AN EVALUATION OF DECISION-THEORETIC TUTORIAL ACTION SELECTION

by

Robert Charles Murray

B.S. in Computer Science, Old Dominion University, 1992

M.S. in Intelligent Systems, University of Pittsburgh, 1999

Submitted to the Graduate Faculty of

the School of Arts and Sciences in partial fulfillment

of the requirements for the degree of
Doctor of Philosophy

University of Pittsburgh

2005

UNIVERSITY OF PITTSBURGH

SCHOOL OF ARTS AND SCIENCES

This dissertation was presented

by

Robert Charles Murray

It was defended on

July 15, 2005

and approved by

Dr. Kevin D. Ashley, Professor, School of Law

Dr. Gregory F. Cooper, Associate Professor, School of Medicine

Dr. Marek J. Druzdzel, Associate Professor, School of Information Sciences

Dissertation Director: Dr. Kurt A. VanLehn, Professor, Department of Computer Science

AN EVALUATION OF DECISION-THEORETIC TUTORIAL ACTION SELECTION

Robert Charles Murray, PhD

University of Pittsburgh, 2005

A novel decision-theoretic architecture for intelligent tutoring systems, *DT Tutor* (DT), was fleshed out into a complete ITS and evaluated. DT uses a dynamic decision network to probabilistically look ahead to anticipate how its tutorial actions will influence the student and other aspects of the tutorial state. It weighs its preferences regarding multiple competing objectives by the probabilities that they will occur and then selects the tutorial action with maximum expected utility.

The evaluation was conducted in two phases. First, logs were recorded from interactions of students with a Random Tutor (RT) that was identical to DT except that it selected randomly from relevant tutorial actions. The logs were used to learn many of DT's key probabilities for its model of the tutorial state. Second, the logs were replayed to record the actions that DT and a Fixed-Policy Tutor (FT) would select for a large sample of scenarios. FT was identical to DT except that it selected tutorial actions by emulating the fixed policies of Cognitive Tutors, which are theoretically based, widely used, and highly effective. The possible action selections for each scenario were rated by a panel of judges who were skilled human tutors. The main hypotheses tested were that DT's action selections would be rated higher than FT's and higher than RT's. This was the first comparison of a decision-theoretic tutor with a non-trivial competitor.

DT was rated higher than FT overall and for all subsets of scenarios except help requests, for which it was rated equally. DT was also rated much higher than RT. The judges preferred that the tutors provide proactive help and the study design permitted this information to be put to use right away to develop and evaluate enhanced versions of DT and FT. The enhanced versions of DT and FT were rated about equally and higher than non-enhanced DT except on help requests. The variability of the actions selected by both non-enhanced and enhanced versions of DT demonstrated more sensitivity to the tutorial state than the actions selected by non-enhanced and enhanced versions of FT.

TABLE OF CONTENTS

PREFACE x
1.0 INTRODUCTION 1
1.1 RESEARCH PROBLEMS 5
1.1.1 Hypothesis 1: Decision-theoretic can be better than random tutoring 6
1.1.2 Hypothesis 2: Decision-theoretic can be better than fixed-policy tutoring 6
1.2 GENERAL APPROACH 7
1.2.1 A decision-theoretic approach 7
1.2.2 DT Tutor's general architecture 9
1.2.2.1 High-level overview of DT Tutor's architecture 10
2.0 SCIENTIFIC CONTRIBUTIONS AND RELATED WORK 12
2.1 COMPARATIVELY EVALUATE DECISION-THEORETIC TUTORING 12
2.2 A NOVEL ARCHITECTURE FOR TUTORIAL ACTION SELECTION 12
2.2.1 Making decisions 13
2.2.2 Deciding the type of tutorial action as well as the topic 14
2.2.3 Modeling change over time 15
2.2.4 Which attributes to model 16
2.2.4.1 Modeling observable and unobservable attributes 17
2.2.4.2 Modeling the user's focus of attention 18
2.2.4.3 Modeling the user's affective state 19
2.2.4.4 Predicting and learning from the user's actions 19
3.0 TECHNICAL APPROACH 21
3.1 THE DOMAIN EXPERT 21
3.1.1 The calculus related-rates problem domain 21
3.1.2 Problem solutions generated by the domain expert 24
3.2 DECISION-THEORETIC ACTION SELECTION ENGINE 26
3.2.1 Tutor Action Cycle Network in more detail 27
3.2.2 Problem solution graph structure 28
3.2.3 Tutor Action Nodes 28
3.2.4 Student Action Nodes 28
3.2.5 Student Focus Subnetworks 29
3.2.5.1 Focus evolution and aging 31
3.2.6 Student Knowledge Subnetworks 32
3.2.7 Discourse State subnetworks 33
3.2.8 Student Independence nodes to model affect 34
3.2.9 Student Help Style nodes 35
3.2.10 Utility subnetwork 35
3.2.11 Filter nodes 36
3.2.12 Rule-based conditional probability table creation 36
3.3 THE STUDENT INTERFACE 39
3.3.1 Reification of goal structure in the Goals Window 39
3.3.1.1 An extended example of student interface displays 39

