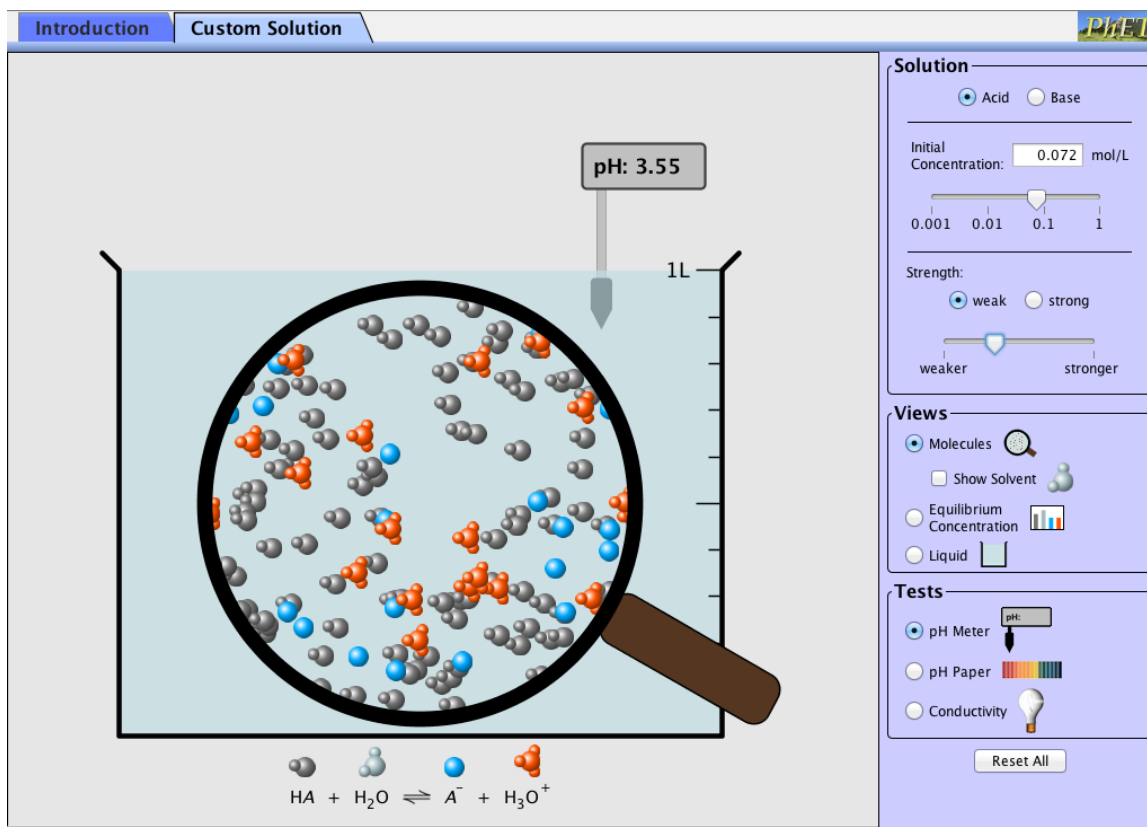
Figure 6.12: Screenshot of the PhET Acid-Base Solutions sim available at `http://phet.colorado.edu/en/simulation/acid-base-solutions`.

the PhET Interactive Simulations group at the University of Colorado Boulder. PhET develops free, interactive, research-based educational simulations on topics in mathematics and science. The goal of these simulations are to enable students to make connections between real-life phenomena and the underlying science concepts through a fun and highly interactive interface. PhET simulations (sims) typically include animated graphics and intuitive, user friendly controls, such as sliders and radio buttons, which help students visually comprehend conceptual relationships through rapid trials and instantaneous feedback. Many of the graphics in PhET sims represent measurement instruments, including pH meters, stop-watches, voltmeters, and thermometers, which respond with animation to student input. Each PhET simulation is tested and evaluated in individual student interviews, and many of the sims are used in classroom activities such as lecture, recitation, and lab. PhET simulations are available for free at the PhET website (`http://phet.colorado.edu`), and are used by educators worldwide.

Recently, the PhET team has begun recording sim event data logs of interactions between

Figure 6.13: Visualization of the log files from the Acid-Base Solutions PhET sim: Light Guidance (Red), Medium Guidance (Green), and Heavy Guidance (Blue) as sequences.

students and various sim controls, as part of their ongoing research on sims. Through our collaboration, BeSocratic is being used to visualize and model this student actions, in order to better understand how students are interacting with the PhET sims. We have focused on one specific chemistry sim, "Acid-Base Solutions," that allows students to explore different properties of acids and bases. A screenshot of this sim is shown in Figure 6.12, (available online and the sim may be accessed freely at http://phet.colorado.edu/en/simulation/acid-base-solutions). Student data was collected as part of a study performed at the University of Colorado [46]. The goal of this study was to explore the effects of administerating different levels of written instruction along with the sim. The study consisted of 113 student groups (usually pairs) split into 3 treatment groups: heavy guidance (HG, n=33), medium guidance (MG, n=40), and light guidance (LG, n=40). Each treatment group received a different activity to complete using the sim. The *HG* treatment received an activity with strict, procedural instructions. The *MG* treatment's activity was composed of fewer instructions. The *LG* students were given minimal instruction in their activity.

