ABSTRACT

EAGLE, MICHAEL JOHN. Data-Driven Methods for Deriving Insight from Educational Problem Solving Environments. (Under the direction of Tiffany Barnes.)

Intelligent tutoring systems provide personalized feedback to students and improve learning at effect sizes approaching that of human tutors. However, the design and development of these systems is expensive and requires the collaboration of experts across multiple fields. An additional benefit of these educational systems is that they allow collection of detailed records of student actions. However, the complex nature of this interaction rich data makes it difficult to analyze. In previous work, researchers were able to use a past corpus of student data to generate one of the basic features of intelligent tutoring systems, a personalized hint about the next step in a problem. We have used a variety of methods to discover the effectiveness of these automatically generated hints in terms of tutor performance, student motivation, and student problem solving strategies.

We have created a complex network model called an interaction network that represents an empirical sample of student problem solving behavior. We have mined this structure to derive high level student approaches to solving problems. Using these Approach Maps we are able to quantitatively evaluate between-group differences in how student approach problems. Interaction networks are designed handle environments where problems have more than one correct answer and multiple solution paths. To further our understanding of student problem-solving within these open-ended well-structured problems we create interaction networks from several different tutor domains and explore the common properties between them. We find evidence that Interaction Networks are scale-free, indicating that even when the problem state-space seems infinite, students visit a relatively small subset of the total state-space. Finally, we explore methods to estimate the size of the unobserved parts of the Interaction Networks. We use these estimates in order to understand how much data is needed before we can use the networks as samples of student problem-solving.
Data-Driven Methods for Deriving Insight from Educational Problem Solving Environments

by
Michael John Eagle

A dissertation submitted to the Graduate Faculty of North Carolina State University in partial fulfillment of the requirements for the Degree of Doctor of Philosophy

Computer Science

Raleigh, North Carolina 2015

APPROVED BY:

__________________________  _________________________
Nagiza Samatova                     James Lester II

__________________________  _________________________
Eric Wiebe                        Tiffany Barnes
Chair of Advisory Committee
DEDICATION

I would like to dedicate this dissertation to my beloved grandmother Geraldeen Eagle.
BIOGRAPHY

Michael Eagle received a Bachelor of Science (BS) in Computer Science, a Bachelor of Art (BA) in International Studies, as well as a minor in Japanese from The University of North Carolina at Charlotte. Michael studied at Gakushuin University in Tokyo, Japan for 18 months as a Freeman-ASIA and Monbu-kagaku-sho scholarship recipient.

He also completed a Masters of Science (MS) in Computer Science with a concentration in artificial intelligence from The University of North Carolina at Charlotte. Michael transferred to North Carolina State University's Ph.D. program in Fall 2013.

Michael began his research career designing and developing educational games for teaching computer science. He completed a series of studies showing a large positive effect on learning gains in an educational game he designed to teach for-loops and arrays. He became interested in the ways that students were behaving within the game environment that could be discovered by going beyond pre and posttest measures, and transitioned into the field of educational data mining.

Michael received a NSF GRFP Honorable Mention award, and was a GAANN fellow. Michael was also the PI on a NSF EAPSI grant, in which he traveled to Japan and collaborated with Japanese researchers also working in the educational data mining field.

Michael has also had extensive industry experience within the Data Science field. In 2013, Michael worked as a Data Science intern at Warner Bros Games’ division Turbine Inc. for nine months working on games such as Infinite Crisis, The Lord of the Rings Online, and Dungeons and Dragon's Online. In summer of 2015, Michael worked with Blizzard Entertainment as a Data Scientist as part of their summer internship program. While there he worked on data from games such as World of Warcraft and Heroes of the Storm.

After completion of his work at North Carolina State University, Michael will start a postdoctoral research position at Carnegie Mellon University within the Human-Computer Interaction Institute.
ACKNOWLEDGEMENTS

Over the past seven years I have received support and encouragement from a great number of individuals. My first experience with research began as an independent study and summer Research Experiences for Undergraduates (REU) under the direction of Dr. Tiffany Barnes. It was these experiences, as well as Tiffany's encouragement that I choose to pursue a graduate level degree. Tiffany has been a wonderful mentor, colleague, and friend. I would also like to thank my dissertation committee Nagiza Samatova, James Lester II, and Eric Wiebe for their support.

Thanks to my labmates for their input and feedback over the years. Many late nights were spent working in the lab on publications with Matt Johnson. Drew Hicks and Acey Boyce spent many hours proofreading and providing feedback on my research. John Stamper, Marvin Croy, and Behrooz Mostafavi all helped with development of the Deep Thought tutor and shared data. I have also received financial support from several sources, the UNC school system, the Department of Education, and the National Science Foundation.

I am grateful for the support I have received from my family and friends over the years. Last, but certainly not least, I would like to thank my wife for her constant support and the countless hours of listening to me talk about research.
# TABLE OF CONTENTS

LIST OF TABLES ................................................................. vii

LIST OF FIGURES ............................................................... viii

Chapter 1  Introduction ...................................................... 1
  1.1  Motivation ............................................................ 2
  1.2  Previous Work ........................................................ 4
    1.2.1  The Deep Thought Logic Tutor ................................. 6
    1.2.2  2009 Deep Thought Study ................................... 6
  1.3  Organization .......................................................... 8
    1.3.1  Chapter 2: Effects of Automatically Generated Next-Step Hints on Student Time-in-Tutor and Retention ......................... 8
    1.3.2  Chapter 3: Exploring Differences in Problem Solving with Data-Driven Approach Maps .................................................. 11
    1.3.3  Chapter 4: Exploring Networks of Problem-Solving Interactions ................................................................. 13
    1.3.4  Chapter 5: Interaction Network Estimation: Predicting Problem-Solving Diversity in Interactive Environments ..................... 14

Chapter 2  Survival Analysis on Duration Data in Intelligent Tutors .......... 17
  2.1  Introduction .......................................................... 17
  2.1.1  Methods and Materials .......................................... 19
  2.1.2  Survival Analysis .................................................. 20
  2.2  Results .............................................................. 22
  2.3  Discussion ............................................................ 24
  2.4  Conclusions .......................................................... 27

Chapter 3  Exploring Differences in Problem Solving with Data-Driven Approach Maps .................................................. 28
  3.1  Introduction .......................................................... 28
  3.2  Related Work ........................................................ 30
  3.3  Methods .............................................................. 31
    3.3.1  Constructing an Interaction Network .......................... 31
    3.3.2  Extracting Regions .............................................. 32
  3.4  Results & Discussion ................................................ 34
    3.4.1  Problem 1.4 ...................................................... 36
    3.4.2  Problem 1.5 ...................................................... 40
    3.4.3  Working Backwards and Trailblazing .......................... 41
    3.4.4  Conclusions ..................................................... 42
LIST OF TABLES

Table 1.1  Number of students that continued or dropped out of the tutor after L1 7
Table 1.2  The effect sizes of the differences between hint group and control group for completion and attempt rates by level. 7

Table 3.1  Summary of Approach Maps for 11 Deep Thought tutor problems. An asterisk (*) indicates problems where the hint group had access to hints. 35
Table 3.2  Detailed information on the regions in the 1.4 Approach Map shown in Figure 3.1. 38
Table 3.3  Number and depth of hints used by the hint group in each region; PS=Problem Start. 38
Table 3.4  Detailed information on the regions in the 1.5 Approach Map shown in Figure 3.3. 41

Table 4.1  Averages across the datasets, $dr$ is the directed assortativity, $r$ is the undirected assortativity, and $S(g)$ is the scale-free metric. 55
Table 4.2  Interaction Networks generated from problems in the 2009 Hint Factory dataset [SEBC13]. Alpha is the exponent of the fitted power-law distribution, logLik is the log-likelihood of the fit with xmin, KS is the Kolmogorov-Smirnov test statistic (smaller score indicates better fit), $p$ is the p-value of the KS test, values less than 0.05 reject a power-law fit. 56
Table 4.3  Averages across all Interaction Networks generated from problems in the B0Ts(Code and World) and DT(DT1 and DT3). Variables are the same as those in Table 4.2. 56
Table 4.4  Example Hint Template which translates the elements of an Action into 4 levels of hints. 58
Table 4.5  Actions Available in the BLG 61
Table 4.6  Example Play Sequence 63

Table 5.1  Dataset summary: the total number of students in the dataset, the number of distinct problems, and the average number of students represented in each network. 72
Table 5.2  Different populations have different spread in problem exploration. 82
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>This example shows two steps within the Deep Thought tutor. First, the student has selected ( Z \land \neg W ) and performed Simplification (SIMP) to derive ( \neg W ). Second, the student selects ( X \lor S ) and performs backward Addition to derive ( S ).</td>
<td>7</td>
</tr>
<tr>
<td>1.2</td>
<td>Attempt and complete rates per level, *indicates a problem where the hint group was given access to automatically generated hints.</td>
<td>8</td>
</tr>
<tr>
<td>2.1</td>
<td>QQ-Plots for the log-normal and Weibull distributions, the primary difference appears to be that the Log-normal is sensitive to very small durations, while the Weibull distribution is sensitive to very large durations.</td>
<td>20</td>
</tr>
<tr>
<td>2.2</td>
<td>Histogram with density plots for the Weibull and log-normal distributions. Both seem to fit reasonably well.</td>
<td>21</td>
</tr>
<tr>
<td>2.3</td>
<td>The probability density functions, represented by the dashed lines, provide the probability of observing tutor completion at a specific time. The hazard functions, the solid lines, are the probability of observing tutor completion at a specific time, given that it has not occurred yet. The probability of completion grows rapidly before becoming stable and eventually decreasing.</td>
<td>25</td>
</tr>
<tr>
<td>2.4</td>
<td>The percent of students who have completed the tutor over time. We have added points with the mean tutor score (max of 13) to each curve at the 20%, 40%, 60%, and 80% quantiles of total duration.</td>
<td>26</td>
</tr>
<tr>
<td>3.1</td>
<td>The Approach Map for problem 1.4. Edges and vertexes can be read as the number of students who performed action(s) to derive proposition(s). Three main approaches are revealed, with the hint group strongly preferring to work the problem forwards. The control group often attempts to solve the problem by wards with addition, there are no goals along this path. More detail is given in Table 3.2.</td>
<td>37</td>
</tr>
<tr>
<td>3.2</td>
<td>The most common attempt paths for each of the main approaches in the approach map for problem 1.4 (figure 3.1.) The highlighted nodes represent unjustified propositions.</td>
<td>39</td>
</tr>
<tr>
<td>3.3</td>
<td>Even in the absence of the automatically generated hints, the hint group still prefers a forward solution. The control group explores regions that do not lead to goals. Details are given in Table 3.4.</td>
<td>40</td>
</tr>
</tbody>
</table>
Figure 4.1  An example Interaction Network. The vertices represent tutor states, edges represent actions that go from state to state, edge thickness is weighted by frequency, goal states are green, and vertices with high attrition rates have red boarders.

Figure 4.2  Example of a problem-space. The black states represent parts of the problem space are are not visited by human walkers. The blue and orange states represent the states observed from two different populations of human walkers.

Figure 4.3  Interesting problems must have meaningful choices. No matter what action the student takes they move from the start state (white) toward the goal state (green), the rule-using problems we want to focus on allow divergence.

Figure 4.4  The vertices represent a convexity, or 'bubble', which is a region within an Interaction Network where the problem expands from one state and then later converges to a single state. If there are multiple paths to get to the goals, and they can cross over, these types of structures should exist in the network.

Figure 4.5  Observed networks rarely have complete bubbles, the grey edges and vertices represent places that the either we have yet to observe a student visiting or that it cannot be reached.

Figure 4.6  An example of an assortative network (right) where vertices connect to vertices with similar degree, and a dissortative network (left) where vertices share edges with vertices of different degree.

Figure 4.7  The in-degree vs. out-degree ratio can reveal information about the complexity of problems. Degree assortativity is the tendency for vertices to share edges with vertices with a similar degree.

Figure 4.8  The Approach Map for problem 2.4. Edges and vertices can be read as the number of students who performed action(s) to derive proposition(s). Several approaches are revealed, with the hint group strongly preferring to work the problem forwards. The control group is more likely to try approaches from which there are no goal paths.

Figure 4.9  Example problem state in the Deep Thought logic tutor.

Figure 4.10  The BeadLoom Game Interface: the goal of the game is to create the pattern on the left in 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.

Figure 4.11  An example problem in BOTS. The goal is to use commands to move the red robot, carrying boxes to their destinations. The players are scored on the number of commands used, with lower scores representing better, more efficient solutions.
Figure 5.1 Example of state to state transitions within the Deep Thought (DT1) propositional logic tutoring system. .................................................. 73
Figure 5.2 The growth of new states as new students are added for each problem, for each dataset. .................................................. 78
Figure 5.3 The average absolute error between the estimated number of new states and the observed new states over the number of students for all problems in each of the four datasets. $P_0$ accurately predicts the observed values after roughly 10 students, rarely being off by more than one after that. .................................................. 79
Figure 5.4 For the hintable states, the average difference between the estimated number of new states and the observed new states over the number of students for all problems in each of the four datasets. $P_0$ accurately predicts the observed values after roughly 10 students, rarely being off by more than one after that. .................................................. 80
Figure 5.5 The estimated network coverage $I_C$ for each of the 5 datasets, note the poor coverage for the BOTS-C dataset. The BOTS-W state is more general and has the much higher coverage. .......................... 81
Figure 5.6 An image of the main gameplay interface for BOTS. The left hand side of the screen shows the user’s program, used to derive code states. The right-hand side shows the game world, where the program output determines the world states. .................................................. 82
Figure 5.7 For the hintable network: the estimated network coverage $I_C$ for each of the 5 datasets. Even the lowest performing hint network BOTS-C reaches roughly 70% coverage by 100 students. .......................... 83
Figure 5.8 Prediction of total final number of states, as observed number of states increases. Note that for small $t$, the estimate is very high (up to 300% over prediction), but becomes fairly accurate after roughly 20% of the sample is measured. .................................................. 84
Figure 5.9 Prediction of total final number of goal states, as observed number of states increases. Note that for small $t$, the estimate is very high, but becomes an underestimate as $t$ increases. $P_0$ can predict the number of additional hintable states that can be added for a additional sample of data. .................................................. 85
Chapter 1

Introduction

The high level goal of this work is to use data-driven methods to understand how people solve problems within problem solving environments, as well as to evaluate the differences in the ways that people solve problems, and finally to use this understanding to change problem-solving behavior.

In this work we take a Educational Data Science approach, using a variety of analytic methods and techniques on user log data collected from educational computer programs. Data Science is the extraction of knowledge and insight from large volumes of structured and unstructured data [Dha13]. This is a relatively new field which combines data mining, predictive analytics, exploratory data analysis, data visualization, inferential statistics, as well as concepts such as big data and scalability. Educational Data Mining is also an emerging discipline, which focuses on deriving knowledge about human learning and educational settings from the increasingly large-scale data collected from learning environments [Soc14].

These fields are intertwined with big data and scalability. Since the 1980s the world has roughly doubled its ability to store data every 40 months [HL11]. Big data is a general term concerned with storing and manipulating “large” sets of data, where the definition of a “large” set of data is constantly moving. Scalability is capability of a system to handle a growing amount of work or its potential to be expanded [Bon00]. In this work we are more concerned with the scalability of developing personalized educational content, as well as ensuring that educators can manage students who are increasingly receiving education in educational computer programs. Our focus is on deriving insights into student problem solving behavior from the logs of their work in educational environments.
The primary contributions of this work are new methods to explore and evaluate: 1) the effects of automatically generated hints on student behavior (tutor performance and learning); 2) duration data in problem solving environments; 3) higher-level approaches to problems; 4) differences in problem solving approaches between groups; and 5) how features of a complex network can be used to characterize user interactions in computer-based tutors.

### 1.1 Motivation

One-to-one tutoring has a dramatic effect on student learning. Bloom famously found that students given one-to-one tutoring, by human tutors expert in both the content and in pedagogy, performed two standard deviations better than students given traditional instruction \[\text{Blo84}\]. This effect brought the average student in the tutor-group to above 98% of the control group \[\text{Blo84}\] in terms of performance. It also greatly lowered the variance within the tutor-group, meaning that there were less individual performance differences within the same class, which means that students that would have otherwise been considered *low performers* were now performing the same as their peers \[\text{Blo84}\]. However, the cost of one-to-one tutoring for all of the students is too high, and Bloom challenged researchers to find methods of group instruction that are as effective as one-to-one tutoring, also known as the 2-Sigma Problem \[\text{Blo84}\].

The market for supplementary education is estimated to be about $13.1 Billion USD in 2012 \[\text{Adv12}\]. One example of the cost of supplemental educational services (SES), as part of a provision of No Child Left Behind law, sees averages hourly rates of $46 USD \[\text{BSD07}\], and this is not necessarily one-to-one tutoring. The high cost associated with private tutoring and other educational materials clearly raises concerns about equal access based on family income. In Korea, where the private tutoring market is even larger at $19 billion USD in 2008, Choi found that family income affected student access to tutoring as well as academic performance \[\text{Cho12}\].

Intelligent tutoring systems are computer based educational systems that attempt to act in the role of a tutor. VanLehn compared human tutoring, computer tutoring, and no tutoring and found that the computer tutor had a similar effect size to the human tutor\[\text{Van11}\]. Early work by Anderson et al. with the LISP tutor found that while the human-tutor was still better, the LISP tutor was not far behind \[\text{AR85}\] and that tutors were likely
to continue getting better. Koedinger et al. showed that the PUMP algebra tutor improved student performance on standardized tests by 15% [KAH97]. Corbett argued that intelligent tutors have already closed this gap, and have perhaps surpassed human tutors [Cor01] given the results of a meta-analysis of several successful tutoring systems.

However, the development of intelligent tutoring systems is also expensive, one estimate is that each hour or tutoring content requires as much as 100–1000 hours of work [Mur99]. The work involved also requires experts in the fields such as cognitive psychology, educational psychology, computer science, cognitive science, and instructional design. Another potential weakness is that intelligent tutoring systems and other computer-aided instructional environments make it difficult for instructors to track how their students are solving problems within the system, this type of detachment can lead to a decreased sense of control and could effect adoption of the tutor [Sel07]. However, once the system is developed the implication is that personalized tutoring can be widely disseminated and made available at reasonable costs [Woo10, Cor01].

The large educational benefits of private tutoring can be matched by intelligent tutoring systems and these computer-based tutors can make the wide scale adaptation of personalized education economically viable. However, intelligent tutors are difficult and expensive to design and develop. The National Academy of Engineering cited Personalized Learning as one of the 14 grand challenges for engineering [OE14].

The Cognitive Tutor Authoring Tools (CTAT) attempted to help improve the efficiency of tutor development by providing a set of authoring tools, as well as simplifying the tutor creation process [AMSK06]. CTAT was able to improve development time by 1.4 to 2 times [AMSK06]. However, this is still a significant amount of time and still requires the use of experts across many fields. CTAT also places constraints on the tools and systems which the authors must use and is designed for building new tutors, but does not provide support for already existing instructional tools.