3.3.1.2 Immediate flag feedback 43
3.3.1.3 Correspondence between dialog windows and types of rules 43
3.3.1.4 Help messages 44
4.0 EVALUATION: DATA COLLECTION PHASE 45
4.1 GOALS OF THE DATA COLLECTION AND TUNING PHASE 45
4.2 DESIGN OF THE DATA COLLECTION EXPERIMENT 46
4.2.1 Subjects 46
4.2.1.1 Printed materials 46
4.2.1.2 The Random Tutor 47
4.2.2 Procedure 48
4.3 PARTITIONING INTO TRAINING AND TEST DATA SETS 49
4.4 LEARNING PROBABILITIES EMPIRICALLY 49
4.4.1 Identifying student help style, including help abuse 51
4.4.2 Learning prior probabilities 54
4.4.3 Learning conditional probabilities 57
4.4.3.1 Learning conditional probabilities related to unobservable variables 57
4.4.3.2 Learning conditional probabilities with sparse data 57
4.4.3.3 Estimating rule knowledge as it changes over time 59
4.4.3.4 Estimating p(guess) and p(slip) 60
4.4.3.5 Estimating effects of help on student rule knowledge 61
4.4.3.6 Estimating effects of help on student step knowledge when rule unknown 63
4.4.3.7 Estimating effects of help on student step knowledge when rule known 66
4.4.3.8 Estimating conditional probabilities for Student Action Topic 66
4.4.3.9 Estimating conditional probabilities for student action type 68
4.5 TUNING UTILITIES 68
4.5.1 Utilities for each tutorial state attribute 69
4.5.1.1 Discourse coherence 70
4.5.1.2 Discourse relevance 70
4.5.1.3 Student rule knowledge 71
4.5.1.4 Student problem-solving progress 71
4.5.1.5 Student help style 71
4.5.1.6 Student independence 71
4.5.1.7 Tutor response preferences 72
4.5.2 Multiattribute utility function 73
5.0 EVALUATION: ASSESSMENT PHASE 77
5.1 GOALS OF THE ASSESSMENT PHASE 77
5.2 DESIGN OF THE ASSESSMENT PHASE EXPERIMENT 78
5.2.1 Subjects 78
5.2.2 Materials 78
5.2.2.1 Printed materials 78
5.2.2.2 Scenario types and stratified sampling 80
5.2.2.3 The Fixed-Policy Tutor 82
5.2.3 Procedure 83
5.3 FAST RESPONSE TIME BY LIMITING PROBLEM SOLUTIONS 84
5.4 DISTRIBUTIONS OF RESPONSES SELECTED BY THE TUTORS 85
5.4.1 The Fixed-Policy Tutor's overall distribution of response selections 86
5.4.2 DT Tutor's overall distribution of response selections 87
5.4.2.1 DT Tutor's large number of teach responses 87
5.4.2.2 DT Tutor's small number of hint responses 88
5.4.3 First-message-opportunity scenarios: pretest-wrong, pretest-right 88
5.4.4 The tutors' response distributions for help requests 90

5.4.4.1 FT and DT response distributions for FMO help requests 91
5.4.5 The tutors' response distributions for errors 91
5.4.5.1 FT and DT response distributions for FMO errors 92
5.4.6 The tutors' response distributions for step starts 93
5.4.6.1 FT and DT response distributions for FMO step starts 93
5.5 THE JUDGES' EVALUATIONS 94
5.5.1 The judges' comments 94
5.5.2 The judges' individual ratings 95
5.5.2.1 Judge x Tutor ratings 96
5.5.2.2 Scenario Type x Tutor Ratings 98
5.5.2.3 Scenario Type x Tutor Interaction for Errors 99
5.5.2.4 Scenario Type x Tutor Interaction for Step Starts 102
5.5.2.5 Judge x Tutor Interaction for Judge 3 103
5.5.3 Composite judges' ratings 104
5.5.3.1 Similarities among judges' ratings for all responses 104
5.5.3.2 Contrasts in ratings for subsets of scenarios 105
5.5.3.3 Composite judges' ratings use the median rating for each response 107
5.6 COMPARING COMPOSITE RATINGS OF THE TUTORS 108
5.6.1 Composite ratings: Random Tutor vs. Decision-Theoretic Tutor 110
5.6.2 Composite ratings: Fixed-Policy Tutor vs. Decision-Theoretic Tutor 111
5.6.2.1 Decision-Theoretic Tutor vs. Fixed-Policy Tutor: Help requests 111
5.6.2.2 Decision-Theoretic Tutor vs. Fixed-Policy Tutor: Errors 111
5.6.2.3 Decision-Theoretic Tutor vs. Fixed-Policy Tutor: Step starts 112
5.6.2.4 Decision-Theoretic Tutor vs. Fixed-Policy Tutor: FMO scenarios 113
5.7 COMPARING ENHANCED VERSIONS OF THE TUTORS: DTe vs. FTe 114
5.7.1 DTe vs. FTe: first-message-opportunity scenarios 117
6.0 DISCUSSION 119
6.1 LEARNING PROBABILITIES 119
6.1.1 Techniques for learning probabilities 119
6.1.2 Learning about students' rule knowledge in the presence of help abuse 121
6.1.3 Learning with sparse data 122
6.1.4 Some surprises in the learned probabilities 123
6.1.5 Expected patterns in the learned probabilities 124
6.2 TUNING UTILITIES 125
6.3 RANDOM VS. DECISION-THEORETIC: SUPPORT FOR HYPOTHESIS 1 128
6.4 FIXED-POLICY VS. DECISION-THEORETIC TUTORING 128
6.4.1 Fixed-Policy Tutor vs. Decision-Theoretic Tutor: Support for Hypothesis 2 128
6.4.2 FT vs. DT: Adapting the tutor's response type to the situation 129
6.4.3 Examples of judges' preferences for more explicit help than FT would select 131
6.4.3.1 Example of preferences for more explicit help for a help request 131
6.4.3.2 Example of preferences for more explicit help for an error 132
6.4.3.3 Example of preferences for more explicit help for a start step scenario 133
6.4.4 FT vs. DT: The role of proactive help 134
6.4.4.1 Effects of enhancing a fixed policy 135
6.4.5 Should you choose fixed-policy or decision-theoretic tutoring? 137
6.5 SHOULD COMPUTER TUTORS PROVIDE PROACTIVE HELP? 138
6.6 LIMITATIONS AND FUTURE WORK 140
6.6.1 Limitations of decision-theoretic approaches 140
6.6.2 Limitations of DT Tutor 141
6.6.3 Limitations of the current study 143
6.6.3.1 The method of comparing the tutors 143