Each control action was first categorized based on its function. For example, several controls were allowed students to test the conductivity of a solution and were thus grouped under the "Conductivity Test" category. Once these categories were established, the log data was converted to BeSocratics Sequence XML format and imported into BeSocratic's analysis tools. Figure 6.13 and Figure 6.14 show the log data loaded in BeSocratic's analysis tools as sequences and feature vectors. It becomes immediately clear from these visualizations that the HG treatment group (blue) performed different actions within the sim compared to the LG group (red) and MG group (green).
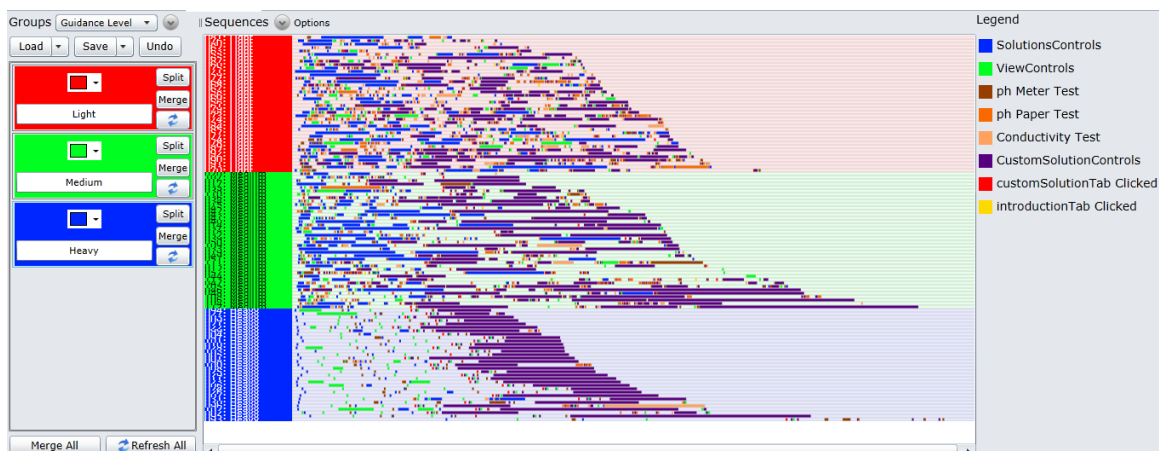
96
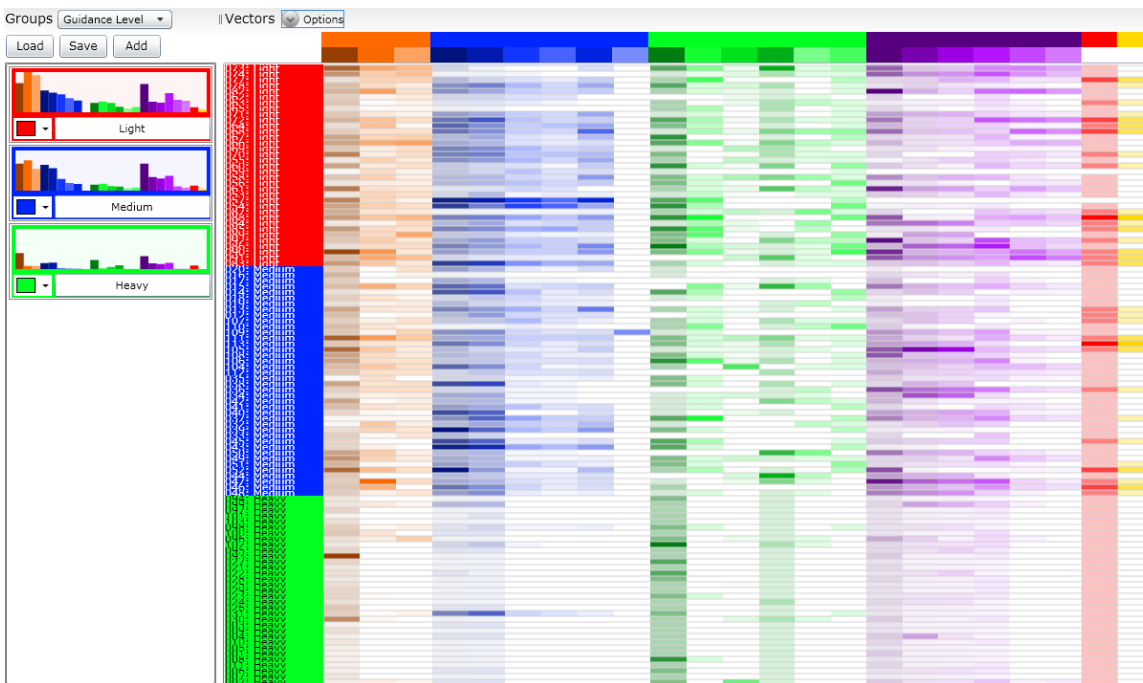
Figure 6.14: Visualization of the log files from the Acid-Base Solutions PhET sim: Light Guidance (Red), Medium Guidance (Green), and Heavy Guidance (Blue) as feature vectors.

Figure 6.15 shows the results of modeling each group (i.e., LG, MG, and HG) as hidden Markov models. We believe these models visually reveal that the MG and LG groups accessed the sim controls with similar levels of variance. This can be seen in the model states with similar internal structure. Furthermore, the model for the HG treatment shows slight yet significant differences from those of the light and medium groups in that there appears to be less variance in the control access patterns. This consistency implies that the students did not explore the controls as much as the other groups. By comparing the HG students' sequences with their given activity instructions, we see high similarity between the instruction actions and those actually performed by the students. Students in this group tended not to stray from the given directions.

