EXPLORING THE VISUALIZATION OF STUDENT BEHAVIOR IN INTERACTIVE LEARNING ENVIRONMENTS

by

Matthew W. Johnson

A dissertation submitted to the faculty of
The University of North Carolina at Charlotte
in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in
Computing and Information Systems

Charlotte

2013

ABSTRACT

MATTHEW W. JOHNSON. Exploring the visualization of student behavior in Interactive Learning Environments. (Under the direction of DR. TIFFANY BARNES)

My research combines Interactive Learning Environments (ILE), Educational Data Mining (EDM) and Information Visualization (Info-Vis) to inform analysts, educators and researchers about user behavior in software, specifically in CBEs, which include intelligent tutoring systems, computer aided instruction tools, and educational games.

InVis is a novel visualization technique and tool I created for exploring, navigating, and understanding user interaction data. InVis reads in user-interaction data logged from students using educational systems and constructs an Interaction Network from those logs. Using this data InVis provides an interactive environment to allow instructors and education researchers to navigate and explore to build new insights and discoveries about student learning.

I conducted a three-point user study, which included a quantitative task analysis, qualitative feedback, and a validated usability survey. Through this study, I show that creating an Interaction Network and visualizing it with InVis is an effective means of providing information to users about student behavior. In addition to this, I also provide four use-cases describing how InVis has been used to confirm hypotheses and debug software tutors.

A major challenge in visualizing and exploring the Interaction Network is network's complexity, there are too many nodes and edges presented to understand the data efficiently. In a typical Interaction Network for twenty students, it is common to have hundreds of nodes, which to make sense of, has proven to be too many. I present a network reduction method, based on edge frequencies, which lowers the number of edges and nodes by roughly 90% while maintaining the most important elements of the Interaction Network. Next, I compare the results of this method with three alternative approaches and show our reduction method produces the preferred results. I also present an ordering detection method for identifying solution path redundancy because of student action orders. This method reduces the number of nodes and edges further and advances the resulting network towards the structure of a simple graph.

Understanding the successful student solutions is only a portion of the behaviors we are interested in as researchers and educators using computer based educational systems, student difficulties are also important. To address areas of student difficulty, I present three different methods and two visual representations to draw the attention of the user to nodes where students had difficulty. Those methods include presenting the nodes with the highest number of successful students, the nodes with the highest number of failing students, and the expected difficulty of each state. Combined with a visual representation, these methods can draw the focus of users to potentially important nodes, which contain areas of difficulty for students. Lastly, I present the latest version of the InVis tool, which is a platform for investigating student behavior in computer based educational systems. Through the continued use of this tool, new researchers can investigate many new hypotheses, research questions and student behaviors, with the potential to facilitate a wide range of new discoveries.

ACKNOWLEDGEMENTS

This project was made possible by the National Science Foundation and their research grant IIS 0845997 - CAREER: Educational Data Mining for Student Support in Interactive Learning Environments. Thank you to Ron Smelser for his edits, suggestions and help throughout my doctoral program. I thank my committee members for providing invaluable feedback, suggestions and discussion. Thank you to my advisor for her guidance, input, conference funding and always keeping the bills paid. Thank you to Michael Eagle for all of his hard work, inspiring conversation and time spent working together. I am truly and deeply thankful for the wonderful parents I have who always expected the most from me and were always there to help me in every way. Thank you to both my siblings, Caitlin and Wes who continuously pushed me to do more and more.

Lastly, for the person who has helped me the most, a person who took upon herself more than anyone will ever know, I thank my brilliant and wonderful wife, Tomoko Okimoto. Every accomplishment I have made, including this one, has been supported with an infinite amount of love, caring and dedication. Her time, energy and effort towards supporting both me and our lovely daughter Wakako, was essential to our success. I am now and always will be grateful to her for all she has given me.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1: INTRODUCTION

The National Academy of Engineers, in 2008, declared the advancement of personalized learning as a Grand Challenge[46]. The increasing use of the web in education and learning management systems is rapidly increasing the amount of data that can be brought to bear on this important challenge. As of 2010, the PSLC DataShop, a repository for educational data, had collected logs from over 42,000 students from different tutors with a wide range of topics, from algebra to Chinese[35]. However, such large data-sets can be unwieldy, and deciding just how to use them to improve learning is a challenge as illustrated by the emergence of the new field and conference on Educational Data Mining. These data sets have the potential to provide invaluable insights into student learning and how it might be better supported or improved. However, there are few tools beyond those supplied by the PSLC that allow researchers and educators to investigate, explore, and understand how students solve problems in particular learning environments.

One way to achieve the goal of providing personalized learning is to improve Interactive Learning Environment (ILE) tools to adapt to individual learners in the way that intelligent tutoring systems (ITSs) do. The goal of this research is to provide tools that will allow researchers to better understand student learning and identify areas where CBE systems can be improved. These tools will focus on interactive visualization of educational data collected from students during problem-solving in ILEs.

My first research question is: Can we build a visualization of problem-solving data that will allow instructors and researchers to better understand student behavior and learning?

To address this question, we developed InVis, a non-domain specific, interactive visualization tool to explore and navigate Interaction Networks. InVis is not guaranteed to work in *all* domains, but rather is not limited to a single domain. The focus of InVis and the related research is on multi-step, problem solving, particularly for STEM fields, science, technology, engineering and math, where problems can easily be modelled as a state-transition diagram with a clear problem start and problem goal. Interaction Networks are the data structure we use to represent the steps students take in problem solving. I demonstrate the use of InVis for visualizing and exploring student problem-solving data with a triangulated approach user study that includes a validated usability study, a quantitative task-analysis, and qualitative feedback. Results show users could successfully use InVis to investigate student behavior, generate and confirm hypotheses, and gain insights into ILE design. I demonstrate the application of InVis through three case studies that illustrate the insights that InVis can provide about student behavior. I show how educators generated hypotheses about students from the data, and then used the tool to confirm or reject those hypotheses.

This work is a first step towards allowing researchers to examine tutor data, and is intended to help inform decision making for educators and tutor developers, it is also likely be helpful to researchers studying ILE systems. There are many types of

ILE tutors, many of which are built with scaffolding in mind. Although a system providing insight about any type of tutor would be wonderful, the focus of InVis is on tutors addressing open-ended problem solving. I define open-ended problems as tutor-problems which require multiple steps to solve the problem and have one or more ways to solve the problem. Furthermore, InVis is most useful when a tutoring systems 'state' can be clearly defined, and students will have solution overlap, which I will in chapter 3.

The initial design of InVis provided a much needed, non-domain specific view of computer tutor data; however, it did not provide users with a good summary or sense of the overall student problem-solving skills or methods. The user-study showed that providing interactive elements for exploring and navigating Interaction Networks alone are not sufficient to support the understanding of these large, complex networks representing student behavior. The major drawback to this initial design is that the number of states increases with the number of students and the number of actions they perform. The state space of some solutions can become very large; for 23 students for a single logic problem, 662 nodes were generated to populate the Interaction Network. When viewing 100 or more students at a time, meaningful tasks become too difficult and tedious to perform, hindering one's ability to understand student problem solving approaches quickly or effectively. In such cases, the Interaction Network is too verbose, and instructors and researchers need better ways to see an overview of the data to better direct their exploration.

My second research question is: Can we leverage network analysis, sequential pattern mining and other data mining techniques to identify important characteristics and the underlying structure of the solution space represented by the Interaction Network?

To address this question, I extend the functionalities of InVis and incorporate three novel concepts with respect to the Interaction Network. I present a strategy reduction algorithm and an ordering consolidation approach that can be performed on the Interaction Network to better focus on the underlying structure of the solution space. These two methods when combined offer a great deal of network size and order reduction, preserve the most common actions and solution-paths, reduce redundant information, draw attention to the important features, and help advance the resulting network towards a simple graph. This manages the complexity and presentation of the student data-driven solution space, in the form of the Interaction Network, and facilitates a greater ability to identify interesting student problem-solving behavior.

The third approach is a state difficulty detector. The theory behind this concept is founded in problem difficulty from classical test theory. InVis has three different methods for displaying state difficulty: we present the states to the user in order of goal-frequency, failure-frequency, as well as expected difficulty on a per state basis. The difficulty detector combined with a difficulty view, where visual characteristics of the presentation are adjusted to facilitate the view, allow users to quickly investigate where students faced difficulty. This type of information provides educators with visual identity of the challenges students faced and focus can be directed to regions of the solution-space where the student data provides evidence of those challenges. In combination with systems like the Hint Factory [55], educators can more efficiently generate well-crafted hints to the regions where students exhibit the highest rates of difficulty.

The research questions asked are directed at finding ways to help instructors and researchers gain an overview of general student behavior while solving a single problem. Specifically, InVis will support instructor and researcher exploration of problem-solving data by providing the following tools to:

- Identify common strategies students used to solve problems.
- Identify areas of difficulty for students.
- Provide a general overview while allowing users to explore interesting student behavior in more detail.

We support these tasks through a usable interface that helps instructors and researchers focus on important information at a higher level, while still allowing detailed inspection of interesting cases. This functionality is based on Interaction Network data. The approach is not specific to any single tutor or domain, but may not work for all domains, I describe our attempts with five different data-sets. Tutor development and interactive learning environment construction are long, complex tasks and their continued refinement is important as this field continues to move forward. Student data visualization can aid in these processes and going forward will likely be an important part in focusing on developmental efforts which should provide maximal impact on educational efforts.

CHAPTER 2: RELATED WORK

It takes 100-1,000 hours of development time to build one hour of Intelligent Tutoring System content [44]. On the other hand, Interactive Learning Environments (ILE) are increasing by the day. However, many of these environments lack intelligent features that extend them from being a ILE tool into an Intelligent Tutoring System(ITS). An ITS is a sub-type of Interactive learning environment, which usually requires some level of personalization or 'intelligence' most often implemented through student modelling. Generating the intelligence, with tools like CTAT [37], is a long and difficult process, requiring a high level of expertise. By improving automated, non-domain specific methods for offering insight, we can make significant improvements in the area of personalized learning.

The three areas we will address in this related work section are Interactive Learning Environments (ILE), Educational Data Mining (EDM) and Information Visualization (Info-Vis). Each one of these areas plays a role in my contribution, leveraging portions of each field in which they are most effective. John Stamper's Markov Decision Process (MDP) graph is the leading inspiration for this work, showing that a graph of tutor-log states can be used for improving tutor interventions. An intervention in this sense is any activity, technique or method for the purpose of teaching a topic. Stamper was able to use the tutor-log graph of states in order to create the Hint Factory [55], a domain independent method of generating context-sensitive hints for users.

This Related Work chapter is presented in three sections. In the Hint Factory section, I present an overview of Stamper's work and how it lays the foundation for ours. Next, in section 2.3 and section 2.4 I present the Deep Thought logic tutor and BeadLoom Game, two different CBEs whose log data I have used for analysis. Following those CBE systems, in section 2.5 I review related visualizations for educational data.

2.1 The Hint Factory

The Hint Factory is an end-to-end approach that builds a domain model from student problem-solving data, creates an MDP on that model, and uses the MDP to automatically generate context-sensitive, individualized, hints upon student request. The Hint Factory approach begins by building a student solution graph of all student attempts for a particular problem. Each state in the graph is like a snapshot of the the problem-solving attempt (from the tutor log), and transitions between states are the actions students take to perform the next problem-solving step. The student solution graph can help us better understand student problem-solving behavior, and build better interventions to improve the tutor or learning. However, student solution graphs, even for simple logic problems, can contain thousands of nodes, for example our 2009 data set for problem 3.5 of Deep Thought has 1375 states for 190 students, making it difficult to build good human understanding. The primary cause of this difficulty is related to complexity of the of graph, with so many nodes and edges, even with basic interaction tools, they are too large to effectively navigate and gain insight from.

The hint factory has been extensively applied to the Deep Thought propositional logic proof tutor [7, 55]. Deep Thought stores student data in a transactional database, like many CBEs represented in the Datashop, a data-repository for CBE-log data [35]. These actions can be used to describe the state of a tutor, or in other words a snapshot of what a student has done so far in a particular tutor or problem. Typically the start state is the problem definition; in Deep Thought, this is defined as the set of premises supplied when the problem is first loaded, along with the conclusion to be derived.

Next, Deep Thought logs student actions as they select logic statements and apply logic axioms to combine them to derive the desired conclusion. The student may select some premise and apply an action, like simplification, and a log-transaction is recorded. The result of that action creates a new state, which would include the newly generated logic statement.

After the student has solved the problem, the history of their transitions is recorded as a sequence of state-action pairs. The hint factory uses a dataset (typically from one or more classes of students) to generate a graph that includes all of the states and actions. Later when a new student is using the tutor, and requests a hint, the system compares the student's current state to all the states available in the hint factory and finds a match. Once a match is found, the hint factory looks at the set of next states, and finds the state which is best to move to next, which is done through a Markov Decision Process (MDP), as described in Stamper's work [56, 7, 9, 8]. Once the next desired state is determined, a hint that guides the student to that next state is presented, and this creates a contextualized hint for the student.

The contextualized hint from the hint factory, is the intelligence of the Intelligent Tutoring System, and is also considered adaptive feedback or personalized learning, because it is unique to the particular student's problem-solving context when they request a hint. Essentially the hint factory looks at the current solution provided by the student and matches that solution to a previously completed solution from past student data. The hint is based on the past student's next state, providing a hint based on the next state the current student should work towards. Stamper, et al. showed that students solving logic problems in the Deep Thought tutor were three times less likely to drop out from ever using the tutor again, when they received hints using the Hint Factory [7]. Hint students spent a similar amount of time in the tutor but were able to complete more problems, and hint students were less likely to drop out, a significant advantage.

The major contribution of Stamper, Barnes, and Croy's work is that hint generation is fast and purely data-driven. Most hints in ILEs are hand-generated by experts, such as those for the geometry tutor [36], the Andes physics tutor [30], and even a domain-independent tutor, The Help Tutor[52] a tutor for help-seeking behavior in any cognitive tutor. Help-seeking behavior in this sense, is the act of requesting help when you need it, an important skill for learners. The downside to manually generating hints is that it requires an expert in the field, and can be expensive in terms of resources, particularly the expert's time. Not only must the expert be a domain expert, but typically an expert in ITS development as well, as most systems lack an intuitive interface. This means it is not easy to alter ILEs, though CTAT aims at alleviating some of these authoring costs. CTAT, the Cognitive Tutor Authoring Tool, which allows teachers to author hints they would like to provide given specified student responses to questions [37]. Though CTAT provides an important service, the Hint Factory addresses an important problem, converting a Computer Aided Instruction tool into an Intelligent Tutoring System, through data-driven means, as was done with the Deep Thought Logic Tutor. An important advantage of this approach is that it reduces ITS development costs, allowing more CBE systems to incorporate personalized tutoring.

A major limitation of the Hint Factory is that it is difficult to build understanding of the state-space that is generated from the data. The Hint Factory's focus is generating a hint for most states, that provides guidance to a new state closer to the goal. However, without InVis, educators, researchers and ILE developers cannot **see** that rich state-space. InVis provides a means of looking at how students *explored* the solution space of a problem. Through visualizing the data and solution-space we should be able to understand the most common strategies of our population, while also identifying states where students experience high failure rates. The purpose of our work is to provide a method of letting developers, educators and researchers explore, visualize and understand their users' data through visualization and data analysis.

2.2 Problem Solving Environments

The focus of InVis is open-ended problem-solving environments. For this work, we define open-ended problem solving environments as tutors where students can apply one of many different actions and must use many actions in order to solve a single problem, as in the Deep Thought tutor [21]. Other examples include tutors from ASSISTments [60], and games like the BeadLoom Game [15, 14].

This is an important distinction as there are other types of ILE systems which rely heavily on scaffolding and knowledge tracing [18]. Scaffolding in problems have had clear pedagogical benefits [48]. The idea of scaffolding comes from the idea that educators should provide the information necessary for the learner to move forward step by step beyond their current knowledge [16]. Though scaffolding has been shown to be successful, there could be benefit in providing open-ended problem solving environments.

In scaffolding and knowledge tracing systems, typically each question is associated to a skill or set of skills, a student is required to answer a single question at a time, much like a multiple choice test. InVis is not designed to visualize data from these types of systems, because typically the solution space is small, or as in the case of multiple choice, the smallest possible space. That is, for a single problem, there is only one action, choose the answer, and the answer is either correct or incorrect.

InVis is designed to explore the open-ended problem-solving environments, which means there are multiple actions required to solve the problem, and different solutions for solving the problem. With more open environments, students have a larger state-space to explore for solutions, consider the multiple ways to calculate a mathematical problem, or programming a program.

Additionally, open problem-solving tutors may have an important role in education. Bloom's taxonomy of learning domains is a categorization of tasks in which domains higher in the taxonomy cannot be performed until the lower domains have been mastered, suggesting greater intellectual skills [13]. Open problem-solving based tutors may encourage learning in higher levels of Bloom's cognitive domain.

By developing InVis in such a way that its techniques not specific to a single domain, we can allow the visualization of data from a wide range of tutors, like the ones mentioned above. This will allow us to identify common strategies, subgoals, and areas of difficulty in a number of tutors. Furthermore, InVis users will be able to explore and navigate the solution-spaces generated from student data in a wide range of domains.

The way InVis works is through the visualization of the Interaction Network, which in short is the solution space of states generated from transactional logs of students performing actions in a tutor. The complete details are covered in section 2.6. Before I explain the Interaction Network in detail, let us first look at two different tutors and how they work, so it will be more clear how we transform their log-data into the Interaction Networks.

2.3 The Deep Thought Tutor

The Deep Thought (DT) logic tutor is a propositional logic tutor developed by Marvin Croy at the University of North Carolina at Charlotte. Dr. Croy has been collecting DT data for over a decade, and the problems are of the appropriate open-ended nature. We collaborate with Dr. Croy to build new features and test our hypotheses with DT [20].

Deep Thought is an Intelligent Tutoring System for propositional logic for perform-

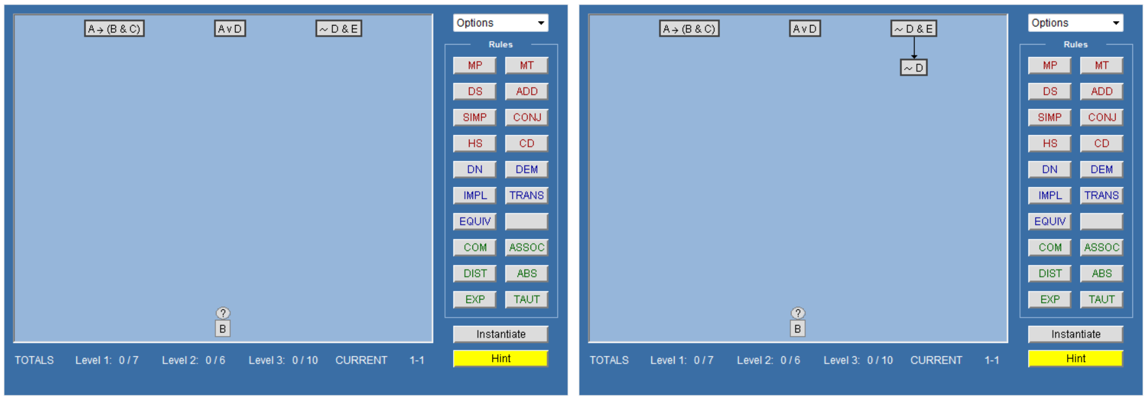


Figure 1: The Deep Thought tutor, on the left we have the start of the problem. The start premises are at the top, the goal is derive B. On the right the student has simplified the premise $\neg D \wedge E$ to $\neg D$.

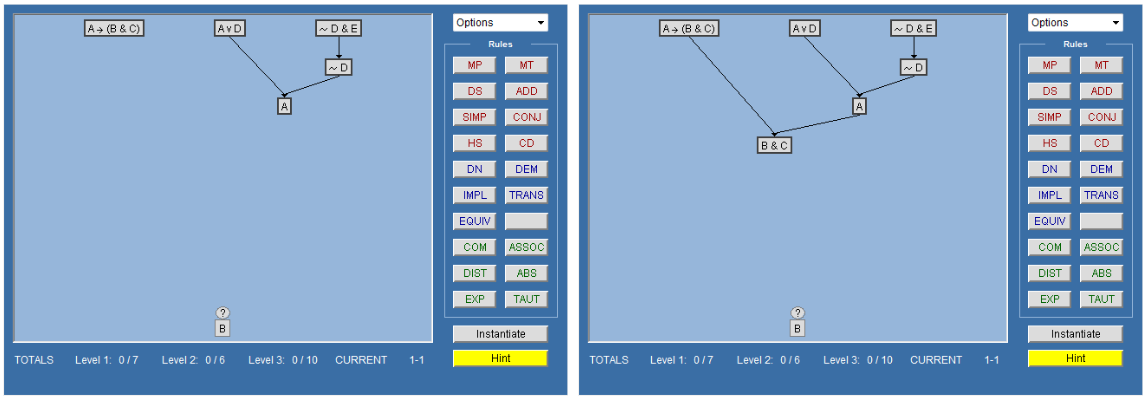


Figure 2: In the left image, the student has selected $A \vee D$ and $\neg D$ and applied the *Disjunctive Syllogism* axiom, deriving A. In the right image, the student has selected two more logic statements and applied *Modus Ponens* to produce $B \wedge C$.

ing first-order logic proofs [21]. Students are given a set of premises and a conclusion; students must use basic logic axioms to prove the conclusion. As the student works through the proof, the tutor records each interaction.

For example, a student starts at state $A \vee D, A \rightarrow (B \wedge C), \neg D \wedge E$, where each premise is separated by a comma. The student performs the interaction $SIMP(\neg D \wedge E)$, applying the simplification rule of logic to the premise $\neg D \wedge E$ and derives $\neg D$. This leads to the resulting-state of $A \vee D, A \rightarrow (B \wedge C), \neg D \wedge E, \neg D$.

Errors are actions performed by students that are illegal operations of logic and

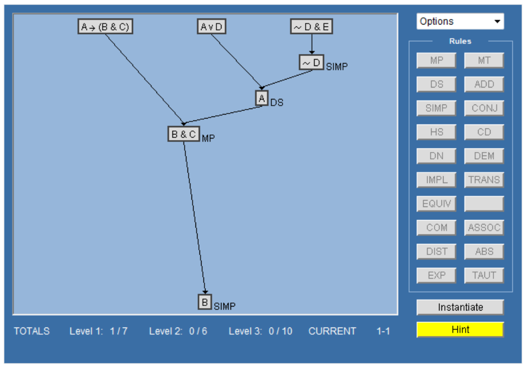


Figure 3: The student then applies the *simplification* axiom to $B \wedge C$ to derive the goal, B.