6.6.3.2 Some other limitations of the current study 147
6.7 CONCLUSIONS 148
6.7.1 A decision-theoretic architecture for making tutorial action selections 148
6.7.2 Development and assessment of a decision-theoretic tutor 150
APPENDIX A. Calculus Tutor Tutorial 154
APPENDIX B. Posttest 167
APPENDIX C. Calculus Tutor Tips 180
APPENDIX D. Expanded Problem Screen Shots with Goal Numbers 183
APPENDIX E. Screen Shots of Dialog Windows 189
APPENDIX F. Sample Scenario Description 195
APPENDIX G. Sample Help Messages 200
BIBLIOGRAPHY 205

LIST OF TABLES

Table 4.1: Help abusers and their scores on help abuse measures 54
Table 4.2: Learned prior probabilities for Calculus Tutor domain rules 55
Table 4.3: Learned prior probabilities by rule type 56
Table 4.4: P(guess) and p(slip) by rule type and help style 60
Table 4.5: P(rule known) by rule type, help type and student help style 63
Table 4.6: P(step known) by rule type, help type, rule known, and student help style 65
Table 4.7: P(Cancel) and p(Help!) for Student Action Topic 67
Table 4.8: P(Help Request) and p(Error) for Student Action Type 68
Table 4.9: Utilities for tutor response preference 73
Table 4.10: Weights for linearly-additive multiattribute utility function 74
Table 5.1: Distributions of response types for all scenarios, percentages 86
Table 5.2: FMO responses for all scenarios, percentages: pretest wrong & right, FT & DT 90
Table 5.3: Distributions of response types for help requests, percentages 90
Table 5.4: FMO responses for help requests, percentages: pretest wrong & right, FT & DT 91
Table 5.5: Distributions of response types for errors, percentages 91
Table 5.6: FMO responses for errors, percentages: pretest wrong & right, FT & DT 92
Table 5.7: Distributions of response types for step starts, percentages 93
Table 5.8: FMO responses for step starts, percentages: pretest wrong & right, FT & DT 93
Table 5.9: Tutor x Judge x Scenario Type, repeated-measures ANOVA 96
Table 5.10: Tutor x Judge, mean ratings: RT vs. FT vs. DT 97
Table 5.11: Tutor x Scenario Type, mean ratings: RT vs. FT vs. DT 98
Table 5.12: Error scenario ratings, means by each judge and overall 100
Table 5.13: Step start scenario ratings, means by each judge and overall 102
Table 5.14: Agreement among judges, all scenarios 105
Table 5.15: Agreement among judges, step start scenarios 106
Table 5.16: Agreement among judges, first-message-opportunity help requests 107
Table 5.17: Tutor x Scenario Type, repeated-measures ANOVA: RT vs. FT vs. DT 108
Table 5.18: Tutor x Scenario Type, composite ratings: RT vs. FT vs. DT 109
Table 5.19: Tutor x Scenario Type, composite ratings, paired t-tests: RT vs. DT, FT vs. DT 110
Table 5.20: Step start scenario composite ratings by response type 113
Table 5.21: FMO scenarios, composite ratings, paired t-tests: FT vs. DT 113
Table 5.22: Tutor x Scenario Type, repeated-measures ANOVA: FTe vs. DTe 115
Table 5.23: Tutor x Scenario Type, composite ratings: RT vs. FT vs. DT vs. FTe vs. DTe 116
Table 5.24: Tutor x Scenario Type, composite ratings, t-tests: FTe vs. DT, FTe vs. DTe 117
Table A1: Equation Form Examples 162
Table A2: Sample equation forms for the operators 163
Table A3: Operator selection heuristics: differentiate, flip derivative, integrate, & restate 165
Table A4: 2 out of 3 heuristic for selecting operator(s) for evaluate operand 1 166