One major benefit of computer-based tutoring methods is the ability to measure many aspects of how students work within tutoring environments. This has paved the way for large databases of transactional logs of student’s work within tutoring systems [KBC10]. The field of Educational Data Mining (EDM) has grown from this abundance of data and used to understand and evaluate data from intelligent tutoring systems, as well as inform educational system design [BY09].
1.2 Previous Work

In 2008 Barnes and Stamper approached the tutor development problem by taking a computer-aided instructional program and using previously collected student data to automatically generate the basic features of an intelligent tutoring program [BS08a]. Barnes and Stamper used a machine learning approach and modeled previous student work as a Markov Decision Process (MDP); they then used this MDP to create a best next-step policy, which allowed them to provide a new student with a next-step hint [BS10]. A pilot study using this technique showed that students used the system, and that the automatic method of providing hints was able to provide hints 91% of the time they were requested [SBLC08b]. Barnes and Stamper also found that visualization of the MDPs themselves revealed surprising insights into how the student’s solved the problems [BS08a]. The MDP method of generating next-step hints, also called Hint Factory, has been applied across domains (linked list tutor, logic tutor, and programming game) [FDEO+09, PIHB14, EJBB13], and been shown to increase student retention in tutors [SEBC13].

In section 1.2.2 we summarize the results from Stamper, Eagle, and Barnes’ 2009 study on automatically generated hints [SEBC13]. In Chapter 2 we provide a followup analysis on student time-in-tutor vs. completion rates and show that the automatically generated hints reduce the time needed to complete the tutor by almost 50/.

The Hint Factory approach provided feedback in an environment without high levels of problem scaffolding, a common feature of most intelligent tutoring systems [Van06]. The typical tutoring system is designed by carefully breaking down the problem domain into knowledge components, basic units of knowledge, and scaffolding the problems such that each step of the problem uses only a single knowledge component [Van06]. Doing this allows the researcher to develop student models using systems such as ACT-R (as seen in the early LISP tutors,) [ACKP95] or later generalizations such as Bayesian Knowledge Tracing [CA94]. However, it is often necessary to strongly restrict the presentation of the problems to use these techniques, which makes it difficult to incorporate in already existing systems.

Section 1.2.2 summarizes the results from Stamper, Eagle, and Barnes’ 2009 evaluation on the Hint Factory method of automatically generating next-step hints [SEBC13]; section 1.2.1 is a description of the Deep Thought tutor that was used in this study. These two sections are included to provide context for the results of Chapter 2 and Chapter 3. Section 2
expands on the results, and offers a more detailed view into how the automatically generated hints affected student's time within the tutor.

Eagle and Barnes abstracted the MDP domain model into a complex network representation of the student-tutor interactions called an Interaction Network [EJB12]. These networks worked well as visualizations of student work within tutors and Johnson et al. successfully created a visualization tool InVis specifically to aid instructors in understanding student-tutor interaction data [JEB13]. Interaction Networks have been used for the development of data-driven mastery learning [EB12]. One of the contributions of this work is that analysis of the Interaction Networks with network mining techniques allowed us to derive useful sub-regions which represented diverse student approaches to solving problems [EB14a] (see Chapter 3).

Chapter 4 presents work on furthering the understanding of Interaction Networks and their interpretation. Specifically, we address treating the Interaction Networks as samples from a population. We also explore the types of educational environments where Interaction Networks work best by exploring the overall shape and distribution of states and connectivity within the network. We present evidence that Interaction Networks are scale-free networks, that is, the vertex connectivity frequency distribution follows a power-law. We also make further use of network invariants such as assortativity to describe the high-level characteristics of the network. We argue that these properties allow Interaction Networks to work across a wide variety of instructional environments, even when the potential problem-space might seem intractable.

Finally, chapter 5 will explore methods to determine how much of an Interaction Network we have not yet observed, how much data we need to collect for a good representation of the network, and how we evaluate state definitions. We introduce a novel method of estimating the size of the unobserved Interaction Network from a sample by leveraging Good-Turing frequency estimation. We use this estimation to predict size, growth, and overlap of Interaction Networks using a small sample of student data. Our estimate is accurate using data from as few as 10–30 students and is a good predictor for the growth of the observed state space for the full network, as well as the subset of the network which is usable for automatic hint generation.
1.2.1 The Deep Thought Logic Tutor

In Deep Thought propositional logic tutor problems, students apply logic rules to prove a given conclusion using a given set of premises. Deep Thought allows students to work both forward and backwards to solve logic problems [Cro00]. Working backwards allows a student to propose ways the conclusion could be reached. For example, given the conclusion $B$, the student could propose that $B$ was derived using Modus Ponens (MP) on two new, unjustified (i.e. not yet proven) propositions: $A \rightarrow B, A$. This is like a conditional proof in that, if the student can justify $A \rightarrow B$ and $A$, then the proof is solved. At any time, the student can work backwards from any unjustified components (marked with a ?), or forwards from any derived statements or the premises. Figure 1.1 contains an example of working forwards and backwards with in Deep Thought.

1.2.2 2009 Deep Thought Study

This section provides detail on an important dataset for this document. The author designed the second half of this study, and performed the evaluation. In 2013, Stamper, Eagle, and Barnes studied the effect of data-driven hints [SEBC13] using the Spring and Fall 2009 Deep Thought propositional logic tutor dataset [SEBC11]. Data was collected from six 2009 deductive logic courses, taught by three professors. Each instructor taught one class using Deep Thought with automatically-generated hints on half of the problems (hint group, $n=105$) and one without access to hints on any problems (control, $n=98$). Students from the 6 sections were assigned 13 logic proofs in Deep Thought as a series of three graded homework assignments, with problems L1: 1.1-1.6, L2: 2.1-2.5, and L3: 3.1-3.2.

Table 1.1 shows retention information for each group after level L1; a $\chi^2$ test of the relationship between group and dropout produced $\chi^2(1) = 11.05$, which was statistically significant at $p = 0.001$. The hint group completed more problems, with the effect sizes for these differences shown in Table 1.2. Stamper et al. found that the odds of a student in the control group dropping out of the tutor were 3.6 times more likely when compared to the group provided with automatically generated hints [SEBC13].

Figure 1.2 charts the attempt and completion rates for hint group and control group for each problem in Deep Thought. Both groups had similar problem attempt rates, shown using solid lines, for L1 (1.1-1.5), but the hint group had significantly higher attempt rates in L2 and L3. The completion rates for each group are shown with dashed lines in Figure 1.2.
Figure 1.1: This example shows two steps within the Deep Thought tutor. First, the student has selected $Z \land \neg W$ and performed Simplification (SIMP) to derive $\neg W$. Second, the student selects $X \lor S$ and performs backward Addition to derive $S$.

Table 1.1: Number of students that continued or dropped out of the tutor after L1

<table>
<thead>
<tr>
<th>Group</th>
<th>Total</th>
<th># Continued</th>
<th># Dropped</th>
<th>% Dropped</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hint</td>
<td>105</td>
<td>95</td>
<td>10</td>
<td>9%</td>
</tr>
<tr>
<td>Control</td>
<td>98</td>
<td>71</td>
<td>27</td>
<td>28%</td>
</tr>
<tr>
<td>Total</td>
<td>203</td>
<td>166</td>
<td>37</td>
<td>18%</td>
</tr>
</tbody>
</table>

Table 1.2: The effect sizes of the differences between hint group and control group for completion and attempt rates by level.

<table>
<thead>
<tr>
<th></th>
<th>L1</th>
<th>L2</th>
<th>L3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Completed</td>
<td>$d = 0.51^*$</td>
<td>$d = 0.64^*$</td>
<td>$d = 0.39^*$</td>
</tr>
<tr>
<td>Attempted</td>
<td>$d = 0.27$</td>
<td>$d = 0.44^*$</td>
<td>$d = 0.33^*$</td>
</tr>
</tbody>
</table>
Note that, after problem 1.4, the differences in attempt rates and completion rates seem to diverge between the groups.

Figure 1.2: Attempt and complete rates per level, *indicates a problem where the hint group was given access to automatically generated hints.

1.3 Organization

Below we have listed each chapter and its problem statement, research questions, our approach, and our conclusions.

1.3.1 Chapter 2: Effects of Automatically Generated Next-Step Hints on Student Time-in-Tutor and Retention

The primary contributions of this chapter are new methods to explore and evaluate: the effects of automatically generated hints on student behavior (tutor performance and learning); and duration data in problem solving environments.
Problem

Intelligent tutoring systems have sizable effects on student learning efficiency — spending less time to achieve equal or better performance. In a classic example, students who used the LISP tutor spent 30% less time and performed 43% better on posttests when compared to a self-study condition [AR85]. While this result is quite famous, few papers have focused on differences between tutor interventions in terms of the total time needed by students to complete the tutor. In many studies of intelligent tutoring systems, time is simply held constant for two groups, and efficiency then boils down to comparing the number of problems each group could solve in the given time and the results of posttest measures. However, it is not clear how to factor students who were not able to complete the tutor into this analysis. Typical time duration distributions violate the normality assumptions of many statistical tests and measures of central tendency. Anderson, Corbett, Koedinger, and Pelletier used mean duration data to compare differences between groups of students with and without intelligent feedback in the LISP tutor [ACKP95]. The authors state that the mean times (for the control group) are underestimates, as many students in the control (no-feedback group) did not complete all assignments. In other words, if the control group persisted, the time they took to complete tasks would have been longer than the observed durations for the few high-performing students who were able to persist without feedback.

Different dropout rates between experimental groups can cause attrition bias [MH07], where groups completing the study are self-selected due to achievement levels; this self-selection causes the sample to become different than the target population and hampers the study’s generalizability [MEHB97]. When dropout exists, more complex analyses are needed to study learning efficiency; not only are results suspect for generalization purposes, but the data itself contains missing values because of high dropout rates.

Research Questions

What was the effect of automatically generated next-step hints on student’s time-in-tutor? What was the difference in time-to-complete between-groups? How is total time-in-tutor distributed?
Approach

We investigate data from a prior study of the Deep Thought logic tutor comparing versions with and without hints. Stamper et al. found that the odds of a student in the control group dropping out of the tutor after the first six problems were over 3.6 times higher when compared to the group provided with (data-driven and automatically generated) hints [SEBC13]. Students given access to hints also had better tutor performance, as well as higher overall course scores. However, comparison of duration means showed no differences in overall time spent in the Deep Thought logic tutor between the hint and control groups. This is likely because this comparison does not take into account student dropout. In this study, we applied survival analysis to data from Stamper et al.’s study to more fully explore the impact of hints on performance, duration, and dropout.

Our exploration of tutor efficiency has three important elements: performance (tutor completion percentage), duration (total time spent interacting with the tutor), and dropout (whether stopped before completion). Dropout can easily confound the results of duration and performance. By modeling tutor data with high dropout rates using survival analysis, we hypothesize that we can build a more detailed understanding of tutor efficiency and explain differences between groups in an educational intervention.

Contribution

In this work, we used survival analysis to re-analyze the data from six 2009 logic courses using the Deep Thought logic tutor both with and without hints. The original study (section 1.2.2) showed that students without hints were over 3.6 times more likely to drop after the first six problems when compared to students offered hints. However, standard analyses were insufficient to show the impact of hints on the time needed to complete the tutor between the two groups. Using survival analysis, we have been able to estimate the total duration for both hint and control groups while taking into account dropout data, showing that students in the hint group take 55% of the time to complete the tutor than students in the control group. Using these estimates, we were able to explain approximate time per problem in the tutor for each group. This analysis sheds light on the probable reasons for dropout in the control group. Without these analyses, we might have concluded that students in the control group gave up sooner or were not persistent. However, in reality we see that these students are in fact persistent and spend a considerable amount of time in
the tutor - equal to the amount of time spent in the tutor by the hint group. The difference is
tutor efficiency: students in the hint group performed more efficiently, and were therefore
able to complete the tutor, while the control group spent a similar amount of time but was
less likely to be able to finish. This is a much richer understanding of the differences in
effects between the two groups than traditional methods provide. The survival function
also allows us to make predictions on how much time is needed for tutor completion, both
for teacher planning and student feedback. These results suggest that survival analysis is a
powerful toolbox for investigating the impact of interventions on learning efficiency while
accounting for performance, duration, and dropout.

1.3.2 Chapter 3: Exploring Differences in Problem Solving with Data-
Driven Approach Maps

The primary contributions of this chapter are new methods to explore and evaluate higher-
level approaches to problems and differences in problem solving approaches between
groups.

Problem

Procedural problem solving is an important skill in STEM (science, technology, engineering,
and math) fields. Open-ended procedural problem solving, where steps are well-defined,
but can be combined in many ways, can encourage higher-level learning [Blo56]. However,
understanding learning in open-ended problems, particularly when students choose
whether or not to perform them, can be challenging. The rich interaction data saved by
transactional tutor logs offers many avenues to explore and understand student problem
solving data, particularly for problems with multiple solutions. Student individual differ-
ences affect the ways that students solve problems [Jon00]. However, it is difficult to examine
the overall approaches that groups of students demonstrate during problem-solving. While
pre and posttests are useful for measuring the change in behavior before and after an
experimental treatment, we are interested in studying not only whether a student can solve
a problem, but how they are solving the problem. In this study, we use Interaction Networks
of student behaviors to investigate how providing hints affects student problem-solving
approaches.
Research Questions

In what way does the availability of automatically generated next-step hints effect the way that students solve problems within the tutor? How can we get both a qualitative and quantitative measure of the differences between two groups in terms of how they approach a problem? Can we break the network up into high-level regions, which can summarize the different approaches to a problem? Do the differences in problem-solving persist to problems even when each group is not given access to next-step hints?

Approach

By mapping Deep Thought transactional data into an Interaction Network, and applying graph mining to derive regions based on the structure of this network, we develop a new Approach Map that illustrates the approaches that groups of students take in solving logic problems. We built Approach Maps for all 13 problems in the tutor, and illustrate a detailed analysis of two of these maps to explore the differences in problem solving between the hint and control groups. We use a two-tailed chi-squared test to look for differences between the hint and control groups in how they visit regions in the Approach Map. The null hypothesis is that there is no difference in the frequency of entering a particular region between attempts in the hint group and the control group. The alternative hypothesis is that the groups enter regions with different than expected frequency. We use Bonferroni correction [Sha95] to compensate for the number of tests that we run. We then compare between-group approaches to problems in which one group had hints and the other did not, as well as problems in which neither group was given access to hints.

Contribution

Approach Maps are novel representations of student-tutor interaction data that allows for the comparison of problem-solving approaches on open-ended logic problems. The Approach Map visualization results in a significant reduction in the space needed to describe a large amount of student-tutor data. It does this by reducing the student attempts into regions that we can consider as higher-level approaches to problem-solving. Deep Thought problems each had an average of 330 solution attempts, which were made up of about 6.5 thousand interactions. Using our Approach Maps, we partition problems into about 15 regions each.
Approach Maps annotated with frequencies of visits by two groups to identify regions where a particular study group was over-represented allowed us to examine the approaches each group took to solving each proof. As we predicted, the automatically generated hints seemed to direct the students in the hint group down a common path, and we were able to detect this with the Approach Maps. Interestingly, even in problems where neither group had hints, the hint group still showed a preference for better approaches, providing some evidence for a persistent effect of the hints. Analyzing Approach Maps also facilitated another important discovery that control group tended enter and remain in unproductive (or buggy) regions. These observed differences help explain how the automatically-generated hints produced the difference in tutor performance and retention in the 2009 Deep Thought study. Our investigations suggest that the patterns of behavior exhibited by students do result in meaningful regions of the solution attempt search space.

1.3.3 Chapter 4: Exploring Networks of Problem-Solving Interactions

The primary contributions of this chapter are new methods to explore and evaluate how the features of a complex network can be used to characterize user interactions in computer-based tutors.

Problem

Problem solving is an important skill across many fields, including science, technology, engineering, and math (STEM). Working open-ended problems may encourage learning in higher ‘levels’ of cognitive domains [Blo56]. We have developed Interaction Networks to represent student problem solving in educational environments. In this chapter we explore the properties of Interaction Networks and the types of educational environments in which they work best.

Research Questions

What are the properties of the Interaction Network, that make it possible to create automated feedback and approach maps etc; what can these properties tell us about how students solve problems; and are there common Interaction Network features across domains?
Approach

We explore our theory on what types of problem solving environments Interaction Networks are best suited for. We look into complex network metrics of degree distribution to explore how network connectivity is distributed. We also explore the self similarity, or scale-free, nature of Interaction Networks. We compare results across data from a logic tutor, as well as a educational game for teaching programming.

Contribution

In this chapter we explore student-tutor Interaction Networks, empirical samples of student-walks though a problem-space modeled as a complex network. Interaction Networks were designed specifically for tutors with problem-solving tasks in which there are many goals and many paths to those goals, and that the user moves along those paths by using a set of actions. We explore Interaction Networks from multiple datasets and find that they exhibit scale-free properties. We find that Interaction Networks from different tutors share similarities in scale-free metrics and find that global and local vertex degree assortativity provide insight into the nature of the problem solving environments.

We present success stories from a variety of tutoring systems and argue that Interaction Networks are useful for a wide variety of problem solving instructional environments, even when the potential problem-space might look to be intractable. The scale-free properties of the network make it possible to discover the important regions of a network even with a small sample of student data. We expect that Interaction Networks are a good representation for describing a large amount of student data; provide a common language for performing cross tutor evaluation of problem solving environments; and that the intuitive nature of the model helps convey results from the data in a way that is interpretable for both research and instructor.

1.3.4 Chapter 5: Interaction Network Estimation: Predicting Problem-Solving Diversity in Interactive Environments.

The primary contributions of this work are new methods to explore and evaluate how the features of a complex network can be used to characterize user interactions in computer-based tutors.
Problem

Data-driven methods to provide automatic hints have the potential to substantially reduce the cost associated with developing tutors with personalized feedback. Modeling the student-tutor interactions as a complex network provides a platform for researchers to automatically generate next step hints. An *Interaction Network* is a complex network representation of all observed student and tutor interactions for a given problem in a game or tutoring system. In addition to their usefulness for automatically generating hints, Interaction Networks can provide an overview of student problem-solving approaches for a given problem.

Data-driven approaches cannot reliably produce feedback until sufficient data has been collected, a problem often referred to as the Cold Start problem. The precise amount of data needed varies by problem and environment. However, some properties of Interaction Networks allow us to estimate how much data is needed. Eagle et al. explored the structure of these student Interaction Networks and argued that networks could be interpreted as an empirical sample of student problem solving [EHIB15]. Students employing similar problem-solving approaches will explore overlapping areas of the Interaction Network. The more similar a group of students is, the smaller the overall explored area of the Interaction Network will ultimately be. Since we expect different populations of students to have different Interaction Networks, and different domains to require varying amounts of student data before feedback can be given, good metrics for the current and predicted quality of Interaction Networks are important.

Research Questions

How much data is needed to provide hints for a new problem? How much do different populations of students overlap in their solution attempts? How can we compare different state representations?

Proposed Approach

In this work, we adapt Good-Turing frequency estimation to interaction level data to predict the size, growth, and “hintability” of Interaction Networks. Good-Turing frequency estimation estimates the probability of encountering an object of a hitherto unseen type, given the current number and frequency of observed objects [GS95]. It was originally developed
by Alan Turing and his assistant I. J. Good for use in cryptography efforts during World War II. We will adapt this method for observed and unobserved tutor states.