Table 1: Actions and Conditions for Deep Thought Problem shown in figures 1-3.

Action	PreCondition	PostCondition
SIMP	$\neg D \wedge E$	$\neg D$
DS	$A \vee D,\ \neg D$	A
MP	$A \rightarrow (B \wedge C),\ A$	$B \wedge C$
SIMP	$B \wedge C$	B

the tutor; when they are detected the tutor saves the error and returns DT to its prior state. For example: The student is in state $A \vee D, A \rightarrow (B \wedge C), \neg D \wedge E, \neg D$. The student performs the interaction $SIMP(A \vee D)$ in an attempt to derive A. The resulting state would remain $A \vee D, A \rightarrow (B \wedge C), \neg D \wedge E, \neg D$, and the log-file saves this action as an error.

In 2008, John Stamper augmented the Deep Thought tutor with the Hint Factory to provide automatically generated hints [7]. This addition to Deep Thought enabled the tutor to provide context-sensitive feedback to users. That is, depending on the state of the user, when a hint was requested the hint-factory process was executed and the corresponding hint provided.

Table 2: Actions Available in the BLG

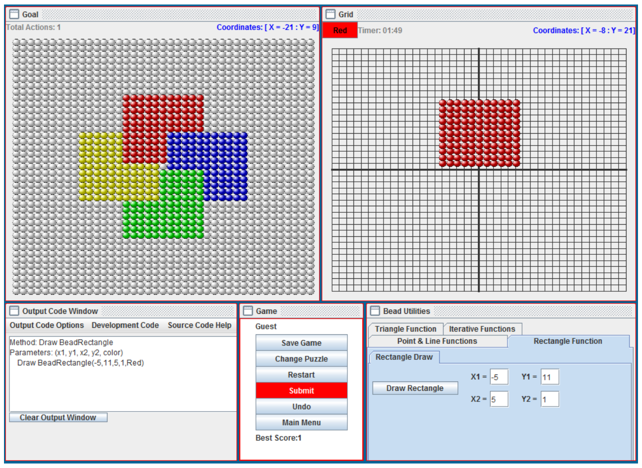
Action	Parameters
Draw Point	X, Y
Draw Line	X1, Y1, X2, Y2
Draw Rectangle	X1, Y1, X2, Y2
Draw Triangle	X1, Y1, X2, Y2, X3, Y3
Linear Iteration	X, Y, Length, +beads, rows, dir.
Triangle Iteration	X, Y, stepHeight, +beads, rows, dir.
Undo	Returns to previous step

2.3.1 Working Backwards

Deep Thought allows students to work both forward and backwards in the logic domain to solve problems [22]. Working backwards allows a student to select the goal premise and use actions to change the conclusion by adding (unjustified) propositions. When a proposition is 'unjustified', the student must derive ways to prove that statement. Students can solve a problem completely by just working forward, working backward, or a hybrid approach working in both directions, Deep Thought offers a diverse set of problem solving strategies. Deep Thought is a rich environment that allows us to explore general ways to learn from diverse problem-solving behavior.

2.4 BeadLoom Game

The BeadLoom Game is a Interactive Learning Environment (ILE) game that provides an open-environment to students to practice plotting with the Cartesian coordinate system [15, 14].The BeadLoom Game(BLG) [14] is a game-based extension of the Virtual BeadLoom (VBL), an online tool to teach math using Native American cultural beadwork practices. The BLG and the original VBL were developed for teaching middle school students.



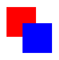
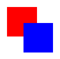
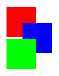
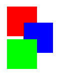
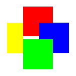
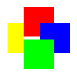
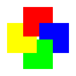
```
Method: Draw BeadRectangle
Parameters: (x1, y1, x2, y2, color)
  Draw BeadRectangle(-5,11,5,1,Red)
```

X1 = -5
Y1 = 11
X2 = 5
Y2 = 1

Figure 4: The BeadLoom Game Interface: the goal of the game is to create the pattern on the left using as few moves as possible. Here the player has started by using the rectangle tool to create the first part of the pattern. Next a player might choose the color blue, and draw a blue rectangle, by entering the coordinates in the panel in the bottom right.

The BeadLoom Game added game elements to VBL to increase motivation and learning [14]. In the BLG, players place beads in a 41x41 Cartesian grid using six different tools, shown in Table 2, and can also use an undo command. All actions take a color parameter; there are 12 different colors available. The loom starts empty and once beads are added they cannot be removed unless players use the undo action. However, beads can be overwritten by future actions. The goal of the game is to create a specific image with the given tools. Figure 4 shows an example from the BLG. In this image the player is attempting to draw the image on the left; the player has started by drawing a red rectangle using the rectangle tool. The goal of the BeadLoom Game is to solve each puzzle in the fewest moves possible. An example sequence is shown in table 3.

Table 3: Example Play Sequence

Start-State	Action	Result-State
	rectangle(-5 1 5 11)	
	rectangle(1 -5 11 5)	
	rectangle(-5 -11 5 -1)	
	rectangle(-11 -5 -6 5)	
	rectangle(-5 -5 -1 0)	

2.5 Visualization & Interactive Learning Environments

Student tutor-log data sets are large, representing hundreds of problem attempts with hundreds of problem states. Interactive Learning Environments(ILE) has the potential to allow teachers to see how students are solving problems, but educational data mining tools require “good visualization facilities to make their results meaningful to educators and e-learning designers” [53]. Instructors are not necessarily savvy with graphs, spreadsheets, or statistics, so they need support in navigating these large domain models to learn about student behavior. One approach of providing that accessibility and exploration of the data is through data visualization.

Ben-Naim and colleagues [11] have developed an authoring tool, called the Solution Trace Graph, that allows teachers to create simulators for science and explore small graphs of student actions in the simulator. However, this visualization is restricted to teacher-created states, where teachers label a step in the simulator as one of interest. This is useful to let teachers see how many students enter their pre-defined interesting states, but imposes a significant limitation by not letting students provide the states through their actions. Since their approach is not fully derived from student data, it is subject to the expert blind spot. The expert blind spot is the idea that expertise in a subject area may make someone blind to the learning processes and instructional needs of novice students, and are unaware of having such a blind spot [45]. Also by definition it does not facilitate exploration of student generated data, meaning it is less likely their users will discover surprising insights from the data, since the educator had a role in creating the data that is captured. In addition their tool is specific to their tutors built through their Adaptive eLearning Platform (AeLP), so large, diverse data sets from other tutors are not supported. However their team does state that the inclusion of this tool and the Solution Trace Graphs has "greatly enhanced and simplified...the development of eLearning content." [12].

CourseVis is another visualization that produces graphical representations of student tracking data collected by a Content Management System, and helps teachers gain an understanding of how students are behaving in their online courses. In CourseVis, the focus is on the behavior of a student over the course of an entire semester. CourseVis supports some techniques to view student performance but the focus is on visualizing knowledge components and assessment performance, not problem-solving behavior as in our work [40]. The focus of InVis is how a group of students solved a specific problem, whereas CourseVis focuses on how student knowledge evolves over many problems throughout the course of a semester.

VisMod provides students with a visualization tool for representing and interacting with their own student-model, thereby helping students to develop their meta-cognitive skills [62]. VisMod focuses on the visualization of a Bayesian Belief Network, which is related to Bayesian Knowledge Tracing. The focus of this tool is to investigate what the students think about their own behavior, but does not provide instructors a view of all the student data.

TADA-Ed, Tool for Advanced Data Analysis for Education, is a tool designed for mining educational data generated from digital tutors, the focus is on visualizing the results of several data-mining techniques, such as k-means clustering and decision trees applied to educational data [41]. TADA-Ed focuses on the relevance of student actions, the evaluation of correctness and, if applicable, the type of mistake made. This is important and very useful but does not provide users a way to interact visually, directly with student data on solving open environment problems. Their tool is focused on the visualization of scaffolded problem tutors, not open-ended problems with large solution spaces. Their visualization applies data-mining techniques to data based on correctness and incorrectness, not on paths to the solution like *InVis*.

Our work is unique among these visualization tools in its focus on visualizing student behavior while solving a single problem. The work by Ben-Naim visualizes data at the problem level of granularity, but is domain dependent. Their visualization tool is built specifically for the tutoring systems that they are also developing, while we provide a domain independent approach. A domain independent approach offers a much wider range of system coverage, but potentially at the risk of offering lower specificity. By building a visualization tool specific for certain tutors one can potentially create powerful visualization techniques specific to the input data. The problem is that the techniques and visualization itself are restricted to a specific set of tutors, thus limiting tutor coverage.

2.5.1 Visualization of Game Log Data

Intelligent Tutoring Systems and video games are similar in that both incorporate rapid feedback loops and scaffolding techniques [6]. Recent research has looked for ways to leverage the ITS and video game communities for future learning environments [51]. Typically an intelligent tutoring system is designed to provide a unique educational experience for each student. To provide this personalized experience, intelligent tutors map student behavior to a student model and from the interactions logged by the user, map a user to some student model and use the model to build intelligent support for the student. These same techniques could also be used in games to tune player experiences.

There have been a few explorations of using visualization for games, which can be extended to educational games. Noobler uses visualization to enhance the quality of the "spectator mode" for multiplayer first-person shooters, incorporating many different visualization techniques to show a more complete overview of the action taking place in the game [32]. Dixit and Youngblood created a tool to visualize player logs to understand how a player navigates a 3D environment [24]. In these examples and others the challenge is providing an overview of the action that takes place in 3D space, and so the visualization tools themselves are much more focused on spatial and temporal visualizations, which is different from our focus.

In games research, Andersen analyzed game data to create game “states” - snapshots of what’s been done in a game, and applied different metrics to game data to assign a value to each of these states [3]. Using these values, Andersen created a dissimilarity matrix, which stores the difference, or distance, between each state-pair, and then used classical multidimensional scaling to project those relative distances to 2D space. This approach works well for visualizing state similarity, but does little to show the overall behavior of the players. It also does not present the information in a way that visually preserves sequence information.

2.6 Interaction Network

The Interaction Network is a data structure for combining a large set of transactional data into a single graph that represents all interactions in an interface. This representation allows us to leverage network analysis techniques to better understand software-user log-data. The Interaction Network is comprised of three main components: states, actions, and cases. We attach additional information, such as student, aggregate, and derived data to nodes and edges.

We model a solution attempt as a sequence of states (vertices) and actions (edges). We use *case* to refer to a single student’s data for a problem. (Note that students may attempt problems more than once, resulting in more than one solution attempt for a single case). We create the Interaction Network for a problem by conjoining the set of all of its solution attempts (cases). We use *state* to describe the state of the software environment, representing enough information so the program’s state could be regenerated in the interface. Of course, the state is specific to the domain being

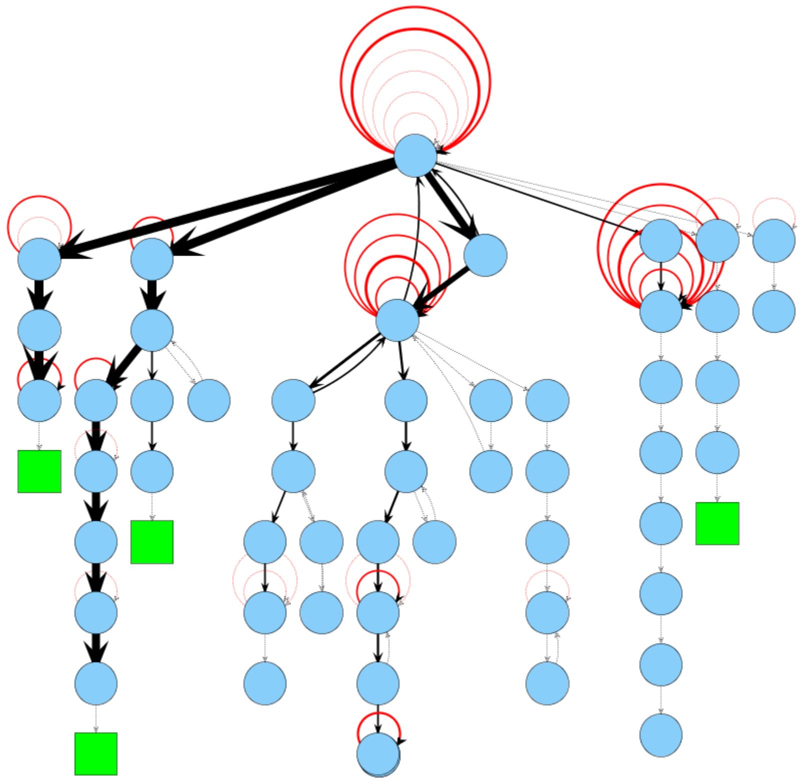


Figure 5: An example of a small Interaction Network.

visualized, more on this below. We use *actions* to describe user interactions and their relevant parameters. We also store the set of all cases that visited any particular state-vertex or action-edge, allowing us to count frequencies and connect case-specific information to the Interaction Network. This representation results in a connected, directed, labeled multigraph with states as vertices, directed action edges to connect the states, and cases that provide additional information about states and edges. (See figure 5.)

This representation allows us to build an Interaction Network model from any system that logs interactions in state, action, resulting-state tuples. The **state-action-state** tuple is the most important part of the system's logging. However, we recommend logging other features including the time spent on each step, any

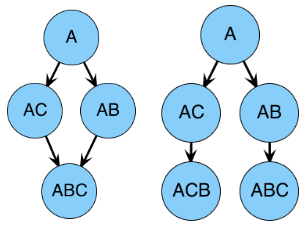


Figure 6: The same Interaction Network, presented in two different ways, un-ordered (left) and ordered (right). By preserving the order, the right side representation, treats states ACB and ABC as two distinct states, as opposed to the ordered states, where those two states are presented as equivalent.

feedback messages given to the user, and information about the problem (such as a problem ID). Frequency information, as well as information about related cases, is embedded into the edges and vertices. This results in a network graph which shows the interactions of a large number of students in a relatively small space.

To build the Interaction Network for a particular problem we combine the interaction sequences, or solution attempts, from each case into a single aggregate graph. States are combined when they are considered equal. Aggregating the states and actions adds frequency information based on which students visit which states, and this frequency allows us to provide context to the solution space. Mainstream solutions and strategies will have higher frequencies, while uncommon or outlier behaviors should be identifiable since their nodes and edges will have very low frequencies.

In different tutors and interfaces, two states could be considered equal as long as the screen looks the same, or all the same actions have been performed, regardless of order, but in other cases, states arrived at by taking the same actions in a different order could be considered distinct. *InVis* will handle either case, as shown in Figure 6.

Before visualizing a data set, we can ask whether order matters for a particular problem-solving domain. For example, in the Deep Thought data should the state A,B be distinct from the state B,A? An order-matters approach results in more distinct program states, and thus more vertices. However, preserving the order of statements derived increases the information of the visualization by making the order of steps a student takes more obvious. By contrast, an un-ordered approach reduces the number of states and reduces the size of the overall network. While the un-ordered graph provides a more generalized view, it can be visually more difficult to follow the precise steps of an individual case, due to the graph-layout applied to the network. For our work with BeadLoom Game data and Deep Thought data, we use an un-ordered state-description, with the goal of reducing the number of states in the network in mind.

For the intelligent tutoring system community, we encourage the use of logging standards such as the PSLC's Datashop[35]. We recommend preservation of the action's parameters and results, since they are useful for labeling the visualization.

One final characteristic of the Interaction Network relates to the method for handling tutor-based errors. These types of errors are actions performed within the targeted tutor but are invalid actions within the tutor. A simple example of such an error would be like making an invalid move with a Chess piece in the game of chess, like moving a pawn three spaces. There are different display options which can be used in the interaction network and those are shown in figures 7-9. The choice of display method is left to the user, when defining their states and preparing the data to be read into InVis.

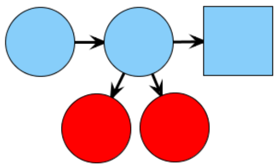


Figure 7: Errors shown as hanging vertices.

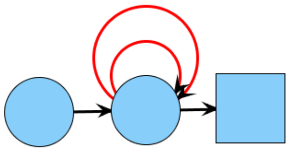


Figure 8: Error is shown as a self-loop, showing that the action did not result in a change in state.

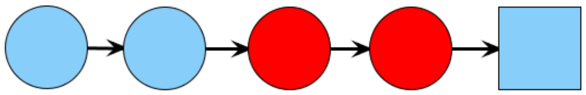


Figure 9: Error is shown as a distinct error state.

2.6.1 Interaction Networks: A Formal Description

In sequential problem solving environments, a solution path describes a sequence of state changes from a starting position towards a desired end position. For this work we will only consider discrete time environments with deterministic state transitions. The reason for this is due to the inherent difficulty with defining a state for a task like writing a novel. To best leverage the Interaction Network it is useful for student solutions, when the same to be considered so, such that the solutions will contain appropriate overlap. This overlap in states among student solutions gives states their frequency. An Interaction Network is a data structure designed to concisely describe the information contained in a large number of such sequences. Interaction Networks provide a structure on which to perform data mining, and are also useful for visually displaying information.

An interaction network is based on individual student-tutor interactions, as recorded in the log file of the tutoring environment. We define an interaction, I, as a 5-tuple $I = (S_t, A_., S_{t+1}, U, I)$, where

- S_t is the state at step t
- A_s is the action performed on S_t
- S_{t+1} is the resulting state after A has been performed
- U is the unique case ID responsible for this interaction
- I is a set of additional information about the interaction. For example, I_{time} would return a value for how long this interaction took. Included here are I_{error}, which stores the error value, and I_{goal}, which is true if this action resulted in a goal state.

A case represents all the information about an individual user in a particular tutoring system. All interactions are associated with some case. A case is represented by the ordered pair $c = (U, I)$, where

- U is a unique identifier
- I is a set of additional information about the individual. For example, $I_{pretest}$ would return a value for this case's pretest score.

The interaction network for a problem P is:

$IN_P = (C, S, A, t, s, S_0, G, I_A, M)$, where

- C is a set of *cases.*

- S is the set of observed tutor program states
- A is the set of observed actions, which connect two states
- $s : A \to S$ and $t : A \to S$ are two maps indicating the *source* and *target* states of an action
- S_0 is the starting state of the problem
- G is the set of goal states
- $I_A : A \to I$ is a map to the source set of Interactions
- M is the set of maps, which allow the lookup of relevant state, action, and case information. For example: $M_{Freq} : S \to Frequency$ will map from the state x to the frequency value for that state.

A single solution attempt is modeled as an ordered sequence of interactions. A *case* refers to an individual student's data, and could be combined with any additional student-specific information, such as test scores. An interaction network for a problem is created by conjoining the set of all the solution attempt graphs into a single graph. *State* describes the state of the software environment, with enough information so the program's state could be regenerated in the interface. *Actions* refer to user interactions and their relevant parameters. States and actions also make reference to all cases associated with them, allowing frequency counts and viewing case-specific information in the interaction network visualization. Pre and post conditions, an idea borrowed from the Strips problem solver[27], store which element of the problem was

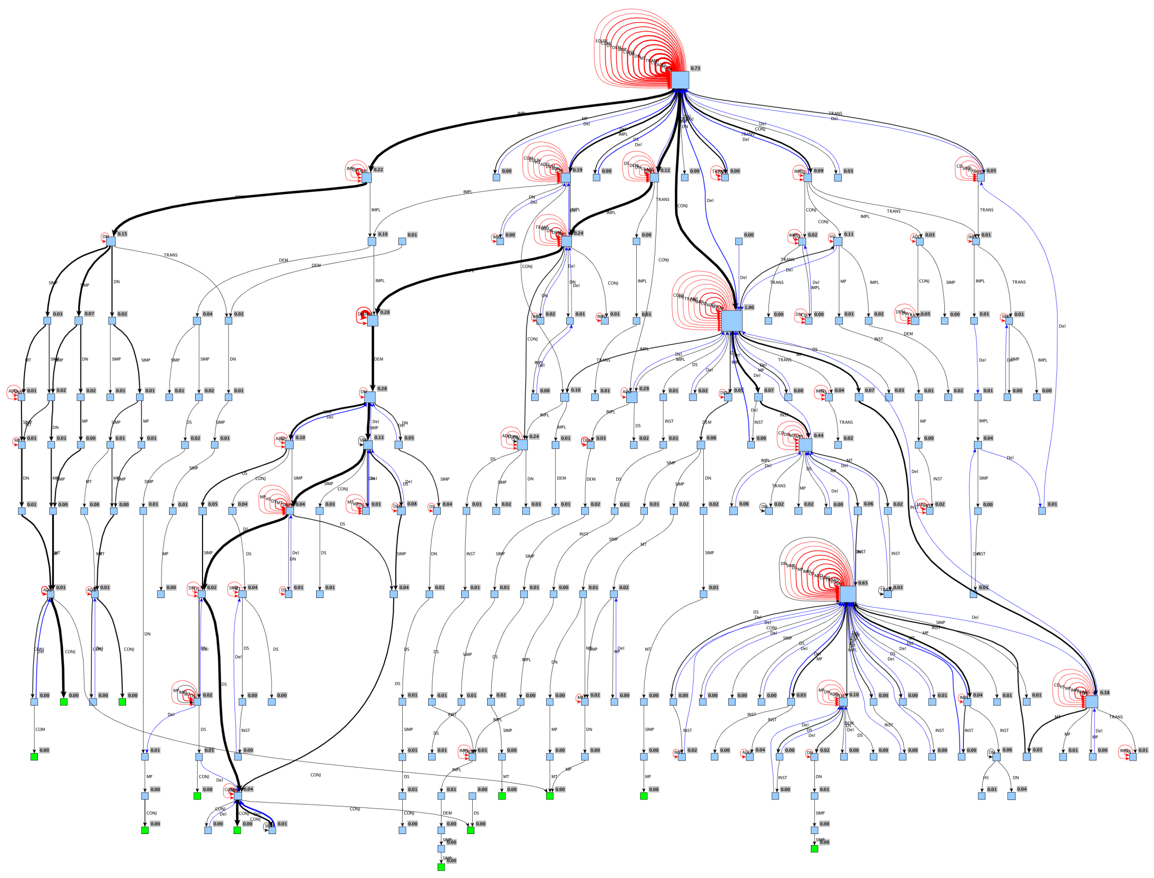


Figure 10: A visualization of an Interaction Network from the Deep Thought data set. Red edges are errors (incorrect actions), edge width depicts frequency, green squares are goal states.

used to facilitate the current action, and what has changed, respectively. In the Deep Thought tutor a simple example is, the premise A∧B simplify B, where A∧B is the pre-condition, simplify is the action, and B is the post-condition of this interaction.