LIST OF FIGURES

Figure 1.1: Tutor action cycle network, high-level overview 10
Figure 3.1: Problem solution graph for problem P1 26
Figure 3.2: DT Tutor's Tutor Action Cycle Network 27
Figure 3.3: Student Focus subnetworks in TACN 30
Figure 3.4: Student Knowledge subnetworks in TACN 32
Figure 3.5: Simplified CPT for a Student Knowledge step node 37
Figure 4.1: Text discouraging help requests on 27 of 60 tip sheets 47
Figure 5.1: Tutor x Judge, mean ratings: RT vs. FT vs. DT 97
Figure 5.2: Tutor x Scenario Type, mean ratings: RT vs. FT vs. DT 99
Figure 5.3: First error scenario ratings by each judge: RT vs. FT 101
Figure 5.4: Subsequent error scenario ratings by each judge: RT vs. FT 101
Figure 5.5: Step start scenario ratings by each judge: RT vs. FT 103
Figure 5.6: Tutor x Scenario Type, Judge 3: RT vs. FT 104
Figure 5.7: Tutor x Scenario Type, composite ratings: RT vs. FT vs. DT 109
Figure 5.8: Tutor x Scenario Type, composite ratings: RT vs. FT vs. DT vs. FTe vs. DTe 116
Figure A1: Calculus Tutor screen shot 155

PREFACE

Kurt VanLehn, my research advisor, has been the rock upon which my research development has been founded. From research ideas, to constant support, to spot-on advice from a deep wellspring of knowledge and experience, Kurt has left an indelible mark on my research and my approach. Kurt's research group has been influential as well, particularly regarding the use of Bayesian techniques for student modeling. Notables, out of many, include Pamela Jordan, Cristina Conati, Abigail Gertner, and Patricia Albacete, who provided much appreciated encouragement early in my graduate career.

For my committee, I chose faculty whom I respect and admire. Greg Cooper can be counted upon for clear insight, making simple what once seemed complex, always delivered with friendly professionalism in a timely manner. Marek Druzdzel has been a mentor and an inspiration to me as well as many others, and I appreciated the chance to work with his Decision Systems Laboratory. The core of the decision-theoretic portion of DT Tutor's implementation is based on the SMILE reasoning engine for graphical probabilistic models contributed to the community by the Decision Systems Laboratory, University of Pittsburgh (http://www.sis.pitt.edu/~dsl). Kevin Ashley was instrumental in bringing me to the Intelligent Systems Program. His enthusiasm, his brilliant use of language, and his impeccable manner are qualities that I will always admire.

I was fortunate to have the chance to work with Jack Mostow and his Project LISTEN at Carnegie Mellon University to develop a prototype reading application for DT Tutor. Jack is a truly stimulating individual. Rarely have I had so much fun while working.

My practical interest in artificial intelligence started with Dennis Ray at Old Dominion University, whose difficult but fun AI-related classes motivated me to switch my major from Psychology to Computer Science. Jim Schwing, also at ODU, likewise inspired me and helped pave the way.

I could not have reached this point without my family. My wife, Raeann, has taken on much more than her share of home duties, cheerfully (mostly) endured a steep drop in income while I returned to school, and delayed some of her career goals so that I could pursue mine. I dedicate this dissertation to her. My young daughter, Allison, has had to understand my absences as I have missed irreplaceable events. My mother, Sally Van Nostrand, was my original inspiration and has provided unwavering support, all the while believing in me.

I gratefully acknowledge financial support for this research from two Mellon Fellowships, the Intelligent Systems Program and the School of Arts and Sciences at the University of Pittsburgh, the Office of Naval Research, and the National Science Foundation.

1.0 INTRODUCTION

Intelligent tutoring systems (ITSs) that coach students as they attempt tasks often emulate the turn taking observed in human tutorial dialog (Graesser et al., 1995; Merrill et al., 1995). Student turns usually consist of attempting a task step or asking for help. The tutor's main task can be seen as deciding what action to take on its turn, or *tutorial action selection*. Selecting tutorial actions involves inherent difficulties.