We also explored using BeSocratic's clustering capabilities to automatically separate the log files into groups. We were curious if it was possible to automatically discover the underlying LG, MG, and HG groups that were present, using only student sim use patterns. To accomplish this, we performed the split and merge procedure described in Section 5.3 to cluster the set of log files using hidden Markov modeling. After the first split, the sequences were separated into two groups with 86 and 27 students. Interestingly, the group with 27 students were all members of the *HG* group,
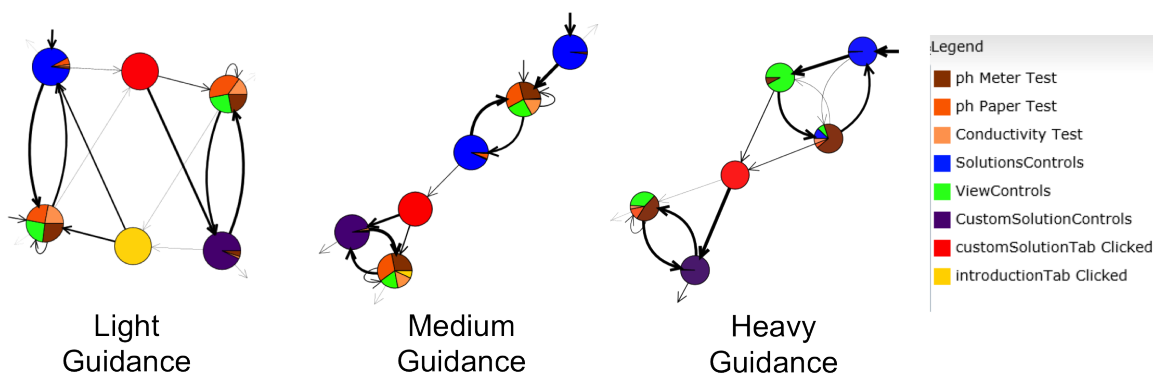
Figure 6.15: Hidden Markov models generated from the PhET sim event data logs.

so it appears that the model-based clustering was able to identify a significant number of the HG students automatically. However, further splitting achieved mixed results where members of the LG and MG groups were intermixed. This seems to indicate that their performances contained a high amount of variability as we found previously.

In addition, feature vector clustering methods were used to cluster the log files and produced similar results. That is, the feature vector clustering separated the HG group relatively easily, but further separation of the LG and MG groups could not be accomplished. Figure 6.16 shows the dendrogram resulting from hierarchically clustering the feature vectors. The dendrogram shows that the HG submissions were quite similar and thus grouped together. The LG and MG submissions are intermixed, as they are relatively similar to each other and thus difficult to automatically separate. This is consistent with the other analysis techniques used and with the LG and MG procedures, which instructed students to explore the sim openly for some or all of the activity.

The analysis performed with the PhET data revealed several interesting things. Primarily, it appears that students in the LG and MG groups used sim controls with high amounts of variance. These instances of high variance correspond to times where students explored the sim and makes sense given students in the LG and MG groups were given specific instructions to explore as part of their directions. With a stricter procedure, it appears that the HG group explored fewer controls within the sim and primarily used the features referred to in their instructions.

We believe this analysis reveals uses for BeSocratic with data from outside sources. Using automated tools such as these, we believe researchers can reduce time spent viewing large sets of system logs and screen recordings. Instead, BeSocratic's analysis tools and visualizations can quickly
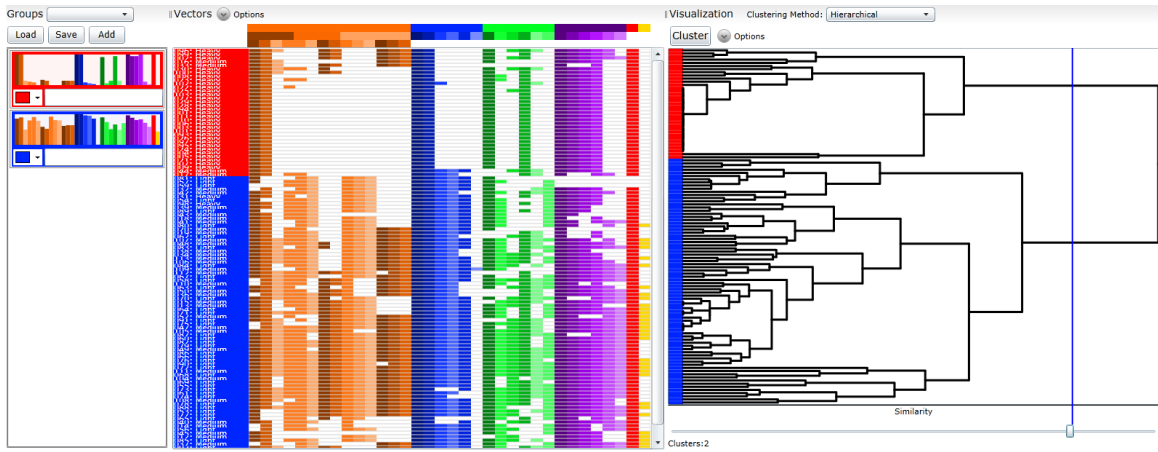
Figure 6.16: Dendrogram generated from hierarchically clustering the PhET log files as feature vectors.

lead researchers to interesting aspects of their data that warrant further investigation.

Chapter 7

Future Extensions

We believe that future extensions could be incorporated into BeSocratic to make it more intelligent. Moreover, follow-up activities and studies using BeSocratic should be performed to evaluate learning gains using the systems. This chapter describes some of our plans and ideas for future work with BeSocratic.