The Interaction Network is a connected, directed, labeled multi-graph with states as vertices, actions as directed-edges to connect the states, student details provide additional information about states and edges. The Interaction Network stores the set of all students who visited any particular state-vertex or action-edge, allowing us to count frequencies and connect other information, like test scores or hint usage values, to the Interaction Network representation. A detailed description of the Interaction

Network is provided in previous work [25]. An example Interaction Network for the problem 1.3 data set is provided in figure 10, note the size of the full network and the likely inherent difficulty involved with following a student or multiple students' paths from start to finish. In figure 11 I have plotted the growth of an Interaction Network for problem 3-5.

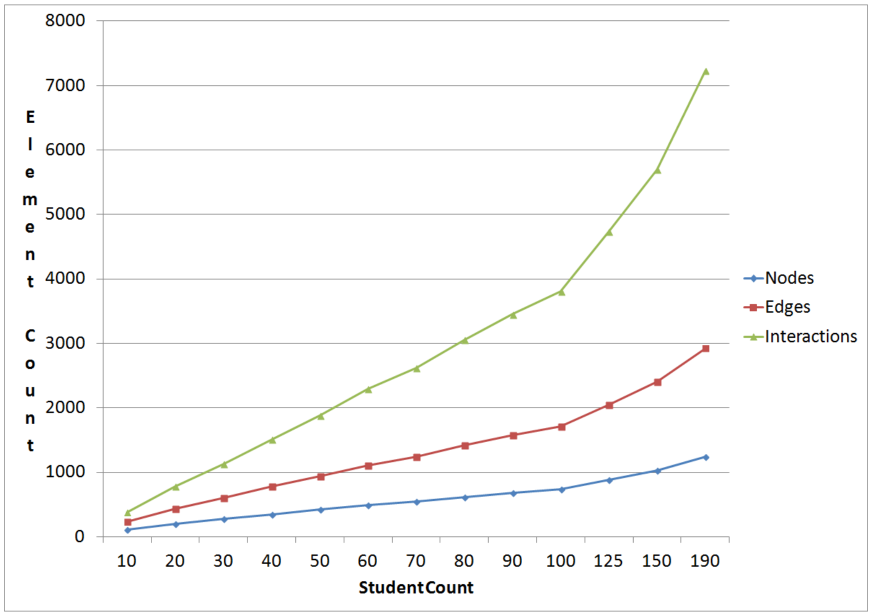


Figure 11: Along the X-axis there are he number of students which were randomly sampled. Along the Y-axis are the number of each respective element of the Interaction Network. Interactions refers to number of transactions from the data-log. The sample for each student count was taken 500 times and these are the average values. As we would expect, the transactions continue to grow, while the nodes and edges grow at a slower rate, because multiple transactions can represent the same node or edge.

The Interaction Network has proven to be an effective data structure for storing transactional data from tutors and lends itself to a variety of data mining techniques. We have applied network community detection [31] to cluster the network and provide some meaningful hierarchical structure to the otherwise overly verbose and cluttered network [42], but work is needed to present useful summarizations of the Interaction

Networks to users. For the first step we will take a look at methods of displaying and interacting with the Interaction Network, in the next chapter.

2.6.2 State Spaces for Various Tutors

The InVis team attempted to visualize tutoring data for, in total, five different tutoring systems. From the information we were provided for each of these tutors we had a variety of different results, based on a number of factors, so I will review those results here. The five systems we worked with, in order to load their data into an Interaction Network include: The Deep Thought logic tutor, the BeadLoom Game, Stoichiometry tutor [1], Algebra [36], and Novice Programming tutor [33].

The greatest success was achieved with data from the Deep Thought tutor and the BeadLoom Game, throughout this document majority of the examples are provided from these tutors and as a result we will not focus on those systems here. However, the relative success with those systems was is likely due to the following factors: first we had access to the tutor and respective developers of both systems which made understanding the state space significantly easier and to greater detail. Next, we were provided with a significant amount of data, in the thousands and tens-of-thousands of transactions, covering hundreds of students. Lastly, generating the respective states for the space was relatively simple. In the case of the BeadLoom Game, the state was defined as the configuration and color of the beads in the loom. Simply a 41x41 two dimensional array with one of twelve colors for each value. The construction of the state for Deep Thought was slightly more complicated but still relatively simple and is described below.

In Deep Thought the state was defined in two parts. First we collected each derived element from the tutor, which was a set of roman characters. The root state, was the set of premises which defines the problem and was provided to the student. For example: K→M, Z→R, ¬(K→R). The conclusion students are tasked with deriving is $M \wedge \neg$Z. The resulting state description for this state, which is also the problem root, is: K→M,Z→R,¬(K→R)/M∧¬Z. The slash '/' separates elements of the solution derived working forwards and working backwards. Within each respective portion, i.e. before the slash and after the slash, elements are ordered lexicographically. This lexicographical ordering is what causes the generation of bubbles described in section 4.5. The effects of this ordering are also depicted in a simple example in figure 6. Due to the innate structure of the BeadLoom Game state space, this type of ordering was not necessary.

2.7 Chapter Summary

The Hint Factory [7, 9, 56] has provided us with the inspiration and starting point for investigating the visualization of solution spaces for Intelligent Tutoring Systems (ITSs) and Computer Based Education (CBE) tools. The Deep Thought logic tutor [21, 22] and the BeadLoom Game [15, 14] can provide us with input data from drastically different domain areas. Although there are some tools for visualizing student-related data, visualizing the Interaction Network is novel and unique. A limitation of the Interaction Network is ones ability to define an appropriate state for which the network will depict along with transitions, we reviewed five different attempts and discussed some of the reasons why we felt we had success or failure with those systems.

CHAPTER 3: INVIS USABILITY STUDY

In the Related Work chapter we discussed the difficulty of understanding how a student worked a problem in a tutor based on transaction logs alone. I argue for the need for visualization tools of computer based tutor logs, to make this information more easily understood and accessible to educators and researchers. These visualization tools will enable educators, researchers and developers to better leverage an important advantage Computer Based Education has over traditional methods, which is the record of *how* students solved problems in a tutor. To this end I led a team to develop *InVis*, a visualization tool for providing insights about student behavior in tutors. As with many visualization tools, our goal is to provide new discoveries and insights into our domain, which is the behavior of students using an Intelligent Tutoring System.

InVis was developed to be a domain independent approach to providing an information visualization solution for Intelligent Tutoring system logs, to better understand student behavior. Once our first version of the tool was completed, we conducted a three pronged visualization study to evaluate the effectiveness and usefulness of InVis. Our evaluation addresses the first of my research questions.

Research Question: Can we use visualization techniques to allow people to understand, and gain insights from, the interactions that are being performed by students, from their transactional log-data, using the Interaction-Network?

I introduce *InVis*, a novel visualization technique and tool for exploring, navigating, and understanding user interaction data. *InVis* creates an interaction network from student-interaction data extracted from large numbers of students using educational systems, and enables instructors to inspect the steps students take during problem-solving. Here I present the *InVis* tool and demonstrate that it is an effective tool for providing instructors with useful and meaningful insights to how students solve problems.

InVis allows educators to explore the novel Interaction Network which represents the interactions students perform in problem-solving environments. InVis displays student behavior across an entire class, enabling educators to develop insights from common strategies and mistakes that groups of students apply in a software tutoring environment. While this work will concentrate on data from computer-aided instructional environments, I have also used *InVis* for exploring user interactions in games as discussed in section 3.1.7.4. If done well, visualizing problem-solving interaction logs can provide insight into how users solve problems, as well as what errors they encounter; and provide more information than a purely summative approach.

3.1 InVis: Interaction Visualization Usability Study

Our "triangulated evaluation" is inspired by Plaisant's aptly named paper, the "Challenges of Visualization Evaluation", where she described the difficulties of performing evaluations with tools that can "answer questions you didn't know you had." [47]. Since there is no single proven technique for visualization evaluation, Plaisant recommends using several complementary methods that can mitigate the weaknesses of single techniques used alone. Our guidelines review shows how *InVis* adheres to commonly accepted visualization guidelines. In the case-studies, educators were able to generate and confirm hypotheses, and discover insights into their data. As shown below, the results of the usability study showed that educators were able to complete tasks at 85% accuracy with minimal training time.

We used three methods of evaluation: 1) a guidelines review, where we compare *InVis* to the commonly accepted visualization guidelines; 2) a set of case-study like success stories with expert users, and 3) a summative usability study, where educators explored data from a university-level logic tutor using guided tasks and completed a validated usability scale, and reflected on their experience through a qualitative survey.

3.1.1 Visual Representation & Graph Layout

A graph is a natural representation for our Interaction Network model. However, there are still a wide variety of graph layouts, and the primary visualization view should be one that is easiest for the novice user to understand. InVis uses a tree-like graph layout to present Interaction Networks. The root node is the starting state, the problem "givens", and is placed at the top of the view, with student interactions branching downward. This layout makes it easy to follow the individual solution-paths of a student, as the vertex depth effectively preserves the number of steps in the solution-attempt.

State vertices can be labeled with the entire state description, or just the result of the latest interaction, since the path of sequential states should reveal the entire state description. If they exist in the data, goal states are represented with a different color and shape, a green rectangle shown in figure. Each edge is labeled by its action, but not its parameters, to minimize textual clutter for readability. Edge thickness is determined by the frequency of observed interactions, with most-frequent edges being thickest. *InVis* uses JUNG [61] to efficiently place nodes in a graph layout.

3.1.2 Modeling Program States

We have successfully built interaction networks from a variety of sequence-oriented interaction data. Here we describe two systems and concentrate on how we modeled the state description.

3.1.2.1 BeadLoom Game States

Previously in section 2.4 I described the the BeadLoom Game (BLG) and its features that increase motivation and learning [14]. In the BeadLoom Game, players place beads in a 41x41 Cartesian grid using six different tools, as well as an undo command. All actions take a color, one of 12 different options as a parameter. The loom starts empty and once beads are added they cannot be removed, aside from the undo action. However, beads can be overwritten by future actions. The goal of the game is to create a target image with the tools available.

In order to gain a better understanding of the BLG data, we used InVis to explore player solutions. The state representation is a 41x41 array containing the 12 color values. Actions are represented by the six bead-placement tools and their parameters. We also store the set of all cases that visited any particular state-vertex or action-edge.

We use an image depicting the player's state as the state label. The BLG does not have error data, meaning users cannot submit invalid parameters as actions. However, users are able to undo previous actions, which we can interpret as an unintended action.

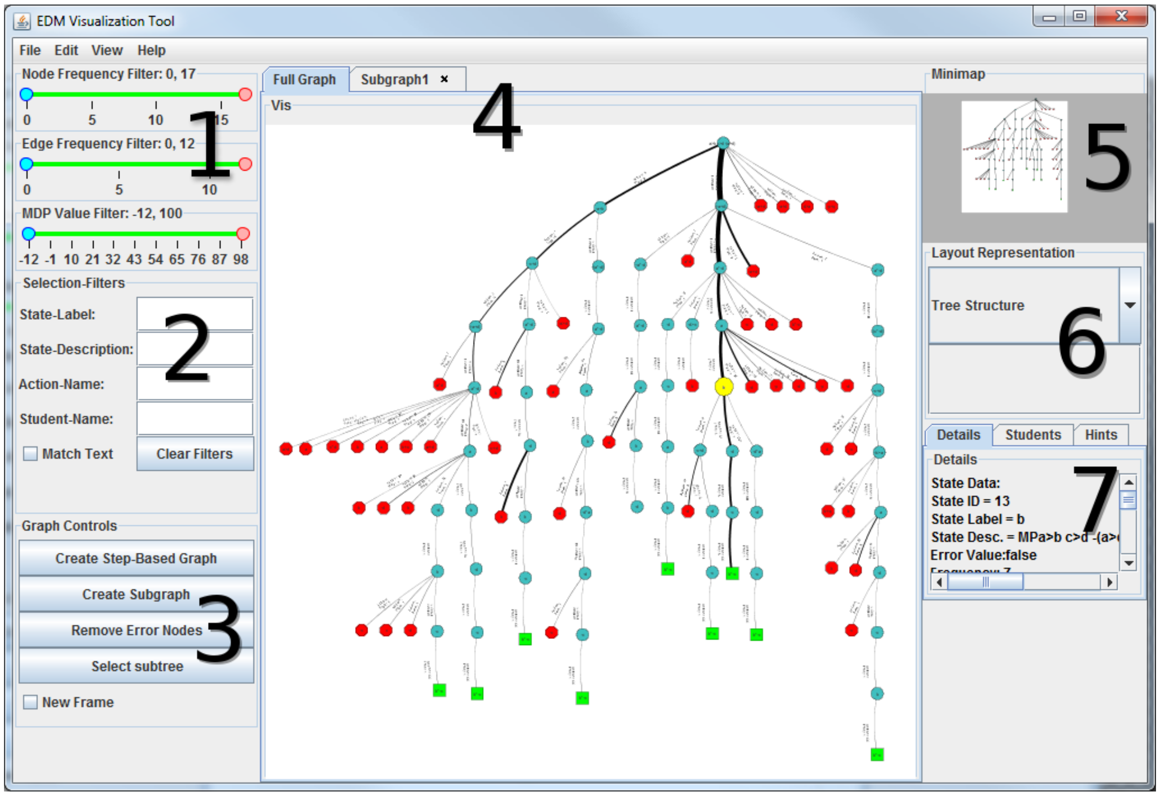
3.1.2.2 Deep Thought Logic Tutor States

For a complete description of how states are generated in The Deep Thought logic tutor, refer to section 2.3, here I will briefly summarize our approach. The tutor provides a set of premises and a conclusion. We model the application of rules as the actions, and the state is the conjoined set of each premise and derived proposition. Here is a simple example:

A student starts at state $A \vee D, A \rightarrow (B \wedge C), \neg D \wedge E$, where each premise is separated by a comma. The student performs the interaction $SIMP(\neg D \wedge E)$, applying the simplification rule of logic to the premise $\neg D \wedge E$ and derives $\neg D$. This leads to the resulting-state of $A \vee D, A \rightarrow (B \wedge C), \neg D \wedge E, \neg D$. This example is described in section 2.3 with related images in figures 1 - 3.

3.1.3 Design of the Visualization

Here we address the design of the interactive visualization, implemented in the software tool we call *InVis.* The tool was designed with the visual information-seeking mantra [54] as our guide. Shneiderman states from his own extensive experience that an effective visualization contains those elements, and Cairns has shown the large number of applications where it has been used to guide visualization design [19]. Thus we feel this to be a reasonable approach to developing such an interactive visualization.



Node Frequency Filter: 0, 17
Edge Frequency Filter: 0, 12
MDP Value Filter: -12, 100
State-Label:
State-Description:
Action-Name:
Student-Name:

Details
State Data:
State ID = 13
State Label = b
State Desc. = MPa>b c>d -(a>
Error Value:false

Figure 12: An example screen shot of the default view in InVis.

Note our visualization is just one method for designing an interactive tool for exploring Interaction Networks, and later we provide evidence that it is effective at its goal, but many different types of implementations for exploring Interaction Networks are likely to exist.

Referring to figure 12 we will explain the major features of InVis.

1. Frequency filters allow the user to filter edges and nodes based on the frequency. The filter removes nodes or edges based on the range. Frequency is a useful metric for investigating the behavior of a large number of users. Nodes and edges with high frequency identify common behaviors, while low frequency nodes represent less common behaviors. Because frequency is an intuitive domain-independent metric we choose to add this filter to the default

view.

2. The selection filters allow users to select states, actions, or cases by entering a search string; the user can select to match with *contains* or *direct match*. This provides an easy measure to select large numbers of specific states, and is especially powerful when combined with the subgraph extraction feature.

3. Graph controls allow users to create subgraphs, which are then loaded into a separate tab. The user can also remove the hanging error nodes from the current graph, if they are interested in correct behaviors more than errors. Creating a subgraph, copies all of the currently selected nodes to a new tab (in the interface) and re-applies the graph-layout to the subgraph. This can be used to clear up clutter. By first selecting desired states, edges, or cases with the selection filters, and then generating a subgraph, a wide range of options for exploring the data is possible. An example of the subgraph generation is shown in figure 13.

4. The Interaction Network is displayed here. This panel has mouse controls for panning, zooming, and selecting, controlled by right-click, mouse-wheel, and left-click respectively. Multiple graphs or subgraphs can be loaded at once, each placed in a separate tab.

5. A mini-map helps users stay oriented even in large data sets and provides context for the user. The mini-map provides a white box which represents the current view of the main visualization panel described above, and updates as

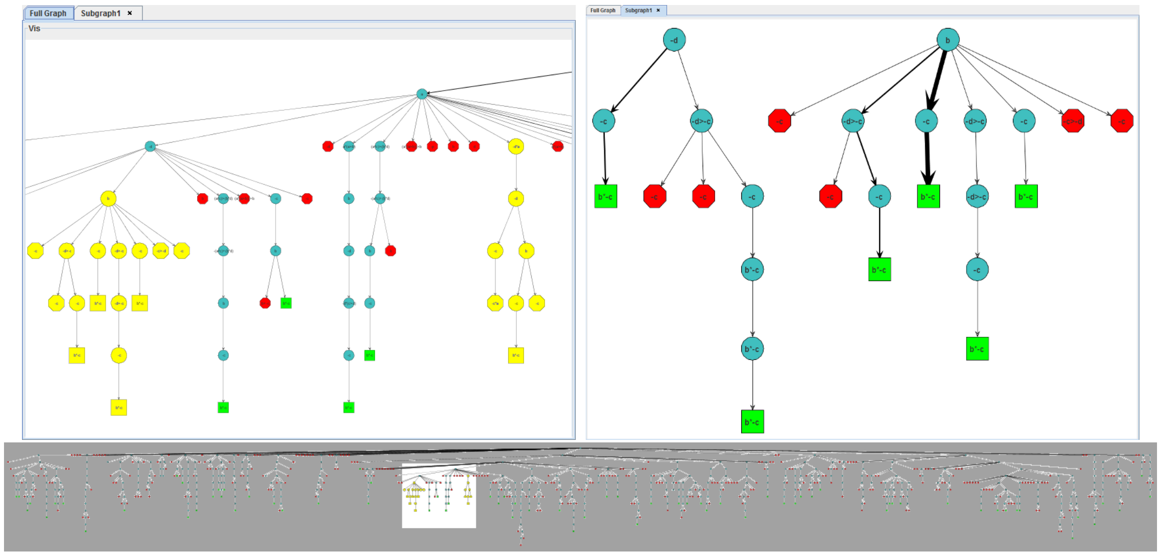


Figure 13: Left) The user has selected several nodes. By constructing a sub-graph, *InVis* presents the selected nodes in a new tab, shown on the Right. Below is an example of the mini-map, with the white box representing the view frame of the left hand image. Note the size of the graph when visualizing 170 students.

the user pans and zooms.

6. In figure 12 the interaction network is shown using the default graph layout; however, there are several layouts available via this drop down menu.

7. Additional information about the states, actions, and cases are available here. Users are able to view the complete state of the node, as well as a list of all case IDs that visited the state or edge. Other information, such as tutor hint messages or test scores can also be displayed here.

3.1.3.1 Guidelines Review

Our visualization was designed with the visual information-seeking mantra in mind. *InVis* supports functionality for overview, zooming and filtering, details on demand, viewing relationships, and extraction. In this section, we describe why each element is important to viewing interaction networks, how each element was included and supported, and improvements that can be made. Craft and Cairns state that many other developers cite the mantra as their guiding source for the development of their visualization but often forgo explaining how and where they used it [19]. We use this evaluation to find strengths and weaknesses in our approach, and confirm we have an implementation based on these guiding principles.

Overview First The hierarchical graph offers an overview representation of the students' behavior as they work through a single problem. Combined with the edge width representing frequency, we provide a quick understanding of student behavior trends. In addition the mini-map consistently orients the user within the context they are working, always providing an overview for the user to reference as they navigate the Interaction Network graph.

Zoom and Filter Zooming and filtering on the interaction network means users can focus on specific behaviors they are interested in. Zooming is not only physically zooming in on the graph-layout, but also zooming in on students who performed a specific action, or visited a particular state. Zoom and filter are supported through a variety of filter and selection tools which allow manipulation based on states, student-IDs and actions. Additionally selection combined with creating subgraphs allows for alternative approaches for filtering and zooming. While the mouse wheel allows the user to zoom on the graph layout, frequency-based filters allow users to focus on either common trends or atypical behaviors exhibited by the students. Identifying common sequences in the interaction network could help identify subgoals to problems and is one main improvement that we explore in chapter 4.

Detail on Demand The details tab is where the user can find specific information regarding a state, action or student. Details are available in a set of tabs, displaying text information about the selected node. These details include cases that visited the node, the frequency of the state, actions leading to or from the node, whether the state is a goal or error state, and the description of the state. These details are the finest granularity we can provide from the log data. One improvement for details is to provide aggregate user statistics that describe the current graph or selected subgraph, for example number of error states, total number of states, number of actions, or average number of actions per student.

View Relationships To "relate" in our *InVis* tool is to compare behaviors between students and to compare sub-graphs of the Interaction Network. Students can be compared to other students by selecting their nodes via the student selection tool and generating sub-graphs. Sub-graphs allow users to compare, at once, all the times a specific action was used by all the students. Viewing the sub-graphs resulting from selecting two frequent paths let a user view how two different strategies were applied to solving the problem. Viewing relationships is supported through the selection and filtering tools combined with creating new sub-graphs in separate tabs. An example of this type of comparison is shown in figure 13. Multiple problems can be loaded into separate tabs and problems can be compared, as well as Interaction Networks for groups of students. Much of the Interaction Network maintains its sequential structure, as students performed actions in order, so it is possible to do slight comparison between similar approaches, but a desirable improvement for comparing could be to allow users to more easily compare strategies between students. In chapter 4 we apply different mining techniques to help users find these strategies automatically.

History Shneiderman notes that history of *InVis* user actions is the most often ignored element due to its high cost of development and is rare in prototypes [54]. In our prototype interface this feature is weakly supported. In InVis, when users edit the main graph in significant ways, such as filtering vertices, the new graph is placed in a new tab. Users are able to keep track of each in the tabbed interface. This allows some measure of preserving the history of the user actions. However, this could be improved further by allowing users to undo and redo actions such as selection even when they do not generate new subgraphs.