**Contribution**

We have adapted Good-Turing frequency estimation for use with networks built from student-tutor interactions. We found that the estimator for the missing proportion of the network $P_0$ was accurate in predicting the number of new states discovered with new data. We also found that we could accurately measure network coverage with $I_C$ for both the regular network, as well as the network of hintable states. This provides us with a metric to compare different state representations as well as determine the suitability of Interaction Network methods to different tutoring environments. We were also able to use these metrics to provide accurate predictions for the size of networks expected given more data samples, which will be useful for predicting the amount of additional data needed to provide a desired amount of hintable network coverage. Additionally, we were able to use these metrics to compare different state representations. Finally, we used the estimate of network coverage to compare different student populations to show that the addition of hints in one environment had an effect on the number of states explored by students.
Chapter 2

Survival Analysis on Duration Data in Intelligent Tutors

Effects such as student dropout and the non-normal distribution of duration data confound the exploration of tutor efficiency, time-in-tutor vs. tutor performance, in intelligent tutors. We use an accelerated failure time (AFT) model to analyze the effects of using automatically generated hints in Deep Thought, a propositional logic tutor. AFT is a branch of survival analysis, a statistical technique designed for measuring time-to-event data and account for participant attrition. We found that students provided with automatically generated hints were able to complete the tutor in about half the time taken by students who were not provided hints. We compare the results of survival analysis with a standard between-groups mean comparison and show how failing to take student dropout into account could lead to incorrect conclusions. We demonstrate that survival analysis is applicable to duration data collected from intelligent tutors and is particularly useful when a study experiences participant attrition.

2.1 Introduction

Intelligent tutoring systems have sizable effects on student learning efficiency — spending less time to achieve equal or better performance. In a classic example, students who used the LISP tutor spent 30% less time and performed 43% better on posttests when compared to a self-study condition [AR85]. While this result is quite famous, few papers have focused on differences between tutor interventions in terms of the total time needed by students
to complete the tutor. In many studies of intelligent tutoring systems, time is simply held constant for two groups, and efficiency then boils down to comparing the number of problems each group could solve in the given time and the results of posttest measures. However, it is not clear how to factor students who were not able to complete the tutor into this analysis. In this chapter, we explore tutor efficiency in terms of time and performance, while taking student dropout (ceasing to interact with the tutor before completion) into account.

College students often use computer-based tools to complete homework assignments, but no specific time limits apply. Typical time duration distributions violate the normality assumptions of many statistical tests and measures of central tendency. Anderson, Corbett, Koedinger, and Pelletier used mean duration data to compare differences between groups of students with and without intelligent feedback in the LISP tutor [ACKP95]. The authors state that the mean times (for the control group) are underestimates, as many students in the control (no-feedback group) did not complete all assignments. In other words, if the control group persisted, the time they took to complete tasks would have been longer than the observed durations for the few high-performing students who were able to persist without feedback. This study illustrates how dropout can obscure the true impact of an intervention.

Our exploration of tutor efficiency has three important elements: performance (tutor completion percentage), duration (total time spent interacting with the tutor), and dropout (whether stopped before completion). Dropout can easily confound the results of duration and performance. Different dropout rates between experimental groups can cause attrition bias [MH07], where groups completing the study are self-selected due to achievement levels; this self-selection causes the sample to become different than the target population and hampers the study’s generalizability [MEHB97]. When dropout exists, more complex analyses are needed to study learning efficiency; not only are results suspect for generalization purposes, but the data itself contains missing values because of high dropout rates. By modeling tutor data with high dropout rates using survival analysis, we hypothesize that we can build a more detailed understanding of tutor efficiency and explain differences between groups in an educational intervention.

In this study, we investigate data from a prior study of the Deep Thought logic tutor comparing versions with and without hints. Stamper et al. found that the odds of a student in the control group dropping out of the tutor after the first six problems were over 3.6
times higher when compared to the group provided with (data-driven and automatically generated) hints [SEBC13]. Students given access to hints also had better tutor performance, as well as higher overall course scores. However, comparison of duration means showed no differences in overall time spent in the Deep Thought logic tutor between the hint and control groups. This is likely because this comparison does not take into account student dropout. In this study, we applied survival analysis to data from Stamper et al.’s study to more fully explore the impact of hints on performance, duration, and dropout. We hypothesize that students given access to hints in the Deep Thought logic tutor, spend less time in tutor while also performing better than students without hints. In other words, the tutor efficiency for Deep Thought with hints is higher than that for they system without hints. We found that students given automatically generated hints take 55% of the time that students in the control needed to complete the tutor.

2.1.1 Methods and Materials

We perform our experiments on the Spring and Fall 2009 Deep Thought propositional logic tutor [Cro99] dataset as analyzed by Stamper, Eagle, and Barnes in 2011[SEBC11]. Data was collected from six deductive logic courses, taught by three professors. Each instructor taught one class using Deep Thought with automatically-generated hints available (hint group) and one without any additional feedback (control). The dataset includes 105 students in the Hint group and 98 students in the Control group. In Deep Thought, students choose the amount of time they spend using the online tutor; however, they were graded on the completion of 13 specific proofs.

The variables we use for this study are:

**Group**  a two level factor (Hint, Control) depicting the student’s experimental condition

**ProblemDuration**  the sum of the time taken over all steps in a problem until 1st completed (max 3min per step)

**Duration**  the sum of problem durations over all 13 problems

**Performance**  a number between 0–13 representing the number of proofs solved by the student

**Dropout** a boolean (True, False) defined as true for students who stop engaging with the tutor without completing the assignment (*Performance ≠ 13*)
Figure 2.1: QQ-Plots for the log-normal and Weibull distributions, the primary difference appears to be that the Log-normal is sensitive to very small durations, while the Weibull distribution is sensitive to very large durations.

Duration data often falls into a set of known distributions [BM11] [ME98]. Q-Q plots (figure 2.1) and histogram/density plots (figure 2.2) allowed us to narrow the possible distributions down to log-normal[CS88] or Weibull[W+W51]. The primary difference appears to be that the log-normal does not fit well to early dropout (small durations), while Weibull does not fit as well for extremely long durations.

2.1.2 Survival Analysis

Survival analysis is a series of statistical techniques that deal with the modeling of “time to event” data [HLM08]. Survival analysis, also known as reliability analysis or duration analysis in economics, is named for its start as a method to measure survival after applying a medical intervention.

Survival analysis includes techniques for unknown values, non-parametric data, log-normal and Weibull probability distributions, and between-groups testing. We use the survival package for R [The14] to perform our analysis of learning efficiency, where the event for survival analysis is tutor completion.

“Censoring” allows for modeling duration with unknown values. Right censoring occurs
when participant data is lost before tutor completion, while left censoring would be when completion time is known but start time is not. For our data, the duration for students who drop out, or stop using the tutor is right censored, since we know the start time but do not know how long it would have taken the student to complete the tutor. For example, a student who has completed 5 problems but then quits is considered right censored as we do not know how long it would have taken the student to complete all 13 problems.

The survival function is defined as:

\[
S(t) = Pr(E > t) = 1 - F(t)
\]  

(2.1)

where \( t \) is the time in question, \( E \) is the time of the event (tutor completion), \( Pr \) is probability, \( F(t) \) is the duration distribution. This function gives the probability that the time of the tutor completion event, \( E \), is later than \( t \). That is, the probability that the student has not completed the tutor.

The duration distribution function, which is found via the cumulative distribution function \( c d f(t) \), is the probability of observing a problem completion time \( E \) less than or equal to some time

\[
F(t) = Pr(E \leq t) = 1 - S(t) = c d f(t)
\]

(2.2)
The derivative of \( F(t) \) is the probability density function (pdf) of the duration distribution,

\[
f(t)Pr(E = t) = F'(t) = \frac{d}{dt}F(t) = \text{pdf}(t), \tag{2.3}
\]

which provides us with the probability of observing a single tutor completion time \( E \) at some time \( t \). The hazard function, which tells us the instantaneous completion rate at time \( t \), is:

\[
\lambda(t) = \lim_{dt \to 0} \frac{Pr(t \leq E < t + dt | E \geq t)}{dt} = \frac{f(t)}{S(t)} = \frac{\text{pdf}(t)}{1 - \text{cdf}(t)}. \tag{2.4}
\]

This is the probability of the event occurring at time \( t \) given that the event has not yet occurred.

There are two models we consider for measuring effects of covariates: the accelerated failure time (AFT) model and the Cox proportional hazards model. AFT assumes that the effect results in one group that completes the tutor more quickly, while the Cox proportional hazards model assumes that the tutor completion rate for one group is a constant multiple of that for the other. We have chosen the AFT model, which assumes that the effect of the covariates, \( \theta \), is to accelerate the time to tutor completion by some constant factor [TP00].

\[
S(t|\theta) = S(\theta t) \tag{2.5}
\]

The AFT model assumptions fit with our hypothesis that hints shorten the time it takes to finish the tutor. In addition, it is easy to interpret \( \theta \) as a direct modifier to tutor completion time, and AFT facilitates using data from log-normal and Weibull distributions.

## 2.2 Results

To explore the differences between the hint and control groups we submitted the data to an AFT model as both a log-normal and Weibull distributions. The log-likelihood scores were -336.6 and -341.2 respectively. We chose to use the log-normal distribution, however both models fit similarly well and had similar results. Investigation showed the log-normal fit less well for early dropout students, while Weibull fit less well for students with extremely long durations. The probability distribution function (pdf) and cumulative distribution function (cdf) for the log-normal distribution are:
\[
pdf(t) = \frac{1}{\sqrt{2\pi} \sigma t} e^{-\frac{(\ln(t) - \mu)^2}{2\sigma^2}}, \quad cdf(t) = \Phi \left( \frac{\ln t - \mu}{\sqrt{2\sigma^2}} \right).
\]

where \( \Phi(x) \) is the cumulative distribution function (cdf) of the standard normal distribution. Note that we use \( \ln(t) \) when using \( \Phi \), we can do this thanks to the assumption that the log of the duration data shows a normal distribution.

The AFT model was statistically significant \( \chi^2 = 9.21 \) on 1 degree of freedom, \( p = 0.0024 \), \( n = 202 \), the coefficients of the model had the intercept (mean) as 5.655, the effect of Hint \( \theta \) as \(-0.599\), and the \( SD \) (scale) as 0.948. The effect of hints is \( e^{-0.599} = 0.55\); this means that it takes the Hint group 55% of the time it takes the control group to solve all 13 tutor problems.

We have plotted the inverse of the survival curve in figure 2.4.

Figure 2.3 shows the hazard function for the duration data, in other words, the instantaneous completion rate for each of the groups. It also shows the probability density function for the completion rate. Overall, these plots give us a good overview of the shape of the duration data, showing that the probable total duration for students in the control group, if they were to complete the tutor, would be much longer than that for students in the hint group. One concrete measure of this is illustrated by the median of the survival function, the location where 50% of people have completed the tutor. The median is found by \( e^\mu \), which is \( e^{5.65} = 284.29 \) for the control group and \( e^{5.65-0.599} = 156.18 \) for the hint group. Comparing these medians illustrates again the considerable difference in duration, or time to tutor completion, between the groups.

We measure the difference between groups with a Student’s t test to explore possible differences in performance between the two groups. We have no reason to believe that the total tutor scores are not normally distributed. We find that the total performance in tutor between the hint group \( (M = 9.26, SD = 4.26) \) and the control group \( (M = 6.78, SD = 4.62) \) was significant, \( t(200) = 3.98, p < .001 \), with the Hint group solving between 1.25 and 3.71 more problems than the control group. To illustrate these differences at different points in time, we have added points to the survival curve in figure 2.4 indicating the mean performance score for students who left the tutor (by completing or dropping out) within the 20%, 40%, 60%, and 80% quantiles of the maximum duration. This lets us compare relative performance in the tutor between the two groups. Both groups have similar scores at about the 30 minute mark, but the hint group experiences a large increase in performance by the 60 minute mark. After this, the rate of growth in score decreases, this is likely because students that take an exceptionally long time are less skilled.
To illustrate the impact of dropout, we compare the results of survival analysis to a more
traditional between-groups testing method. To explore differences in overall time in tutor
between the two groups, we subjected the total elapsed time on all 13 problems to a 2-tailed
Student’s t-test. The total time in tutor between the hint group \( M = 86.05, SD = 69.80 \) and
the control group \( M = 122.95, SD = 122.94 \) was not significant, \( t(200) = -1.34, p = 0.183. \)

However, since we know the data isn’t from a normal distribution, we can improve on
this accuracy by using a data transformation. To normalize the data, we use a logarithmic
transformation (common log, base 10) to make the data more symmetric and homoscedas-
tic. We subjected the log-transformed data to a 2-tailed Student t-test. The difference in
the logs of duration between the hint group \( M = 8.20, SD = 1.02 \) and the control group
\( M = 8.17, SD = 1.21 \) was not significant, \( t(200) = .168, p = 0.867. \) The ratio of the duration
between groups is calculated by taking the difference between the means of the groups,
since \( \lg(x) - \lg(y) = \lg \left( \frac{x}{y} \right) \). The confidence interval from the log-data estimates the difference
between the population means of log transformed data. Therefore, the anti-logarithms of
the confidence interval provide the confidence interval for the ratio. The anti-log of the
log-transformed means provides us with the geometric mean, the anti-log of the trans-
formed standard deviation is not interpretable. However, we can use the anti-log of the
confidence intervals. The most useful statistic we can derive is the difference ratio, and its
corresponding confidence intervals. A difference ratio of 0.026 between the means of the
logged data equates to \( 10^{0.026} = 1.06 \) with a 95% confidence interval of \( CI (0.52, 2.19) \).

2.3 Discussion

The results of the survival analysis allow us to reveal striking differences between the hint
and control groups in terms of the time needed to complete the tutor. Students in the hint
group complete the tutor in just over half the time needed for students in the control group.
It is interesting that the control group and the hint group do not have observed differences
in overall tutor time; in other words - students in the control group don’t generally complete
the tutor, so we can’t observe that they would take twice as long. It is likely that, given the
nature of the online access tutor, students are only willing to spend a certain amount of
time on this homework assignment. This can explain the observations of differences in
tutor progress observed at different times in figure 2.4.

Using survival analysis, we have estimated that the median duration (tutor completion
Figure 2.3: The probability density functions, represented by the dashed lines, provide the probability of observing tutor completion at a specific time. The hazard functions, the solid lines, are the probability of observing tutor completion at a specific time, given that it has not occurred yet. The probability of completion grows rapidly before becoming stable and eventually decreasing.

time) is 284 minutes for the control group and 156 minutes for the hint group. Dividing this by the number of problems in the tutor (13) gives us an estimate of efficiency, since it gives a time per problem needed for solution. The control group is therefore spending about 21 minutes per problem on average, while the hint group is spending an average of 12 minutes per problem. Although this estimate is derived using curves to estimate the (unknown) completion time for the control group, it does in fact fit with the observed data in the first several problems, before significant dropout in the control group occurred. Given these very different rates, we can see that the control group could be discouraged by solving less than 3 problems in an hour, while the hint group could solve 5 in the same amount of time. We were in fact surprised, after realizing these estimates, that students in the control group did not drop out sooner than they did!

This back-of-the-napkin estimate of efficiency is one objective measure that suggests reasons for differences in student behavior (e.g. choice to persist or not). Perception may also play a role in explaining why students in the control group drop out. One possible reason is that the students perceive that the time they are spending is not “worth it.”
et al. [BLJS01] defined the efficiency of a tutor, for how the student perceives it, as the “belief or judgement that information can be accessed without wasting time or effort.” Scanlon and Issroff [SI05] posit that computer-based instruction can conflict with the student’s perceptions of division of labour within learning context. In other words, students using computer-based instruction must be more self-directed and manage their own learning. The feedback provided by the tutor with hints might have helped students in the hint group feel more directed, while also helping them when they were stuck. This could have led to improved student perceptions of efficiency.

Without survival analysis, we would not be able to use observed duration to make any conclusions regarding potential differences between the hint and control groups. Using survival analysis, we can estimate the differences between groups by accounting for unknown values - the total time it would have taken students who dropped out (in both groups) to complete the tutor. Survival analysis has also enabled us to answer questions like “How much time is needed so that 50% of the students can complete the tutor”. Using the survival function $S(t) = .5$, we can estimate that the control group needs about 4.76 hours before 50% of students are done, while the hint group needs just 2.61 hours for half the group to complete the tutor. The survival function can be used to decide how much time needs to
be allocated in schools for students to use a tutor. We are considering using these estimates to proactively indicate to students when they might need to seek outside help. For example, if a student has taken more than the estimated time for half of students in their group to complete the tutor, we could suggest they speak to a teaching assistant.

### 2.4 Conclusions

As more learning systems become used outside of traditional classrooms it is imperative that educational data mining researchers leverage methods such as survival analysis that can handle non-normal data with high dropout rates. In this paper, we have used survival analysis to re-analyze the data from six 2009 logic courses using the Deep Thought logic tutor both with and without hints. The original paper showed that students without hints were over 3.6 times more likely to drop after the first six problems when compared to students offered hints. However, standard analyses were insufficient to show the impact of hints on the time needed to complete the tutor between the two groups. Using survival analysis, we have been able to estimate the total duration for both hint and control groups while taking into account dropout data, showing that students in the hint group take 55% of the time to complete the tutor than students in the control group. Using these estimates, we were able to explain approximate time per problem in the tutor for each group. This analysis sheds light on the probable reasons for dropout in the control group. Without these analyses, we might have concluded that students in the control group gave up sooner or were not persistent. However, in reality we see that these students are in fact persistent and spend a considerable amount of time in the tutor - equal to the amount of time spent in the tutor by the hint group. The difference is tutor efficiency: students in the hint group performed more efficiently, and were therefore able to complete the tutor, while the control group spent a similar amount of time but was less likely to be able to finish. This is a much richer understanding of the differences in effects between the two groups than traditional methods provide. The survival function also allows us to make predictions on how much time is needed for tutor completion, both for teacher planning and student feedback. These results suggest that survival analysis is a powerful toolbox for investigating the impact of interventions on learning efficiency while accounting for performance, duration, and dropout.
Chapter 3

Exploring Differences in Problem Solving with Data-Driven Approach Maps

Understanding the differences in problem solving behavior between groups of students is quite challenging. We have mined the structure of interaction traces to discover different approaches to solving logic problems. In a prior study, significant differences in performance and tutor retention were found between two groups of students, one group with access to hints, and one without. The Approach Maps we have derived help us discover differences in how students in each group explore the possible solution space for each problem. We summarize our findings across several logic problems, and present in-depth Approach analyses for two logic problems that seem to influence future performance in the tutor for each group. Our results show that the students in the hint group approach the two problems in statistically and practically different ways, when compared to the control group. Our data-driven approach maps offer a novel way to compare behaviors between groups, while providing insight into the ways students solve problems.

3.1 Introduction

Intelligent tutors have been shown to be as effective as human tutors in supporting learning in many domains, in part because of their individualized, immediate feedback, enabled by expert systems that diagnose student’s knowledge states [Van11]. For example, students provided with intelligent feedback in the LISP tutor spent 30% less time and performed 43% better on post-tests when compared to other methods of teaching [AR85]. Similarly,
Eagle, and Barnes showed that students with access to hints in the Deep Thought logic tutor spent 38% less time per problem and completed 19% more problems than the control group [EB14b]. In another study on the same data, Stamper, Eagle, and Barnes showed that students without hints were 3.6 times more likely to drop out and discontinue using the tutor [SEBC13].