7.1   Further Activity Testing

In the Results section, three activities were described that have been tested with students: Mechanisms, Gibbs Energy, and Splay Trees. As was noted in the section, we feel that each of the studies performed warrant further revisions and testing. The Mechanisms activity is currently being rerun with a new group of students to confirm determine the source of the effect sizes seen in the previous studies. The Gibbs Energy activity is currently being revised to address some of its current weaknesses, and a new version will be student-tested in 2013. In addition to these activities, we are developing additional activities for use with chemistry and molecular biology students and will be testing them in the spring of 2013.

The Splay Tree activity is also undergoing changes to target common student errors that were discovered though pilot testing. Even with the shortcomings of the pilot test, we feel that the activity was beneficial to students and have plans to develop additional activities for a data structures class. These activities will focus on many of the data structures typically taught, including various lists, queues, stacks, trees, heaps, and graphs. We also plan on running experiments to determine

the teaching effectiveness of these activities using control/treatment groups. We plan on developing and using these activities in the spring semester of 2013.

7.2   Expansion to Other Disciplines

So far, BeSocratic activities have primarily been created for chemistry, molecular biology, and computer science classes. We have begun developing activities for physics and mathematics courses as well. We believe that properly developed activities could improve student learning in these disciplines as they have in chemistry.

We also believe the potential to develop additional modules would extend BeSocratic to more disciplines. An example of such a module would be a math equation module that allows students to create mathematical equations using ink or using a button-based interface. Once the equation was created, the module could convert the equation to a common language such as MathML so it could be compared to solution equations. Following the similar pattern with SocraticGraphs, OrganicPad, and GraphPad, the module could provide feedback to students based on how their equations match the teacher's solutions.

7.3   Supporting Additional Devices

As described in Section 4.3, BeSocratic was created with the Microsoft Silverlight framework. Unfortunately, Microsoft has recently indicated that it will not be updating Silverlight in the future. Therefore, we have begun making plans to port BeSocratic to a Javascript application.

In general, we feel that developing a Javascript application would be the best course for the future. Every year new devices are being produced. Recent examples include the Apple iPad, Android Tablet, Kindle Fire, Microsoft Surface, and Barnes and Noble Nook, which each uses a different language and software development kit for app development. This makes reusing code difficult if not impossible. Fortunately, each of these devices has a built-in web browser capable of running Javascript applications, and because of this, we feel that shifting away from Silverlight and towards Javascript will lead to better device support in the future.

7.4   Text Coding

One possible functionality extension we feel would be useful relates to evaluating textual data. There has been a wealth of research on the subject of text mining that we feel BeSocratic could benefit from. Currently, teachers must manually evaluate text responses from students. Ideally, BeSeocratic would be able to better assist in this analysis. For example, one extension of the text-coding tool could enable the prediction of codes within text responses based on how other answers have been coded. We believe that classifiers could be created based on previously coded text responses. BeSocratic could then use these classifiers to predict codes for the answers yet to be manually coded. Teachers would have the ability to accept or reject the codes, and over time the classifiers would improve and help speed up the time to code answers.

Furthermore, more advanced techniques may be able to assist in the generation of codes themselves. We believe that techniques such as latent semantic analysis could be used cluster text responses and identify similarities within student answers. These similarities might make it easier and faster for teachers to identify common student wording and perhaps problems.

Chapter 8

Conclusions

The majority of pedagogical software being used today rely on asking students either free-response text questions or constrained multiple-choice questions. The free-response questions allow students to answer freely and demonstrate their knowledge, but these questions require time-consuming manual evaluation by teachers. Multiple choice questions can be automatically assessed but often lack the ability to deeply probe student understanding. Ideally, a system is able to ask open-ended questions in which students cannot simply guess the correct answer and yet can be evaluated and analyzed automatically. In the subsections that follow, we summarize the software system that we developed to address these shortcomings and state our contributions. Finally, we conclude by highlighting the expected impact of these contributions on the development of more robust pedagogical software.

8.1   Thesis Summary

This dissertation defends the following 3-part statement. It is possible to devise a system that

1. enables teachers to create intelligent tutors, which are able to recognize, evaluate, and provide feedback to student drawings, including Euclidean graphs, computer science graphs, chemistry molecules, and simple free-form drawings.

2. allows teachers and researchers to track individual students and identify concepts in which

students needs help.

3. allows teachers and researchers to analyze large groups of student sequence data and generate informative reports and visualizations that can be used to gain insights into a class' knowledge and improve future activities.

8.2 Contribution Summary

The work described in this dissertation can be summarized in the following contributions:

Contribution 1 - *Free-Form Tutor Authoring Tool.* We believe the majority of current intelligent tutors rely on overly simplistic forms of input from students (e.g. multiple-choice questions) because the evaluation of such input is trivial; however, it is common for students to find ways to game these systems or simply guess their way to the correct answer. There are a number of intelligent tutors that allow for more free form student input; however, we believe these systems tend to be complicated for authoring tutors and usually require the students to provide extra hints to the system in order for the work to be analyzed. The problem with adding these hints is that it increases a student's cognitive load, thus making it more difficult to measure a student's understanding of a topic.

We have developed the first online intelligent tutoring system that allows teachers to ask mathematical graph-based questions and provides students with multi-tiered feedback based on their answers. We have chosen to focus on graphs because (1) graphs frequently appear throughout curriculum, especially STEM disciplines, and because (2) their free-form nature makes it difficult for students to guess the correct answer. Our authoring tool allows teachers to quickly create intelligent tutors in an intuitive and visual manner by setting constraints for the characteristics that are desired from a *correct* drawing along with feedback to present students when these constraints are not met. In addition to mathematical graphs, our system is able to evaluate and analyze student-drawn chemistry molecules, computer science graphs, and simple free-form drawings. All of this functionality results is a flexible system that can be used in a variety of disciplines. Furthermore, preliminary results show learning gains for some activities.