Extract Sharing one's findings about student behavior is important, particularly for teachers, so challenging issues for students can be addressed. In *InVis* users can save the image of the visualization panel so that it can be shared. Teachers can show the sequence of steps to their students and highlight situations where errors were common. Though not currently supported, an improved sharing between colleagues could be supported by saving the current layouts, subgraphs, and graph annotations, so they can be stored and shared with others easily and they too can interact with the data via *InVis*. By extension this would also facilitate the convenience of being able to save one's work and making it available at a later time.

3.1.4 InVis Usability Study

One goal of InVis is to make complex interaction data accessible to non-expert users. To test the usability of InVis we created a quantitative task-based test as a measure of summative usability testing. We developed 15 tasks based on the propositional logic tutor interaction data. These questions were designed based on common use-cases, and cover the range of features in the tool. A list of these types of questions is provided in table 4, and were created to address activities that are in line with the focus of the tool, namely general behaviors, and outlier behavior. We ran a user study consisting of three sections, a quantitative portion, usability portion and qualitative portion, each supporting different types of evidence that the tool is effective at allowing users to make discoveries about their data.

3.1.4.1 InVis quantitative task-based survey

The quantitative survey presents 15 questions that represent tasks of finding or summarizing data from the tutor logs, and the questions were generated to have just one answer each, with no more than 5 minutes spent per task. We evaluated the usefulness of InVis in performing the required tasks by measuring how well users were able to answer these quantitative questions. Table 4 highlights the 15 questions as a representative sample of the nature of the tasks.

To evaluate the usability of InVis, we used the validated CSUQ survey written by James Lewis at IBM [39]. CSUQ stands for The Computer System Usability Questionnaire, and is divided into four scores, an Overall CSUQ score, a SYSUSE, INFOQUAL, and INTERQUAL scores. SYSUSE stands for system use, INFOQUAL is information quality and INTERQUAL is interface quality. Questions 9, 10 and 11 from his survey CSUQ were removed because they were deemed irrelevant to *InVis* because it does not contain any error messages. One difference between Lewis' survey and ours was in our CSUQ, the score has high scores being preferable rather than low scores; Strongly Agree is equal to seven instead of one. Our overall CSUQ score was 4.65.

To determine how well InVis met user expectations for data exploration, users were asked to complete a qualitative survey where they could provide open ended responses directed towards functionality that they would like to see available in future versions of the tool.

We ran a user study with seven participants to determine the level of usability of *InVis* for understanding interaction networks representing data from the Deep Thought logic tutor described above. All teaching material was conducted in the classroom, and the Deep Thought logic tutor is strictly used for conducting homework. The students were given a set of premises, $[(A \rightarrow B), (C \rightarrow D), \neg(A \rightarrow D)]$, and were tasked with generating a first-order logic proof for the conclusion of $B \wedge \neg C$. To prepare the data for the study, we duplicated or removed students to ensure each task-question had a single correct answer, necessary since some of the questions were based on state or edge frequency. In total there were 17 students and 186 interactions were selected for this study. The original data-set was from the Proof Solver tutor, problem 1-3 with 73 students, and 1866 interactions.

Of the seven participants, we met with four individually and they used our computer with *InVis* with the target dataset loaded. The participants were all university educators for logic courses, who have used the Deep Thought logic tutor as part of their course. The data set was from the logic tutor, but not from the professor's courses specifically. We gave them a brief overview of how the tool worked, how to zoom, pan and select. We also demonstrated how to generate a subgraph (using a

GUI button), as well as how to use the selection tools (where selection criteria are entered in text boxes). The demonstration lasted 5–10 minutes. The other three participants were emailed a 2.5 page description of how the interactive elements of the tool works. This document served as the resource for the 5–10 minute demonstration. These three participants conducted the study by themselves, as I sent the InVis software and appropriate dataset to the participants via email. They were instructed to contact us if any issues arose that they felt were unintended or prevented them from conducting the study. If any task took more than five minutes, they were asked to stop and move to the next one. Participants were instructed not to ask how to complete any of the tasks. After the quantitative section was complete they were asked to do the usability survey then the qualitative survey.

3.1.5 InVis User Study Results

In this section we report on the results of our 7 participants using InVis to perform 15 tasks on the Deep Thought logic tutor dataset. We report the results in three sections: task performance, usability survey results, and open-ended qualitative feedback.

3.1.5.1 Task Performance Results

The participants completed 85% of the tasks successfully ($M = 12.71, SD = 2.66$). From the questions in table 4 the most commonly missed questions were Q4 with 64% success and Q6 with 71% success. Participants spent an average of about 43 minutes on the quantitative survey, with a standard deviation of 19.95 minutes, based on survey start and end times. However, the participants reported they felt that tasks

Table 4: Task Performance Instrument

Q1	Find the shortest correct solution-path to this problem.
Q2	How many students found the shortest solution-path(s)?
Q3	Find the most frequent solution-path to this problem.
Q4	Find the error(s) with the highest frequency and write the State ID(s).
Q5	Write the state ID to a state, after which no students reach the goal. (Non-error state).
Q6	Write the state ID of the state that leads to the most unique errors?
Q7	Write the action-label and the corresponding final state ID for the last action of student X?
Q8	How many unique solutions lead to the goal?
Q9	After filtering nodes and edges to frequency 5 and greater, how many complete solution paths exist?
Q10	What is the state-ID(s) of the state(s) with MDP Value 14?
Q11	What is the student ID of the student(s) with longest solution to the goal?
Q12	Who are the students on the node with the following node label: –a*-d (note the label is: minus minus a * minus d)
Q13	From the starting state, the root of the tree, how many unique actions did students perform?
Q14	Highlight the node with node-label: $\neg\neg A \wedge \neg D$ and select the sub-tree, then build a sub-graph. How many error nodes exist in this sub graph?
Q15	Answer yes or no, did student 81 find a goal solution?

took an average of 23 minutes, with a standard deviation of 13 minutes, meaning participants felt it took less time to complete the tasks than the time spent with the survey open. This could potentially be explained if the internet participants were not solely focused on the study.

3.1.5.2 Usability Survey Results

To evaluate the relationship between the task performance and usability we submitted the results to a bivariate correlation analysis. Task performance strongly correlated$(M = 12.71, SD = 2.70), r = 0.87, n = 7, p = 0.01$, with the overall usability survey $(M = 4.65, SD = 1.89)$. The *r-squared* value was 0.76, which means that 76% of the variance in the quantitative skills test-scores is accounted for by variance in usability scores. We examined the three subsections of the usability survey. Task performance strongly correlated with the Sysuse score $(M = 4.84, SD = 2.04, r = 0.78, n = 7, p = 0.04)$, the Infoqual score $(M = 4.54, SD = 1.98, r = 0.90, n = 7, p < 0.01)$; and the Interqual score $(M = 4.39, SD = 1.70, r = 0.96, n = 7, p < 0.01)$.

These results provide evidence for the validity of using the task performance test as a measure of the visualization quality. This provides insight into what functionalities the developers should focus on in future development of *InVis*. We cannot make strong causal inferences between the quantitative test and the usability scale; were participants able to complete the skills test because InVis was usable, or did they report that it was usable because they were able to complete the tasks? The fact that 85% of the tasks were correctly completed, with little time spent on training, provides evidence that our technique is usable by our target population. The questions that

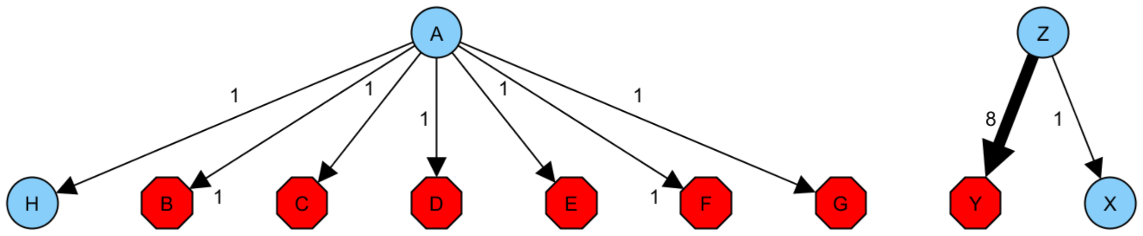


Figure 14: This is a general example, without details, of how the two examples regarding errors would be represented. On the left we have six errors, each with one frequency. On the right we have a single error with eight frequency. The error example on the left has more unique errors, while the right example has more errors. This ambiguity in the questions could be causing some of the difficulties.

were most commonly missed, were tasks related to understanding the most frequent error, and the state from which the most *unique* errors were made. This highlights a potential problem with the current error-state representation; which seems to make it difficult to distinguish unique errors from frequent errors, a simple example is shown in figure 14.

3.1.5.3 Qualitative Results

In the open-ended survey questions we asked participants about specific ways to improve *InVis*. Here we will mention some of the most important suggestions from participant utterances during InVis usage and from the survey.

Two issues regarding the graph layout were raised. Participants felt the layout would be more intuitive and easy to navigate if it were possible to order the layout along the breadth (x-axis). By applying a more informed layout to the graph, we could order the states in some manner along the X-axis, for example by making the most frequent path on the left, and the least frequent on the right.

The second issue raised was in regard to strategies, sub-strategies and ordered states. Participants mentioned they would like the layout to group or cluster approaches based on similar strategies. If two students each have nine identical steps, but the first step in each approach is different, then the layout does not necessarily put the states from these two approaches close to one another. When looking at a hundred or more students, this makes understanding the number of strategies difficult to understand. A graph layout which places similar paths next to one another could provide a more intuitive visualization of the Interaction Network.

All participants reported that they would like to use InVis to augment their understanding of current classes' behavior and learning. This underscores the need for visualization tools that help instructors explore and understand student behaviors in software tutors. One participant's response summarizes our findings: "The tool provides a sense of how broadly varying students are in their approaches, how many get stuck, and how many make similar mistakes." This statement is a good representation of the kinds of insights *InVis* was designed to help detect.

3.1.6 InVis User Study Conclusions

The main contribution of the study reported in this chapter is the implementation and evaluation of InVis, a tool for the interactive visualization of problem-solving data. The use of InVis in three case studies and in our usability study showed that the use of interactive visualization techniques combined with an interaction network model in InVis allows users to explore and gain insight from interaction-log data. We performed a user study on InVis to show that users can successfully complete relevant tasks, and paired these results with a standardized method for testing the usability of a software tool. Users were able to explore the data from an entire class and were able to confirm some of the hypotheses they had about students, which was a primary goal. This is the first step in creating a domain independent visualization tool for understanding student behavior in software tutors, and these results informed our further research to improve *InVis*.

This study has a few limitations, first users were provided the data to investigate, and were tasked with a specific set of challenges. This was useful to determine whether or InVis is useful for completing those tasks, tasks which we felt were important, but does not address open exploration by users. Next the data set our users were navigating was small, with only 17 students worth of data, so similar success may not be true for larger data-sets for example 200 students. This limitation in fact is one of the reasons we built the study with a small data-set in mind, and also to provide tasks that contained only a single correct answer, so that task-success could be evaluated. Lastly, there seems to be a misconception between unique errors and error frequency which should be addressed in future versions of the tool. It seems a high number of unique errors, takes up more screen space and may lead our users to believe that some particular state, the left image in figure 14.

3.1.7 Case Studies and Success Stories

Plaisant comments on the usefulness of case studies and success stories in her work, "The Challenge of Information Visualization Evaluation" [47]. Case studies provide useful information from experts and can help address whether a user was able to find answers to questions they did not know they had. As mentioned by Plaisant, evaluation of tools meant to discover features which you did not know existed is difficult. Tory and Moller offer some support for ways in which this can be done [59]; and argue that expert reviews are useful because they can identify aspects of systems which non-experts may not recognize. In order for tools like *InVis* to be adopted it is important to have success stories so that the early majority will adopt the tool [47]; for this reason we provide four *InVis* success stories. These success stories also test InVis' ability to be used as a tool for exploration, one of the limitations of the previously described study.

We met with the developers of two separate propositional logic tutors. Both developers are university professors of logic and have extensive knowledge about the actions their tutors provide and how their tutors log data. We modeled their logic tutor data and provided each with InVis. We observed as they used the visualization tool to explore their tutor log-data. We also met with the developers of an educational game and visualized student-player data in a similar way.

3.1.7.1 Case One: Common Student Mistakes

We visualized data from Deep Thought[21] and interviewed the professor responsible for its development. We met for one hour and had him explore data from the tutor and inform us of different insights he was able to discover and hypotheses he generated and confirmed from using *InVis*.

We prepared a data set of thirty students, representing a classroom of students. The professor noticed a student had performed addition rather than conjunction in order to derive $A \wedge B$, which is an incorrect application of the rule. After he recognized this, he mentioned that it was a common mistake made by students; this was his

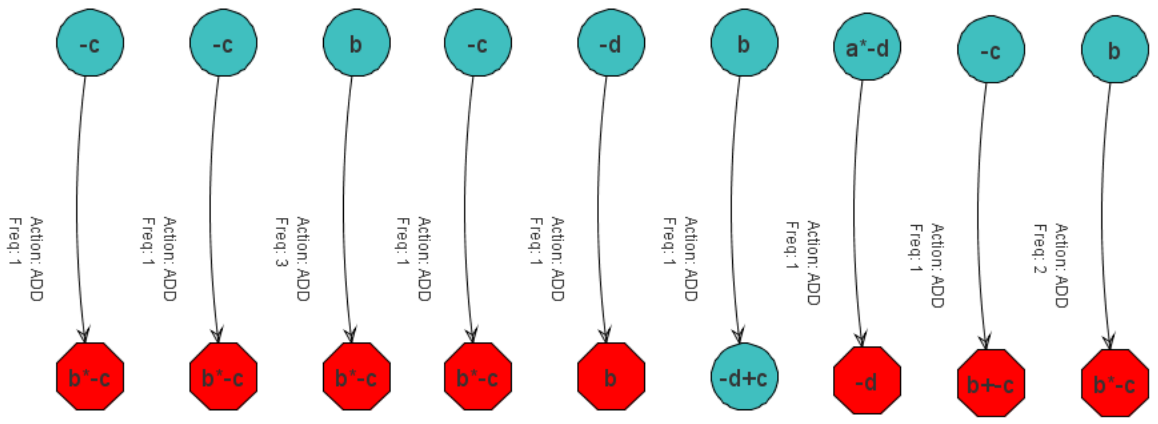


Figure 15: Eight out of nine applications of the addition rule were errors, denoted by the red octagon nodes. Five of nine of these actions would have been correct had students used conjunction, evidenced by the fact students enter $B \wedge \neg C$ as the derived data. In our case study, the user generated and confirmed this hypothesis through the support of data visualized in InVis.

hypothesis. He then used the action selector and entered ADD which selected all instances of students performing the ADD action in his logic tutor. Next he built a sub-graph, moving all those actions and their corresponding nodes to a new tab. Last, he was able to confirm his hypothesis; the data showed that eight of nine applications of the addition rule (ADD) were errors, five of which would have been correct had the student performed conjunction (CONJ) instead. See figure 15 for the subgraph generated by the process. With the same hypothesis in mind, a larger separate data set of Deep Thought tutor data was loaded, this time with 170 students. The larger dataset was made up of data from eight different sections of the course, that is four different professors who taught the course twice a year, in Fall and Spring.

Again all addition actions were selected, a sub-graph generated, and the hypothesis confirmed. This time, 11 out of 12 applications of the addition rule were in fact errors, and 7 of the 12 would have been correct had the students used conjunction instead. Within minutes, our user was able to identify a hypothesis, check the data using *InVis* and confirm the hypothesis. In the second data set, the task certainly would have been time prohibitive had he scoured the 170 student logs of the data. This shows one example of the usefulness of being able to generate sub-graphs of the behavior network in a separate tab for zooming and filtering purposes.

3.1.7.2 Case Two: Finding Students without Strategies

In the success story above, the user was interested in investigating a behavior exhibited by a sub-population, the group of students who used the addition rule. In this second case, a different professor was interested in the general behavioral trends of students. By including frequencies on nodes and edges, we are able to identify strategies that students perform in order to solve problems. In this case the hypothesis was that students who change all the implications in a problem into 'ORs' likely had no true strategy for completely solving the problem. The reason for this is over the years the professor has recognized students who employ this strategy often have difficulty actually solving the problem and thus students are explicitly instructed in class not to use this approach.

After loading that data into *InVis*, the professor looks for the two main strategies performed by the students, which can be selected in *InVis* by searching for sequences of frequent states. The first strategy, having the highest frequency, is the strategy she teaches to her class. We call this most frequent strategy the professor's strategy; the first node in this strategy has 74 student cases. The next most common first step has 29 students, and is the start of the unfocused strategy, that is to change an implication into an OR. Next the professor selected the first node of the professor's strategy and

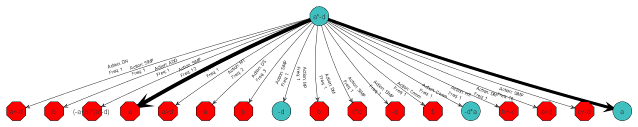


Figure 16: This is a screen-shot from our process of checking the hypothesis regarding the no-strategy approach. The highlighted node is the state in which afterwards many students have no-strategy.

Table 5: Success Rates for Different Strategies

Strategy	Student Count	Percent of Error Nodes	Success Rate
Professor Strategy	74	37%	74%
Unfocused Strategy	29	43%	59%
9 of 11 Success	11	28%	82%
7 of 11 Success	11	36%	64%

performed the select subtree action, which selected all states derived after the current state, effectively selecting all the different variations of the professor's strategy. Then she created a sub-graph. The same was done for the unfocused strategy. Next she selected all of the goal nodes of each sub-graph in turn and looked at the combined frequency of the goal nodes for each sub-graph. For the 74 students who applied the professor's strategy, 55 of those students arrived at the goal, giving a 74% success rate. For the unfocused strategy, the sub-graph has a combined goal node frequency of 17 out of the 29 students, resulting in a 59% success rate which is noticeably lower. In total there are 174 students, and the two strategies highlighted above are the most common. The next two most common strategies both have frequencies of 11, each with 9 and 7 students successfully solving the problem with their respective strategies. This suggests that the unfocused strategy approach has a particularly low success rate.

3.1.7.3 Debugging Tutors

One interesting application of InVis is found in the debugging of tutors. During the previously described case studies, the Interaction Network uncovered bugs in the tutor systems; that is, places where the recorded interactions should not have been legal actions. This is interesting, as both of these tutors have been used for many years and their data have been subject to extensive analysis. However, these bugs were not discovered until their data were visualized with InVis. Viewing the entire group of user behaviors at once improves the ability to spot 'peculiar' or outlier behaviors. In *ProofSolver* several solutions were noticeably shorter than the average, or skipped to the goal in 'strange' and invalid ways. After examining the series of actions these students performed, the professor confirmed that the interactions were illegal and should not have been permitted in the tutor.

In the case study described in section 3.1.7.1, some students were able to reach the goal by repeatedly performing the same action. In this case, the students were able to use the instantiation-action inappropriately to add any proposition to the proof. As a result of this, students could simply add items directly to the proof rather than use the rules, allowing them to game the system and illegally solve the problem.

3.1.7.4 Case Three: BeadLoom Game

I provide an in-depth analysis of the BeadLoom Game use case with the following dataset. We collected game log-files from a study performed on the BeadLoom Game (BLG) in 2010. Data came from a total of 6 classes, ranging from 6th to 8th grade; for a total of four sessions. There were 132 students, and 2,438 game-log files. The students were split into two groups (called A-day and B-day) and were presented with BLG features in different orders. The A-Day students were given access to custom puzzles (a free play option,) while B-Day students were given a competitive game element in the form of a leader board. Due to differences in student schedules some B-Day classes missed session three. To investigate whether or not there were different problem solving patterns between the groups, we colored vertices based on the percentage of students who visited each state from each group. The values were normalized from green (A-Day) to red (B-Day.) We then loaded the data into the InVis tool and presented it to the BeadLoom Game developers.

3.2 In-depth BeadLoom Game Case Study

Designing games is a complex task, and players are increasingly expecting highly individualized game experiences. One common method for tuning the player experience is play-testing, where players try the game before its release, exposing glitches in gameplay and difficulty in both surveys and game-play logs. However, effectively using the large amount of data available from a playtest can be challenging. Larger game companies such as Microsoft Studios have developed their own data analytics tools to understand these large complex player interaction datasets. In three dimensional game environments, these tools act like geographic information tools, overlaying player trace data on a game's level maps. However, in puzzle games and many educational games, there is a need to visualize the sequence of player behavior, but there is no logical spatial way to overlay player behavior over the usual visual representation of the game - except perhaps as a video. This makes it much more challenging to understand how, when, and where a puzzle or educational game design may need improvement.

In this chapter, we demonstrate a more in-depth application of InVis to help designers visualize sequential game states for a large number of playtesters at once. InVis was originally designed to visualize data from intelligent tutoring systems - software built to support learners in interactive problem solving. At the core of InVis is the novel idea of creating a ‘solution space’ out of player behavior networks. All game developers have an idea of a game’s state - an inventory of the current values for all variable aspects of a game. In educational software and in puzzle games, this state corresponds to where someone is in a problem-solving sequence - and can often be understood by taking a simple screen-shot. In both intelligent tutoring systems and in puzzle games, the experience can be individualized with feedback to direct attention to mistakes, or provide hints on what might be done next.

We believe InVis can support the iterative development and testing of 2D puzzle and educational games, by helping designers visually analyze player behavior to identify ways to better craft the player experience. We demonstrate how we have used InVis to explore game-log playtest data from BeadLoom Game, a puzzle game designed to teach math concepts. This case study shows that InVis helped developers discover surprising player behaviors, identify bugs and inefficient aspects of the interface, and design potential algorithms for individualizing the game experience.

As with the Deep Thought tutor logs, a vertical tree layout is used to preserve the step order of solution paths; therefore, the y-axis roughly corresponds to path length in the number of steps. For this in-depth study with BeadLoom Game, we modified

InVis to draw states and edges from left to right in order of most frequent, to provide a partially meaningful x-axis. Cycles were also removed before the layout phase and these edges were added again after the nodes' positions have been determined. The new ordering makes it so that the most frequent paths to the goal are easily identified, and solution paths are easily traced. The other modification to InVis for this study was to augment the display of nodes to allow a visual display of the puzzle state. This made it visually easy to understand what action sequence the player has taken. Hundreds of log-files can be visualized at one time, making it easy to see classrooms of students and their similarities as well as identify outliers.

In InVis we focus on evaluating and representing sequences of actions and states which closely model the behaviors of players. States retain all the details of their dimensions, (easily visible through the use of 'details on demand' features), and provide some filtering options for focusing a user's investigation. This allows game designers to explore all the information and aggregate statistics gathered by their game-logs, allowing for an effective method for gaining insights into how players behave.