Procedural problem solving is an important skill in STEM (science, technology, engineering, and math) fields. Open-ended procedural problem solving, where steps are well-defined, but can be combined in many ways, can encourage higher-level learning [Blo56]. However, understanding learning in open-ended problems, particularly when students choose whether or not to perform them, can be challenging. The Deep Thought tutor allows students to use logic rules in different ways and in different orders to solve 13 logic proof problems for homework. In this paper, we analyze the 2009 Deep Thought data set analyzed by Stamper, Eagle, and Barnes to further understand the differences between the hint and control groups.

The rich interaction data saved by transactional tutor logs offers many avenues to explore and understand student problem solving data, particularly for problems with multiple solutions. By mapping Deep Thought transactional data into an Interaction Network, and applying graph mining to derive regions based on the structure of this network, we develop a new Approach Map that illustrates the approaches that groups of students take in solving logic problems. We built Approach Maps for all 13 problems in the tutor, and illustrate a detailed analysis of two of these maps to explore the differences in problem solving between the hint and control groups.

The Approach Maps for problems 1.4 and 1.5 show that the hint group explored productive regions of the Interaction Network, while students in the control group were more likely to explore unproductive regions that did not lead to solutions. Problem 1.4 had available hints for the hint group. Even though problem 1.5 has no hints for either group, the Approach Map shows that the two groups still explore the problem space differently, illustrating that prior access to hints had a lasting effect. The Approach Maps help us discover unproductive regions of the problem-solving space, that we believe contributed to lower retention rates for the control group. In these regions, proactive hints could be used to direct students toward more productive approaches.

In section 3.2, we discuss related work and the prior study with Deep Thought. In section 3.3, we describe our algorithm for extracting Approach Maps from data. Section 3.4 presents
the results and illustrates two detailed Approach Maps on problems 1.4 and 1.5. Finally, we discuss the results, conclusions, and future directions for this work.

3.2 Related Work

Although they can be very effective, the construction of intelligent tutors can be costly, requiring content experts and pedagogical experts to work with tutor developers to identify the skills students are applying and the associated feedback to deliver [Mur99]. One way to reduce the costs of building tutoring systems is to build data-driven approaches to generate feedback during tutor problem-solving. Barnes and Stamper built the Hint Factory to use student problem-solving data for automatic hint generation in a propositional logic tutor [BS08a]. Fossati et al. implemented Hint Factory in the iList tutor to teach students about linked lists [FDEO+09]. Evaluation of the automatically generated hints from Hint Factory showed an increase in student performance and retention [SEBC13]; more details about this study are provided in section 1.2.2.

Although individual differences affect the ways that students solve problems [Jon00], it is difficult to examine the overall approaches that groups of students demonstrate during problem-solving. While pre and posttests are useful for measuring the change in behavior before and after an experimental treatment, we are interested in studying not only whether a student can solve a problem, but how they are solving the problem. In this study, we use Interaction Networks of student behaviors to investigate how providing hints affects student problem-solving approaches.

Interaction Networks describe sequences of student-tutor interactions [EJB12]. Johnson et al. showed that visualizations of Interaction Networks in the InVis tool could be used to better understand how students were using the Deep Thought logic tutor [JEB13]. Interaction Networks form the basis of the data-driven domain model for automatic step-based hint generation by the Hint Factory. Eagle et al. applied Girvan-Newman clustering to Interaction Networks to determine whether the resulting clusters might be useful for more high-level hint generation [EJB12]. Stamper et al. demonstrated the differences in problem solving between the hint and control groups by coloring the edges between Girvan-Newman clusters of Interaction Networks based on the frequencies between two groups, revealing a qualitative difference in attempt paths [SEBC13]. In this paper we expand on these works to develop Approach Maps that concisely illustrate the approaches that students take while
solving problems.

The Girvan-Newman algorithm (GN) was developed to cluster social network graphs using edge betweenness to find communities of people [GN02]. The technique also works in other domains. Wilkinson et al. applied GN in gene networks to find related genes [WH04]. Gleiser et al. used GN to discover essential ingredients of social interactions between jazz musicians [GD03]. We are the first to apply GN to Interaction Networks consisting of problem-solving steps.

In this paper, we mine the interactions from student problem solving data to summarize a large number of student-tutor transaction data into an Approach Map, demonstrating the diverse ways students solve a particular problem. We use Approach Maps to better understand the differences in behavior between two groups, students who were given access to hints, and those who were not, while completing homework in the Deep Thought logic proof tutor.

3.3 Methods

In Section 3.3.1, we describe how we use Deep Thought tutor logs to create an Interaction Network of all the student-tutor interactions within a single problem. We then show how we refine this network into regions of densely connected subgraphs (Section 3.3.2) using the Girvan-Newman (GN) algorithm. Finally, in Section 3.3.2 we define how we construct Approach Maps from the GN regions. For both steps in the process, we use the statistical environment \textit{R} [R C12], and the complex network research library \textit{iGraph} [CN06].

3.3.1 Constructing an Interaction Network

We construct an Interaction Network using all observed solution attempts to a single problem. Each solution attempt is a sequence of \{state, action, resulting-state\} interactions from the problem start to the last step a student performs. The \textit{state} represents enough information to regenerate the tutor’s interface at each step. An \textit{action} is defined as a step taken, and consists of the name of the rule applied, the statements it was applied to, and the resulting derived statement. For example, Figure 1.1 displays two Deep Thought interactions. The first interaction works forward from STEP0 to STEP1 with action \textit{SIMP} (simplification) applied to \((Z \land \neg W)\) to derive \(\neg W\). The second interaction works backward from STEP1
to STEP2 with action $B - ADD$ (backwards addition) applied to $(X \lor S)$ to derive the new, unjustified statement $S$.

We use a state matching function to combine identical states, that consist of all the same logic statements, but may have been derived in a different order. This way, the state for a step STEP0, STEP1, or STEP2 in Figure 1.1 is the set of justified and unjustified statements in each screenshot, regardless of the order that each statement was derived. We use an action matching function to combine actions, and preserve the frequency of each observed application.

If we treat the interactions used to create the networks as samples of observed behavior from a population, we could expect that the Interaction Networks constructed from different populations may have observable differences. However, rather than building two separate Interaction Networks and attempting to compare them, we construct a single network but keep track of the frequencies of visits by the hint and control groups for each state (vertex) and action (edge).

### 3.3.2 Extracting Regions

We partition the Interaction Network into densely connected subgraphs we call regions using Girvan and Newman's edge-betweenness clustering algorithm [GN02] and modularity score, a measure of the internal verses external connectedness of the regions [NG04]. We use the following algorithm to apply region labels to nodes in a Deep Thought Interaction Network. First, we remove the problem start state and goal states from the Interaction Network $IN$ to create $G_1$. Then, we iteratively remove all edges in $G_1$, in order of edge betweenness. Edge betweenness (EB) for a particular edge $e$ is calculated by computing all shortest paths between all pairs of nodes, and counting the number of shortest paths that contain the edge $e$. At each GN iteration $i$ and graph $G_i$, we find the edge with the highest EB, and call this bridge $b_i$. We remove the bridge $b_i$ from the graph $G_i$, and compute the modularity score for the resulting graph $G_{i+1}$. The process is repeated until all edges have been removed. Then, we assign identifiers to all nodes in the disjoint regions in the intermediate graph $G_n$ with the best modularity score. At the end of this process, we use $G_n$ to construct the Approach Map with nodes for the original start and goal states, and a new node for each region in $G_n$. The Approach Map edges are the edges that connect the start state and goals to the regions, and the bridges between regions that were removed from the Interaction Network to create $G_n$. 

Regions represent sets of steps that are highly connected to one another. When a solution attempt is within a region, new actions will stay within the region, or take a bridge edge into another region or goal. If an attempt is in a region with no goal bridges, the student must take a bridge to another region to reach a goal. Therefore, paths on the Approach Map can be interpreted as a high-level approaches to solving the problem. We hypothesize that we can use the Approach Map to discover different problem-solving approaches. In the next section, we investigate Approach Maps for two problems in Deep Thought, after which the hint and control groups diverged in performance.

**Approach Map**

Here we provide a more detailed description of the algorithm we use to generate an Approach Map from the Interaction Network for a problem after its nodes have been labeled with region identifiers. A region \( A \) (or action \( a \)) dominates a region \( B \) if every path from the start of the problem to \( B \), must go through \( A \) (or \( a \)).

1. Combine all nodes with the same region identifier into a single region node labeled with the identifier, and remove all the edges with the same region identifier.
2. Combine all goal states that are dominated by a single region into a single goal node.
3. Calculate chi-squared to find in-edge frequencies that are different than expected between the groups (described in more detail below).
4. Combine parallel bridge edges between two regions into complex edges that represent the combination of the actions.
5. Label each region with the post conditions (derived statements) that result from the most frequent in-edge actions.
6. Provide new region identifiers that indicate the significant regions by the group with larger than expected frequency, with a number indicating the order in which the region was formed. For example, the regions the hint group visits more than expected are \( H1, H2, \ldots \), the regions the control group visits more than expected are \( C1, C2, \ldots \), and those that are visited as expected by both groups are labeled \( N1, N2, \ldots \).

We use a two-tailed chi-squared test to look for differences between the hint and control groups in how they visit regions in the Approach Map. The null hypothesis is that there is no difference in the frequency of entering a particular region between attempts in the hint group and the control group. The alternative hypothesis is that the groups enter regions with different than expected frequency. We use Bonferroni correction [Sha95] to compensate for
the number of tests that we run. When the \( p \) is less than the Bonferroni-corrected alpha, we label the regions H1, H2, etc., blue for significantly higher than expected participation by the hint group. Regions C1, C2, etc., are bordered in orange and represent regions where the control group was represented more frequently than expected. Regions N1, N2, etc., satisfy the null hypothesis in that both groups visit these regions as expected.

The Approach Map for problem 1.4 is shown in Figure 3.1. Each region node contains statements derived on the most frequent in-edge. The bridge edges are those actions that most frequently lead into and out of each region. The edges are labeled with the action(s) taken and the number of attempts using these actions. A bridge and its resulting region can be read as, this many students performed the following action(s) to derive the following proposition(s). For clarity we do not draw edges with frequency less than ten, and we delete actions and regions that become disconnected due to these edge removals. The edges on the map are colored on a spectrum based on the ratio between the groups from blue (hint group) to orange (control group.) Paths in the Approach Map can be interpreted as empirically-observed problem solving approaches.

Each approach map is accompanied by a region table which provides more detail about the frequencies of observed solution attempts from each group. The columns Hint and Control are the total frequencies of in-edges by each group, or in other words, the number of solution attempts from each group that visit at least one node in the region. Time refers to the mean time a solution attempt stays in the region before exiting. Goals refers to the sum of the frequencies of out-edges that lead to goal states. The \( p \) values are the results of the chi-squared tests to compare group representation to expected values.

### 3.4 Results & Discussion

We perform our experiments on the Spring and Fall 2009 Deep Thought propositional logic tutor dataset as analyzed by Stamper, Eagle, and Barnes in 2012[SEBC13]. The data set is made up of 4301 student-attempts which contain 85454 student-tutor interactions across 13 problems. The prior study compared the performance between the hint (\( n=105 \)) and control (\( n=98 \)) groups, showing that students with available hints on the first 5 problems in L1 were 3.6 times more likely to complete the tutor. In addition, the hint group spent about 12 minutes per problem in the tutor, while the control group took 21 minutes per problem. Although the average total time in tutor between groups was not significantly different,
more in-depth analysis of time revealed that this was because many students in the control group dropped out of the tutor, and were less likely to complete problems attempted in levels L2 and L3 [EB14b]. In this section we present the results of applying Approach Maps to 11 problems in this data set, and illustrate the Approach Maps to two problems 1.4 and 1.5, just before the retention gap begins between the hint and control groups.

Table 3.1 summarizes our results from constructing Approach Maps for 11 of the 13 Deep Thought problems (records for problems 1.6 and 2.1 have not been normalized into our standard format). It is difficult to summarize the information from each map in to a single row in a table, however we have selected a few measures that provide an overview. In Table 3.1, the Hint and Control columns count the number of problem attempts for each group. Regions refers to the total number of regions in each Approach Map. Sig-H and Sig-C denote the number of regions visited significantly more than expected by the hint and control groups respectively. Sig-G denotes the number of significant regions that were also goal regions. This table shows that most problems in Deep Thought have 10-17 regions. In problems 1.4 and 1.5, more than half of the regions were visited more than expected by the hint or control groups.

Table 3.1: Summary of Approach Maps for 11 Deep Thought tutor problems. An asterisk (*) indicates problems where the hint group had access to hints.

<table>
<thead>
<tr>
<th>Prob</th>
<th>Hint</th>
<th>Control</th>
<th>Regions</th>
<th>Sig-H</th>
<th>Sig-C</th>
<th>Sig-G</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1*</td>
<td>348</td>
<td>447</td>
<td>16</td>
<td>1</td>
<td>7</td>
<td>2</td>
</tr>
<tr>
<td>1.2</td>
<td>196</td>
<td>187</td>
<td>16</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>1.3*</td>
<td>171</td>
<td>152</td>
<td>15</td>
<td>2</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>1.4*</td>
<td>138</td>
<td>219</td>
<td>16</td>
<td>5</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>1.5</td>
<td>155</td>
<td>218</td>
<td>18</td>
<td>4</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>2.2*</td>
<td>150</td>
<td>150</td>
<td>15</td>
<td>4</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>2.3*</td>
<td>129</td>
<td>108</td>
<td>14</td>
<td>4</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>2.4*</td>
<td>99</td>
<td>80</td>
<td>10</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2.5*</td>
<td>112</td>
<td>79</td>
<td>10</td>
<td>3</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>3.1</td>
<td>173</td>
<td>114</td>
<td>17</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>3.2</td>
<td>147</td>
<td>100</td>
<td>12</td>
<td>1</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>

We present detailed Approach Maps for problems 1.4 and 1.5 for three reasons. First,
they occur before a large increase in control group dropout, as shown in Figure 1.2 in Section 1.2.2. After these problems, the odds of the control group dropping (no longer logging into the tutor) was 3.6 times that of the hint group [SEBC13]. Second, these problems stand out in Table 3.1, with high goal regions and more than half the extracted regions being significantly different between the groups. Third, in problem 1.4 the hint group had access to hints, however in problem 1.5 neither group received hints. This allows us to look for differences in behavior between the groups when working in the tutor on equal terms. For each of these problems we generated the Approach Map and corresponding reference table and visualization as described in Section 3.3.2.

### 3.4.1 Problem 1.4

**Problem 1.4:** Prove \( X \lor S \)

*Given:* \( Z \rightarrow (\neg Y \rightarrow X), Z \land \neg W, W \lor (T \rightarrow S), \neg Y \lor T \)

Problem 1.4 was designed to teach the Constructive Dilemma (CD) rule \([(P \rightarrow Q) \land (R \rightarrow S)] \land (P \lor R) \rightarrow (Q \lor S)\). For this problem, students in the hint group had access to hints. Table 3.2 describes the regions of the Approach Map. Figure 3.1 shows the Approach Map for problem 1.4. To show differences in more detail, we have provided the most common attempts for each group in figure 3.2. In particular, this figure shows that the control group has derived an unjustified statement \( T \) that cannot be proven.

Hints were available for the hint group on problem 1.4; Table 3.3 shows the number of hint requests at depths D1 to D4, where students could request up to four consecutive hints while in a single state. In Table 3.3, \( R \) is the region, D1–4 is the depth of the hint, Target Proposition refers to the proposition the student is directed to derive, and Rule is the rule that the student is directed to use. Depth D1 hints direct students to the Hint column, while depth D2 hints direct the students to the Rule column. Depth D3 tells the student the preconditions needed to derive the target proposition. The depth D4 hint is a bottom out hint that directly tells the student what interface elements to click to derive the target step.

There are three obvious paths in the Approach Map in Figure 3.1, one for the hint group, one for the control, and one with no differences between the groups. Figure 3.2 shows the most common solution paths for the hint and control group, with the same edges as the Approach Map. The Hint group tends to work forward using simplification (SIMP) (H1 to H2), while the control group was more likely to work backwards with addition (B-ADD) (C4 to C1). This backward addition path is a buggy strategy, that does not lead to any goals. We
Figure 3.1: The Approach Map for problem 1.4. Edges and vertexes can be read as the number of students who performed action(s) to derive proposition(s). Three main approaches are revealed, with the hint group strongly preferring to work the problem forwards. The control group often attempts to solve the problem by wards with addition, there are no goals along this path. More detail is given in Table 3.2.
Table 3.2: Detailed information on the regions in the 1.4 Approach Map shown in Figure 3.1.

<table>
<thead>
<tr>
<th>Region</th>
<th>Hint</th>
<th>Control</th>
<th>Time</th>
<th>Goals</th>
<th>p</th>
</tr>
</thead>
<tbody>
<tr>
<td>H1</td>
<td>109</td>
<td>65</td>
<td>1.59</td>
<td>2</td>
<td>&lt;0.001</td>
</tr>
<tr>
<td>H2</td>
<td>89</td>
<td>43</td>
<td>1.71</td>
<td>81</td>
<td>&lt;0.001</td>
</tr>
<tr>
<td>H3</td>
<td>19</td>
<td>3</td>
<td>1.34</td>
<td>22</td>
<td>&lt;0.001</td>
</tr>
<tr>
<td>C1</td>
<td>9</td>
<td>106</td>
<td>0.41</td>
<td>0</td>
<td>&lt;0.001</td>
</tr>
<tr>
<td>C2</td>
<td>6</td>
<td>68</td>
<td>0.41</td>
<td>0</td>
<td>&lt;0.001</td>
</tr>
<tr>
<td>C3</td>
<td>5</td>
<td>62</td>
<td>1.95</td>
<td>0</td>
<td>&lt;0.001</td>
</tr>
<tr>
<td>C4</td>
<td>24</td>
<td>134</td>
<td>0.32</td>
<td>0</td>
<td>&lt;0.001</td>
</tr>
<tr>
<td>N1</td>
<td>22</td>
<td>51</td>
<td>0.9</td>
<td>0</td>
<td>0.089</td>
</tr>
<tr>
<td>N2</td>
<td>9</td>
<td>15</td>
<td>1.41</td>
<td>20</td>
<td>0.811</td>
</tr>
<tr>
<td>N3</td>
<td>10</td>
<td>23</td>
<td>1.47</td>
<td>2</td>
<td>0.261</td>
</tr>
<tr>
<td>N4</td>
<td>14</td>
<td>38</td>
<td>0.13</td>
<td>0</td>
<td>0.056</td>
</tr>
</tbody>
</table>

Table 3.3: Number and depth of hints used by the hint group in each region; PS = Problem Start

<table>
<thead>
<tr>
<th>R</th>
<th>D1</th>
<th>D2</th>
<th>D3</th>
<th>D4</th>
<th>Target Proposition</th>
<th>Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>PS</td>
<td>50</td>
<td>13</td>
<td>13</td>
<td>4</td>
<td>¬W</td>
<td>SIMP</td>
</tr>
<tr>
<td>H1</td>
<td>36</td>
<td>17</td>
<td>11</td>
<td>5</td>
<td>Z</td>
<td>SIMP</td>
</tr>
<tr>
<td>H1</td>
<td>32</td>
<td>16</td>
<td>11</td>
<td>5</td>
<td>T → S</td>
<td>DS</td>
</tr>
<tr>
<td>H1</td>
<td>29</td>
<td>17</td>
<td>10</td>
<td>3</td>
<td>¬Y → X</td>
<td>MP</td>
</tr>
<tr>
<td>H2</td>
<td>36</td>
<td>19</td>
<td>18</td>
<td>2</td>
<td>(¬Y → X) ∧ (T → S)</td>
<td>CONJ</td>
</tr>
<tr>
<td>H2</td>
<td>21</td>
<td>17</td>
<td>12</td>
<td>3</td>
<td>X ∨ S</td>
<td>CD</td>
</tr>
</tbody>
</table>

note that there are no backwards hints given in Deep Thought, so students on this path do not get hints regardless of group. Data on hint usage, shown in Table 3.3 and the statements derived in the H1-H3 regions suggest that students in the Hint group are being “routed” toward a successful strategy.