A significant source of difficulty is that the tutor is uncertain about the student's internal state because it is not directly observable. This includes both (1) the student's cognitive state, such as task-related knowledge, mental inferences, and focus of attention; and (2) the student's affective state. Compounding the difficulty, the student's internal state changes over the course of a tutoring session as the student interacts with the tutor, learns, attempts task steps, and experiences successes and failures. Furthermore, the tutor is uncertain about the effects of the tutor's actions on the student's internal state. The tutor may also be uncertain about other aspects of the tutorial state, such as task progress if it is not entirely observable. To glean uncertain information about the tutorial state and how it is influenced by both tutor and student actions, a tutor must make inferences based on observable phenomena and guided by the tutor's beliefs about the situation. In recent years, many ITSs (see, e.g., Jameson, 1996) have modeled the tutor's uncertainty in terms of probability using Bayesian techniques (Pearl, 1988) for mathematically sound yet relatively efficient inference.

Another significant difficulty is that just what constitutes effective tutorial action depends upon the tutor's objectives and priorities among them. The tutor's objectives are likely to include student-centered objectives such as increasing the student's knowledge, helping the student complete tasks and bolstering the student's affective state, along with other objectives such as being a cooperative discourse partner. It may not be possible to maximize attainment of all the tutor's objectives over the course of a tutoring session. For instance, the tutor may want to maximize both the student's knowledge and task progress, but focusing on increasing the student's knowledge could take time away from helping the student complete tasks, and conversely, helping the student complete tasks (e.g., by telling the student exactly how to do task steps) could take time away from increasing the student's task-related knowledge. When the tutor has competing objectives, the effectiveness of the tutorial action alternatives depends

upon the tutor's priorities. Tutors must often strike a "delicate balance" among multiple competing objectives (Lepper et al., 1993; Merrill et al., 1992, p.280; Reye, 1995).

Decision theory extends probability theory by considering, in addition to the decision-maker's (e.g., the tutor's) uncertainty, the decision-maker's objectives and priorities in terms of utility as a rational basis for making decisions (Russell & Norvig, 1995). This work features a decision-theoretic approach for tutoring, called *DT Tutor*, that involves explicitly looking ahead to anticipate how the tutorial action alternatives will influence the student and other aspects of the tutorial state. In broad terms, for each tutorial action alternative, the tutor looks ahead to compute (1) the probability of every possible outcome of that tutorial action, (2) the utility of each possible outcome relative to the tutor's objectives and priorities, and then (3) the alternative's *expected utility* by weighing the utility of each possible outcome by the probability that it will occur. The tutor then selects the tutorial action with maximum expected utility. This approach unifies considerations regarding the tutor's objectives and priorities, the tutor's uncertain beliefs about the changing tutorial state, and the tutor's uncertain beliefs about the effects of tutorial actions. One advantage of a decision-theoretic approach is the capability to balance multiple tutorial objectives in a principled way when computing the utility of each outcome. DT Tutor leverages this capability by considering multiple objectives related to a rich model of tutorial state outcomes such as the student's knowledge, focus of attention, and affective state, along with task progress and the discourse state. Few other tutoring systems have modeled any tutorial state attribute other than the student's knowledge probabilistically, let alone all in combination.

While many ITSs and user-modeling systems have used Bayesian networks for reasoning under uncertainty (for examples, see, e.g., Horvitz et al., 1998; Jameson, 1996), decision-theoretic approaches for selecting tutorial actions remain novel. Reye (1995) proposed a decision-theoretic approach for ITSs which considered uncertainty about the student's knowledge and next action, as well as multiple concurrent objectives. That paper mentioned an SQL tutor in progress but left unspecified many details of the both the approach and its concrete implementation. Murray and VanLehn (2000) presented DT Tutor's approach in the context of a prototype tutorial action selection engine for calculus related-rates problems. The only other decision-theoretic ITS work of which the author is aware are the recent contrasting approaches embodied by CAPIT (Mayo & Mitrovic, 2001) and iTutor (Pek, 2003). DT Tutor appears to be unique among implemented decision-theoretic ITS approaches in several respects, including considering a rich model of the tutorial state to adapt how it responds to the student from turn to turn – i.e., to decide what *type* of help (e.g., what type of hint) to provide in addition to the help *topic*. Decision-theoretic methods have been used more often in the user modeling community. Conati's proposal (e.g., 2002) for educational games is probably the closest to DT Tutor in that it too uses a dynamic decision network (defined below) designed to balance objectives regarding the user's cognitive and affective

states, although versions published so far model only the user's affective state. Chapter 2.0 of this dissertation, Scientific Contributions and Related Work, further describes related work in the context of the scientific contributions of this research.

There are at least three ways that a decision-theoretic approach to tutorial action selection could fail. First is the *knowledge representation problem*: For real-world tutorial contexts, it might not be feasible to decision-theoretically represent the tutorial state with enough fidelity. The real tutorial state is of course hopelessly complex. It includes the student's knowledge (which is changing, we hope), the student's focus of attention, the student's affect, progress on the tutorial task, the tutor's domain knowledge and pedagogical objectives, the discourse history, etc. Some of these tutorial state attributes can be approximated. For instance, student affect could be represented by a variable with just two values: *high* or *low*. However, such coarse approximations may make it impossible to predict future tutorial states and rate their utilities accurately enough. Moreover, to build a decision-theoretic system on the scale of DT Tutor, thousands of probabilities and utilities must be specified. It is probably not necessary for all of these values to be precise (e.g., Henrion et al., 1996), but they must be accurate enough for DT Tutor to have a sufficiently realistic model of the tutorial state to make effective decisions. Thus, one challenge is to develop a representation of the tutorial state that is accurate enough to make good decisions but not so complex that it is computationally infeasible.