Contribution 2 - *Student Data Analysis Tool.* One of the main advantages of an intelligent tutoring system (ITS) is the removal of a human teacher. This is especially important when the system is used with large group sizes and individual attention from teachers is not possible due

to time-constraints. Once a tutor is authored, ITSs are able to provide automatic and personalized instruction to each student. However, the information that can be obtained from the these systems is mostly quantitative, and as such it removes the rich, temporal nature of the actions students are performing.

The tools presented are designed to try and meet this shortcoming. Our system records the actions of each student. This process generates a large amount of sequence data, especially when it is used in large class sizes. The tools described use a new set of analysis techniques and visualizations that enable the analysis of these data sets. Using our system to collect and replay student work, teachers can track individual students and understand the source of their errors. Furthermore, our system contains the ability to perform analysis of groups of students using hidden Markov model-based clustering, the results of which give teachers a summary of a class's performance and common student strategies. With these abilities, teachers have the ability to provide further instruction to students who are struggling with concepts and to directly use the results of their analysis to improve the tutor with more directed feedback for future students. In addition, our analysis tools can be used with data sources outside of our system.

8.3   Expected Impact

We have developed an authoring tool for creating intelligent tutors with semi-free form input and we have built an analysis system to automatically cluster and visualize students based on their sequences of actions. We believe this research will impact several aspects of pedagogical software. By demonstrating various ways to recognize and evaluate free-form input from students, we believe students will be able to interact with software in a more natural way and with less frustration. Furthermore because these sytems make it harder for students to make correct *guesses*, we will have more meaningful engagement with students and thus produce deeper levels of learning. We expect other systems to follow our lead and move away from rigid and simplified input methods (e.g., multiple choice/matching) and look to incorporate more natural forms of inputs for students. By clustering students with similar solution patterns, we are able to provide more targeted instruction to students who are struggling with concepts. We expect that other educational systems will take advantage of our analysis techniques to make similar discoveries within their disciplines. In summary, this work will set a new direction for the design of future intelligent tutoring systems with respect

to student input and data analysis.

Appendices

Appendix A   SocraticGraphs Context Free Grammar

```
Max → MaxValue NumberConditional IntValue

MaxValue → maximums in CurveLocation


Min → MinValue NumberConditional IntValue

MinValue → maximums in CurveLocation


AreaUnderCurve → AUCValue NumberConditional DoubleValue

AUCValue → area under the curve in CurveLocation


Slope → SlopeValue NumberConditional DoubleValue

SlopeValue → slope of CurveLocation


CurveShape → CurveShapeValue CurveShapeConditional Shape

CurveShapeValue → curve shape of CurveLocations

Shape → linear | exponential | logarithmic | parabolic | sigmoidal

CurveShapeConditional → is | is not


XOrYCoordinate → XOrYValue NumberConditional DoubleValue

XOrYValue → XOrY coordinate of the MajorPoint point of CurveLocation

XOrY → x | y


XAndYCoordinate → XAndYValue XAndYConditonal Area

XAndYValue → (x,y) coordinates of the MajorPoint point of CurveLocation

XAndYConditional → = | !=


Intersection → intersection in which CurveLocation IntersectionConditional Area

IntersectionConditional → crosses | does not cross | is contained within | is not contained within


NumberOfDrawnCurves → NumberOFDrawnCurveValue NumberConditional IntValue

NumberOfDrawnCurvesValue → the number of drawn curves


IntValue → int | MaxValue | MinValue | NumberOFDrawnCurvesValue

DoubleValue → double | AreaUnderCurve | Slope

Area → Coordinates | CoordinateList | MathEquation | UserDrawnArea

Coordinates → (double, double)

CoordinatesList → Coordinates | Coordinates CoordinatesList
```

```
MathEquation → CoordinatesList

UserDrawnArea → CoordinatesList

CurveLocation → curve double | CurveSegment

CurveSegment → curve segment in curve int from CurveSegmentLocation to CurveSegmentLocation

CurveSegmentLocation → absolute double | X = double | Y = double

NumberConditional → = | != | < | > | <= | >=

MajorPoint → first | last | minimum | maximum | mean | mid
```