3.2.0.1 Generating the Interaction Network from BeadLoom Game Data

We used the InVis Tool to explore player solutions. The state representation is a 41x41 array containing the 12 color values from the BLG game. Actions are represented by the six available bead-placement tools, as well as the relevant parameters for each tool. We also store the set of all cases visiting any particular state-vertex or action-edge, to present frequency and to connect case specific information to the

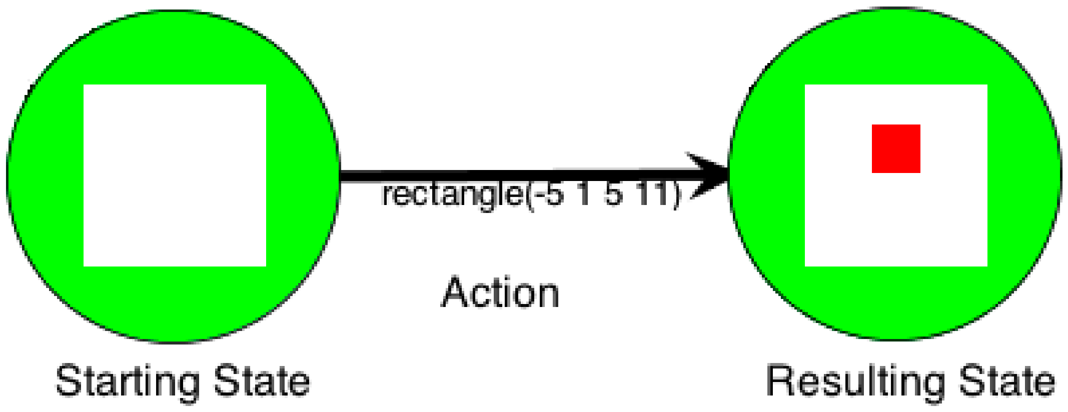


Figure 17: Each interaction in the log system is defined by a state-action-state tuple. This example shows what the state-action-state would look like for the action performed in figure 4.

Table 6: Design for the 2010 BLG study testing for ordering effects of different gameplay modes.

	A-Day	B-Day
Session One	Introduction	Introduction
Session Two	BLG	BLG
Session Three	Custom Puzzles	Leader board
Session Four	All Features	All Features

network representation.

In figure 17 we show how the BLG data is translated into an Interaction Network. Players move from a blank starting state to a state containing a red square by using the rectangle tool.

3.2.1 BeadLoom Game Case Study

From the same data set as before, we collected game log-files from a study performed on the BeadLoom Game (BLG) in 2010. Data came from a total of 6 classes, ranging from 6th to 8th grade; for a total of four sessions. There were 132 students, and 2,438 game-log files. The students were split into two groups (called A-day and B-day) and were presented with BLG features in different orders, presented in table

6. The A-Day students were given access to custom puzzles (a free play option,) while B-Day students were given a competitive game element in the form of a leader board. Due to differences in student schedules some B-Day classes missed session three. These students followed an abbreviated A-Day schedule during session four. In order to investigate whether or not there were different problem solving patterns between the groups, we colored vertices based on the percentage of students who visited from each group. The values were normalized from green (A-Day) to red (B-Day.) We then loaded the data into the InVis tool and presented it to the developers of the BeadLoom Game.

Next we met with the designer and a developer from the BeadLoom Game and asked them to explore their log data using our visual analytics tool, InVis. We hypothesized that InVis would:

1. Illuminate something unexpected or surprising about their data.

2. Provide insights about the behavior of their players.

3. Provide an efficient means to understand the general behavior of their players.

4. Allow observers to identify 'strange' or outlier behavior.

As discussed earlier, InVis supports a variety of straightforward interactions which can be used to explore the data contained in a behavior network. Users can filter the network based on the data they load into the vertices, edges, or aggregate data like frequency - the number of people who visit the same state or use the same action. The mouse provides panning, zooming, and selection, while GUI buttons provide the generation of sub-graphs, which can be used to further zoom and filter a behavior network. Lastly, search options exist for finding specific players and the states and actions they visited and used respectively.

3.2.1.1 Data Highlights

Figure 18 shows a set of students who worked on the same problem on two different days, the first and third session of the study. By looking at the number of states we can see a more diverse set of attempts on the first day. As mentioned before, edge width represents frequency, green vertices are from the A-Day students and red vertices are from B-Day; the goal has a square vertex. At the start of our investigation we colored the vertices to see if we could discover differences but it does not seem the classes had any significant differences between them. It is possible that the change in the number of states over time is the visual representation of learning, which figure 18 might suggest; additional research would be necessary to determine if that is so. What makes this interesting, is as students answer questions over time, they should become more skilled with the knowledge components associated with the questions. As a result the topology of Interaction Networks created from later solutions should have fewer states suggesting fewer unnecessary actions, like what we see in figure 18.

Though outside of the scope of this research it would be interesting to see if learning, or skill proficiency can be detected through the analysis of Interaction Networks. A potential study of this phenomenon would be to compare solutions created by skilled and unskilled students, then determine features which define the two solutions. Next, have a new group of students take a skills test to assign students to a skilled and

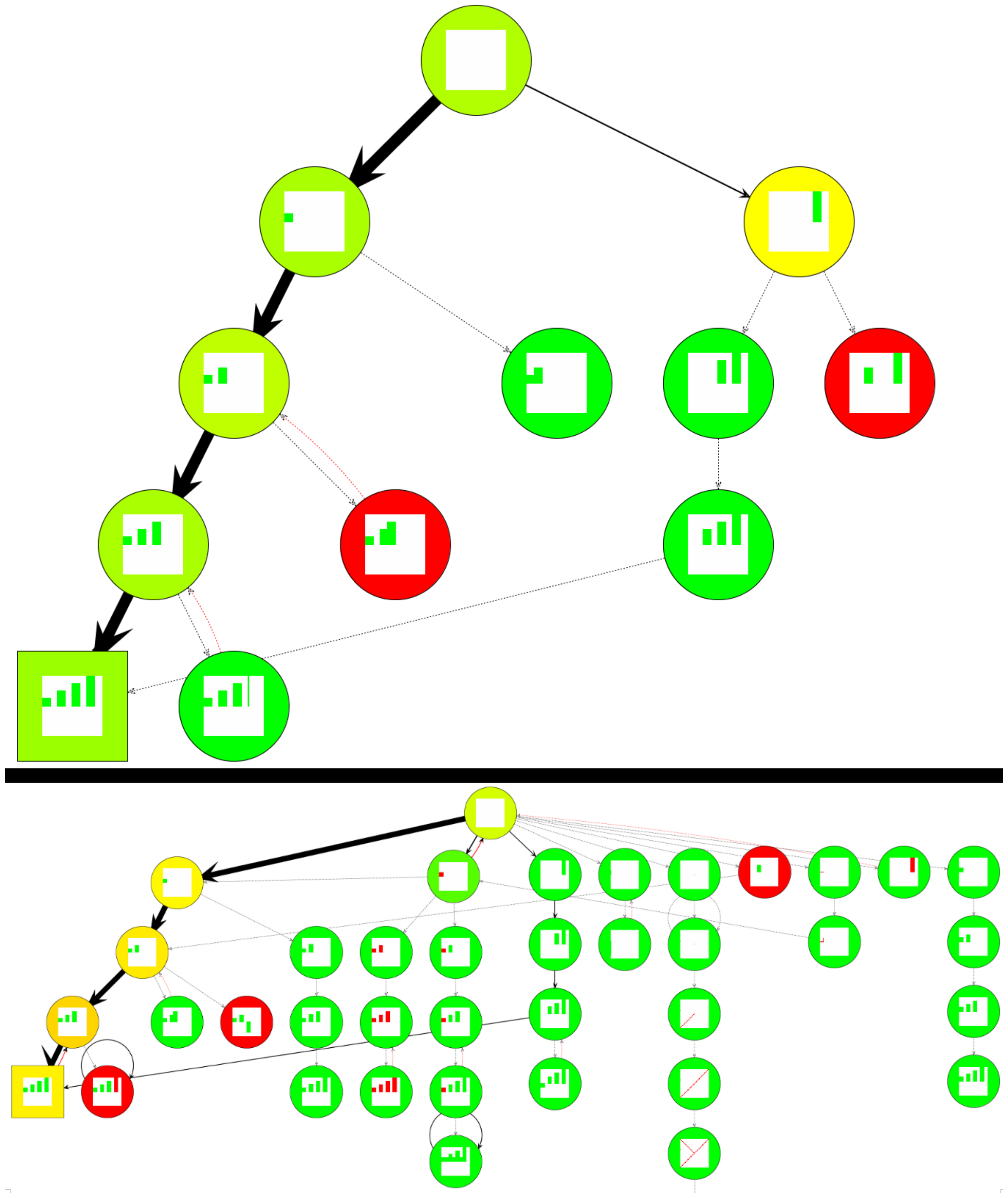


Figure 18: This image shows the students' attempts from their third session on the top, with their first session attempts on the bottom. This image suggests that as students become more familiar with the tool they are better at solving the problem and make fewer mistakes; thus fewer states are visited.

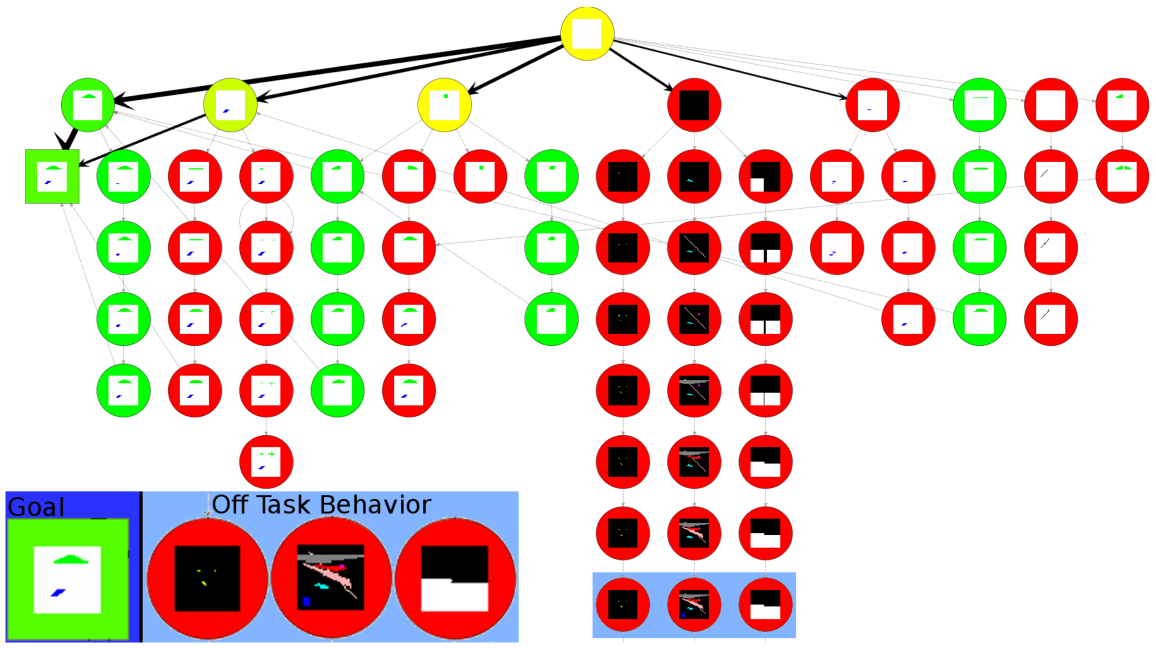


Figure 19: This image shows students' solutions to one problem, where we can recognize some off-task behavior. Here we have highlighted the final state of three students and enlarged their images. It is clear the students were not working towards the actual goal, which is shown on the left.

unskilled group. Then use the features of the solutions to predict the skill level of the students, the skills-test would be used to validate the prediction method. Performing this type of study across multiple problems and multiple tutors and datasets could address this method's level of generalizability.

The BeadLoom Game developers were able to identify a variety of design changes they would like to make to the game after spending roughly 20 minutes using In-Vis. The most surprising detail the developers were interested in was the number of students who seemed to participate in off-task behavior. Off-task behavior is easily identified as student solution-paths with low frequency and unusual length, as well as the visible evidence of no interest in working towards the goal state. For example, a student may opt to draw a picture rather than solve the puzzle. This will result in

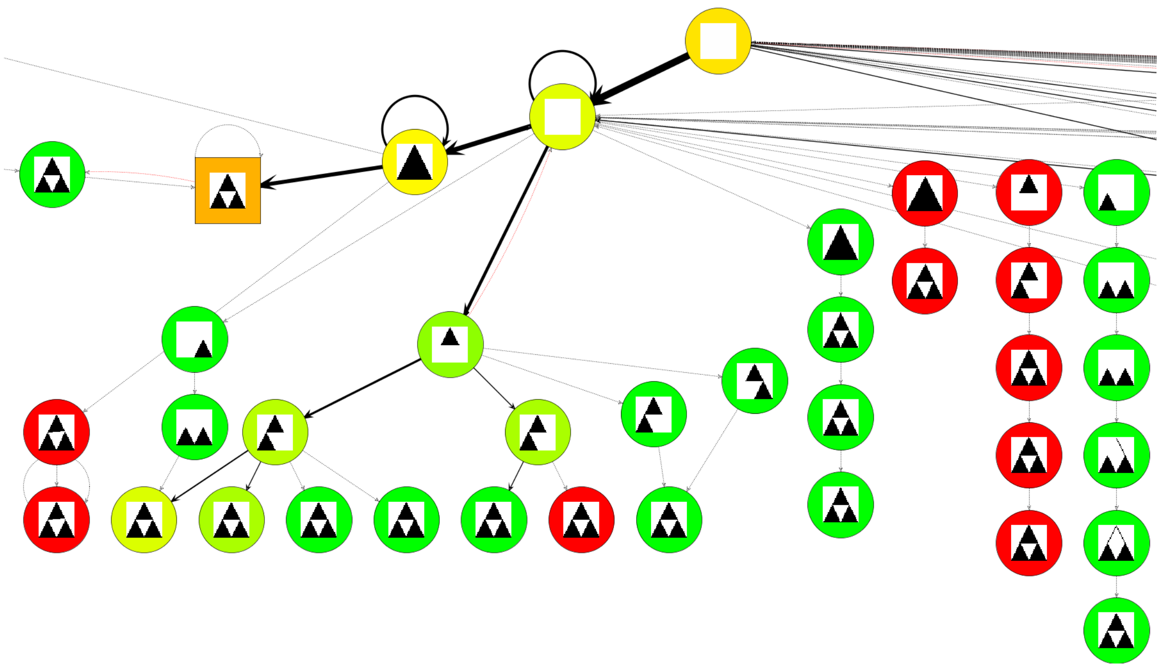


Figure 20: Here we can see that several states look very similar to the goal image. The goal image is in the orange box, while all other images are not recognized as the goal. Yet many of those other attempts look very similar and in most cases are merely off by a pixel. After using InVis, BLG developers added feedback about how many beads were incorrect in a solution to help students see and correct errors.

a path visually jutting out of the interaction network.

In game off-task behavior was a common problem as seen in figure 19, with a number of students foregoing the goal and designing their own images in the game instead. Some form of this type of behavior was present in roughly 90 percent of the puzzles solved by the students. A free-play mode is supported with the BeadLoom Game and can be found in the custom puzzle section of the game; it is arguable that students found the step of changing modes an unnecessary one.

Figure 20 shows an unintended flaw in the BLG: the states along the bottom of this image are almost indistinguishable from the goal vertex. Some of those solutions are only off by 1-2 pixels. This error seems to be due to the way the triangle tool works

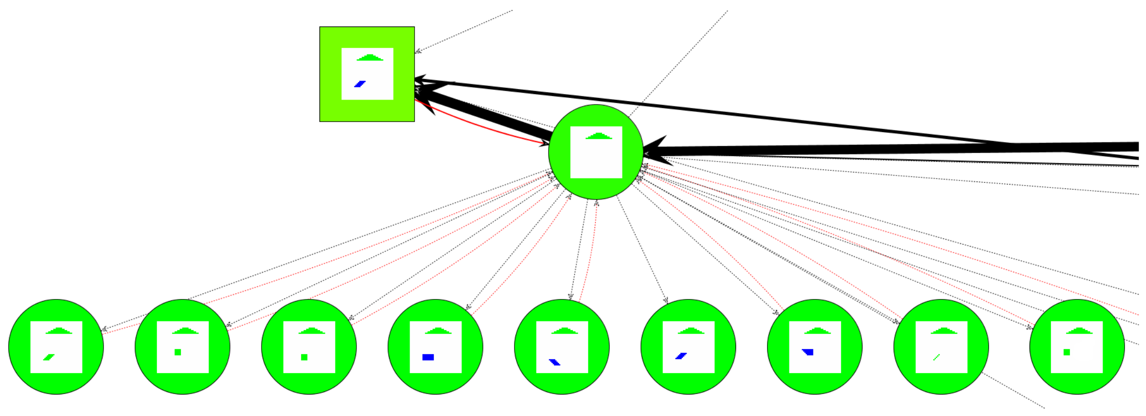


Figure 21: In this image we demonstrate how easy it is to look at errors. Each of the states on the bottom represents a student performing an incorrect move, and then undoing. This could perhaps be modeled by keeping track of states that are returned to frequently.

in comparison to the triangle iteration tool within the BLG; both of these tools can be used to create triangles, and depending on which one is used, the triangles can be very similar. Identifying this case in the data has lead the developers to make two design changes to the game. The new changes allow the player to see a count of how many pixels or beads remain incorrect, as well as a pop up window to inform players when they have successfully completed a puzzle, rather than forcing the student to click a button on the user interface to submit their puzzle for checking. Discovering this particular case in the data was a surprising and helpful discovery for the game's developers.

In figure 21 we can see a number of child states generated from a single state. The interesting feature of this image, is most of those states have a red returning action which represents the undo. This state, for whatever reason contained a number of mistakes made by students, but were quickly identified by the students.

3.2.2 BeadLoom Game Case Study Discussion

The designers were able to identify a variety of design changes they would like to make to the game after spending roughly 20 minutes using InVis. First a design issue regarding automatic feedback, after viewing and exploring the Behavior Network of the BLG data, the developers recognized a number of people performed actions, even after arriving at the goal state. In the original game users were required to click when they were finished, however after discovering the undesirable behavior, an automated method to detect the goal was designed so the level will end once the goal-state is reached. There were also a number of cases where students created the goal-image, but the whole plot was offset by 1 in some direction, so the game did not recognize it as a correct solution, as with the Triforce in figure 20. This could be frustrating since it is visually difficult to determine how each of these solutions in incorrect. The proposed design change in this case, was to have a counter on the user interface which lets the players know how many beads are still incorrect as compared to the goal.

Baker et al., analyzed how off-task behavior affects learning and they recognize two different types of that behavior. Furthermore they show that some off-task behavior does not affect learning gains, while other types of being off-task does [4]. Since this time the off-task behavior is within the game, the students must still use the coordinate tools in order to create their images, suggesting perhaps the students still may learn about coordinate systems.

One of the goals of InVis is to facilitate the construction of behavior models by improving the researchers' understanding of the data. Baker et al. also created a model for gaming-behavior, identifying when students try to leverage the interface so as to proceed through a software tutor's curriculum [5]. It could be possible to use the insights gained from InVis to help create a model for off-task behavior in the BLG, perhaps if a solution-path has too many actions, or a bead's color is incorrect, the game can offer an intervention to re-direct the player to the appropriate goal, or to another part of the game that allows them to make their own custom puzzles.

Another way to prevent excessive off-task behavior is through the use of game-mechanics; simply put, the game itself could give each player X number of actions they can perform. This could have two effects, one is for a simple puzzle that can be completed in, for example just two moves - like a tutorial, the game will only allow five actions to be performed, thus preventing a student from drawing complex images. A second potential effect could be that students may spend more time considering their actions and try to think of more efficient ways to reach the goal, i.e. avoid using only the point tool which colors one bead at a time.

Another pattern discovered using InVis occurred when students performed a large number of undo actions from particular states, such as in figure 21. These states represent locations where the addition of real-time feedback could prove the most valuable. The decision of when and where to add hints is an important topic in ITS research. If players make an error early the game does not currently provide any form of feedback. This allows players to continue down a path which is unlikely to reach the goal. Hint feedback could be added to redirect the player once they have entered one of these paths.

At the end of the interview, the developers were not only interested in adding the functionality mentioned above, but were also interested in how the Interaction Network could be used as in Stamper's Hint Factory to recognize appropriate behaviors and provide hints when students got off-track.

3.2.3 BeadLoom Game Case Study Conclusion

In this chapter we presented the application of InVis to data from the BeadLoom Game to understand player behavior. We were able to identify a number of interesting situations in the log-data that helped the BeadLoom Game developers improve the game and understand their players. By modeling the player data as a behavior network and visualizing each state graphically in a graph layout, we were able to display hundreds of game-logs in a concise form.

The most important results of this study were that InVis could be used to discover useful and surprising aspects of student behavior in BLG, that its developers have used to improve the game. Based on the feedback from BLG developers it seems that information visualization has a place in game analytics and can be used to improve our understanding of player behavior in games. It is particularly useful for the generation of new hypotheses about user behavior; which when followed up with confirmation studies will aid in the research and development of video games.

3.3 Chapter Summary

In this section we have presented two different studies conducted to determine the effectiveness of the InVis tool for exploring and understanding student behavior in two very distinct contexts: logic homework problems for colleges students, and a game designed to teach math to middle school students. Plaisant inspired our "triangulated evaluation" from her work on the challenges of visualization evaluation and her recommendation of using several complementary methods in order to mitigate the weaknesses of single techniques used alone [47]. We used James Lewis' validated usability survey [39], a task performance test, and qualitative feedback and a set of use cases to evaluate InVis from several perspectives. The use cases include data from two different logic tutors, the Deep Thought Logic Tutor [21], the Proof Solver, and an educational game, the BeadLoom Game [15, 14].

Based on the results of these studies, there are a few design adjustments we can make to improve InVis. One of the major points of challenge was the complexity of the Interaction Networks, which is important as I address the scalability of InVis and its application to even larger datasets such as those in the PSLC Datashop.