Another potential point of failure is the *real-time inference requirement*: A tutor must select actions quickly enough to keep the student engaged. DT Tutor's decision-theoretic approach uses an extension of Bayesian networks. Its networks have many uninstantiated variables, are multiply-connected, and can be large. Such characteristics, which appear to be necessary for many complex, real-world domains (Cooper, 1990), can make probabilistic inference NP-hard (Cooper, 1990; Dagum & Luby, 1993) and thus can make real-time inference challenging.

A third way that a decision-theoretic approach could fail is in its *tutorial action selection capabilities*. By definition, a decision-theoretic approach should be able to select actions rationally. However, DT Tutor's architecture is novel. Its networks can be complex because they model multiple outcomes as they change over time, and they may include hundreds of nodes and thousands of probabilities and utilities. Yet these networks are just approximations of the tutorial state. At the outset of this research it was unclear just what action selection capabilities would emerge from such a complex yet approximate representation, or how these capabilities would compare to other approaches. Previously, decision-theoretic approaches had been compared only to random action selection (Mayo & Mitrovic, 2001) and to no tutoring at all (Mayo & Mitrovic, 2001; Pek, 2003). Even if DT Tutor's action selections should prove to be better than those selected by unintelligent approaches (e.g., random or no tutoring it all), it is still important to know how a decision-theoretic approach compares to more reasoned

approaches. DT Tutor's approach is computationally intensive and so it must provide some benefit over less computationally intensive approaches in order to be worthwhile.

In work with DT Tutor prior to the current study, the feasibility of its approach was tested with prototype action selection engines for tutoring diverse domains: calculus related-rates problems and reading aloud. These action selection engines did not yet have modern user interfaces and so they could not be used to test DT Tutor's action selection capabilities for tutoring with real students. However, the knowledge representation problem was addressed by encoding DT Tutor's approach for the two domains (e.g., Murray & VanLehn, 2000; Murray et al., 2001b). To see whether the real-time inference requirement could be met, the response times of both action selection engines were tested on a variety of problem sizes with improving but still modest results (Murray & VanLehn, 2000; Murray et al., 2004). The remaining problem, tutorial action selection capabilities, was partially addressed by presenting both action selection engines with a variety of simulated scenarios and checking to see whether the actions selected were both (1) rational in light of the system's probabilities and utilities, and (2) comparable to the actions of human or other automated tutors (Murray & VanLehn, 2000; Murray et al., 2004). Section 1.2 introduces DT Tutor's solutions to these challenges, beginning with its general decision-theoretic approach and concluding with an overview of the architecture of its action selection engine.

The purpose of the current study was to assess DT Tutor's action selection capabilities for situations involving real students and to compare these capabilities with a competing approach. In order to assess DT Tutor's capabilities for situations involving real students, it was first necessary to flesh out DT Tutor by developing a modern user interface for calculus related-rates problems along with other components necessary for a complete ITS. A research plan was developed not only to comparatively assess DT Tutor, but also to further address the knowledge representation problem by empirically learning key probabilities and further tuning DT Tutor's utilities. Along the way, further steps were taken to address and informally assess progress on the real time inference requirement as well.

First, besides fleshing out DT Tutor, two other methods of selecting tutorial actions were developed for comparison purposes. One, the Random Tutor, selected randomly from tutorial actions that were relevant to the current tutorial state. The other, the Fixed-Policy Tutor, emulated a fixed policy for selecting tutorial actions employed by the Cognitive Tutors (Anderson et al., 1995), which are theory-based (Anderson & Lebiere, 1998), widely-used and highly effective (Koedinger et al., 1997). Both of these tutors shared the same user interface and help messages as DT Tutor with only the methods used for selecting tutorial actions being different. The tutorial action selections consisted only of whether to provide a help message and, if so, which help message to provide. By using the same interface and the same pool of help messages for all three tutors, the only differences between them would be which help

messages they selected and when, and these differences would be solely due to their action selection methods.

Next, after taking a pretest, students used the Random Tutor while both student and tutor actions were logged, and then they took a posttest. This was the start of the *data collection and tuning phase* of the experiment, which had three purposes: The first purpose was to collect data from the pre- and posttests and logged student-tutor interactions to learn key probabilities about student knowledge, student behavior, and the effects of tutorial actions. The Random Tutor was used to collect data about the effects of *individual* tutorial actions by statistically controlling for the effects of *sequences* of tutorial actions by randomizing over the sequences in which the individual actions occurred. The second purpose of the data collection and tuning phase was to tune DT Tutor's utilities. The final purpose of this phase was to collect logs of student-tutor interactions for use during the *assessment phase* of the study.