Listing 1: Full Context Free Grammar for the evaluating graph drawings with the SocraticGraphs module
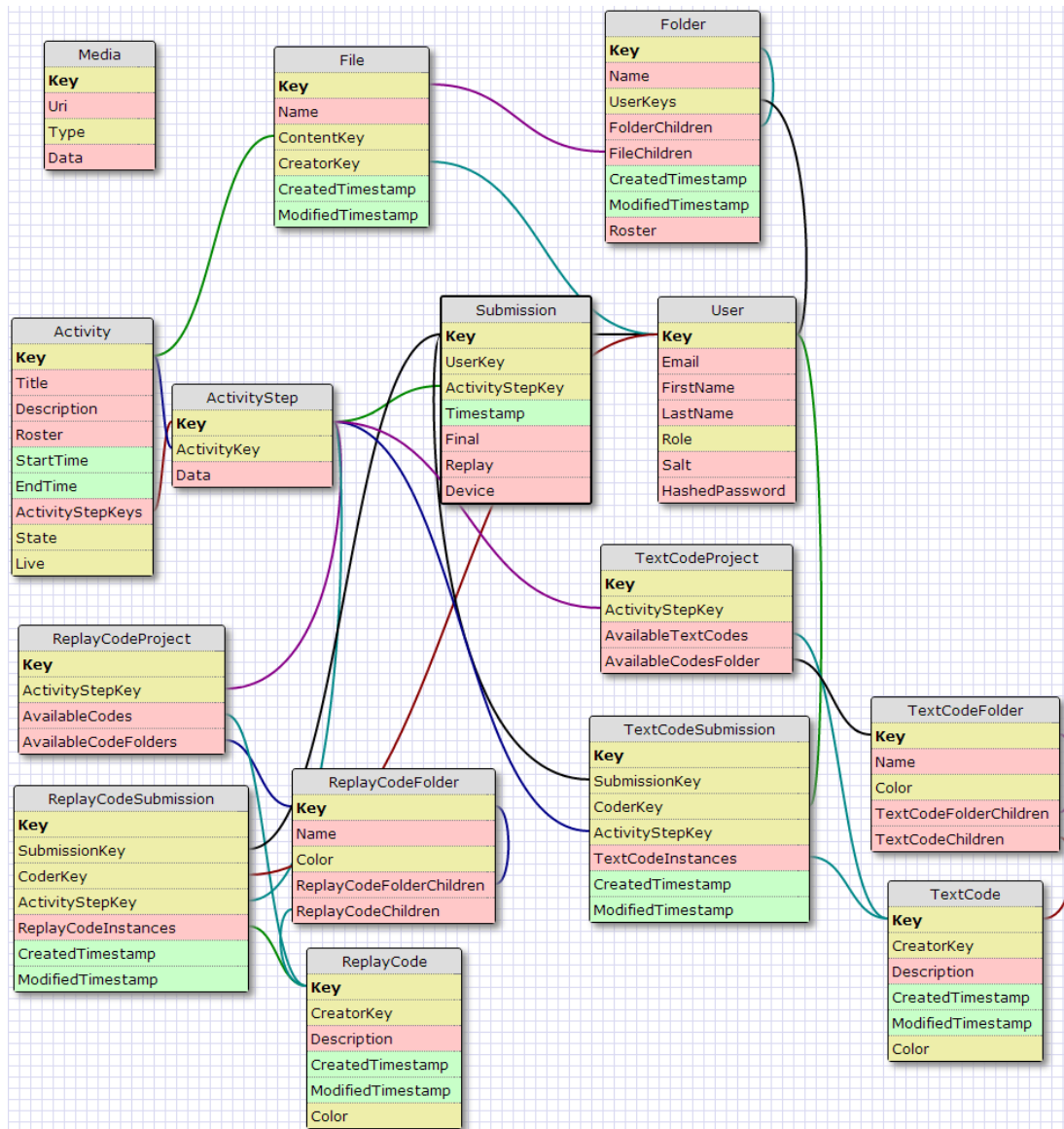
Appendix B   BeSocratic Database Diagram

Figure 1: Diagram of BeSocratic's database

Bibliography

[1] Education and human resources (ehr) active funding opportunities, 2011.

[2] Ieee educational software, 2011.

[3] S. Ainsworth and S. Grimshaw. Evaluating the redeem authoring tool: Can teachers create effective learning environments? *Int.J.Artif.Intell.Ed.*, 14(3,4):279–312, December 2004.

[4] T. Aleahmad, V. Aleven, and R. Kraut. Creating a corpus of targeted learning resources with a web-based open authoring tool. *Learning Technologies, IEEE Transactions on*, 2(1):3–9, 2009.

[5] V. Aleven, B. M. McLaren, and J. Sewall. Scaling up programming by demonstration for intelligent tutoring systems development: An open-access web site for middle school mathematics learning. *IEEE Trans.Learn.Technol.*, 2(2):64–78, April 2009.

[6] V. Aleven, J. Sewall, B. M. McLaren, and K. R. Koedinger. Rapid authoring of intelligent tutors for real-world and experimental use. In *Proceedings of the 6th IEEE International Conference on Advanced Learning Technologies*, pages 847–851, 2006.

[7] M. W. Alibali. Gesture in spatial cognition: Expressing, communicating, and thinking about spatial information. *Spatial Cognition and Computation*, 5(4):307–331, 2005.

[8] R. S. Baker, A. T. Corbett, K. R. Koedinger, and I. Roll. *Detecting When Students Game the System, Across Tutor Subjects and Classroom Cohorts*, pages 220–224. User Modeling 2005. 2005.

[9] L. E. Baum and J. A. Eagon. An inequality with applications to statistical estimation for probabilistic functions of markov processes and to a model for ecology. *Buletin of the American Mathematical Society*, 73(3):360–363, 1967.

[10] L. E. Baum and T. Petrie. Statistical inference for probabilistic functions of finite state markov chains. *The Annals of Mathematical Statistics*, 37(6):1554–1563, 1966.

[11] L. E. Baum, T. Petrie, G. Soules, and N. Weiss. A maximization technique occurring in the statistical analysis of probabilistic functions of markov chains. *The Annals of Mathematical Statistics*, 41(1):164–171, 1970.

[12] C. Beal, S. Mitra, and P. R. Cohen. Modeling learning patterns of students with a tutoring system using hidden markov models. In *Proceedings of the 2007 conference on Artificial Intelligence in Education: Building Technology Rich Learning Contexts That Work*, pages 238–245, Amsterdam, The Netherlands, The Netherlands, 2007. IOS Press.

[13] S. B. Blessing, S. B. Gilbert, S. Ourada, and S. Ritter. Authoring model-tracing cognitive tutors. *International Journal of Artificial Intelligence in Education*, 19(2):189, 2009.