The Approach Map in Figure 3.1 shows that the control group is more likely to visit regions that do not contain successful goals. It seems that the effect of hints is to keep students along a particular solution path, or prevent them from following the unproductive one taken by the control group. As a prior study of this data suggests [SEBC11], these students without hints are likely to abandon the tutor altogether. We hypothesize that hints help students achieve small successes and remain in the tutoring environment.
Figure 3.2: The most common attempt paths for each of the main approaches in the approach map for problem 1.4 (figure 3.1.) The highlighted nodes represent unjustified propositions.
3.4.2 Problem 1.5

Problem 1.5: Prove $A \lor \neg C$, given: $B \rightarrow (A \rightarrow E)$, $B \lor (A \rightarrow \neg C)$, $D \land \neg (A \rightarrow \neg C)$, $E \rightarrow \neg C$.

Problem 1.5 was designed to teach the Hypothetical Syllogism (HS) axiom $[(P \rightarrow Q) \land (Q \rightarrow R)] \rightarrow (P \lor R)$. Problem 1.5 is interesting, as this problem had no hints, but still has large differences between the groups. The Approach Map is shown in Figure 3.3, and additional information on the regions is available in Table 3.4.

Figure 3.3: Even in the absence of the automatically generated hints, the hint group still prefers a forward solution. The control group explores regions that do not lead to goals. Details are given in Table 3.4.

The Hint group approaches problem 1.5 by working forward using simplification (SIMP) on $D \land B$ to derive the separate statements $D$ and $B$; this could be a result of the forward directed hints they received in the earlier problems. The hints may have helped students develop a preference to working forwards, as doing so allowed them to request help if they became stuck. This preference carried over to the problems where hints were not available.

When working problem 1.5, the control group systematically derives statements that do not lead to goals. The most common attempt is to work backwards with disjunctive
Table 3.4: Detailed information on the regions in the 1.5 Approach Map shown in Figure 3.3.

<table>
<thead>
<tr>
<th>Region</th>
<th>Hint</th>
<th>Control</th>
<th>Time</th>
<th>Goals</th>
<th>p</th>
</tr>
</thead>
<tbody>
<tr>
<td>H1</td>
<td>53</td>
<td>39</td>
<td>0.42</td>
<td>82</td>
<td>0.002</td>
</tr>
<tr>
<td>H2</td>
<td>89</td>
<td>58</td>
<td>1.16</td>
<td>26</td>
<td>&lt;0.001</td>
</tr>
<tr>
<td>C1</td>
<td>17</td>
<td>55</td>
<td>0.69</td>
<td>0</td>
<td>0.002</td>
</tr>
<tr>
<td>C2</td>
<td>36</td>
<td>106</td>
<td>0.19</td>
<td>0</td>
<td>&lt;0.001</td>
</tr>
<tr>
<td>C3</td>
<td>24</td>
<td>65</td>
<td>2.41</td>
<td>0</td>
<td>0.005</td>
</tr>
<tr>
<td>C4</td>
<td>16</td>
<td>51</td>
<td>0.72</td>
<td>0</td>
<td>0.003</td>
</tr>
<tr>
<td>C5</td>
<td>30</td>
<td>81</td>
<td>1.2</td>
<td>0</td>
<td>0.002</td>
</tr>
<tr>
<td>C6</td>
<td>3</td>
<td>19</td>
<td>0.22</td>
<td>0</td>
<td>0.007</td>
</tr>
<tr>
<td>N1</td>
<td>7</td>
<td>14</td>
<td>2.27</td>
<td>0</td>
<td>0.434</td>
</tr>
<tr>
<td>N2</td>
<td>7</td>
<td>8</td>
<td>1.16</td>
<td>11</td>
<td>0.700</td>
</tr>
<tr>
<td>N3</td>
<td>2</td>
<td>12</td>
<td>3.38</td>
<td>0</td>
<td>0.037</td>
</tr>
<tr>
<td>N4</td>
<td>8</td>
<td>15</td>
<td>0.26</td>
<td>0</td>
<td>0.498</td>
</tr>
</tbody>
</table>

Syllogism (B-DS) (region C2) to derive \( B \lor (A \rightarrow \neg C), \neg B \) from the conclusion \( A \rightarrow \neg C \). This is likely because connecting with the premise \( B \lor (A \rightarrow \neg C) \) seems like a promising direction. However, it is not possible to justify the proposed proposition \( \neg B \) in this problem. This discovery is important as interventions can be added to warn away from regions that do not lead to goals. For example, we could offer a message that warns them that most students who attempt the same type of proof are not successful. Fossati et al. showed that human tutors helping students with the iList tutor, suggest that students delete unproductive steps [FDEO+09].

### 3.4.3 Working Backwards and Trailblazing

Although working backwards seems be unproductive for the control group, we note that there are productive approaches that work backwards, for example N1-N4 regions in problem 1.4 explored evenly by both groups. There are some advantages to working backwards in Deep Thought. When a student works backwards, Deep Thought asks whether they would like to target the premises (extraction) or construct their own hypothesized statement from the conclusion. Then, the student clicks on one of just a few rules that can be used backwards, limiting the search space for the next step. Next, students are prompted to fill
in the blanks in statements derivable from the chosen rule.

Region N1 in Figure 3.1, shows variables $p$ and $r$ that students can set to any proposition. Should the newly derived statements seem to match the patterns of existing premises, students keep them; otherwise they delete and try again. *Deep Thought* will sometimes warn students when they try to work backwards with something that is not justifiable. However, this may lead students to think that the tutor can always determine when working backwards is a viable strategy. In this case, students might mistakenly suppose that if there is no error message, they are closer to the solution. This is not the case, as *Deep Thought* has no built-in measures to determine closeness to completion. Rather, a few buggy rule applications are included in Deep Thought’s automated error detection.

**Trailblazing Effect**

Barnes and Stamper proposed that hints might limit the breadth of student approaches to problems, causing a hint ‘trailblazing’ effect that might bias students toward expert solutions when originally building the Hint Factory [SBLC08a]. In this analysis, we see some evidence of this effect. The difference in solution breadth between the two groups seems to be significant on several problems. The hints provided were limited to working forward, and the hint group demonstrated a strong preference for working forward. It remains to be seen whether providing hints for working backward will allow for more breadth of the search space. In any case, our results suggest that hints can cause a trailblazing effect, even when no hints are provided. Therefore, hints should be carefully constructed to include the diversity that a tutor designer wishes to promote in the tutor.

**3.4.4 Conclusions**

In this paper, we have presented Approach Maps, a novel representation of student-tutor interaction data that allows for the comparison of problem-solving approaches on open-ended logic problems. The Approach Map visualization results in a significant reduction in the space needed to describe a large amount of student-tutor data. It does this by reducing the student attempts into regions that we can consider as higher-level approaches to problem-solving. Deep Thought problems each had an average of 330 solution attempts, which were made up of about 6.5 thousand interactions. Using our Approach Maps, we partition problems into about 15 regions each (including 2–3 goal regions, as shown in
Table 3.1).

We have shown that we can use Approach Maps annotated with frequencies of visits by two groups to identify regions where a particular study group was over-represented. This allowed us to examine the approaches each group took to solving each proof. As we predicted, the automatically generated hints seemed to direct the students in the hint group down a common path, and we were able to detect this with the Approach Maps. Interestingly, even in problem 1.5, where neither group had hints, the hint group still showed a preference for working forwards, providing some evidence for a persistent effect of the hints. Analyzing Approach Maps also facilitated another important discovery that control group tended enter and remain in unproductive (or buggy) regions. These observed differences help explain how the automatically-generated hints produced the difference in tutor performance and retention in the 2009 Deep Thought study. Our investigations suggest that the patterns of behavior exhibited by students do result in meaningful regions of the solution attempt search space. We believe that, since the algorithms we applied to derive Approach Maps work on general graphs, we may be able to apply Approach Maps to understand problem-solving in domains where students solve open-ended problems in a procedural way.
Chapter 4

Exploring Networks of Problem-Solving Interactions

Intelligent tutoring systems and other computer-aided learning environments produce large amounts of transactional data on student problem-solving behavior, in previous work we modeled the student-tutor interaction data as a complex network, and successfully generated automated next-step hints as well as visualizations for educators. In this chapter we discuss the types of tutoring environments that are best modeled by Interaction Networks, and how the empirical observations of problem-solving result in common network features. We find that Interaction Networks exhibit the properties of scale-free networks such as vertex degree distributions that follow power law. We compare data from two versions of a propositional logic tutor, as well as two different representations of data from an educational game on programming. We find that statistics such as degree assortativity and the scale-free metric allow comparison of the network structures across domains, and provide insight into student problem solving behavior.

4.1 Introduction

Problem solving is an important skill across many fields, including science, technology, engineering, and math (STEM). Working open-ended problems may encourage learning in higher 'levels' of cognitive domains [Blo56]. Intelligent tutors have been shown to be as effective as human tutors in supporting learning in many domains, in part because of individualized, immediate feedback, enabled by expert systems which diagnose the knowledge
state of the student [Van11]. An additional benefit of computer-based environments is that they record extensive logs of student work, at a detail otherwise not possible. Stamper et al. used these logs to automatically build intelligent feedback into a otherwise non adaptive system [SEBC13]. One potential weakness of tutoring systems and other computer-aided instructional environments is that they can make it difficult for instructors to track how their students are solving problems within the system, this type of detachment can lead to a decreased sense of control and can affect adaptation of the tutor [Sel07].

In this chapter we explore student-tutor Interaction Networks, an empirical sample of student-walks though a problem-space modeled as a complex network. Interaction Networks were designed specifically for tutors with problem-solving tasks in which there are many goals and many paths to those goals, and that the user moves along those paths by using a set of actions. We explore Interaction Networks from multiple datasets and find that they exhibit scale-free properties. We find that Interaction Networks from different tutors share similarities in scale-free metrics and find that global and local vertex degree assortativity provide insight into the nature of the problem solving environments.

We present success stories from a variety of tutoring systems and argue that Interaction Networks are useful for a wide variety of problem solving instructional environments, even when the potential problem-space might look to be intractable. The scale-free properties of the network make it possible to discover the important regions of a network even with a small sample of student data. We expect that Interaction Networks are a good representation for describing a large amount of student data; provide a common language for performing cross tutor evaluation of problem solving environments; and that the intuitive nature of the model helps convey results from the data in a way that is interpretable for both research and instructor.

4.1.1 Previous Work

Creation of adaptive educational programs is expensive. Intelligent tutoring systems require content experts and pedagogical experts to work with tutor developers to identify the skills students are applying and the associated feedback to deliver [Mur99]. In order to address the difficulty in authoring intelligent tutoring content, Barnes and Stamper built an approach called the Hint Factory to use student data to build a Markov Decision Process (MDP) of student problem-solving approaches to serve as a domain model for automatic hint generation [BS08b]. Hint factory has been applied across domains [FDEO+09, PIHB14, EJBB13], and
been shown to increase student retention in tutors [SEBC13]. Eagle and Barnes abstracted this domain model into a complex network representation of the student-tutor interactions called an Interaction Network [EJB12]. These networks worked well as visualizations of student work within tutors and Johnson et al. successfully created a visualization tool InVis specifically to aid instructors in understanding student-tutor interaction data [JEB13]. Preliminary results on mining the Interaction Networks found that applying network mining techniques to Interaction Networks can help uncover useful sub-regions that represent diverse student approaches to solving a particular problem [EB14a]. In this chapter we expand on the theoretical framework of the Interaction Networks, explore their structure, and the processes that generate them.

4.2 Modeling Problem Solving

Before we can model user interactions within a problem-solving environment, we first need to define our theoretical foundations for what we consider problem solving, as well as what types of environments we will be studying. There are many different ways to define problem solving. Anderson referred to problem solving as "any goal-directed sequence of cognitive operations" [And93]. Newell and Simon [NS+72] break classical problem solving down into mental representations and Problem Space; with problem solving being the internal (cognitive) and external activity based manipulation of that Problem Space. We will use Newell and Simon's characterization of internal and external processes for the remainder of this discussion. Our goal is to model transactional data, which is human-computer interactions that occur in interactive systems; specifically, we are interested in educational systems such as Intelligent Tutoring Systems or other Computer Aided Instructional programs. The external processes can be tracked and measured with user logs from these environments. We will also use these external process logs to make inferences into differences between users internal processes.

The distinction between internal and external problem representation is important, as our goal is not necessarily to model individual cognitive processes such as those represented in ACT-R’s [And07] Imaginal module, which contains steps the user makes in internal cognition. The external manipulation is closer to the Manual Control module in ACT-R. We know the result of the cognitive processes, but do not know precisely what those processes were. In this sense, we treat the students as a black-box, measure their outputs, and then
use that to make inferences into the processes within groups of students. In order to further specify the type of problem solving environment we are focusing our study on we will borrow from Jonassen in *Toward a Design Theory of Problem Solving* which identified 11 different types of problem solving, based on problem features such as structuredness, complexity, and context [Jon00]. Some of these problem types more easily lend themselves towards computer-based education than others.

We will focus on Jonassen’s *Rule-Using Problems*, defined as problems with correct solutions, multiple solution paths, and multiple rules that govern the process. “Rule-using problems constitutes a new class of problem solving, so no research about these specific kinds of problems exists. Cognitive processes and design principles will have to be generalized from any research on prototypic examples of this class (e.g., online searching) in addition to cognitive task analysis [Jon00].” The nature of building a computer aided instructional program often requires a set of well-defined that students must use to manipulate the tutor environment in order to find a desired solution. Modeling student problem solving behavior with Interaction Networks will work best in these rule-using and puzzle environments. However, by defining clear problem representations, researchers could model other problem solving types with Interaction Networks.

### 4.2.1 Interaction Network

An Interaction Network is an empirical result of a sample of human-walkers on a problem-space. In sequential problem solving environments, or Rule-using Problems, a solution path describes a sequence of state changes from a starting position towards a desired end position. Actions are used to change from state to state. For this chapter we will only consider discrete time environments with deterministic state transitions. An Interaction Network is created from transactional student-logs recorded when students work in the tutor. We describe Interaction Networks with terms inspired from classical planning, such as STRIPS [FN71]. The Interaction Network concisely describes the information from a large number of problem solving sequences. It is modeled as a weighted network, with relevant information; such as time spent, frequency, number of errors, and type of action, encoded into the edges and vertices. Interaction Networks represent empirical student-tutor observations and do not represent students’ internal cognitive states that may occur between recorded actions.

Each *Interaction* within the network is an empirical result (represented in transactional
log-files) of some cognitive operations that we are defining as Activity-based manipulations of the environment State. Activities are the Rules or Actions available within the environment. We use State rather than Problem Space as we are modeling the empirical problem-solving State rather than the internal Problem Space manipulations.

We define the State as a representation of the environment, it should initially represent enough information to regenerate the tutor's interface at each step and be generalized as needed with matching functions. We could then describe the State Space in a way similar to game-theory as the set of states that represent every possible legal configuration of the environment from an initial state, and State Space Complexity as the number of elements in this set [All94]. Depending on the types of actions available in the environment, the State Space and Problem Space could easily be infinite. However, as an empirical model, we have the Observed State Space which refers to the set of program States that we have observed and the Interaction Network itself is a proxy to the Observed Problem Space.

We use a State Matching Function to determine which states we consider equivalent. Matching functions that allow some variation help provide more overlap and smaller state space, since problem-solving attempts have more of a chance of matching with prior approaches.

We define Action as:
- Action ID — unique identifier for the rule used
- Precondition — parameters for the action
- Postcondition — action results

We define an Interaction as:
- Start State — the state before the action is taken
- End State — the state after the completion of the action
- Action — the action causing the state change
- Property Map — extra interaction-specific information

An Interaction Network is constructed by treating the set of interactions for a specific program as an edge list, and collapsing parallel edges with identical actions (as defined by a action matching function) into weighted edges, these weighted edges represent a set of transactions. Program states and actions are compared with their respective Matching Functions. Matching functions determine when interactions will be consolidated into a single Vertex/Edge. Property maps are attached to the network to hold additional information about states and edges, such as student IDs, elapsed time, start states, goal states, etc.
Defined as a Property Map: Set of Vertices:
- Unique ID — defined by the State Matching Function
- Set of out Edges — actions taken to leave this state
- Set of in Edges — actions taken to end in this state
- Property Map — contains key/value pairs to a variety of state specific attributes

Set of Edges:
- Unique ID — defined by the Action Matching Function
- Start State — the state before the action is taken
- End State — the state after the completion of the action
- Property Map — contains key/value pairs to edge specific attributes

Matching Functions:
- State Matching Function — determines which states are considered the same
- Action Matching Function — determines which actions are considered the same

Interaction Networks can be built using any system logs that can be mapped into \{state, action, resulting-state\} tuples. In our prior work, we have also represented the states as the set of post conditions (results of performing actions in the tutor.) We have proposed both ordered and unordered matching functions. Ordered matches mean that the two matching states have exactly the same steps executed in the same order. Unordered matches mean that the two matching states have all the same parts, but may not have been done in the same order. The less restrictive the matching function, the more concise the Interaction Network, since it lowers the number of vertices in the network at the cost of having potentially less contextual information. Figure 4.1 uses unordered matches from a single problem solved in a propositional logic tutor.

### 4.2.2 Interpretation

An Interaction Network is a subset of the problem solution space, which includes a number of states and solutions that are not only not observed in data, but are also unlikely to ever be visited by a human. If we consider each attempt to solve a problem as a black-box walker across the problem solution space we can see how different biases in the walkers choice of actions result in different Interaction Networks.

Consider a population of random walkers that starts at the problem initial state and chooses random actions. Now consider "biased walkers" from a population that have a particular bias in the actions chosen at particular states. This bias will cause states to
Figure 4.1: An example Interaction Network. The vertices represent tutor states, edges represent actions that go from state to state, edge thickness is weighted by frequency, goal states are green, and vertices with high attrition rates have red boarders.

be visited differently, and could result in large parts of the problem solution space to go unvisited. Figure 4.2 shows an example with three different populations of walkers. The black states refer to states that exist within the state space and are visited by random walkers, but are never visited by either group of biased walkers.

We interpret an Interaction Network is a sample of problem solving behavior from a population for a particular problem. While different populations of student-walkers will share the same potential solution space, they could have very different Interaction Networks. In Eagle et al. [EB14a] two experimental groups were compared by observing the areas of the Interaction Network that each group tended to visit more than would be expected if the groups were from the same population.