During the assessment phase, a replay mechanism developed for this study was used to replay the logged student-tutor actions while recording the responses that DT Tutor and the Fixed-Policy Tutor would provide for the exact same tutorial situations. The actions selected by the Random Tutor, the Fixed-Policy Tutor and DT Tutor were then rated, mainly quantitatively but also qualitatively, by a panel of judges who were skilled in tutoring calculus. The primary purpose of this phase was to compare the judges' ratings of DT Tutor's tutorial action selections with their ratings of the action selections of the Random Tutor and the Fixed-Policy Tutor. A secondary purpose was to learn details about the preferences of skilled human tutors for tutoring within the domain and about what might be done to improve the performance of computer tutors.

This study design cannot provide conclusive information about the bottom line of which tutor is most effective with students, but it has other advantages. First, it provided data for learning many of DT Tutor's key probabilities. Second, it can be used to compare the action selections of different tutoring approaches in *identical situations*. Third, it can provide information that is much more detailed than the bottom line about what makes the tutors' actions effective or not in particular situations (Mostow et al., 2001), information that can be used to improve not only DT Tutor but other tutors as well. Both the logged data and the judges' ratings remain a rich source of information about tutoring.

1.1 RESEARCH PROBLEMS

This study had a number of goals, as discussed in the Introduction. First, was addressing the *knowledge representation problem* by fleshing out DT Tutor into a complete ITS and learning key probabilities for

its decision-theoretic representation. Along the way, it was hoped that progress would be made towards meeting the *real-time inference requirement*. These issues are discussed in some detail in this report. But the primary focus of this study was evaluating DT Tutor's *tutorial action selection capabilities* by comparing its action selections with those selected by two other action selection methods, according to the ratings of a panel of judges who are skilled tutors in the domain. Comparison between DT Tutor's decision-theoretic approach and the two other action selection approaches was conducted using standard statistical hypothesis testing. The specific hypotheses are described in the following subsections.

1.1.1 Hypothesis 1: Decision-theoretic can be better than random tutoring

Hypothesis 1 is as follows:

> **According to ratings by skilled human tutors, tutorial action selections by decision-theoretic methods can be better than selections made randomly among relevant tutorial actions.**

The procedure for obtaining ratings from skilled human tutors is described in section 5.2. The exact method used to randomly select tutorial actions is described in section 4.2.1.2. The provision that random selection would be made among *relevant* tutorial actions was made to ensure that the Random Tutor would not be at a disadvantage because its action selections were not relevant to the current tutorial state.

This hypothesis was made for two purposes. First, if the other, more reasoned methods for selecting tutorial actions could do no better than an unreasoned, random method, then the validity of the experimental procedure might be in question. Second, random action selection provides a baseline control condition against which to compare decision-theoretic methods to see if they are useful at all.

1.1.2 Hypothesis 2: Decision-theoretic can be better than fixed-policy tutoring

Hypothesis 2 is as follows:

> **According to ratings by skilled human tutors, tutorial action selections by decision-theoretic methods can be better than selections made by a fixed policy that emulates the fixed policies of theory-based, widely accepted and highly effective computer tutors.**

The procedure for obtaining ratings from skilled human tutors is described in section 5.2. The exact fixed policy to be compared is described in section 5.2.2.3. This fixed policy emulates the tutorial action selection policy of the Cognitive Tutors (Anderson et al., 1995), discussed above, and is also very similar to the fixed policy of Andes1 (Conati et al., 2002) except for a difference in responses to student errors (Gertner & VanLehn, 2000). It must be emphasized that this hypothesis is made with respect to this fixed policy only (albeit one whose success sets a high standard) since a fixed-policy can be made arbitrarily complex – e.g., at the extreme, table lookup to match or surpass the tutorial action selections of DT Tutor for any finite enumeration of combinations of tutorial state attributes.

1.2 GENERAL APPROACH

This section first describes the decision-theoretic basis of DT Tutor's approach. Next is a description of how that decision-theoretic basis is put into action with DT Tutor's general architecture.

1.2.1 A decision-theoretic approach

The term decision-theoretic has been used in various ways (Jameson et al., 2001). DT Tutor's approach is described in this section, which first reviews prerequisite concepts to work up to a description of the dynamic decision network that is at the heart of DT Tutor.