The user study conducted on the InVis tool informs us of two primary aspects of the Interaction Network and the method in which we make it accessible to our users. The Interaction Network is an understandable representation of the user interaction-logs, particularly the case studies where the InVis users were able to confirm hypotheses. We also learned that though there is a great deal of information contained within the network, there is a substantial amount information presented by displaying all the states, and for larger data sets likely too verbose of a description. By presenting a network, with so many nodes, it is not clear that for larger datasets with a thousand students worth of solutions, users will be able to take away their desired information quickly. From Shneiderman et. al [54] we can re-address the concept of overview first, though we have provided users an overview, it seems our overview is too verbose. From visual analytics [34] we can consider that our tool should identify what is important and what is not, and present the appropriate information to the user accordingly. These findings suggest the important contributions we propose in the following chapter. In this next section we will investigate using data mining techniques to present overviews, guide discovery, and reduce complexity of the information we present to InVis users.

CHAPTER 4: STRATEGY REDUCTION

4.1 Introduction

The Interaction Network can be used to store student solution data to open-ended problems. The Interaction Network also opens a new direction of educational data mining based on graph theory and network analysis which have gained recent popularity. We present a new application of using the Interaction Network and associated results, to provide evidence of the value of this method of storing student solution data to open-ended problems. The application is to extract a common-solution reduced network for providing a clearer overview of the solutions explored by students from tutor-log data.

One major advantage of computer based tutoring systems, is the ability to log data showing 'how' students solved homework problems, including student mistakes, an aspect not often aggregated from traditional paper based homework. However, one major challenge facing these systems is to provide educators an efficient method of presenting the data logged by these open-ended problem tutors, for the purpose of exploring student solutions and errors in hopes of help students and improving instruction.

One use of the Interaction Network is to provide a visualization tool to display the network to educators and researchers, providing insight into how students solved open-ended multi-step problems. We borrow the definition of open-ended problem solving from Becker and Shimada who define open-ended problems as, '... a problem that has several or many correct answers, and several ways to the correct answer(s)' [10]. These types of problems are often seen in Intelligent Tutoring Systems (ITS) and similar computer based instruction, like the Deep Thought Logic Tutor [21].

One important challenge facing the Interaction Network and InVis, the tool built for exploring those networks, is the size of the network, often resulting in thousands of nodes and edges for roughly a hundred students worth of data. These large networks make it difficult to retrieve a general overview of the student solutions. In this chapter, I present an algorithm that provides a summary presentation that drastically reduces the number of nodes in the network, and allows users to focus on the most common approaches used by students to solve problems.

We compare the reduction of nodes and edges between our summary network and the original, as well as other metrics, like the percent coverage of student solutions. We also compare our approach to two alternative filtering processes to show the benefits of our method.

Next we provide a set of domain experts with one of the problems from the Deep Thought logic tutor and ask that they describe the different approaches students use to solve these problems, as well as the common mistakes. We then compare the expert provided solutions to the reduced Interaction Network to confirm whether or not our method has appropriately captured the solution paths.

We show that the summary reduction algorithm when applied to the Interaction Network successfully reduces the number of nodes of the Interaction Network by between 86 and 96 percent, while still preserving the solution traces to an average of 52 percent of the goals and 40 percent of the student action frequencies. Furthermore, our method preserves all the solutions suggested by experts.

This work has two main contributions. First, the summary reduction algorithm effectively reduces an Interaction Network to significantly fewer nodes while still preserving the majority of solutions crafted by students. These types of reduced networks could aid in the improvement of adaptive tutors and focusing educator efforts, mainly by limiting the network to the most important solutions. In this way ITS developers can focus on hints and feedback needed for the highest impact areas. As well, ITS developers and experts may become aware of common, unexpected solutions students may apply. This could help bootstrap the development of example tracing tutors[2], by providing common solutions derived from actual student data. Lastly, when combined with InVis, this could provide an efficient method of understanding how students solved problems in computer based systems, potentially providing educators a way to respond to user work and mistakes in a timely manner.

4.2 Related Work

Previous research in the fields of EDM and Learning Analytics has focused on clustering as a means to identifying interesting characteristics student groups. These include research on social networks [58], as well as research on interaction in forums and chat logs [17], [26]. Our work differs, as we focus on clustering different student solutions to complex problems in order to reduce the space of student strategies. Others have looked at reducing the state space in learning environments for purposes of improving intelligent tutor efficiency and improving interpretation of the data for use by course developers and instructors[49]. We believe the Interaction Network will provide similar kinds of efficiencies.

The Interaction Network evolved out of work from Barnes and Stamper, and is the basis used for the Hint Factory, an MDP-based approach for providing automatically generated hints [55]. InVis is a visualization tool that we have been developing for visualizing and analyzing Interaction Networks.

There has been an increase in systems that log data for open-ended problems, where the Interaction Network can be applied. Sudol et al. describe a method of generating a similar state space that we use here, but for the domain of programming. In their work, they present the probability distance metric for states in programming problems for introductory students[57]. Menzel and Le have also focused on exploring the state-space of 'ill-defined' domains, using a constraint based system[38]. Mitovic explores open-ended problems in their web-based SQL tutor which is a constraint based system, but their system has also incorporated a student model to aid users[43].

In recent years the growth and need for tools to facilitate educational data mining has begun to grow. Tools like the EDM Work Bench [50] which is built to make handling tutor log data simpler. TADA-Ed is a tool designed for aiding in mining educational data generated from digital tutors. TADA-Ed's focus is on visualizing the results of several data-mining techniques, such as k-means clustering and decision trees applied to educational data[41]. We view InVis and the data structure it visualizes, the Interaction Network, as a similar tool. The purpose of these tools, to enlighten and inspire through the discoveries made from investigating your data from the perspective these tools facilitate. In addition we feel the Interaction Network allows for growth of educational data mining for procedural problem-solving, and also allows this community to leverage network analysis and graph based data mining techniques.

From our experience with earlier versions of the InVis tool, large networks made it difficult for educators and researchers to efficiently decipher the types of solutions students are using to solve problems from the Deep Thought logic tutor. In our previous work, users explored networks for nearly 20 students at a time. However, our goals for the InVis tool are to make understanding student solutions efficient, which can be achieved by viewing more students at a time. Another advantage to looking at more students at once, is it can be easier for users to compare different solutions, as they will not have to maintain those different solutions in their working memory but can quickly make comparisons based on the visualization. Finally, the visualization research community provides us with the Visual Analytics mantra, which argues when there is too much data, visualizations should leverage the machine to analyze the data, identify the important, and present that, rather than providing an overview of all the data and relying on the user to filter[19].

4.3 Reduction Algorithm

As described earlier, one major challenge is that when the data sets are large and diverse, the state space often is as well. One of the goals of InVis is to provide an efficient understanding of common student behaviors. However, exploring networks with thousands of nodes can be slow, and is also subject to hardware limitations.

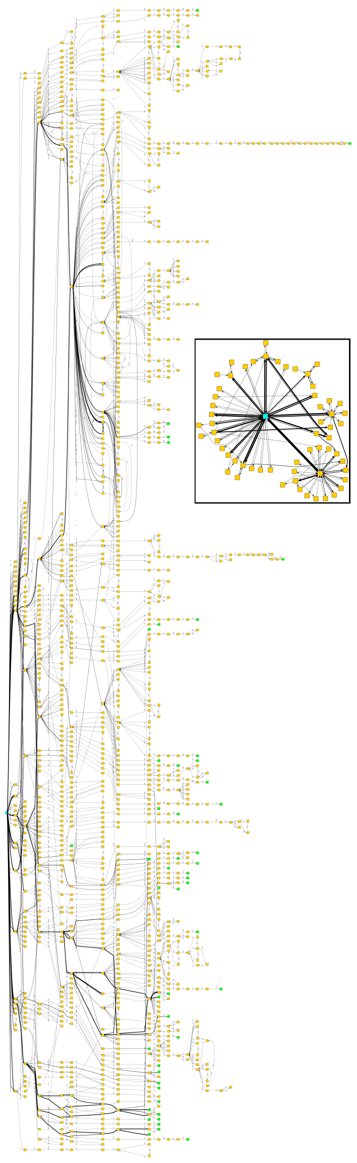


Figure 22: The complete Interaction Network for problem 3-5. This data sets has 192 students, 1252 nodes and 1835 edges. The light blue node at the top of the layout represents the problem start, light green are goal states, successful completions. In the inset box, we have re-created the root and successor nodes within distance two to depict the hub-like structure that is created from the undo action.

Table 7: Problem & Interaction Network Size Table

Problem	Students	Vertices	Edges	Interactions
1.1	295	1362	1853	13607
1.2	163	566	802	4978
1.3	241	978	1519	9593
1.4	234	1119	1659	9353
1.5	229	1348	1986	10749
1.6	177	1501	2149	11435
3.2	192	1459	2183	7943
3.5	192	1252	1835	7287
3.6	195	246	363	1823
3.9	177	1713	2377	7370
3.10	144	1353	1873	2733

In our experience, even professional software tools for viewing graphs start to slow down when the node counts exceed 1500, on typical PC hardware. Table 7 shows node counts for varying problems of Deep Thought based on student counts. Figure 22 shows the Interaction Network for problem 3-5. To address this problem we have developed the summary reduction algorithm for reducing the network by roughly 90%, while still preserving important information.

The purpose of the summary reduction algorithm is to maximize the amount of information we can gain from the data, while minimizing the number of nodes and edges, to make common approaches more clear. We also want to be as close as possible to a simple graph, since they easier to read when following the flow of state-transitions, because they have no parallel edges; technically our graph will not be simple since we have directed edges. Next, we want to preserve as many paths from the problem start to the goals as possible. We would also like to provide continuity and solution variations. Continuity in this case, implies the reduced network maintain complete solution paths, so the graph is understandable, as opposed to a list of the most frequent nodes, which means there should be no disjoint sub-graphs. By providing depicting all the routes to similar solutions we should be able to provide better estimations to the numbers of students who performed a particular solution. Without the context of the progression of the states, users would be unable to understand how the problems were solved. We want to provide a means for understanding how many students solved the problem, not just which actions were most frequent.

We will use four metrics for measuring our success.

1. Vertex and Edge Reduction Rates
2. Number of Goals
3. Number of Interactions
4. Average Student Interaction Frequency per Edge

The vertex and edge reduction will inform us how well we performed at reducing the number of states and actions. Next the goal counts will let us know how many of the solution paths we have maintained, from start to finish. For this metric, not only the count is important, but to maintain continuity all goals must have a path from the start of the problem to the respective goal state. The sum of edge frequencies will inform us of the total number of actions performed by all students, and in turn what percent of student actions are being preserved in our reduction. Arguably, if we must choose one of two edges with student frequencies one and ten, the ten frequency edge is more informative for an overview because ten students performed that action, over an action performed by a single student. Lastly, the average student frequency per edge will give us an indicator of how important each edge is in the network.

We asked two professors with more than a decade of experience teaching logic,

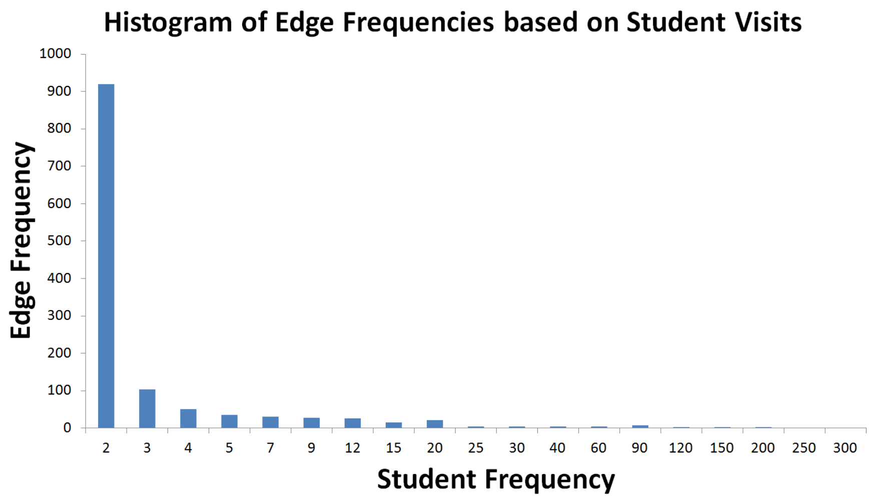


Figure 23: This histogram shows the frequency of edges, based on the number of students who traversed that edge in the Interaction Network. This is the histogram for the Interaction Network of problem 3.5 where there are hundreds of low frequency edges, and few high frequency edges.

as well as two graduate students who have either taught the course or performed a teaching assistant role, to provide us with the set of solutions they expected students to use to solve problem 3-5. These four experts provided us with eight solutions total, four of which differed in either the direction or actions used to solve the problem. Problem 3-5 was chosen because it has one of the larger ranges of possible solutions in our problem set. We will use these provided solutions in comparison to the reduced network and compare how many of those solutions are preserved in the reduced network.

4.3.1 Summary Reduction Algorithm

The idea for the summary reduction algorithm is inspired by compression algorithms. We want to identify the edges with the highest frequencies and preserve them, then find goal states that complete those paths. The Interaction Network for the problem 3-5 data set has 1252 nodes and 1835 edges. In figure 23 we provide a histogram of the edge frequencies from the Interaction Network. The summary reduction algorithm works by accepting three parameters, the Interaction Network on which to act upon, the percent of desired reduction, and a growth parameter.

Prior to the reduction, we first calculate a set of values in a pre-reduction step. In tutors which do not contain ‘undo’ or ‘delete’ actions, this step will not be necessary. In figure 22, the blue node represents the problem start state. In the figure we can see hub-like structure around that state and others. These states contain a number of single actions, some with high frequencies, immediately followed by high frequency ‘undo’ or ‘delete’ actions. These artifacts, though potentially interesting, are not what we want to preserve in our summary. For users who would like to detect these features, sorting nodes by degree, sum of in and out edges, is most effective. To adjust for this behavior of moving forward, followed by an undo, we calculate a table of negative weights. For each state, an incoming action followed by an ‘undo’, will increment a negative weight counter for the incoming action. This will be used to devalue the frequency of these actions. Next we remove the ‘undo’ edges from the network, this reduces the number of cycles and parallel edges presumably making the flow of state-transitions in the Interaction Network easier to follow.

Next we calculate the adjusted edge frequencies, which are equal to an edge’s original frequency minus the negative weight calculated in the previous step. Now, the network is reduced using the percent reduction parameter. We aimed for an order of magnitude reduction and so this parameter was set to 10%. For the reduction step, we generate a new network using the edges with the top 10% of student frequencies, and their source and target nodes. Depending on the network a set of disjoint graphs

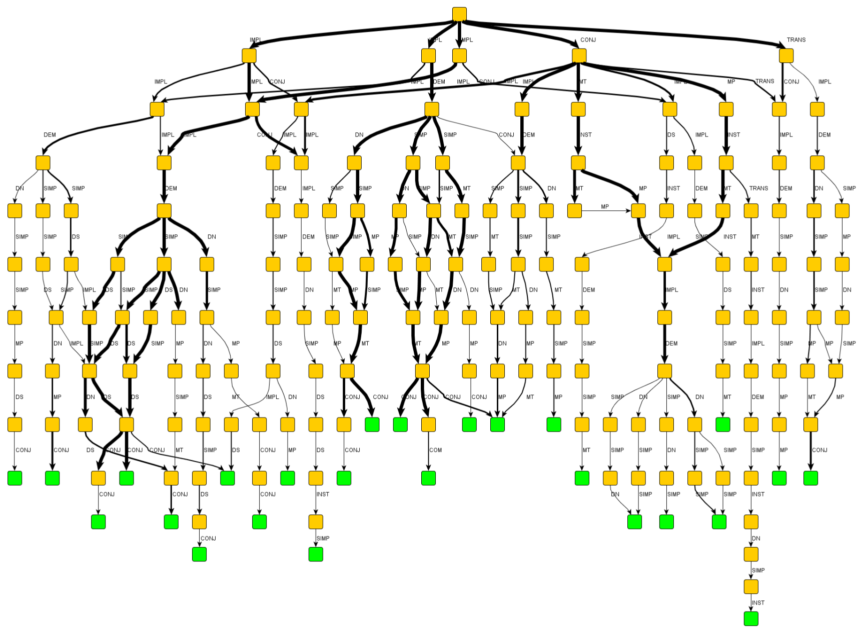


Figure 24: This is the reduced network for problem 3-5. This reduced network contains 186 nodes (85% reduction) and 219 edges (88% reduction).

will be created, we find the roots of each disjoint graph, which are the nodes with zero in-degree. We then calculate the shortest paths from the problem start to each disjoint-root, and inject the necessary edges and nodes to reconstruct a connected graph. Following this step we check the list of all goal nodes and attempt to connect any node in the reduced graph to any of the goal nodes, again using the shortest path in the original network. We use the growth parameter to limit the distance of the shortest path, for this work we used a value of ten. That is, if a goal node can be reached within ten edges, the path is added, otherwise it is ignored. As a final step, we attempt to connect all the nodes within the reduced graph to any other node in the reduced graph, again using shortest path. An alternative to shortest path could be to include all paths connecting the nodes, or the path with the highest total frequency. The shortest path provides the minimum number of edges to be added, and is what was chosen here. The reduced network for problem 3-5 is provided in figure 24.

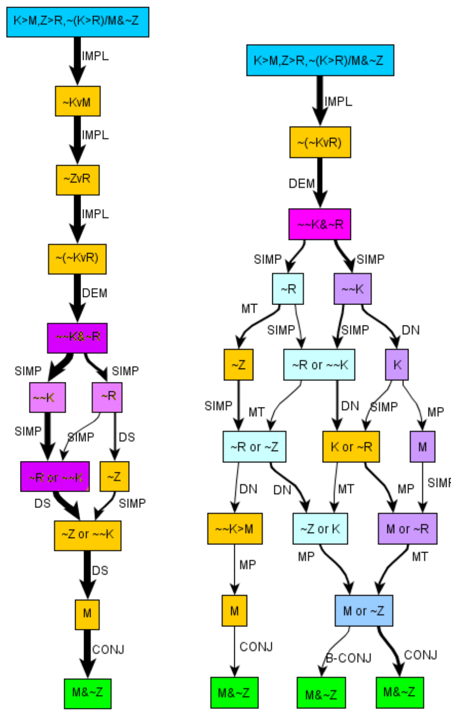
4.4 Results

For each of the 11 problems we have from our dataset, we generate the Interaction Network, the reduced Interaction Network using our algorithm described here, as well as two other reductions described below. Next we average the values across all 11 problems and compare. From this image we can see the start of the problem at the top of the network, with various paths leading to goal-states shown in green. In the next section we discuss the results of table 8 and the comparisons with the other methods of reduction.

Table 8: The average metric scores across 11 problems and 2239 problem sessions. Original refers to the full network values. Each column is a method and its score, with percentile comparison to the Original network in parenthesis. For vertices and edges the percent is the amount of reduction, for goals and interactions it is the amount of coverage or inclusion.

	Original	Reduced	Shortest Paths	Greater than Frequency One
Vertices	1172	**114 (90.26%)**	238 (77.85%)	203 (81.73%)
Edges	1690	**132 (92.23%)**	237 (84.75%)	348 (78.33%)
Goals	38	20 (52.54%)	**38 (100.00%)**	12 (33.04%)
Interactions	3332	1283 (39.90 %)	1162 (36.89)	**1990 (60.77%)**
Avg. Edge Freq.	2.10	**12.34**	6.60	6.03

The experts provided eight total solutions independently, four of which differed in either the actions or direction in which the problem was solved. All four participants provided us the solution found in 25(b), which uses a Modus Tollens and Modus Ponens rule. The two teaching assistants provided us with the solution found in 25(a). One backward solution was provided, and one solution which used both disjunctive syllogism and Modus Tollens were provided. Our reduced graph contained three out of the four solutions provided by experts. The fourth solution, which is a forward disjunctive syllogism and Modus Tollens approach was not present, however a working backward version of this solution is presented in the reduced graph. Furthermore, when looking at the full network, only a single student solved the problem with both a disjunctive syllogism and Modus Tollens approach, and they were working backwards.



(a) Dys. Syl. (b) M. Tollens M. Ponens

Figure 25: Left) A solution trace for problem 3-5 using two Disjunctive Syllogisms. The two experts did not suggest this solution, though the teaching assistants did. Right) An alternative solution trace for problem 3-5 using Modus Ponens and Modus Tollens.

4.4.1 Comparison

We compare with a shortest path approach and a frequency one removal approach. The shortest path method of reduction, takes in the start state of the problem and the set of goal nodes for the problem. Next, Dijkstra's shortest path algorithm[23] is run and the result is the union of the shortest paths to each goal. The Frequency one filter approach, simply removes all edges from the network with student frequency one.

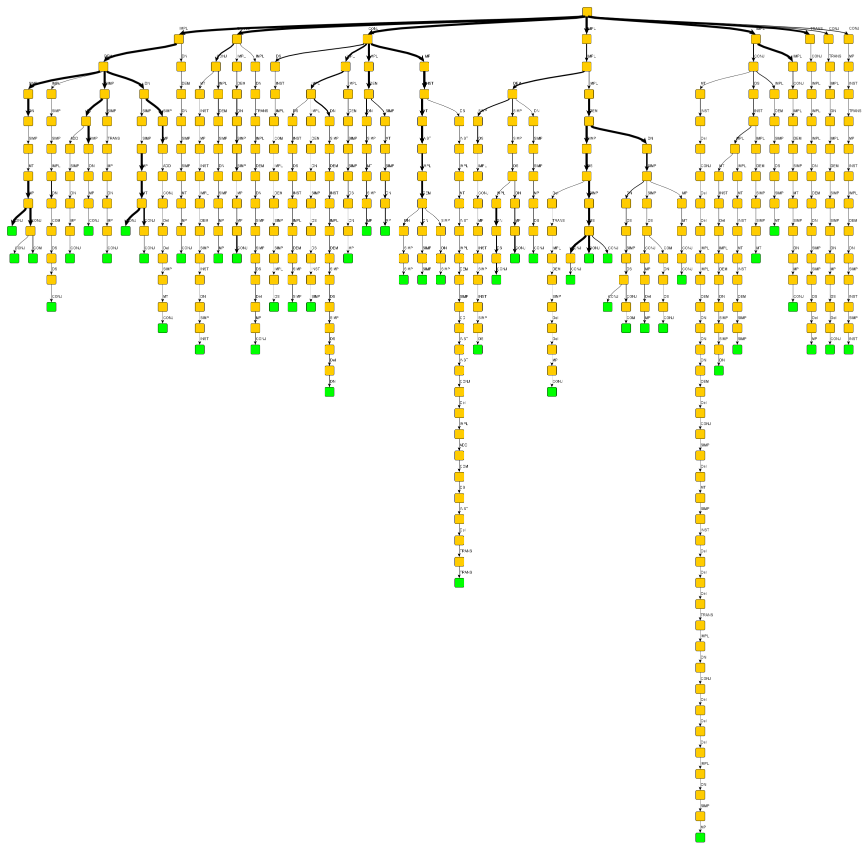


Figure 26: This is the problem 3-5 data set after the shortest paths reduction applied. This network contains 383 nodes(69% reduction), and 382 edges (79% reduction).