[14] J. E. Bresenham. Algorithm for computer control of a digital plotter. *IBM Syst.J.*, 4(1):25–30, March 1965.

[15] S. Bryfczynski. Graphpad: a cs2/cs7 tool for graph creation. In *Proceedings of the 47th Annual Southeast Regional Conference*, ACM-SE 47, pages 53:1–53:3, New York, NY, USA, 2009. ACM.

[16] S. P. Bryfczynski. Genericpad: A framework for capturing and analyzing the cognitive processes of students, 2009.

[17] S. P. Bryfczynski, S. M. Underwood, N. P. Grove, R. P. Pargas, and M. M. Cooper. *OrganicPad as a research tool: Investigating the development of representational competence in chemistry*, pages 3–10. The Impact of Tablet PCs and Pen-based Technology on Education: Going Mainstream, 2010. Purdue University Press, 2010. 2010040248.

[18] E. Case, R. Stevens, and M. M. Cooper. Is collaborative grouping an effective instructional strategy?: Using immex to find new answers to an old question. *Journal of College Science Teaching*, 36(6):42, 501. ID: 6711; Acquisition Information: National Science Teachers Association. 1840 Wilson Boulevard, Arlington, VA 22201-3000. Tel: 800-722-6782; Fax: 703-243-3924; e-mail: membership@nsta.org; Web site: http://www.nsta.org; Language: English; Education Level: Higher Education; Reference Count: 14; Journal Code: JUL2007; Level of Availability: Not available from EDRS; Publication Type: Journal Articles; Publication Type: Reports - Evaluative; Entry Date: 2007.

[19] J. Cohen. *Statistical Power Analysis for the Behavioral Sciences*. Lawrence Erlbaum Associates, Hillsdale, New Jersey, 2 edition, 1988.

[20] S. Cook, Z. Mitchell, and S. Goldin-Meadow. Gesturing makes learning last. *Cognition*, 106:1047–1058, 2008.

[21] S. W. Cook and S. Goldin-Meadow. The role of gesture in learning: Do children use their hands tochange their minds? *Journal of Cognition and Development*, 7(2):211–232, 2006.

[22] M. M. Cooper, C. T. Cox, M. Nammouz, E. Case, and R. Stevens. An assessment of the effect of collaborative groups on students' problem solving strategies and abilities. *Journal of Chemical Education*, 85(6):866–872, 2008.

[23] M. M. Cooper, N. P. Grove, R. Pargas, and T. Bryfczynski, S. P. Gatlin. Organicpad: An interactive freehand drawing application for organic chemistry. *Chemical Education: Research and Practice*, 10(4):296–301, 2009.

[24] M. M. Cooper, S. M. Underwood, N. Grove, S. Bryfczynski, and R. Pargas. Organicpad: a freehand interactive application for the development of representational competence. *ConfChem*, pages November 6, 2010, 2010.

[25] K. K. A. J. Corbett, A. *Intelligent tutoring systems*, pages 849–874. Handbook of human-computer interaction. Elsevier, Amsterdam, 1997.

[26] D. H. Douglas and T. K. Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica The International Journal for Geographic Information and Geovisualization*, 10(2):112–122, 1973.

[27] T. G. Evans. *A program for the solution of a class of geometric-analogy intelligence-test questions*, pages 271–353. Semantic Information Processing. MIT Press, Cambridge, Massachusetts, 1968.

[28] K. Forbus, J. Usher, A. Lovett, K. Lockwood, and J. Wetzel. Cogsketch: Sketch understanding for cognitive science research and for education. *Topics in Cognitive Science*, 3(4):648–666, 2011.

[29] R. Freedman. What is an intelligent tutoring system? *Intelligence*, 11(3):15–16, 2000.

[30] J. K. Gilbert. *Visualization: A Metacognitive Skill in Science and Science Education*, pages 9–28. Visualization in Science Education. Springer, Dordrecht, The Netherlands, 2005.

[31] J. K. Gilbert. *Visualization in Science Education*, volume 1 of *Models and Modeling in Science Education*. Springer, The Netherlands, 2005.

[32] R. Glaser. Cognitive and environmental perspectives on assessing achievement. In *Assessment in the Service of Learning ETS Invitational Conference*, Princeton, NJ., 1988.

[33] R. Glaser. *Expertise and Assessment*. Testing and Cognition. Prentice-Hall, Englewood Cliffs, NJ, 1991.

[34] S. Goldin-Meadow. *Hearing Gesture: how our hands help us think*. Harvard University Press, Cambridge, MA, 2005.

[35] C. M. Grinstead and L. J. Snell. *Grinstead and Snell's Introduction to Probability*. The CHANCE Project; American Mathematical Society, version dated 4 july 2006 edition, 2006.

[36] N. P. Grove and M. M. Cooper. Decorating with arrows. *Journal of Chemical Education*, "submitted".

[37] R. A. R. Gurung. *Teaching of Psychology*, 30(2):92, 2003.

[38] L. Hagen and A. B. Kahng. New spectral methods for ratio cut partitioning and clustering. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 11(9):1074–1085, sep 1992.

[39] A. Harrer, B. M. McLaren, E. Walker, L. Bollen, and J. Sewall. Creating cognitive tutors for collaborative learning: steps toward realization. *User Modeling and User-Adapted Interaction*, 16(3-4):175–209, September 2006.

[40] A. B. Hostetter and M. W. Alibali. Visible embodiment: Gestures as simulated action. *Psychonomic Bulletin and Review*, 15(3):495–514, 2008.