As stated above we treat the students as black boxes, measure their outputs, and then use that to make inferences into the processes used by groups of students. Interaction Networks are an empirical sample of student-walks through a problem-space modeled as a complex network. Interaction Networks were designed specifically for open-ended rule-using problems, where each problem and possible actions are well-defined. There can be many goals, and many paths to those goals. There is little to gain from modeling problems that do not offer meaningful choices [Rab10], such as the example in figure 4.3, where any action will move the student closer to the goal.
Figure 4.2: Example of a problem-space. The black states represent parts of the problem space that are not visited by human walkers. The blue and orange states represent the states observed from two different populations of human walkers.

We have observed similar features across a variety of different educational domains and expect that there are some problem-solving aspects of rule-using tutors that are shared between different domains. Figure 4.4 shows an example of one common feature we would expect to occur within a rule-using problem with a single goal-state. The students branch from a shared start state and then later either converge on the goal or go down a path without a goal. Rabin, referred to this structure as a convexity [Rab10], in the context of designing or describing meaningful decisions in a video game. We have targeted these bubbles and used them as a method to simplify the Interaction Networks for visualizations [JESB13]. We expand upon this idea, by adding multiple goals in figure 4.5, as well as areas where students converge that are not the goal.

### 4.2.3 Network Features

Even though the theoretical state space and problem-space can be very large or infinite, the empirical state-space is small enough to explore. We can explore the high level features of Interaction Networks by computing some common network invariants. For example, the degree of a state represents how many states it is connected to. A state with a high
Figure 4.3: Interesting problems must have meaningful choices. No matter what action the student takes they move from the start state (white) toward the goal state (green), the rule-using problems we want to focus on allow divergence.

Figure 4.4: The vertices represent a convexity, or 'bubble', which is a region within an Interaction Network where the problem expands from one state and then later converges to a single state. If there are multiple paths to get to the goals, and they can cross over, these types of structures should exist in the network.

Figure 4.5: Observed networks rarely have complete bubbles, the grey edges and vertices represent places that the either we have yet to observe a student visiting or that it cannot be reached.
out-degree is a part of the problem in which there are a large number of possible actions. Likewise, a state with a high in-degree is a part of the problem that is converged on by a large number of other states see figure 4.7.

We will look at two particular metrics that can help describe the overall nature of the Interaction Networks. Scale-free networks are a broad class of complex networks that have degree distributions that follow power law [BA99]. We will also look at assortativity, the degree to which vertices with the same degree are connected to each other [New02].

**Scale-Free**

Qualitative observation of Interaction Networks from data, such as Deep Thought in [SEBC13], revealed that a small portion of the network states had degree much higher than the average degree. Several largely studied complex networks such as World Wide Web links, biological networks, and social networks are conjectured to have degree distributions which follow a power law [CSN09]. These networks are called scale-free networks [BA99]. Barabási and Albert theorized that scale-free networks are caused by preferential attachment processes; these processes bias the distribution of a value over objects by how much of the value the objects already have [BA99]. Barabási and Albert’s preferential attachment is one of several different methods of generating scale-free networks [DM02]. Our theory is that the samples of non-random student walkers who generate the Interaction Network are more likely to perform similar actions to other walkers from the same population, this results in shared preferences for certain types of actions or states.

An important feature of scale-free networks is that they are resilient to error and attack [AJB00]. Although several vertices were removed from Figure 4.5, however the network remained connected and in more or less the same shape. The high degree vertices act as hubs and hold the network together; random vertex removals will not matter unless a hub is removed. One method to visually check a network to see if it could be scale-free is to judge a log-log plot, the logarithms of the x and y axises are plotted and a straight line indicates a power relationship [HSK06]. Another method for verifying a power law distribution is to fit the parameters [New05], [CSN09], and use the Kolmogorov-Smirnov test to check the hypothesis that the degree distribution is from a distribution other than power law [MJ51].

To investigate the scale-free nature of Interaction Networks, we measured the examined networks generated from three sets of data:

- Deep Thought (see Section 4.1) — 11 problems
• Deep Thought 3 (see Section 4.1) — 40 problems
• BOTS (see Section 4.4) — 13 problems

For the problems in the BOTS dataset, we looked at Interaction Networks generated from two different state matching functions, (codestates and worldstates.) We continue here to define assortativity, and then we will discuss measurements of both scale-free and assortativity properties for these three problem-solving environments.

**Assortativity and Self Similarity**

The power law degree distribution indicates that there are a few vertices that contain the majority of the overall degrees. Degree assortativity describes the correlation of degree between vertices and their neighbors [New02]. The assortativity metric $r$ ranges between -1 and 1. A network with $r = 1$ would have each vertex only sharing edges vertices of the same degree. Likewise, if $r = -1$ vertices in the network would only share edges with vertices of different degree. Several commonly studied classes of networks tend to have patterns in their assortativity. Social networks tend to have high assortativity, while biological and technological networks tend to have high dissortativity (negative $r$ values) [New02]. Randomly generated scale-free graphs, such as those generated from Barabási and Albert’s preferential attachment, tend to have $r$ values closer to 0.

Li et al. proposed further classifying networks with power law degree distributions into *scale-free* (self-similar) and *scale-rich* (self-dissimilar) networks [LADW05]. Li et al. proposed the metric:

$$s(g) = \sum_{(i,j) \in E} d_i d_j,$$

which measures the extent to which the graph $g$ has a *hub-like* core, $s$ is higher when high-degree vertices share edges [LADW05]. The normalized version of the $s$-metric:

$$S(g) = \frac{s(g)}{s_{\text{max}}},$$

where $s_{\text{max}}$ is a graph which maximizes the $s$-metric for a degree distribution, and is a measure of how close the network is to having a central core of highly connected centrally-located vertices [LADW05]. A low $S(g)$ indicates a *scale-rich* network in which high degree vertices do not form hubs. This is important as in the context of Interaction Networks, these hubs represent states from which students make a larger proportion of actions. Not
being able to reach a state in the program has the potential to prevent the student from finding the goal. High values of $S(g)$ also indicate self-similar “fractal” like patterns, which we expect to see in parts of the network (figure 4.5.)

We computed the assortativity and $S(g)$ metrics for problems from both Deep Thought as well as the BOTS game, shown in Table 4.1. For the most part the Interaction Networks do not exhibit high levels of assortativity, with the BOTS data leaning slightly towards dissortativity. The high $S(g)$ values indicate that the high degree vertices are connected, when paired with assortativity we can get an idea of how the lower degree vertices are attached. Dissortative scale-free networks have low degree vertices attached to the high degree hub vertices and are less likely to share edges with other low degree vertices. Assortative scale-free networks have regions of highly connected hubs, and regions of low degree vertices that are rarely connected directly to the hubs. Finally, the $r = 0$ scale-free networks are likely to contain mixtures of assortative and dissortative regions. Piraveenan et al. showed that networks with low values of assortativity could be composed of assortative and dissortative connected components [PCU12].

Table 4.1: Averages across the datasets, $dr$ is the directed assortativity, $r$ is the undirected assortativity, and $S(g)$ is the scale-free metric.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>$dr$</th>
<th>$r$</th>
<th>$S(g)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code</td>
<td>-0.25</td>
<td>-0.30</td>
<td>0.79</td>
</tr>
<tr>
<td>World</td>
<td>-0.33</td>
<td>-0.31</td>
<td>0.92</td>
</tr>
<tr>
<td>DT1</td>
<td>-0.03</td>
<td>0.01</td>
<td>0.54</td>
</tr>
<tr>
<td>DT3</td>
<td>0.16</td>
<td>0.16</td>
<td>0.79</td>
</tr>
</tbody>
</table>

### 4.3 Applications

Interaction Networks have three primary applications. The first, is to automatically generate next-step hints by learning a *Hint Policy* from a network built from a corpus of student data, and then using that policy and a *Hint Template* to turn the policy output into human readable next-step hints. The second is to allow instructors to visualize the problem solving
Figure 4.6: An example of an assortative network (right) where vertices connect to vertices with similar degree, and a dissortative network (left) where vertices share edges with vertices of different degree.

Table 4.2: Interaction Networks generated from problems in the 2009 Hint Factory dataset [SEBC13]. Alpha is the exponent of the fitted power-law distribution, logLik is the log-likelihood of the fit with xmin, KS is the Kolmogorov-Smirnov test statistic (smaller score indicates better fit), p is the p-value of the KS test, values less than 0.05 reject a power-law fit.

<table>
<thead>
<tr>
<th>Network</th>
<th>N</th>
<th>alpha</th>
<th>xmin</th>
<th>logLik</th>
<th>KS</th>
<th>p</th>
</tr>
</thead>
<tbody>
<tr>
<td>dt1-1</td>
<td>1295</td>
<td>3.50</td>
<td>8.00</td>
<td>-107.92</td>
<td>0.05</td>
<td>1.00</td>
</tr>
<tr>
<td>dt1-2</td>
<td>1030</td>
<td>3.49</td>
<td>6.00</td>
<td>-185.74</td>
<td>0.02</td>
<td>1.00</td>
</tr>
<tr>
<td>dt1-3</td>
<td>1264</td>
<td>3.20</td>
<td>4.00</td>
<td>-350.76</td>
<td>0.04</td>
<td>0.97</td>
</tr>
<tr>
<td>dt1-4</td>
<td>1278</td>
<td>2.89</td>
<td>4.00</td>
<td>-409.83</td>
<td>0.03</td>
<td>0.99</td>
</tr>
<tr>
<td>dt1-5</td>
<td>1420</td>
<td>3.63</td>
<td>6.00</td>
<td>-190.10</td>
<td>0.03</td>
<td>1.00</td>
</tr>
<tr>
<td>dt2-2</td>
<td>1313</td>
<td>3.93</td>
<td>9.00</td>
<td>-86.94</td>
<td>0.06</td>
<td>1.00</td>
</tr>
<tr>
<td>dt2-3</td>
<td>1197</td>
<td>3.23</td>
<td>4.00</td>
<td>-313.22</td>
<td>0.03</td>
<td>1.00</td>
</tr>
<tr>
<td>dt2-4</td>
<td>232</td>
<td>3.14</td>
<td>2.00</td>
<td>-243.77</td>
<td>0.03</td>
<td>0.98</td>
</tr>
<tr>
<td>dt2-5</td>
<td>758</td>
<td>3.67</td>
<td>4.00</td>
<td>-162.06</td>
<td>0.02</td>
<td>1.00</td>
</tr>
<tr>
<td>dt3-1</td>
<td>1524</td>
<td>3.30</td>
<td>4.00</td>
<td>-328.82</td>
<td>0.06</td>
<td>0.54</td>
</tr>
<tr>
<td>dt3-2</td>
<td>1276</td>
<td>3.49</td>
<td>4.00</td>
<td>-201.36</td>
<td>0.05</td>
<td>0.97</td>
</tr>
</tbody>
</table>

Table 4.3: Averages across all Interaction Networks generated from problems in the BOTs(Code and World) and DT(DT1 and DT3). Variables are the same as those in Table 4.2.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>N</th>
<th>alpha</th>
<th>xmin</th>
<th>logLik</th>
<th>KS</th>
<th>p</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code</td>
<td>224.38</td>
<td>2.94</td>
<td>3.38</td>
<td>-78.88</td>
<td>0.08</td>
<td>0.74</td>
</tr>
<tr>
<td>World</td>
<td>51.69</td>
<td>3.34</td>
<td>7.00</td>
<td>-71.01</td>
<td>0.10</td>
<td>0.94</td>
</tr>
<tr>
<td>DT1</td>
<td>1144.27</td>
<td>3.41</td>
<td>5.00</td>
<td>-234.59</td>
<td>0.04</td>
<td>0.95</td>
</tr>
<tr>
<td>DT3</td>
<td>330.69</td>
<td>3.78</td>
<td>2.22</td>
<td>-184.16</td>
<td>0.03</td>
<td>0.99</td>
</tr>
</tbody>
</table>

56
behavior of a large group of students. The third is as a source for graph mining and network clustering. Although many of these applications can overlap, such as using network clustering to create better visualizations as in [EJBB13].

4.3.1 Hint Policies

Interaction Networks are a generalization from previous work of Stamper, Barnes, and Eagle [SEBC13] on automatically generating hints based on previously collected student data. One of the primary uses of Interaction Networks is to generate a Hint Policy, that is, a policy that selects a next-step action for a given state. Table 4.4 shows an example Hint Template which translates the Action recommended by the Hint Policy into a human readable hint. Hints can be generated from an Interaction Network by applying the Hint Factory MDP method as presented in [SEBC13]. When modeling data from a problem-solving environment into an Interaction Network, it is important to consider the information that will be contained within the Action (edge), as this is where the Hint Template will draw its information.

4.3.2 Visualization

The Interaction Network summarizes a large number of transactions into a representation that is much more accessible for instructors for understanding how their students are using a learning environment. Johnson et al. developed a visualization tool called InVis to help instructors explore the Interaction Networks by adding features such as Zoom, Filter, and Details on Demand[JE]. Professors using InVis were successful in performing a series of

Figure 4.7: The in-degree vs. out-degree ratio can reveal information about the complexity of problems. Degree assortativity is the tendency for vertices to share edges with vertices with a similar degree.
Table 4.4: Example Hint Template which translates the elements of an Action into 4 levels of hints.

<table>
<thead>
<tr>
<th>Hint #</th>
<th>Hint Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Try to derive [postcondition] working forward</td>
</tr>
<tr>
<td>2</td>
<td>Highlight [precondition1] and [precondition2] to derive it</td>
</tr>
<tr>
<td>3</td>
<td>Click on the rule [action ID]</td>
</tr>
<tr>
<td>4</td>
<td>Highlight [precondition1] and [precondition2] and click on [action ID] to get [postcondition]</td>
</tr>
</tbody>
</table>

data searching tasks, they also able to create hypotheses and test them by exploring the data [JEB13]. InVis was also used to explore the behavior of students in a educational game for Cartesian coordinates, exploration of the Interaction Networks revealed off task behavior, as well as a series of common student mistakes, the developers used the information gained from the Interaction Networks to change some of the user interface to reduce these undesirable behaviors [EJBB13].

### 4.3.3 Graph Mining

Eagle et al. proposed using network clustering techniques on Interaction Networks as a method of extracting potential sub-goals from student work [EJB12]. Eagle et al. extended this by leveraging edge-betweenness, a metric that determines edge importance based on the number of shortest paths between all nodes it is on, to extract Approach Maps from a Interaction Network [EB14a]. The Approach Map method was able to reduce the number of vertices required to describe the Interaction Networks for the Deep Thought tutor from around 1200 states to around 15 regions (highly connected sub graphs.) The Approach Maps were useful in qualitatively representing the different student approaches to problems. Eagle et al. was also able to show differences in between-group behavior for students, by preforming statistical analysis on each group’s representation within the approach map regions. We have included one approach map in figure 4.8. The vertices in this representation are regions (densely connected sub graphs,) the Hint group (blue) visited several regions along the goal path with greater than expected frequency when compared
to the control group. The control group, (which never received hints,) was significantly more likely to visit several regions when compared with the experimental group (who had access to next-step hints). Approach Maps provide a powerful summary of the student behavior within a tutoring system by taking advantage of the Interaction Network model.

Figure 4.8: The Approach Map for problem 2.4. Edges and vertices can be read as the number of students who performed action(s) to derive proposition(s). Several approaches are revealed, with the hint group strongly preferring to work the problem forwards. The control group is more likely to try approaches from which there are no goal paths.

4.4 Success Stories

In this section we provide examples of environments in which Interaction Networks or the precursor “MDP method” Hint Factory techniques have been used to either automatically generate feedback for new users based on previously collected data, or for the visualization of student work.
4.4.1 The Deep Thought Tutor

Deep Thought is a propositional logic tutor in which students are given a set of propositions and tasked with finding a conclusion via the manipulation of the givens with basic logic axioms [Cro99]. Deep Thought allows students to work both forward and backwards to solve logic problems.

For example a student starts at state $A \lor D, A \rightarrow (B \land C), \neg D \land E$, where each premise is separated by a comma. The student performs $SIMP(\neg D \land E)$, applying simplification (SIMP) to the premise $\neg D \land E$ and derives $\neg D$. This leads to the resulting-state of $A \lor D, A \rightarrow (B \land C), \neg D \land E, \neg D$. Errors are actions performed by students that are illegal operations of logic and the tutor. For example: The student is in state $A \lor D, A \rightarrow (B \land C), \neg D \land E, \neg D$. The student performs the interaction $SIMP(A \lor D)$ in an attempt to derive $A$. The resulting state would remain $A \lor D, A \rightarrow (B \land C), \neg D \land E, \neg D$; the log-file would mark this edge as an error.

There are several ways to model the state of the proof. The default tutor state would be the problem premises and conclusion, with the steps derived added to the state in the order they were derived. This could be thought of as a direct translation of the directed graph visible in figure 4.9. We relax this definition and use a matching function which treats each state as a partially ordered set, with lexicographically ordered forward-derived steps (resulting from an unordered matching function) listed with commas between them, followed by a sign, then a lexicographically ordered set of backward derived propositions. Our action matching function compares the axiom, direction (forward or backward), and pre-conditions and post-conditions.

4.4.2 BeadLoom Game

The Culturally Situated Design Tools (CSDT) [EBO+06] were designed to teach mathematics from a cultural context. The BeadLoom Game (BLG) [BB10] is a game-based extension of the CSDT: Virtual BeadLoom. The BLG and the original CSDT were built to give their users experience and practice with Cartesian coordinates, the intended audience was middle school aged children.

The BLG added game elements in order to increase motivation and learning [BB10]. In the BLG, players place beads on a 41x41 Cartesian grid using six different tools, shown in
Table 4.5, as well as an undo command. All actions take a color parameter; and there are 12 different colors available. The loom starts empty and once beads are added they cannot be removed unless players uses the undo action. However, beads can be overwritten by future actions. The goal of the game is to create a specific image with these tools. Figure 4.10 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.

<table>
<thead>
<tr>
<th>Action</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Draw Point</td>
<td>X, Y</td>
</tr>
<tr>
<td>Draw Line</td>
<td>X1, Y1, X2, Y2</td>
</tr>
<tr>
<td>Draw Rectangle</td>
<td>X1, Y1, X2, Y2</td>
</tr>
<tr>
<td>Draw Triangle</td>
<td>X1, Y1, X2, Y2, X3, Y3</td>
</tr>
<tr>
<td>Linear Iteration</td>
<td>X, Y, Length, +beads, rows, dir.</td>
</tr>
<tr>
<td>Triangle Iteration</td>
<td>X, Y, stepHeight, +beads, rows, dir.</td>
</tr>
<tr>
<td>Undo</td>
<td>Returns to previous step</td>
</tr>
</tbody>
</table>

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. Table 4.6 shows how the BLG data is translated into the network model. The player moves from a blank starting state to a state containing a red square by using the rectangle tool. For the BLG data, order of actions is not preserved in the state description. That is, if the player had reached the same state (same red square) by repeatedly using the point tool, that final state would be considered equal.

### 4.4.3 iList

The developers of iList (a linked-list tutor) also used the Hint Factory approach, however their underlying model is based on snapshots of the tutor's internal state rather than the transactional logs of user interactions [FDEO+09]. The authors use the results of the student
Figure 4.9: Example problem state in the Deep Thought logic tutor.