By referring to table 8 we can see some advantages and disadvantages of each approach. First, as expected, the shortest path approach naturally has 100% goal coverage, that is we can see *a* path to every goal from the original Interaction Network. The disadvantages of this approach is that the paths chosen do not optimize the frequencies of edges, because the shortest path can contain many frequency one edges. Next the overall reduction rates are half as effective as our method, leaving on average twice as many nodes and edges. This method preserves fewer actions performed by students while having lower rates of reduction. The resulting shortest paths network for problem 3-5 is shown in figure 26. One interesting aspect of our approach over the shortest path method, is if the growth parameter is set to infinity, the path to all goals will be preserved, though naturally reduction rates will be affected. In this sense, our method can facilitate 100% goal coverage, if desired.

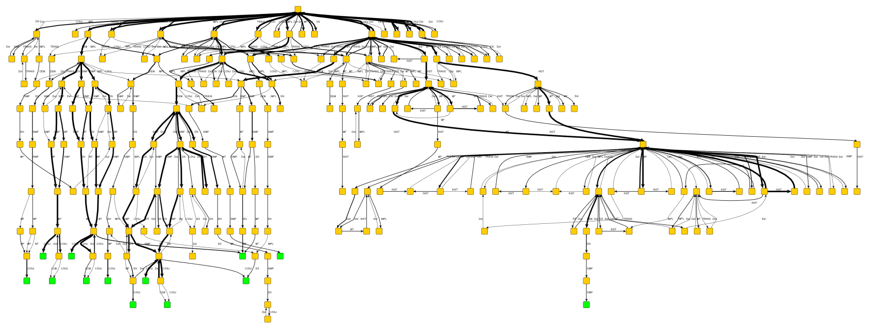


Figure 27: This is the problem 3-5 data set after the frequency one filter reduction is applied. This filtered network contains 235 nodes(81% reduction), and 400 edges(78% reduction).

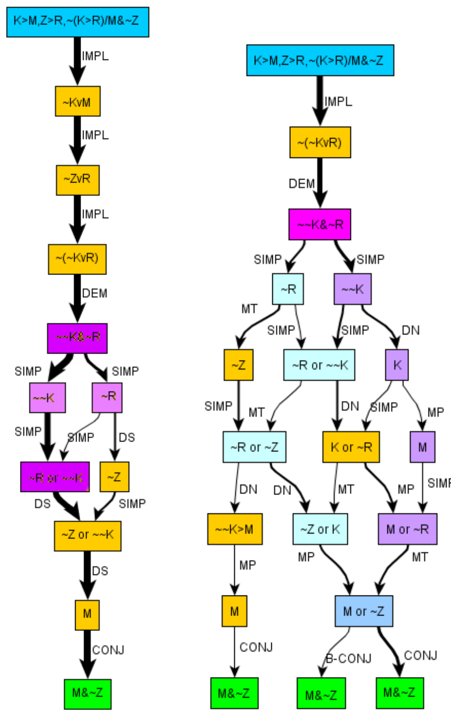
Alternatively, the frequency one filter, maintains a higher rate of interactions, as we would expect since fewer edges are removed. However, frequency one filtering suffers from low rates of reduction, having double the number of nodes and triple the number of edges on average when compared with summary reduction, while also having lower rates of goal coverage, 33% compared to our method which achieved 52%. This method has lower reduction rates and lower goal coverage. Figure 27 shows the resulting network for problem 3-5 using the frequency one filtering process.

Finally, by comparing the average edge frequencies in table 8 we can see our method has double the value than either of the other two approaches. This score is meaningful because it is the average number of actions performed by students, per edge within the network. This means our summary reduction algorithm uses many fewer nodes and edges to represent a significant portion of the problem goal states and student interactions.

4.5 Ordering Detection

First, we will analyze the following problem from Deep Thought, students are given three premises: K→M, Z→R, ¬(K→R) and tasked with deriving the solution M∧¬Z. Deep Thought provides a visual interface to deriving conclusions in propositional logic problems. We will discuss two common solutions to this problem; the first is as follows: perform three implications, deriving ¬K∨M, followed by ¬Z∨R, and¬(¬K∨R). Next, the students perform DeMorgan's to derive:¬¬K∧¬R. Following are two simplifications, to derive¬¬K, and¬R, two disjunctive syllogisms, which generate M and ¬Z, the final step is to perform conjunction which results in the conclusion of M∧¬Z. This solution is provided in figure 28(a). An alternative solution to this problem is shown in figure 28(b). For the tutor log, each application of a rule is a single transaction, in practice students generally have more actions than are required to solve the problem, partially facilitated by an available delete action. For this problem, 3-5, there are eight or nine actions necessary to solve the problem in the above solutions. However, the average number of actions per student is 40 for our data.

Let us look at the related Interaction network for these two solutions. For the Deep Thought problems, we collect the actions in the order that they occurred, but then perform lexicographical ordering on each state, which removes order from arguably equivalent states. An example of this can be seen in figure 28(a). In this case, one



(a) Dys. Syl. (b) M. Tollens M. Ponens

Figure 28: Left) A solution trace for problem 3-5 using two Disjunctive Syllogisms. The two experts did not suggest this solution, though the teaching assistants did. Right) An alternative solution trace for problem 3-5 using Modus Ponens and Modus Tollens.

set of students has simplified for ¬¬K, while the other set simplified for ¬R. Next we can see the students perform the reverse of their previous steps. In this sense, we make the assumption that cognitively students are at the same 'state' within the tutoring system, and do not need to be considered different. The Logic domain makes this equivalence matching convenient but for other domains this could be more complicated. The 'bubble' created by these two actions can be considered an unordered set of actions, it is common in this domain to see even larger bubbles, as shown in figure 28(b). This type of bubble in the Interaction Network captures some of the diversity of solutions generated by students, so we aim to preserve them in our reduction method; more on this topic will be addressed in section 4.5.

4.5.1 Detection Algorithm

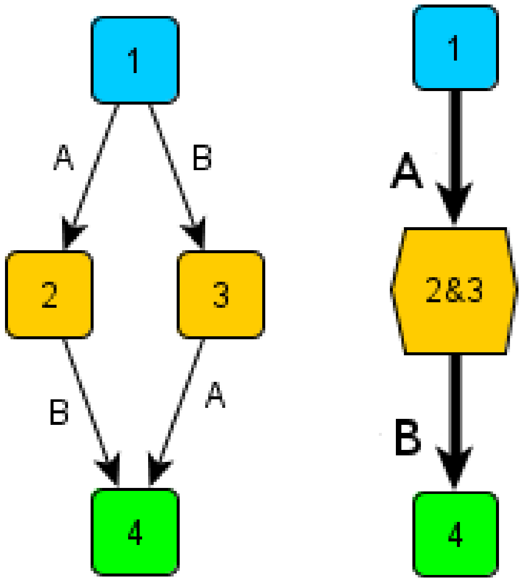
The bubble detection algorithm is simple. We iterate through all of the vertices and check specifically for vertices with either in-degree or out-degree of greater than two. These vertices populate two lists, which are potential starts and ends to an ordering *bubble*. That is in order for a bubble to exist between two nodes, there must be at least two paths between the start and ending states of a bubble. Next, we focus specifically on bubbles of length two, because larger bubbles can be computationally expensive. Now we iterate through the In-degree list, and calculate Dijkstra's Shortest Path (DSP)[23] to each node in the out-degree list. By limiting to length's of two, we can terminate the DSP algorithm early in processing, which has a processing time of $O(|E| + |V|log|V|)$ [29].

When a bubble is detected there is a four-tuple of nodes, or sub-graph of the structure depicted in image 29(a). The children nodes of the start of a bubble are combined and so are the edges, an example is shown in figure 29(b).

It is possible for larger bubbles to exist as shown in figure 28(b). We do not want to ignore these larger bubbles, as they too can be further reduced. By iteratively running the ordering detection and combining algorithm these larger bubbles will also be combined and reduced into a single path.

4.6 Consolidating Frequencies

We can consider for a moment an option of ordering when considering the summary reduction and bubble detection algorithm. The options are to run the summary



(a) Bubble detected

(b) Resulting Compression

Figure 29: Left) An example of the structure created by a bubble, detected when considering different orders to arguably the same strategy. The blue node 1 at the top depicts the bubble-start, while the green node-4 identifies the end. Note the edges through each respective path are the same, but simply ordered differently. Right) The structure of the bubble after it has been combined. Note the size and shape of the combined node are altered to depict the difference, also the labels have changed. The edges have a larger width based on the combined frequencies of the original edges.

reduction first, followed by the bubble detection, or the reverse order. If we run bubble detection first, we could consolidate frequencies, so the ordering will not affect the frequency of the edges. This could be important in the summary reduction stage, where we select the top ten percent of edges based on frequency. It would be reasonable that by combining the frequencies of edges for the same strategy, different edges could be selected by the edge selection portion. However in practice we see that this ordering has little effect on the edges selected.

The reason we see little effect of ordering on the edges selected is based in the concept of graph flow, similar to the maximal flow problem [28]. In order to see a diversity in paths between two nodes, each path must contain at least one student who progressed down a particular path. So, if for example, only two students enter a particular state *i*, some later state *i+2* can have at most two paths between them. To have three separate paths, state *i* would need at least 3 students who entered that state. As a result, low frequency states do not have a high number of diverse paths, by definition.

Table 9: The problem column refers to problem from the 2009 Deep Thought data-set. Minimum Edge Frequency is the lowest frequency edge from the summary reduction algorithm when the top 10% of edges are selected based on frequency. Insufficient Frequency count is the number of bubbles which contain an edge that has a frequency lower than the minimum edge frequency value. Insufficient Frequency count is the number of edges that would not be removed if bubble detection and combining were run first. The final column is the number of orderings detected for the data set.

Problem	Minimum Edge Frequency	Insufficient Frequency Bubble Count	Number of Bubbles Detected
1-1	3	0	113
1-2	2	0	3
1-3	2	0	24
1-4	2	0	23
1-5	2	0	47
1-6	2	0	39
3-2	2	0	39
3-5	2	3	123
3-6	4	4	37
3-9	2	2	29

The results are shown in table 9 and almost counter-intuitive, since it shows that few sets of edges with even arguably low frequencies are affected by the ordering. The main reason for this phenomenon is because many bubbles have high edge frequencies, as was previously discussed above. For example in the case of Problem 3-5 in the table, there are 123 bubbles detected, yet only three of the edges in that entire set of bubbles has frequency greater than two and are ignored in the top 10 % of edges. Referring back to figure 27 we can see that only three out of 400 edges would differ based on the ordering of the algorithms.

By combining the summary reduction step, followed by the bubble detection we obtain the resulting network shown in figure 30.

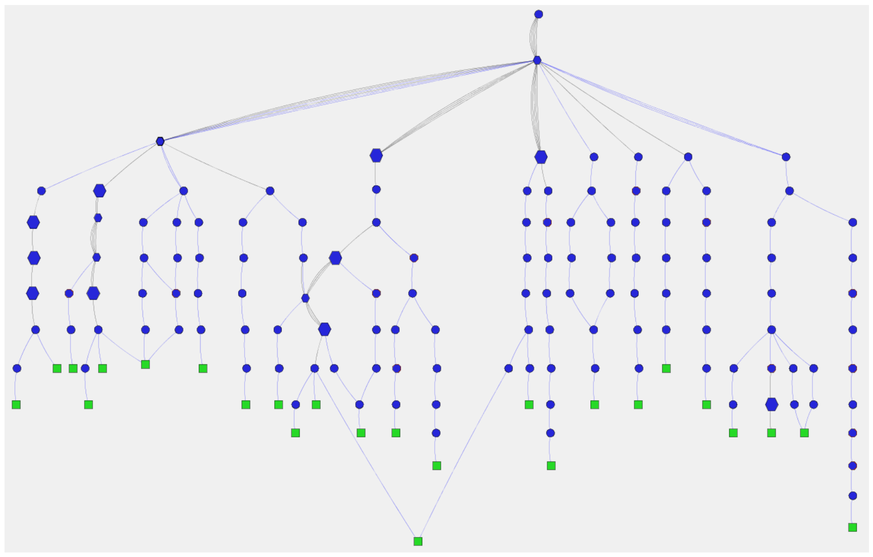


Figure 30: This is the network for problem 3-5, with the summary reduction algorithm applied, and then bubbles combined into single nodes, after five iterations. Hexagon shaped nodes contain 'multiple' nodes, and have been combined to help reduce the complexity of the Interaction Network.

4.7 Conclusions

We presented the Interaction Network data structure, which is a method for storing student solution data from open-ended multiple step problem solving domains. We then provide an algorithm for reducing the complete Interaction Network to a summary of the most common problem solving approaches used by students. We showed that the summary reduction algorithm was capable of drastically reducing the number of vertices and edges of the Interaction Network by an average of around 90%, while still depicting more than half the of the solution paths and accounting for 40% of interactions performed by students.

4.8 Chapter Summary

The number of edges with low and often size one frequency are common but arguably offer little into understanding the generalizable behaviors of student solutions. The summary reduction approach focuses on the edges with the highest frequencies, along with other features that help depict complete solution paths. Next I consider the order of actions that people have performed to further reduce the Size and Order of the resulting network and further approach identifying specific approaches students have exhibited in their solution attempts. This work helps facilitate a concise summary of the solutions that students have used to solve the task they are provided.

CHAPTER 5: INVIS TOOL AND STUDENT DIFFICULTY

Previous methods and features discussed have focused on removing redundant information, focusing on student approaches and how students have progressed to problem goals. However, there is another important aspect of problem solving which should be addressed, which is identifying where students have faced difficulties. With summary reduction and bubble detection the focus has been on high-frequency states of the solution space. For difficulty detection we want to identify the states in the solution space where students have the greatest difficulty. For this we offer three methods of identifying states where students might be having trouble.

The challenge is that educators need to see where students are having trouble. Although errors can be useful, these types of errors are constrained to tutor errors and may not fully capture what is difficult in a problem. Next simple filters, or orderings could leave the graph disjoint or provide incoherent collections that are difficult to understand for the user. Our goal then should be to detect states with evidence of difficulty, present them within a context of the solution network, and display them in a readable manner.

5.1 Expected Difficulty

The foundation for our difficulty detection is based in the definition of *item difficulty* supported by classical test theory, a heavily researched concept in the field of psychometrics. Item Difficulty is the number of correctly answered questions by the population of students who have attempted the item. We have applied this to each state that is contained in the Interaction Network. A slight variation is applied here to treat the student population as the number of students who visit a particular state, since not all students visit all graph states. This means the item difficulty for each state is the number of goals reached from a particular state, divided by the number of students visiting that state. For example, if one out of four students arrive at a goal from a particular state, the difficulty will be equal to 0.25, while another state could have four out of 12 students achieve a goal state, in which case it too will have a difficulty of 0.25. The reduced network for problem 3-5 is shown in figure 31, the level of red is based on the expected difficulty.

In this case, the network is colored based on the inverse level of difficulty. To explain, imagine some state has ten percent success, meaning ten students enter the state and one successfully solves the problem. This state would have a difficulty of one over ten or 0.1. Next, we are interested in the level of error, so we subtract that expected difficulty from 1.0 and get 0.9. This value is the percent level of failures. We use this value to be converted to a color which is normalized by all expected difficulty values, again the results are shown in figure 31.

The disadvantage of this approach is that states with high numbers of failures may not be recognized. If there are two states, one with 100 students and another with three students, with 22 and 1 failure students respectively. The expected difficulty of these two states would be 22% and 33%, suggesting that the 1 failure student state, should be given attention. However, if a professor has time to address only a

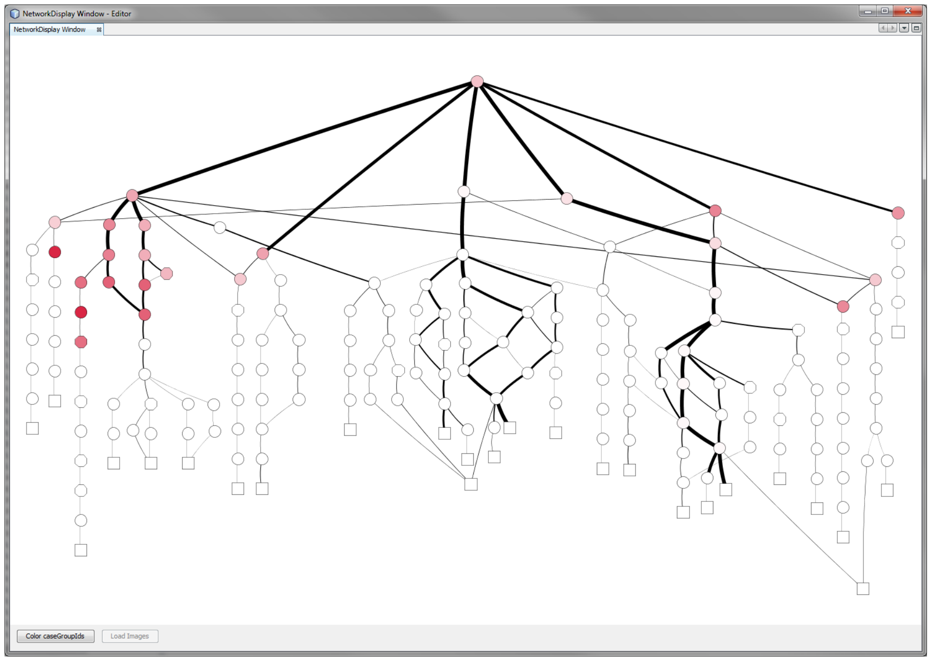
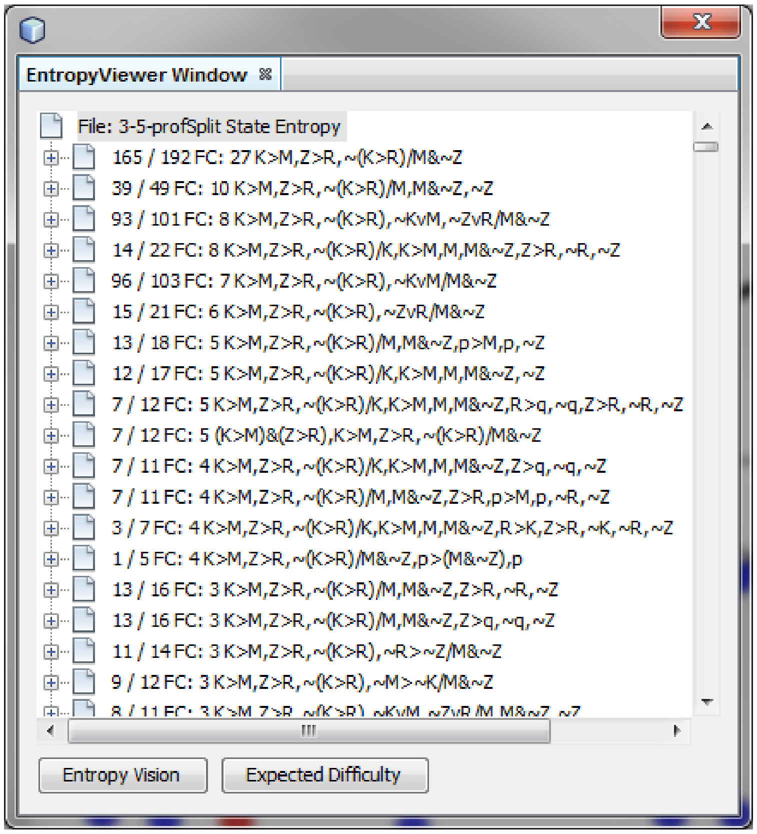


Figure 31: This is the strategy reduced network from the problem 3-5 dataset. The color corresponds to the expected difficulty for each state, which is equivalent to the failure-count divided by the total frequency of the state.

single state, then it is arguable that priority should be given to the frequency 100 state, as addressing the misconception here may help as many as 22 students. To address this issue we created a similar view, which is no longer normalized based on student frequency, but is instead the absolute value of difficulty, that is the number of students that fail from a particular state.

5.2 Absolute Value Difficulty

The absolute value method is supported through a difficulty window which is populated with all of the states. An ordering function can be applied to the window to focus the user's attention to the states with highest failure rates. An example of this window is shown in figure 32.The user can sort the nodes in terms of absolute goal counts, or absolute failure counts. The advantage of using absolute values is educators



```
File: 3-5-profSplit State Entropy
165 / 192 FC: 27 K>M,Z>R,~(K>R)/M&~Z
39 / 49 FC: 10 K>M,Z>R,~(K>R)/M,M&~Z,~Z
93 / 101 FC: 8 K>M,Z>R,~(K>R),~KvM,~ZvR/M&~Z
14 / 22 FC: 8 K>M,Z>R,~(K>R)/K,K>M,M,M&~Z,Z>R,~R,~Z
96 / 103 FC: 7 K>M,Z>R,~(K>R),~KvM/M&~Z
15 / 21 FC: 6 K>M,Z>R,~(K>R),~ZvR/M&~Z
13 / 18 FC: 5 K>M,Z>R,~(K>R)/M,M&~Z,p>M,p,~Z
12 / 17 FC: 5 K>M,Z>R,~(K>R)/K,K>M,M,M&~Z,~Z
7 / 12 FC: 5 K>M,Z>R,~(K>R)/K,K>M,M,M&~Z,R>q,~q,Z>R,~R,~Z
7 / 12 FC: 5 (K>M)&(Z>R),K>M,Z>R,~(K>R)/M&~Z
7 / 11 FC: 4 K>M,Z>R,~(K>R)/K,K>M,M,M&~Z,Z>q,~q,~Z
7 / 11 FC: 4 K>M,Z>R,~(K>R)/M,M&~Z,Z>R,p>M,p,~R,~Z
3 / 7 FC: 4 K>M,Z>R,~(K>R)/K,K>M,M,M&~Z,R>K,Z>R,~K,~R,~Z
1 / 5 FC: 4 K>M,Z>R,~(K>R)/M&~Z,p>(M&~Z),p
13 / 16 FC: 3 K>M,Z>R,~(K>R)/M,M&~Z,Z>R,~R,~Z
13 / 16 FC: 3 K>M,Z>R,~(K>R)/M,M&~Z,Z>q,~q,~Z
11 / 14 FC: 3 K>M,Z>R,~(K>R),~R>~Z/M&~Z
9 / 12 FC: 3 K>M,Z>R,~(K>R),~M>~K/M&~Z
8 / 11 FC: 3 K>M,Z>R,~(K>R),~KvM,~ZvR/M,M&~Z,~Z
```

Figure 32: These are the top states based on absolute failure count supplied by the InVis tool, by selecting one of the states, the corresponding state is highlighted in the Network Display window.

can focus on states that impact the highest numbers of students. Through this type of approach an intervention given to the students will maximize the amount of affect on students having difficulties successfully solving the problem being visualized.