[41] H. Jeong, A. Gupta, R. Roscoe, J. Wagster, G. Biswas, and D. Schwartz. Using hidden markov models to characterize student behaviors in learning-by-teaching environments. In *Proceedings of the 9th international conference on Intelligent Tutoring Systems*, ITS '08, pages 614–625, Berlin, Heidelberg, 2008. Springer-Verlag.

[42] M. Johnson and T. Barnes. *Visualizing Educational Data from Logic Tutors*, pages 233–235. Intelligent Tutoring Systems. 2010.

[43] D. E. Knuth. *METAFONT: the program*. Addison Wesley Pub. Co, 1986. lc86001230.

[44] K. R. Koedinger, J. R. Anderson, W. H. Hadley, and M. A. Mark. Intelligent tutoring goes to school in the big city. *International Journal of Artificial Intelligence in Education*, (8):30–43, 1997.

[45] S. E. Kuehne, K. D. Forbus, D. Gentner, and B. Quinn. Seql: Category learning as progressive abstraction using structure mapping, 2000.

[46] K. Lancaster, J. M. Chamberlain, R. Parson, and K. K. Perkins. Examining the effect of guidance on student engagement with an interactive simulation. 2012.

[47] A. Lesgold, S. Lajoie, M. Bunzo, and G. Eggan. *SHERLOCK: A coached practice environment for an electronics troubleshooting job*, pages 201—238. Computer assisted instruction and intelligent tutoring systems Shared goals and complementary approaches. Erlbaum, 1992.

[48] V. I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. Technical Report 10, 1966.

[49] M. Mark and J. E. Greer. The vcr tutor: Effective instruction for device operation. *The Journal of the Learning Sciences*, 4(2):209–246, 1995.

[50] D. L. Martin L., Schwartz. Prospective adaptation in the use of external representations. *Cognition and Instruction*, 24(4):370–400, 2009.

[51] A. Mitrovic, K. R. Koedinger, and B. Martin. A comparative analysis of cognitive tutoring and constraint-based modelling. pages 313–322. Springer-Verlag, 2003.

[52] A. Mitrovic, B. Martin, P. Suraweera, K. Zakharov, N. Milik, J. Holland, and N. McGuigan. Aspire: An authoring system and deployment environment for constraint-based tutors. *International Journal of Artificial Intelligence in Education*, 19(2):155, 2009.

[53] T. Murray. Authoring intelligent tutoring systems: An analysis of thestate of the art. *International Journal of Artificial Intelligence in Education*, (10):98—129, 1999.

[54] M. J. Nathan and A. Petrosino. Expert blind spot among preservice teachers. *American Educational Research Journal*, 40(4):905–928, 2003.

[55] S. B. Needleman and C. D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48(3):443–453, March 1970.

[56] S. Ohlsson and A. Mitrovic. Fidelity and efficiency of knowledge representations for intelligent tutoring systems. *Joseph M. Scandura (Ed.)*, pages 101—132, 2007.

[57] R. Pargas, M. M. Cooper, C. Williams, and S. Bryfczynski. Organicpad: A tablet pc based interactivity tool for organic chemistry. May 24 2007.

[58] R. P. Pargas and S. Bryfczynski. Using ink to expose students' thought processes in cs2/cs7. In *Proceedings of the 40th ACM technical symposium on Computer science education*, SIGCSE '09, pages 168–172, New York, NY, USA, 2009. ACM.

[59] L. R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. In *Proceedings of the IEEE*, pages 257–286, 1989.

[60] L. B. Resnick and D. P. Resnick. *Assessing the Thinking Curriculumn: New Tools for Education Reform*, pages 37–76. Changing Assessments: Alternative Views of Aptitude, Achievement and Instruction. Kluwer Academic Publishers, Boston, MA, 1992.

[61] S. Ritter, J. Anderson, M. Cytrynowicz, and O. Medvedeva. Authoring content in the pat algebra tutor. *Journal of Interactive Media in Education*, 1998(2), 1998.

[62] D. L. Schwartz, R. Lindgren, and S. Lewis. *Constructivism in an age of non-constructivist assessments*. Constructivist Instruction: Success or Failure? Routledge, NY, NY, 2009.

[63] P. H. Sellers. On the theory and computation of evolutionary distances. *SIAM Journal on Applied Mathematics*, 26(4):787–793, 1974.

[64] L. Shepard. Interview on assessment issues with lorrie shepard. *Educational Researcher*, 20(2):21–23, 1991.

[65] J. Shi and J. Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22:888–905, 1997.

[66] B. Shih, K. R. Koedinger, and R. Scheines. Discovery of learning tactics using hidden markov model clustering. In *3rd International Conference on Educational Data Mining*, 2010.

[67] J. Stamper, T. Barnes, and M. Croy. *Enhancing the Automatic Generation of Hints with Expert Seeding*, pages 31–40. Intelligent Tutoring Systems. 2010.

[68] R. Stevens, D. F. Johnson, and A. Soller. Probabilities and predictions: Modeling the development of scientific problem-solving skills. *Cell Biology Education*, Spring 2005.

[69] D. Tenneson and S. Becker. Chempad: Generating 3d molecules from 2d sketches. In *In SIGGRAPH '05: ACM SIGGRAPH 2005 Posters*, page 87. ACM Press, 2005.

[70] E. R. Tufte. *The Visual Display of Quantitative Information*. Graphics Press, second edition, 2001.

[71] U. von Luxburg. A tutorial on spectral clustering. *CoRR*, abs/0711.0189, 2007.

[72] B. P. Woolf and P. Cunningham. Multiple knowledge sources in intelligent teaching. Technical report, University of Massachusetts, 1987.