Figure 4.10: The BeadLoom Game Interface: the goal of the game is to create the pattern on the left in 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.
actions rather than the actions themselves, to define states. An *action* in iList is a change between two states, and is defined by the modification made between the two states rather than by specific commands from the user.

As a state-matching function, the authors find isomorphic internal representations for the tutor. Any of the tutor's internal states which are isomorphic to each other will be mapped to the same state in the Interaction Network. Similarly, the Action Matching Function selected by the authors was based on precisely the start state and end state, regardless of the actual commands input by the user. Therefore, any move from state A to state B will be matched to the same action in the Interaction Network.

To evaluate the efficacy of the data-driven feedback, the authors ran an experiment including five conditions; Three versions of iList, one group with human tutors, and one group with an unrelated task. The version of iList that provided data-driven feedback was referred to as iList-3. Of the three versions of iList, iList-3 had the largest average change in pre-to-post test scores, and the authors found these results to be similar to those achieved by a human tutor.

<table>
<thead>
<tr>
<th>Start-State</th>
<th>Action</th>
<th>Result-State</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>rectangle(-5 1 5 11)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>rectangle(1 -5 11 5)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>rectangle(-5 -11 5 -1)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>rectangle(-11 -5 -6 5)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>rectangle(-5 -5 -1 0)</td>
<td></td>
</tr>
</tbody>
</table>
After working with iList-3, the authors continued using data-driven methods to provide feedback, this time in the form of what they called (proactive feedback) [FEO+10]. This feedback was intended to guide the user towards a more desirable state, which is analogous to hints provided in other systems using Hint Factory. This tutor was shown to outperform all previous versions of iList.

### 4.4.4 BOTS

BOTS is a serious game designed to teach basic programming concepts to novice computer users and programmers [Hic12]. BOTS features open-ended problems with many possible solutions. In order to be able to generate hints for these problems without expert intervention, the developers used data-driven hint generation.

![Figure 4.11: An example problem in BOTS. The goal is to use commands to move the red robot, carrying boxes to their destinations. The players are scored on the number of commands used, with lower scores representing better, more efficient solutions.](image)

In BOTS, players are tasked with moving robots around a grid-based world, picking up objects and placing them on buttons. Once every button in the level is pressed down, the level is complete. Players control the robots’ paths by writing programs in a graphical language, using loops and functions as well as simple commands like "Step Forward" and "Turn left". Even the very simplest levels in BOTS have many correct (though not optimal) solutions. For this reason, the developers chose to represent students solutions
using programs and output (taken at runtime) rather than each addition or modification of an instruction. The states generated from programs are referred to as *codestates* while those generated from output are referred to as *worldstates*.

Using these representations, if two students end up running the same program (or getting the same output) they are considered to be in the same state, regardless of the commands they used to get there. These representations were chosen in order to increase the amount of overlap between students, allowing hints to be generated more frequently. For BOTS, the *states* of the Interaction Network are states in the world and the positions of each individual object in the world. Analogous objects are strictly identical, so swapping the place of two identical blocks, for example, would result in an identical state. The *action* is the change to the robot’s program made to result in the new state when the program is run. This means that one *interaction* in the network might consist of several lower-level GUI interactions within the system. For BOTS, the *state matching function* is implicit based on the ordering of elements in the state representation. The *action matching function* considers actions to be the same if they have precisely the same previous state and result state, regardless of differences in the program; however, the differences in programs are retained for the purpose of offering lower-level feedback. The Hint Policy used for the existing work with BOTS is based on the policy used for Deep Thought; however, due to the differences in state representation, the developers have to consider "step cost" as well as the number of edges when scoring states. This is because it is possible for a student to solve a problem in a single action by solving the problem on their first attempt. However, even without taking that into consideration, the developers were able to generate hints that suggest a next-state for the robot, which worked particularly well on paths where previous students had made many attempts.

### 4.5 Discussion

Despite differences in educational domain, we can model student-tutor interactions from puzzle games, programming tutors, and propositional logic environments in a common structure. While the educational objectives are different, the problem solving nature of the environments is shared. The potential state-space of many of these domains is infinite, however the majority of that space is not likely to be traversed by human walkers. This is related to the idea of preferential attachment and we predict that networks of interactions
will tend to have a degree distribution that follows power law. It also helps explain the results from Barnes and Stampers cold start [BS08b] where they found that they could start offering automatically generated hints for the majority of students even with a small initial sample; this is because the important high degree nodes are likely to be visited within the first few student attempts.

This scale-free nature results in networks that have a small number of vertices containing a large degree, we can use metrics such as assortativity and the $S(g)$ metric of self-similarity to gain a high level understanding of the network's organization. For example, the BOTS networks tend to have high degree vertices connect to other high degree vertices through low degree ‘bridges.’ For problem solving this means that students move from high choice areas to bridge vertices with low action choice, from which they move into areas of high action choice. The BOTS data is interesting because it shows the effect that different state matching functions have on the network topology; the assortativity does not change greatly, but $S(g)$ increases a moderate amount indicating the higher overall connectedness of the network.

In the DT3 data the trend is toward moving between high degree regions with occasional connections to low degree regions, for problem solving this means the students move from high choice states to other high choice states or into bridging regions of low choice states. This makes sense for the domain as each step in the propositional logic proof adds a new proposition to the state; as the student gets closer to the goal there are less reasonable actions to perform. It is interesting to see the differences between DT and DT3, the major difference in the tutors is that DT3 selects next problems for students based on their performance while DT has static problem sequence. The DT3 problem selection results in networks built from samples of less diverse populations of student data. Less diverse bias-walkers explore the same regions of the state space and will have smaller networks overall networks (evidenced by the lower average vertices.) The DT dataset had a static set of problems, when a student became stuck they would have no choice other than to search for a solution or quit the problem. This could be part of the reason why the $S(g)$ score is lower, as the increased walker diversity combined with greater search depth resulted in a network that was less self-similar.

Shared bias in choosing actions results in common paths and solution states, and if different populations of walkers have different bias we can expect to see them have different common paths and states. Interaction Networks provide a means for exploring
problem solving behavior in rule-using tutors, and looking for between-group differences. Previous work confirmed between-group differences in region visitation with approach maps [EB14a], see figure 4.8. We expect to see that local, region sub-graph, versions of $S(g)$ and assortativity differ between regions of the network. Highly self-similar networks are likely to contain patterns, such as those we discussed in section 4.2.2. Future work could look to explore the generative processes of these network shapes, we hypothesize that there are some common network sub-graphs, or motifs, that are shared across tutor domains. We are currently working on modeling more tutoring, game, and simulation environments in order to further explore the differences and similarities between them and discover cross-domain techniques to improve student learning.

Several diverse domains have had success using Interaction Networks to model student problem solving, such as puzzle games, propositional logic tutors, and programming tutors. Features of rule-using interactive tutors and student individual differences (bias) in action selection can be compared across educational domains by describing the shape of the network. Interaction Networks tend to have power law degree distributions, fairly high measures of self-similarity, $S(g)$, scale-free scores. Combining these metrics with assortativity reveals the overall shape of the network. Interaction Networks can be a common language in which to discuss hint generation and student work in rule-using problem solving environments. Even when the theoretical problem-space seems intractable, Interaction Networks created from observed transaction logs can provide a surprising amount of useful information.
Chapter 5

Interaction Network Estimation: Predicting Problem-Solving Diversity in Interactive Environments.

Intelligent tutoring systems and computer aided learning environments aimed at developing problem solving produce large amounts of transactional data which make it a challenge for both researchers and educators to understand how students work within the environment. Researchers have modeled student-tutor interactions using complex networks in order to automatically derive next step hints. However, there are no clear thresholds for the amount of student data required before the hints can be produced. We introduce a novel method of estimating the size of the unobserved Interaction Network from a sample by leveraging Good-Turing frequency estimation. We use this estimation to predict size, growth, and overlap of Interaction Networks using a small sample of student data. Our estimate is accurate in as few as 10–30 students and is a good predictor for the growth of the observed state space for the full network, as well as the subset of the network which is usable for automatic hint generation. These methods provide researchers with metrics to evaluate different state representations, student populations, and general applicability of Interaction Networks on new datasets.
5.1 Introduction

Data-driven methods to provide automatic hints have the potential to substantially reduce the cost associated with developing tutors with personalized feedback. Modeling the student-tutor interactions as a complex network provides a platform for researchers to automatically generate next step hints. An Interaction Network is a complex network representation of all observed student and tutor interactions for a given problem in a game or tutoring system. In addition to their usefulness for automatically generating hints, Interaction Networks can provide an overview of student problem-solving approaches for a given problem.

Data-driven approaches cannot reliably produce feedback until sufficient data has been collected, a problem often referred to as the Cold Start problem. The precise amount of data needed varies by problem and environment. However, some properties of Interaction Networks allow us to estimate how much data is needed. Eagle et al. explored the structure of these student Interaction Networks and argued that networks could be interpreted as an empirical sample of student problem solving [EHIB15]. Students employing similar problem-solving approaches will explore overlapping areas of the Interaction Network. The more similar a group of students is, the smaller the overall explored area of the Interaction Network will ultimately be. Since we expect different populations of students to have different Interaction Networks, and different domains to require varying amounts of student data before feedback can be given, good metrics for the current and predicted quality of Interaction Networks are important.

In this work, we adapt Good-Turing frequency estimation to interaction level data to predict the size, growth, and “hintability” of Interaction Networks. Good-Turing frequency estimation estimates the probability of encountering an object of a hitherto unseen type, given the current number and frequency of observed objects [GS95]. It was originally developed by Alan Turing and his assistant I. J. Good for use in cryptography efforts during World War II. In our context, network states (vertices) are the object types, and the student interactions (edges) leading to those states are observations.

We present several metrics, derived from Good-Turing frequency estimation. Our hypotheses are that these metrics: \textbf{H1:} Predict the probability that a student interaction will result in a state which was not previously observed \textbf{H2:} Describe the proportion of the network that has been observed for a population \textbf{H3:} Predict the expected size and growth
when additional student data is added. **H4:** Provide a quantitative comparison of different state representations for their ability to represent greater proportions of the network. **H5:** Are useful for comparing different populations of users in how they explore the problem space.

Additionally, we use the metrics to explore the subset of the Interaction Network that is useful for providing automatically generated hints. This provides us with estimates of the size, growth, and coverage of automatically generated hints. We find that our metrics quickly become accurate after collecting a sample of about 10 students. This has value as a metric to compare the quality of the Interaction Networks, and will aid future researchers in determining an adequate state representation. We also show how two experimental groups, despite having the same amount of network coverage, have substantially different numbers of unique states. This supports previous work, suggesting that different populations of students produce different Interaction Networks [EHIB15], which has broad implications for generating hints as well as using the networks to evaluate student behavior.

### 5.1.1 Previous Work

Creation of adaptive educational programs is costly. This is, in part, because developing content for intelligent tutors requires multiple areas of expertise. Content experts and pedagogical experts must work with tutor developers to identify the skills students are applying and the associated feedback to deliver [Mur99]. In order to address the difficulty in authoring intelligent tutoring content, Barnes and Stamper built an approach called the Hint Factory to use student data to build a Markov Decision Process (MDP) of student problem-solving approaches to serve as a domain model for automatic hint generation [BS08b]. Hint Factory has been applied in tutoring systems and educational games across several domains [FDEO+09, PIHB14, EJBB13], and been shown to increase student retention in tutors [SEBC13].

Early work with the Hint Factory method used a Markov Decision Process constructed from students’ problem-solving attempts. Eagle and Barnes further developed this structure into a complex network representation of student interactions with the system, called an Interaction Network [EHIB15]. Complex networks are graphs or networks which contain non-trivial topological features unlikely to appear in simple or random networks. The Interaction Network representation can be used as a visualization of student work within tutors. The effectiveness of Interaction Networks as visualizations was shown by Johnson.
et al. who created a visualization tool *InVis* to aid instructors in analyzing student-tutor data [JEB13].

Other approaches to automated generation of feedback have attempted to condense similar solutions in order to address sparse data sets. One such approach converts solutions into a canonical form by strictly ordering the dependencies of statements in a program [RK14]. Another approach compares *linkage graphs* modelling how a program creates and modifies variables, with nested states created when a loop or branch appears in the code [JBS+12]. In the Andes physics tutor, students may ask for hints about how to proceed. Similarly to Hint Factory-based approaches, a solution graph representing possible correct solutions to the problem was used. However their solution space was explored procedurally rather than being derived from student data, and they used plan recognition to decide which of the problem derivations the student is working towards [VLS+05].

Interaction Networks are scale-free networks. This is a property of complex networks whose degree distribution is heavy-tailed, often a power law distribution. In practice, this means that a few vertices have degree that is much larger than the average, while many vertices have degree somewhat lower than average [EHIB15]. Eagle et al. argued that students with similar problem solving ability and preferences would travel into similar parts of the network, resulting in some states being more important to the problem than others [EHIB15]. Using these “hub” states, sub-regions of the network corresponding to high-level approaches to the problem were derived. These sub-regions captured problem-solving differences between two experimental groups [EB14a].

### 5.2 Methods and Materials

For the purposes of this work, we are using datasets from three different environments to build our Interaction Networks. Summaries of these datasets are found in Table 5.1. The first dataset is from the Deep Thought tutor, used in previous work by Stamper et al. [SEBC13]. This dataset was collected for a between groups experiment investigating the use of data-driven hints, so we split the dataset into two groups, DT1-C, the control group from that experiment, and DT1-H, the group that received hints. We selected this dataset to explore and evaluate H5.

The second dataset comes from the game BOTS. Here, we have the same students and interactions represented in two different ways: First, using *codestates* (the programs users
wrote) and second using worldstates (the output of those programs). The advantages and
disadvantages of these state representations were explored in previous work by Peddycord
and Hicks [PIHB14]. We split this dataset into two groups as well (BOTS-C and BOTS-W)
one for each state representation used. We selected this dataset for evaluation of H4.

Our third and largest dataset comes from an updated version of the Deep Thought
tutor, called Deep Thought 3. Unlike with the other datasets, Deep Thought 3 features an
AI problem selection component [MEB15]. This means that not all students will have had
access to all problems. In addition, there is a larger number of problems in this dataset.
We selected this dataset, as the larger number of problems effectively splits student data
across multiple networks. H1–H3 are relevant towards measuring the quality of networks
produced for new problems.

Table 5.1:  Dataset summary: the total number of students in the dataset, the number of
distinct problems, and the average number of students represented in each network.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Total N</th>
<th>Num Problems</th>
<th>Mean Net N</th>
</tr>
</thead>
<tbody>
<tr>
<td>DT1-H</td>
<td>203</td>
<td>11</td>
<td>83.73</td>
</tr>
<tr>
<td>DT1-C</td>
<td>203</td>
<td>11</td>
<td>63.82</td>
</tr>
<tr>
<td>DT3</td>
<td>341</td>
<td>59</td>
<td>78.41</td>
</tr>
<tr>
<td>BOTS-C</td>
<td>125</td>
<td>12</td>
<td>99.75</td>
</tr>
<tr>
<td>BOTS-W</td>
<td>125</td>
<td>12</td>
<td>99.75</td>
</tr>
</tbody>
</table>

5.2.1 Constructing an Interaction Network

An Interaction Network is a complex network representation of all observed student and
tutor interactions for a given problem in a game or tutoring system. To construct an Inter-
action Network for a problem, we collect the set of all solution attempts for that problem.
Each solution attempt is defined by a unique user identifier, as well as an ordered sequence
of interactions, where an interaction is defined as {initial state, action, resulting state},
from the start of the problem until the user solves the problem or exits the system. The
information contained in a state is sufficient to precisely recreate the tutor's interface at
each step. Similarly, an action is any user interaction which changes the state, and is defined
as \{\text{action name, pre-conditions, post-conditions}\}. In Deep Thought, for example, an action would be the logical axiom applied, the statements it was applied to, and the resulting derived statement. Figure 5.1 displays two Deep Thought interactions. The first interaction works forward from STEP0 to STEP1 with action \textit{SIMP} (simplification) applied to \((Z \land \lnot W)\) to derive \(\lnot W\). The second interaction works backward from STEP1 to STEP2 with action \textit{B - ADD} (backwards addition) applied to \((X \lor S)\) to derive the new, unjustified statement \(S\).

Figure 5.1: Example of state to state transitions within the Deep Thought (DT1) propositional logic tutoring system.

Once the data is collected, we use a \textit{state matching function} to combine similar states. In Deep Thought, we combine states that consist of all the same logic statements, regardless of the order in which those statements were derived. This way, the resulting state for a step STEP0, STEP1, or STEP2 in Figure 5.1 is the set of justified and unjustified statements in
each screenshot, regardless of the order that each statement was derived. In BOTS, two state matching functions were used: one which combined states based on the code in students’ programs, and another which instead used the output of those programs. Similarly, we use an action matching function to combine actions which result in similar states, while preserving the frequency of each observed interaction.

5.2.2 Providing Hints

Stamper and Barnes’ Hint Factory approach generates a next step Hint Policy by modeling student-tutor interactions as a Markov Decision Process [BS08b]. This has been adapted to work with Interaction Networks by using a Value Iteration algorithm on the states [EHIB15]. We generate a graph of all student interactions, combining identical states using a state matching function. Then, we calculate a fitness value for each state. We assign a positive value (100) to each goal state, that is a state configuration representing a solution to the problem. We assign an error cost (-5) for error states. We also assign a small cost to performing any action, which biases hint-selection towards shorter solutions. We then calculate fitness values $V(s)$ for each state $s$, where $R(s)$ is the initial fitness value for the state, $\gamma$ is a discount factor, and $P(s, s')$ is the observed frequency with which users in state $s$ take an action resulting in state $s'$. After this, we use value iteration [Bel57] to repeatedly assign each state a value based on its neighbors and action costs, weighted by frequency.

After applying this algorithm, we can provide a hint to guide the user toward the goal by selecting the child state with the best value. We can do this for any observed state, provided that a previous user has successfully solved the problem after visiting that state. In the original work with Hint Factory on the Deep Thought tutor, the algorithm was permitted to backtrack to an earlier state if it failed to find a hint from the current state. However, not all environments allow the user to backtrack and there are risks of the backtracking hints to provide irrelevant information. Because of this inconsistency across domains, we did not permit backtracking for the purposes of the comparisons in this paper.

We define a state, $S$ to be Hintable if $S$ lies on a path which ends at a goal state. We define the Hintable network to be the subset of the Interaction Network containing only Hintable states and edges between hintable states; That is, the induced subgraph on the set of Hintable states.
5.2.3 Cold Start Problem

Barnes and Stamper [BS08b] approached the question of how much data is needed to get a certain amount of overlap in student solution attempts by incrementally adding student attempts and measuring the step overlap over a large series of trials. This was done with the goal of producing automatically generated hints, and solution attempts that did not reach the goal were excluded. Peddycord et al. [PIHB14] used a similar technique to evaluate differences in overlap between two different Interaction Network state representations.