There is one difference between the expected difficulty view show in figure 31 and figure 33. With expected difficulty, we simply colored the vertex based on the expected difficulty. With the absolute value view, we color each vertex based on the failure count, but in addition to this, the node is sized on the number of people who came to the state and reached a goal. This helps depict the overall frequency of the vertices as well as any paths of high frequency states.

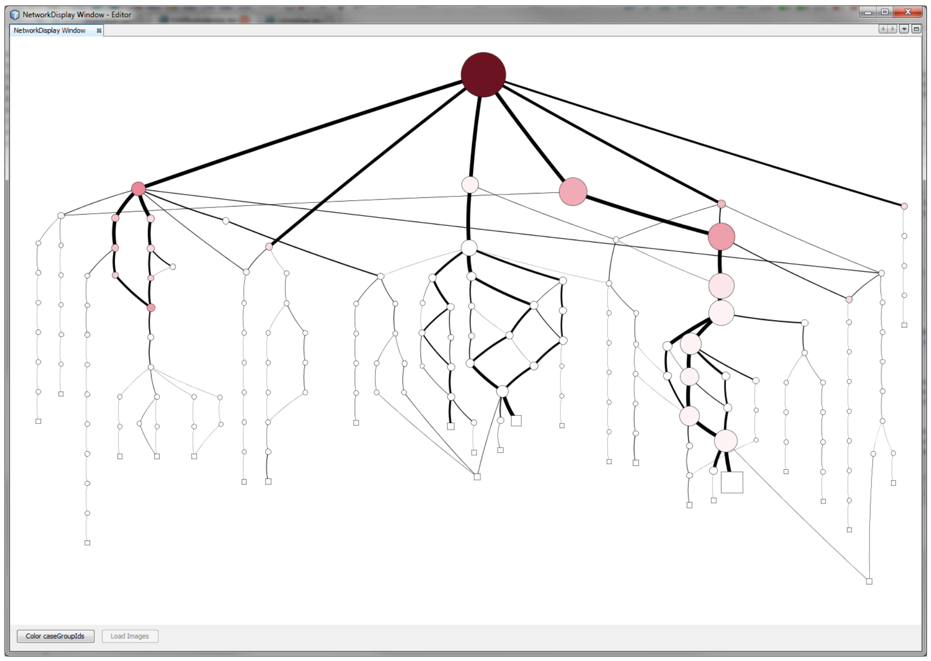


Figure 33: This is the strategy reduced network from the problem 3-5 dataset. The color corresponds with the fail-count of each state, that is the number of students who visited the state but failed to arrive at a goal state. The size of the node relates to the number of students who visited the state and later reached a goal state. As a result, large white nodes denote highly successful states, while large red nodes show states with a high number of failures and a high number of goals reached.

5.3 Conclusions

We mapped expected difficulty to the states of the Interaction Network which provides one type of view regarding evidence of difficulty for students. However this view alone is insufficient to describe to educators where to focus their efforts in terms of addressing students' challenges. A similar view based on difficulty counts, that is the number of students which fail to reach a goal, provides an alternative approach which highlights the states which affect the most number of students. This visualization approach highlights areas with high potential success and high numbers of failures for educators to focus.

5.4 Chapter Summary

Through the use of InVis I support two audiences, the first, researchers, can load data into the InVis tool and explore a variety of aspects in regards to student solution attempts. The next audience are educators whose students are using a computer based educational tool. For the second audience, the set of tools developed for InVis can provide the most common solutions and approaches demonstrated by students. Furthermore through the use of the Difficulty Detection method, we can present to educators the top ten difficult states in the solution space identified directly from the students' log data. Through the combination of these tools and methods InVis is capable of receiving thousands of transaction logs and transforming them into a meaningful and efficient summary of aspects, common solutions and areas of student difficulty, that educators are interested in.

CHAPTER 6: CONCLUSIONS

Educators and researchers need to gain a better understanding of student problem solving behavior and learning. I addressed this challenge through the design, development and testing of the InVis tool. Computer based educational tools have the potential to improve the amount and rate at which students learn, particularly when teachers can use data to better understand student learning. However, current Interactive Learning Environments (ILE) tools alone are insufficient to allow deep investigation into how students use ILE and similar educational tools. InVis provides a means of investigating student solution attempts and provides insight into how students are solving problems. This leverages an important benefit of ILE systems by visually displaying the interactions students have performed, transforming verbose unreadable interaction logs into a manageable, accessible network of nodes and edges.

6.1 InVis Usability Study

A preliminary version of the InVis tool and the Interaction Network model were developed. Next data from the Deep Thought logic tutor, the BeadLoom Game and other similar tutors were loaded into the tool. This provided us with one of the first uses of an interactive tool built to explore and interact with the problem solution spaces as explored by students. The resulting network, known as the Interaction Network, served as the model for our data. The InVis tool was built to interact with the

Interaction Network and assist users in exploring the data. Following the development of the prototype, we conducted a study of how logic professors navigated, explored and interacted with logic data from the Deep Thought logic tutor. A quantitative task analysis, qualitative feedback, and usability survey provide evidence of which methods of interaction allowed effective investigation of the ILE log data from the Interaction Network. In addition, we provide use cases as well as an in-depth case study where InVis analysis improved the BeadLoom game and ultimately fostered a re-design and re-construction through identification of important errors. We also note that errors in other ILE systems were discovered through the visualization of ILE log data provided by the InVis tool.

InVis and many of the interactive techniques developed proved to be effective at exploring data; one challenge remained in dealing with large datasets. The prototype usability study dataset contained only 18 students. This in turn had relatively few states and actions. It became clear that to gain an understanding of large numbers of student solutions, in the hundreds, new methods and techniques would be necessary to properly filter and focus the users' attention on the important aspects of the Interaction Network. This observation is also supported by visual analytics research.

6.2 Strategy Reduction and Ordering Detection

The prototype and related analysis of its use provided important insight into the development of the current version of InVis. The importance of interactions like predecessor and successor selection as well as the generation of sub-graphs became clear. InVis required new methods to filter and focus the user's attention on the major features of the Interaction Network. To facilitate this, we developed two methods. The first is Strategy Reduction. We recognize a great majority of edges have low student frequency and many a frequency value of one. We used this observation of the Interaction Network to develop a systematic filter to eliminate the less important edges and states based on the student frequency. Next, the remaining states and near-by goals were linked by algorithms for networks like Dijkstra's shortest path to provide a meaningful summary of common behaviors driven by the student-data. We compare our technique of Strategy Reduction to other approaches and conclude our method provides the preferable output.

Although the Strategy Reduction method substantially reduces the number of edges and nodes in the Interaction Network, we recognize there still exist some redundant information. This redundant information remains as an artifact caused by the order of the student-performed actions. To address ordering based redundancy, I developed an ordering detection algorithm that analyzes an Interaction Network and looks for multiple paths between pairs of nodes. The algorithm effectively identifies where student solution paths diverge and merge within the Interaction Network. I subsequently developed and implemented a method of recombining the ordering artifacts into single paths. Although the reduction in nodes and edges to the overall Interaction Network by ordering detection is small, the method's value is the advancement of the resulting Interaction Network to a simple graph.

6.3 Difficulty Detection

Not all students succeed in solving problems and addressing student challenges is an important aspect of education. Though errors are one method of detecting difficulty, this approach requires the tutoring system to log invalid actions applied by students. The challenge then is developing methods of addressing student difficulty which does not depend on error actions. By using expected difficulty from classical test theory, based on failure rates of the entire problem, InVis provides an alternative method of identify difficult states within the Interaction Network. This lets educators easily identify discriminating states, that separate students between success and failure. In addition to this, an absolute difficulty value is also calculated and presented in InVis which takes into consideration the number of students who have suffered failure. Through this view and related tools, users can quickly identify the states which have the highest potential of failure. By addressing these states, the maximum number of students could benefit, particularly if resources limit the number of states an educator can address.

The InVis tool is a platform for new research about student learning and ILE tool use. InVis is not a single use or single method for analyzing a specific data set. We built InVis to provide a means through which researchers can consider, study and conclude a wide range of research questions. InVis should provide new methods of investigation into student learning, as well as new investigations on how to gain insight about how students learn using computer based educational tools.

REFERENCES

[1] V. Aleven, B. M. Mclaren, J. Sewall, and K. R. Koedinger. A new paradigm for intelligent tutoring systems: Example-tracing tutors. *Int. J. Artif. Intell. Ed.*, 19(2):105–154, Apr. 2009.

[2] V. Aleven, B. M. McLaren, J. Sewall, and K. R. Koedinger. A new paradigm for intelligent tutoring systems: Example-tracing tutors. *I. J. Artificial Intelligence in Education*, 19(2):105–154, 2009.

[3] E. Andersen, Y.-E. Liu, E. Apter, F. Boucher-Genesse, and Z. Popović. Gameplay analysis through state projection. In *Proceedings of the Fifth International Conference on the Foundations of Digital Games*, FDG '10, pages 1–8, New York, NY, USA, 2010. ACM.

[4] R. S. Baker, A. T. Corbett, K. R. Koedinger, and A. Z. Wagner. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, CHI '04, pages 383–390, New York, NY, USA, 2004. ACM.

[5] R. S. Baker, A. T. Corbett, I. Roll, and K. R. Koedinger. Developing a generalizable detector of when students game the system. *User Modeling and User-Adapted Interaction*, 18:287–314, August 2008.

[6] R. S. Baker, M. P. Habgood, S. E. Ainsworth, and A. T. Corbett. Modeling the acquisition of fluent skill in educational action games. In *Proceedings of the 11th international conference on User Modeling*, UM '07, pages 17–26, Berlin, Heidelberg, 2007. Springer-Verlag.

[7] T. Barnes and J. Stamper. Toward automatic hint generation for logic proof tutoring using historical student data. *Proceedings of the 9th International Conference on Intelligent Tutoring Systems (ITS 2008)*, pages 373–382, 2008.

[8] T. Barnes and J. Stamper. Toward automatic hint generation for logic proof tutoring using historical student data. In *Proceedings of the 9th international conference on Intelligent Tutoring Systems*, ITS '08, pages 373–382, Berlin, Heidelberg, 2008. Springer-Verlag.

[9] L. L. C. M. Barnes T., Stamper J. A pilot study on logic proof tutoring using hints generated from historical student data. *Proceedings of the 1st International Conference on Educational Data Mining (EDM 2008)*, pages 197–201, 2008.

[10] J. P. Becker and S. Shimada. *The Open-Ended Approach: A New Proposal for Teaching Mathematics.* ERIC, 1997.

[11] D. Ben-Naim, M. Bain, and N. Marcus. A User-Driven and Data-Driven Approach for Supporting Teachers in Reflection and Adaptation of Adaptive Tutorials, 2009.

[12] D. Ben-Naim, N. Marcus, and M. Bain. Visualization and Analysis of Student Interactions in an Exploratory Learning Environment. In *Proceedings of the 1st International Workshop on Intelligent Support for Exploratory Environments (part of ECTEL 2008)*, 2008.

[13] B. S. Bloom. *Taxonomy of Educational Objectives: The Classification of Educational Goals*. Taxonomy of educational objectives: the classification of educational goals. Longman Group, New York, 1956.

[14] A. Boyce and T. Barnes. Beadloom game: using game elements to increase motivation and learning. In *Proceedings of the Fifth International Conference on the Foundations of Digital Games*, FDG '10, pages 25–31, New York, NY, USA, 2010. ACM.

[15] A. K. Boyce, A. Campbell, S. Pickford, D. Culler, and T. Barnes. Experimental evaluation of beadloom game: how adding game elements to an educational tool improves motivation and learning. In *Proceedings of the 16th annual joint conference on Innovation and technology in computer science education*, ITiCSE '11, pages 243–247, New York, NY, USA, 2011. ACM.

[16] J. S. Bruner. Belknap of Harvard UP, Cambridge, MA, 1966.

[17] G. Cobo, D. García-Solórzano, J. A. Morán, E. Santamaría, C. Monzo, and J. Melenchón. Using agglomerative hierarchical clustering to model learner participation profiles in online discussion forums. In *Proceedings of the 2nd International Conference on Learning Analytics and Knowledge*, LAK '12, pages 248–251, New York, NY, USA, 2012. ACM.

[18] A. T. Corbett and J. R. Anderson. Knowledge tracing: Modeling the acquisition of procedural knowledge. *User Modeling and User-Adapted Interaction*, 4:253–278, 1994. 10.1007/BF01099821.

[19] B. Craft and P. Cairns. Beyond guidelines: What can we learn from the visual information seeking mantra? In *Proceedings of the Ninth International Conference on Information Visualisation*, pages 110–118, Washington, DC, USA, 2005. IEEE Computer Society.

[20] M. Croy, T. Barnes, and J. Stamper. Towards an intelligent tutoring system for propositional proof construction. In *Proceeding of the 2008 conference on Current Issues in Computing and Philosophy*, pages 145–155, Amsterdam, The Netherlands, The Netherlands, 2008. IOS Press.

[21] M. J. Croy. Graphic interface design and deductive proof construction. *J. Comput. Math. Sci. Teach.*, 18:371–385, December 1999.

[22] M. J. Croy. Problem solving, working backwards, and graphic proof representation. *Teaching Philosophy*, 23:169–188, 2000.

[23] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[24] P. N. Dixit and G. M. Youngblood. Understanding information observation in interactive 3d environments. In *Proceedings of the 2008 ACM SIGGRAPH symposium on Video games*, Sandbox '08, pages 163–170, New York, NY, USA, 2008. ACM.

[25] M. Eagle, M. Johnson, and T. Barnes. Interaction networks: Generating high level hints based on network community clusterings. In *EDM*, pages 164–167, 2012.

[26] D. Feng, J. Kim, E. Shaw, and E. Hovy. Towards modeling threaded discussions using induced ontology knowledge. *In Proceedings of National Conference on Artificial Intelligence (AAAI-2006)*, 2006.

[27] R. E. Fikes and N. J. Nilsson. Strips: a new approach to the application of theorem proving to problem solving. In *Proceedings of the 2nd international joint conference on Artificial intelligence*, IJCAI'71, pages 608–620, San Francisco, CA, USA, 1971. Morgan Kaufmann Publishers Inc.

[28] L. R. Ford and D. R. Fulkerson. Maximal flow through a network. *Canadian Journal of Mathematics*, 8(3):399–404, 1956.

[29] M. L. Fredman and R. E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *J. ACM*, 34(3):596–615, July 1987.

[30] M. A. Gertner, A. S. Gertner, C. Conati, and K. Vanlehn. Procedural help in andes: Generating hints using a bayesian network student. In *Proceedings of the 15th National Conference on Artificial Intelligence*, pages 106–111. AAAI Press, 1998.

[31] M. Girvan and M. E. J. Newman. Community structure in social and biological networks. *Proceedings of the National Academy of Sciences*, 99(12):7821–7826, June 2002.

[32] N. Hoobler, G. Humphreys, and M. Agrawala. Visualizing competitive behaviors in multi-user virtual environments. In *VIS '04: Proceedings of the conference on Visualization '04*, pages 163–170, Washington, DC, USA, 2004. IEEE Computer Society.

[33] W. Jin, T. Barnes, J. Stamper, M. J. Eagle, M. W. Johnson, and L. Lehmann. Program representation for automatic hint generation for a data-driven novice programming tutor. In *Proceedings of the 11th international conference on Intelligent Tutoring Systems*, ITS'12, pages 304–309, Berlin, Heidelberg, 2012. Springer-Verlag.

[34] D. Keim, F. Mansmann, J. Schneidewind, J. Thomas, and H. Ziegler. Visual analytics: Scope and challenges. In S. Simoff, M. Bhlen, and A. Mazeika, editors, *Visual Data Mining*, volume 4404 of *Lecture Notes in Computer Science*, pages 76–90. Springer Berlin / Heidelberg, 2008.

[35] K. Koedinger, R. Baker, K. Cunningham, A. Skogsholm, B. Leber, and J. Stamper. *A Data Repository for the EDM community: The PSLC DataShop.* Boca Raton, FL: CRC Press, 2010.

[36] K. R. Koedinger, J. R. Anderson, W. H. Hadley, and M. A. Mark. Intelligent Tutoring Goes to School in the Big City. *Proceedings of the 7th World Conference on AIED*, 1995.

[37] K. R. Koedinger and N. Heffernan. Toward a rapid development environment for cognitive tutors. In *in Proceedigns of the International Conference on Artificial Intelligence in Education*, pages 455–457. IOS Press, 2003.

[38] N.-T. Le and W. Menzel. Using constraint-based modelling to describe the solution space of ill-defined problems in logic programming. In *Proceedings of the 6th international conference on Advances in web based learning*, pages 367–379, Berlin, Heidelberg, 2008.

[39] J. R. Lewis. Ibm computer usability satisfaction questionnaires: psychometric evaluation and instructions for use. *Int. J. Hum.-Comput. Interact.*, 7:57–78, January 1995.

[40] R. Mazza and V. Dimitrova. Visualising student tracking data to support instructors in web-based distance education. In *Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters*, WWW Alt. '04, pages 154–161, New York, NY, USA, 2004. ACM.

[41] A. Merceron and K. Yacef. Tada-ed for educational data mining. *Interactive Multimedia Electronic Journal of Computer-Enhanced Learning*, 7(1):267–287, 2005.

[42] M. J. Michael Eagle and T. Barnes. Interaction networks: Generating high level hints based on network community clusterings. In Z. O. H. H. Y. M. Yacef, K. and J. Stamper, editors, *Proceedings of the 5th International Conference on Educational Data Mining*, pages 164–167, 2012.

[43] A. Mitrovic. An intelligent sql tutor on the web. *Int. J. Artif. Intell. Ed.*, 13(2-4):173–197, Apr. 2003.

[44] T. Murray. Authoring intelligent tutoring systems: An analysis of the state of the art. *International Journal of Artificial Intelligence in Education*, 1999.

[45] M. J. Nathan and A. Petrosino. Expert Blind Spot among Preservice Teachers. 40(4):905–928, 2003.

[46] N. A. of Engineering. Grand Challenges for Engineering, 2008.

[47] C. Plaisant. The challenge of information visualization evaluation. In *Proceedings of the working conference on Advanced visual interfaces*, AVI '04, pages 109–116, New York, NY, USA, 2004. ACM.

[48] M. Ringenberg and K. VanLehn. Scaffolding problem solving with annotated, worked-out examples to promote deep learning. In *Intelligent tutoring systems*, pages 625–634. Springer, 2006.

[49] S. Ritter, T. K. Harris, T. Nixon, D. Dickison, R. C. Murray, and B. Towle. Reducing the knowledge tracing space. In *EDM*, pages 151–160, 2009.

[50] B. R. M. B. J. A. . D. T. Rodrigo, M. M. T. Development of a workbench to address the educational data mining. In *Proceedings of the 5th International Conference on Educational Data Mining (EDM 2012)*, pages 152–155, 2012.

[51] M. M. Rodrigo, R. S. Baker, S. D'Mello, M. C. Gonzalez, M. C. Lagud, S. A. Lim, A. F. Macapanpan, S. A. Pascua, J. Q. Santillano, J. O. Sugay, S. Tep, and N. J. Viehland. Comparing learners' affect while using an intelligent tutoring system and a simulation problem solving game. In *Proceedings of the 9th international conference on Intelligent Tutoring Systems*, ITS '08, pages 40–49, Berlin, Heidelberg, 2008. Springer-Verlag.

[52] I. Roll, V. Aleven, B. M. McLaren, E. Ryu, R. S. J. de Baker, and K. R. Koedinger. The help tutor: Does metacognitive feedback improve students' help-seeking actions, skills and learning? In M. Ikeda, K. D. Ashley, and T.-W. Chan, editors, *Intelligent Tutoring Systems*, volume 4053 of *Lecture Notes in Computer Science*, pages 360–369. Springer, 2006.

[53] C. Romero and S. Ventura. Educational data mining: A survey from 1995 to 2005. *Expert Syst. Appl.*, 33:135–146, July 2007.

[54] B. Shneiderman. The eyes have it: A task by data type taxonomy for information visualizations. *Visual Languages, IEEE Symposium on*, 0:336, 1996.

[55] J. Stamper, T. Barnes, L. Lehmann, and M. Croy. The hint factory: Automatic generation of contextualized help for existing computer aided instruction. *Proceedings of the 9th International Conference on Intelligent Tutoring Systems Young Researchers Track*, pages 71–78, 2008.

[56] J. C. Stamper, M. Eagle, T. Barnes, and M. Croy. Experimental evaluation of automatic hint generation for a logic tutor. In *Proceedings of the 15th international conference on Artificial intelligence in education*, AIED'11, pages 345–352, Berlin, Heidelberg, 2011. Springer-Verlag.

[57] L. A. Sudol, K. Rivers, and T. K. Harris. Calculating probabilistic distance to solution in a complex problem solving domain. In K. Yacef, O. R. Zaane,

A. Hershkovitz, M. Yudelson, and J. C. Stamper, editors, *EDM*, pages 144–147. www.educationaldatamining.org, 2012.

[58] D. D. Suthers, H. U. Hoppe, M. de Laat, and S. B. Shum. Connecting levels and methods of analysis in networked learning communities. In *Proceedings of the 2nd International Conference on Learning Analytics and Knowledge*, LAK '12, pages 11–13, New York, NY, USA, 2012. ACM.

[59] M. Tory, I. Lab, and T. Moller. Evaluating visualizations: Do expert reviews work. *IEEE Computer Graphics and Applications*, 25:8–11, 2005.

[60] T. E. Turner, M. A. Macasek, G. Nuzzo-jones, N. T. Heffernan, and K. Koedinger. The assistment builder: A rapid development tool for its. In *In*, pages 929–931. ISO Press, 2005.

[61] S. White, D. Fisher, P. Smyth, S. White, and Y. B. Boey. Analysis and visualization of network data using jung. *Journal Of Statistical Software*, VV(Ii):1–35, 2005.

[62] D. Zapata-Rivera and J. E. Greer. Exploring various guidance mechanisms to support interaction with inspectable learner models. In *Proceedings of the 6th International Conference on Intelligent Tutoring Systems*, ITS '02, pages 442–452, London, UK, UK, 2002. Springer-Verlag.