The “Cold Start problem” is an issue that arises in all data-driven systems. For early users of the system, predictions made are inaccurate or incomplete [SPUP02, SFHS07]. If there are insufficient data to compare to (not enough user ratings, or not enough student attempts) then the quality of the recommendations suffers and in some cases no recommendation can be provided. The term is commonly used in the field of collaborative filtering and recommender systems, but it can be used to describe three related issues, the “new user,” the “new item,” and the “new community” [BOHB12] Cold Start problems. The “new user” problem refers to the difficulty of making recommendations to a user who has performed no actions. The “new item” problem refers to the difficulty of suggesting users visit a newly added, unobserved state. The new community Cold Start problem refers to situations where not enough observations exist to make recommendations for new users. The “new community” definition corresponds most closely to the difficulty of generating hints for an entirely new problem in an intelligent tutoring system or educational game.

To measure our ability to address this problem, we add all interactions from a single student, one at a time, to the Interaction Network. This is in order to simulate the growth of the network. We repeat this process for each student, measuring the performance of our model each time. We measured the proportion of currently observed states to total observed states for the entire data set, as well as for the subset of states from which a goal is reachable. To control for ordering effects, we repeated this trial 1000 times using a different random ordering of students each time, and aggregated the results.

5.2.4 Good-Turing Network Estimation

We present a new method for estimating the size of the unobserved portion of a partially constructed Interaction Network. Our estimator makes use of Good-Turing frequency estimation [GS95]. Good-Turing frequency estimation estimates the probability of encoun-
tering an object of a hitherto unseen type, given the current number and frequency of
observed objects. It was originally developed by Alan Turing and his assistant I. J. Good for
use in cryptography efforts during World War II. Gale and Sampson revisited and simplified
the implementation [GS95]. In its original context, given a sample text from a vocabulary,
the Good-Turing Estimator will predict the probability that a new word selected from that
vocabulary will be one not previously observed.

The Good-Turing method of estimation uses the frequency distribution, the “frequency
of frequencies,” from the sample text in order to estimate the probability that a new word
will be of a given frequency. Based on this distribution, the probability of observing a new
word in an additional sample is estimated with the observed proportion of words with
frequency one. This estimate of unobserved words is used to adjust the probabilities of
encountering words of frequencies greater than one.

We adapt the Good-Turing Estimator to Interaction Networks by using the states with
an observed frequency of one to estimate the proportion of “frequency zero” states. Inter-
action Networks represent the observed interactions and therefore we also use this value to
estimate the probability that a new interaction will transition into a new state. We use \( P_0 \)
as the expected probability of the next observation being an unseen state. \( P_0 \) is estimated by:

\[
P_0 = \frac{N_1}{N}
\]  

(5.1)

Where \( N_1 \) is the total number of frequency 1 states, and \( N \) is the total number of interaction
observations. Since \( N_1 \) is the largest group of states, the observed value of \( N_1 \) is a reasonable
estimate of \( P_1 \). \( P_0 \) can then be used to smooth the estimation proportions of the other states.
The proportion of states with observed frequency \( r \) is found by:

\[
P_r = \frac{(r + 1)S(N_{r+1})}{N}
\]  

(5.2)

where \( S() \) is a smoothing function that adjusts the value for large values of \( r \) [GS95].

Our version of \( P_0 \) is the probability of encountering a new state (a state that currently
has a frequency of zero,) on a new interaction. We also interpret this as the proportion of
the network missing from the sample. We will refer to an interaction with a unobserved
state as having fallen off of the Interaction Network. We will use the complement of \( P_0 \) as
the estimate of network coverage, \( I_C \), the probability that a new interaction will remain on
the network: \( I_C = 1 - P_0 \).
The state space of the environment is the set of all possible state configurations. For both the BOTS game and the Deep Thought tutor the potential state space is infinite. For example, in the Deep Thought tutor a student can always use the addition rule to add new propositions to the state. However, as argued in Eagle et. al. [EHIB15], the actions that reasonable humans perform is only a small subset of the theoretical state space; the actions can also be different for different populations of humans. We will refer to this subset as the Reasonable State Space, with unreasonable being loosely defined as actions that we would not expect a human to take. An Interaction Network is an empirical sample of the problem solving behavior from a particular population, and is a subset of the state space of all possible reasonable behaviors. Therefore, our metrics $P_0$ and $I_C$ are estimates of how well the observed Interaction Network represents the reasonable state space.

5.3 Results

In order to evaluate the performance of the unobserved network estimator, $P_0$, and the network coverage estimator, $I_C$, for each problem in each of our 5 datasets we randomly added students from the sample, one at a time until all student data had been included. At each step, $T$, we recorded the values of our estimators using only the data that had been encountered up until then. This simulates a real world use-case, where additional students are added over time. We repeated this process 1000 times and averaged the results. Figure 5.2 shows the growth of unique states as students are added for the Interaction Networks generated by each problem (line) in each of the five datasets.

5.3.1 H1: Prediction of New States

In order to evaluate $P_0$ for the prediction of new states (states that are frequency = 0 on time $T_i$, but will be frequency = 1 on $T_{i+1}$). At each $T$ we add an additional student and compare the expected number of frequency 1 states, $E_{S1}$, vs. the observed number, $O_{S1}$. Across all five datasets, Figure 5.3 shows the differences between the expected and observed number of new states. The $P_0 \times Interaction$ prediction for new states follows closely with the observed number, the estimates increase in accuracy rapidly over the first ten students and are rarely off by more than a fraction of a state afterwards. Figure 5.4 shows the results of running this process on only the hintable portion of the Interaction Network for each data
5.3.2 H2: Network Coverage

We have defined network coverage $I_C$ as the proportion of interactions which lie within the previously observed network. Another interpretation is that $I_C$ is the probability of an interaction resulting in a state that has been previously observed. This value is the complement of $P_0$. Figure 5.5 and 5.7 display the results of network coverage and its growth as additional students are added.

5.3.3 H3: Predicting Future Network Size

In order to further evaluate the use of $P_0$ and $I_C$ we calculated a prediction for the final size of the network, given the number of students in each dataset, at each time stamp. The equation for this prediction is:

$$|V(IN)| = (NewSample \cdot P_0) + U_T. \quad (5.3)$$
Figure 5.3: The average absolute error between the estimated number of new states and the observed new states over the number of students for all problems in each of the four datasets. $P_0$ accurately predicts the observed values after roughly 10 students, rarely being off by more than one after that.

Where $|V(IN)|$ is the number of unique vertices (states) in the final network, $NewSample$ is the number of new interactions added, $P_0$ is the estimation of new states added, and $U_T$ is the number of unique states observed at time $T$. The results are averaged across all problems for each dataset and are presented in figures 5.8 and 5.9. This prediction rapidly improves and after roughly 20% of the sample is added, can accurately predict the final number of unique states for the network. This combined with the accuracy of $P_0$ reveals the short term and long term accuracy for the estimator.

5.3.4 H4: Comparing State Matching Functions

The network coverage metric, $I_C$, allows an easy method of estimating the differences in state matching functions and student network overlap. We can use $I_C$ with two potential matching functions, and get an estimate of the remaining network, to quickly compare different potential state representations as well as to find a state generalization that will allow for a desired amount of network coverage.
Figure 5.4: For the hintable states, the average difference between the estimated number of new states and the observed new states over the number of students for all problems in each of the four datasets. $P_0$ accurately predicts the observed values after roughly 10 students, rarely being off by more than one after that.

The estimate based on the above methods has proven useful for comparing State Matching functions to help determine which produces more relevant hints. Figure 5.6 shows the BOTS interface, with the user’s program (codestate) and the game world (worldstate) both illustrated. In previous work investigating the Cold Start problem on the BOTS data set, we measured "coverage" in terms of how much of the newly added test data was already present in the training set [HPIB14, PIHB14]. Compare this analysis to Figure 5.5 which shows the estimated probability that a student’s next action will result in an observed state, $I_C$. After 100 students, the probability that a student will generate a new codestate is still quite high, $P_0 > .25$. In comparison, after the same number of students, the probability of generating a new worldstate is extremely low, $P_0 < .02$. This result supports both our intuition and our results from the previous work, that students will continue to generate new codestates, but that these different codestates will collapse to previously observed worldstates.
Figure 5.5: The estimated network coverage $I_C$ for each of the 5 datasets, note the poor coverage for the BOTS-C dataset. The BOTS-W state is more general and has the much higher coverage.

5.3.5 H5: Comparing Populations

Samples from different populations have different resulting Interaction Networks. The size of the represented network can tell us about the similarity of student approaches in the sample. If students are more alike in the types of actions they perform, fewer students will be needed to achieve a similar amount of overlap. We can also see that adding students from a dissimilar population will not always increase estimated network coverage ($I_C$), and can potentially decrease it. This has implications about the importance of building hints for one population and applying it for another. In other work we have already shown that different groups are likely to visit different parts of the networks [EB14a]. Here we expand on that analysis by showing that the two groups, while having the same amount of network coverage, have a different number of unique states. Table 5.2 shows the results between the Hint group, which received hints on a subset of the problems, and the Control group which never received hints. This corresponds with results from Eagle et al. [EB14a] in which they uncovered significant differences in the student overall approaches. This result adds
Figure 5.6: An image of the main gameplay interface for BOTS. The left hand side of the screen shows the user’s program, used to derive code states. The right-hand side shows the game world, where the program output determines the world states.

Table 5.2: Different populations have different spread in problem exploration.

<table>
<thead>
<tr>
<th>Group</th>
<th>$P_0$</th>
<th>States</th>
<th>Interactions</th>
<th>$F_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hint</td>
<td>0.09</td>
<td>514.61</td>
<td>2709.84</td>
<td>250.09</td>
</tr>
<tr>
<td>Control</td>
<td>0.10</td>
<td>720.12</td>
<td>3904.92</td>
<td>340.00</td>
</tr>
</tbody>
</table>

to that an estimation of how complete each network was, revealing that additional data was not likely to change the result. It also shows some evidence for a trail blazing effect. When provided hints, students collectively explore a smaller area of the state space.

5.3.6 Estimating the effect of filtering

Visualizations must struggle with an "information to ink" ratio. There is a trade-off between displaying full information and overwhelming the viewer, and displaying only the most frequent states and potentially misleading the viewer by eliminating information. InVis, a visualization tool for exploring Interaction Networks allowed users to filter by frequency[JEB13]. We can use the Good-Turing Estimation to calculate the amount of in-
Figure 5.7: For the hintable network: the estimated network coverage $I_C$ for each of the 5 datasets. Even the lowest performing hint network BOTS-C reaches roughly 70% coverage by 100 students.

formation removed by filtering frequency of a certain degree. $P_0$ is the proportion of the network missing, $I_{C,r} = I_C - P_{1-r} + P_0$, where $r$ is a threshold value for removing low frequency states, and $P_{1-r}$ is the sum of $P_1$ through $P_r$. This should be a useful metric for visualizations for measuring the amount of network that is hidden by filtering. It is also useful to show that sometimes a large number of graphical elements can be removed, with only a small amount of interaction information lost.

5.4 Discussion

Good-Turing Estimation works well in the contexts of Interaction Networks. We were able to provide an easily calculable estimate of the proportion of the network not yet observed $P_0$. This value alone is a useful high level metric for the percentage of times a student interaction results in a previously unobserved state. The $P_0$ score for the hintable network is likewise an estimate of the probability that a student will “fall off” of the network from which we can provide feedback. Our network coverage metric $I_C$ allows a quick and easy to calculate
Figure 5.8: Prediction of total final number of states, as observed number of states increases. Note that for small $t$, the estimate is very high (up to 300% over prediction), but becomes fairly accurate after roughly 20% of the sample is measured.

method of comparing different state representations, as well as quantifying the difference. We believe that this metric can replace the commonly used cold start method of evaluating the “hintability” of a network. $I_C$ is also valuable to quickly gauge the applicability of a new domain to Interaction Networks. The majority of the calculations can be performed on the transactional data. The growth trends for our five datasets were often clear after only ten students.

Our network estimators also have implications given our previous theories on the network being a sample created from biased (non-random) walks on the problem-space, as the more homogeneous the biased walkers are, the faster the network will represent the population and the fewer additional states will be explored. We revisited our previous results [EB14a], and found that students with access to hints explored less overall unique states. This implies that the students were more similar to each other in terms of the types of actions and states they visited within the problem. Overall, this result supports the idea that different populations of students will have different Interaction Networks. The implications of this for generating hints are great. Building hints on one population might not work as
Figure 5.9: Prediction of total final number of goal states, as observed number of states increases. Note that for small $t$, the estimate is very high, but becomes an underestimate as $t$ increases. $P_0$ can predict the number of additional hintable states that can be added for a additional sample of data.

As you can see in figure 5.8, our estimator starts out drastically overestimating the number of unobserved states in the network. As we collect data, this eventually becomes a slight underestimate, eventually converging on the correct number of states. One explanation for why this might be the case is the method by which undiscovered states are added to the network. By using this model for our estimator, we are making an assumption that states are selected independently of one another. At the beginning, when data is sparse, this assumption is not particularly harmful, since undiscovered states are relatively common. However, as our dataset becomes richer, we underestimate the probability of adding an unobserved state because we do not take into account the effect of “trail-blazing” which increases the probability of adding additional unobserved states after the first. Eagle and
Barnes found that Interaction Networks had properties of scale-free networks. [EHIB15]. In particular, their degree distributions follow a power law, with a few vertices having much higher degree than the average for the network. It is likely that taking into account the scale-free and hierarchical nature of the networks will provide methods to improve on our estimators.

5.5 Conclusions and Future Work

We have adapted Good-Turing frequency estimation for use with networks built from student-tutor interactions. We found that the estimator for the missing proportion of the network $P_0$ was accurate in predicting the number of new states discovered with new data. We also found that we could accurately measure network coverage with $I_C$ for both the regular network, as well as the network of hintable states. This provides us with a metric to compare different state representations as well as determine the suitability of Interaction Network methods to different tutoring environments. We were also able to use these metrics to provide accurate predictions for the size of networks expected given more data samples, which will be useful for predicting the amount of additional data needed to provide a desired amount of hintable network coverage. Finally, we used the estimate of network coverage to compare different student populations to show that the addition of hints in one environment had an effect on the number of states explored by students.

Future work will include expanding on these global measures of the network and exploring local measures of coverage. Rather than compute coverage for the entire network we can use methods such as approach map regioning [EB14a] to find meaningful sub-networks and calculate the metrics for those. The region level values of $P_0$ can estimate the “riskyness” of certain approaches to the problem. The $I_C$ metric can direct attention to parts of the network that are not well explored, perhaps allowing additional hints to be obtained by starting advanced users in those areas.
Chapter 6

Conclusions

In this dissertation we have shown several successful results from the modeling of student interactions in educational problem solving environments as networks. The development of new evaluation techniques is inherently related to deriving insight from this kind of complex data. We have used multiple methods of Educational Data Science to describe a full story of student problem solving within several educational environments.

In Chapter 2 we were able to greatly expand our previous understanding about how hints affected students in terms of engagement time and overall tutor performance. Our previous understanding was that the control group was 3.6 times more likely to drop midway though the tutor when compared to the hint group. We also knew that the hint group out performed the control group. However, we did not fully understand why the control group was more likely to quit. Our hypothesis was that they would become stuck or frustrated and be more likely to dropout of the tutor assignment.

Using survival analysis we were able to find that the hint group completed the assignment in 55% of the time when compared to the control group. However, there was no difference in the overall time spent engaged with the tutor. We concluded that both groups were willing to spend a certain amount of time on the assignment, and that access to automatically generated hints enabled students to complete more problems in the same amount of time. This is a much richer understanding of the differences in effects between the two groups than traditional methods provide.

An additional contribution of this analysis was a demonstration of the application of survival analysis techniques to data from intelligent tutoring systems. The survival function also allows us to make predictions on how much time is needed for tutor completion, both
for teacher planning and student feedback. These results suggest that survival analysis is a powerful toolbox for investigating the impact of interventions on learning efficiency while accounting for performance, duration, and dropout.

This leads us to the next major question. We know that there are some significant ways in which these two groups had different outcomes. How can we find the ways in which the two groups differed in their overall problem-solving strategies? For this analysis we made use of Interaction Networks and network community clustering techniques.

In Chapter 3, we derived a high-level view of student problem-solving strategy that we call a Approach Map. These maps serve as a visualization method, as they greatly reduce the amount of visual space needed to convey information about how students solved problems. They also allow us to perform quantitative analysis to show statistically significant differences in how students solved problems between groups. We have shown that we can use Approach Maps annotated with frequencies of visits by two groups to identify regions where a particular study group was over-represented. This allowed us to examine the approaches each group took to solving each proof. As we predicted, the automatically generated hints seemed to direct the students in the hint group down a common path, and we were able to detect this with the Approach Maps. Interestingly, even in problem 1.5, where neither group had hints, the hint group still showed a preference for working forwards, providing some evidence for a persistent effect of the hints. Analyzing Approach Maps also facilitated another important discovery that control group tended enter and remain in unproductive (or buggy) regions. These observed differences help explain how the automatically-generated hints produced the difference in tutor performance and retention in the 2009 Deep Thought study. Our investigations suggest that the patterns of behavior exhibited by students do result in meaningful regions of the solution attempt search space. We believe that, since the algorithms we applied to derive Approach Maps work on general graphs, we may be able to apply Approach Maps to understand problem-solving in domains where students solve open-ended problems in a procedural way.

These results have painted a picture of how hints affect student performance and behavior in the Deep Thought Tutoring environment. That led us to investigate how to interpret the overall networks. We are also interested in what the general properties of these networks are, and which types of other educational environments would benefit from Interaction Networks.

In Chapter 4, we explore alternative applications of Interaction Networks in multiple
environments as well as the assortativity and scale-free nature of the networks. We outline a theory of how shared bias in choosing actions results in common paths and solution states, and if different populations of walkers have different bias we can expect to see them have different common paths and states. Interaction Networks provide a means for exploring problem solving behavior in rule-using tutors, and looking for between-group differences. Several diverse domains have had success using Interaction Networks to model student problem solving, such as puzzle games, propositional logic tutors, and programming tutors. Features of rule-using interactive tutors and student individual differences (bias) in action selection can be compared across educational domains by describing the shape of the network. Interaction Networks tend to have power law degree distributions, fairly high measures of self-similarity, $S(g)$, scale-free scores. Even when the theoretical problem-space seems intractable, Interaction Networks created from observed transaction logs can provide a surprising amount of useful information.

The scale-free nature of Interaction Networks results in the ability to start providing hints, even in large problem domains, with less data than might be expected. We argue that this is due to the underlying similarities in how humans would approach the problems. That is, while the number of possible next-step actions could be infinite the number of next-step actions that students will actually do are limited. This helps explain why Interaction Networks are good for generating hints, but also supports the network visualization as a representation of student work.

Our final questions are related to the amount of data needed to create reasonable views of the problem-solving networks. Evaluating how well the network represents the students problem solving behavior is important for producing hints and interpreting visualizations. In Chapter 5, we have adapted Good-Turing frequency estimation for use with networks built from student-tutor interactions. We found that the estimator for the missing proportion of the network $P_0$ was accurate in predicting the number of new states discovered with new data. We also found that we could accurately measure network coverage with $I_C$ for both the regular network, as well as the network of hintable states. This provides us with a metric to compare different state representations as well as determine the suitability of Interaction Network methods to different tutoring environments. We were also able to use these metrics to provide accurate predictions for the size of networks expected given more data samples, which will be useful for predicting the amount of additional data needed to provide a desired amount of hintable network coverage. Finally, we used the estimate of
network coverage to compare different student populations to show that the addition of hints in one environment had an effect on the number of states explored by students.
REFERENCES


97