Probability has long been the standard for modeling uncertainty in diverse scientific fields. In recent years, algorithms for *belief networks* (Pearl, 1988, equivalently, Bayesian networks) have made probabilistic modeling of complex domains more feasible. A belief network is a directed acyclic graph with (1) a chance node for each modeled attribute to represent beliefs about its value, and (2) arcs between nodes to represent conditional dependence relationships among the beliefs. Beliefs are specified in terms of probability distributions for the attribute's possible values. For a node with incoming arcs, a conditional probability table specifies its probability distribution conditioned on the possible values of its parents. For a node without parents, a prior probability table specifies its probability distribution prior to observation of actual node values. In many real-world scenarios, a substantial number of conditional independence relationships exist. When this is the case, a belief network can concisely represent the entire joint probability distribution – the probabilities for every possible combination of attribute values – with exponentially fewer probability entries, making it possible to model more complex domains. Belief

networks provide a mathematically sound basis for updating beliefs about any set of nodes in the network given any set of observations. Prior and conditional beliefs may be determined subjectively, theoretically, or empirically. Using Bayes' rule and a variety of inference algorithms, belief networks can be used to perform diagnostic, causal and intercausal reasoning, as well as any combination of these (Russell & Norvig, 1995).

Each node within a belief network represents possibly changing beliefs about an attribute whose value is fixed even though it may be unknown. *Temporal probabilistic networks* (Dean & Kanazawa, 1989) support reasoning under uncertainty in domains where the values of attributes may change over time (as tutorial state attributes often do). For each attribute whose value may change, a sequence of nodes represents the attribute's value at each point in time. Typically, a new slice is created for each time point at which attribute values may change, where a slice is a set of nodes representing attributes at a specific point in time. For tutoring, slices can be chosen to represent the tutorial state after a tutor or student action, when attribute values are likely to change (Reye, 1998). In addition to atemporal arcs between nodes within the same slice, temporal arcs extend between nodes across time slices to represent the fact that attribute values may also depend on earlier values of the same and other attributes. The set of temporal arcs represents the network's state evolution model (Russell & Norvig, 1995). Typically (e.g., Albrecht et al., 1998), each slice is constructed so that the Markov property holds true, by adding additional nodes if necessary (Russell & Norvig, 1995): attribute values in one slice depend only on attribute values in the same slice and in the immediately preceding slice.

In static temporal networks, the number of slices is fixed in advance. *Dynamic temporal networks* (e.g., *dynamic belief networks*) avoid this limitation by creating additional slices dynamically and removing old slices when they are no longer required. They rely on the Markov property to roll up beliefs from an old slice into the following slice so that beliefs in the following slice summarize all accumulated evidence and the old slice can be removed. However, attributes that are conditionally independent in one slice may eventually be influenced by a common historical cause, making them conditionally dependent in later slices. This can cause nodes in later slices to become fully connected (Boyen & Koller, 1998) – i.e., to lose all conditional independencies between nodes – eliminating the conciseness advantage of belief network representations. To avoid this situation, rollup schemes that approximate a slice's belief state without full connectivity can be used (e.g., Boyen & Koller, 1998).

Decision theory extends probability theory to provide a normative account of how a rational decision-maker should behave (Keeney & Raiffa, 1976). The decision-maker's preferences in light of her objectives are quantified in terms of a numeric utility value for each possible outcome of the decision-maker's action. To decide among alternative actions, the expected utility of each alternative is calculated by taking the sum of the utilities of all possible outcomes weighted by the probabilities of those outcomes

occurring. Decision theory holds that a rational agent should choose the alternative with maximum expected utility, thereby maximizing the utility achieved when averaged over all possible outcomes (Russell & Norvig, 1995). Explicitly quantifying the decision-maker's preferences facilitates comparing and prioritizing outcomes, helps to clarify the rationale underlying decisions (Jameson et al., 2001), and supports modifying the agent's behavior simply by changing utility values. The expected utility mechanism integrates considerations about probability and utility over a continuous range of values. Decision theory thus provides a rational, transparent, flexible and integrated mechanism for comparing decision alternatives in light of probabilities and priorities regarding any number of competing objectives. A belief network can be extended into a *decision network* (equivalently, an *influence diagram*) by adding *decision* and *utility* nodes along with appropriate arcs (Howard & Matheson, 1984).

A *dynamic decision network* combines the capabilities of a dynamic belief network and a decision network by combining chance, decision and utility nodes in a dynamic temporal representation (Dean & Wellman, 1991). Dynamic decision networks model scenarios in which decisions, attribute values, or priorities among objectives can vary over time. They provide a unified mechanism for computing the decision with maximum expected utility considering both uncertainty about the changing state and multiple competing objectives. As with dynamic belief networks, dynamic decision networks are typically constructed so that they can rely on the Markov property to dynamically add new slices and remove old slices. Rollup methods are similar to those for dynamic belief networks.

DT Tutor uses a dynamic decision network to make tutorial action decisions by looking ahead to anticipate their effects on the changing tutorial state in light of the tutor's uncertain beliefs and multiple competing objectives. For DT Tutor, chance nodes represent the tutor's beliefs about tutorial state attributes, decision nodes represent tutorial action alternatives, and utility nodes represent the tutor's preferences among the possible tutorial states.

1.2.2 DT Tutor's general architecture

DT Tutor's dynamic decision network is formed from dynamically created decision networks. These networks are called *tutor action cycle networks* (TACNs) because they each represent a single cycle of tutorial action, where a cycle consists of deciding a tutorial action and carrying it out, observing the next student action, and updating the tutorial state based on these two actions.

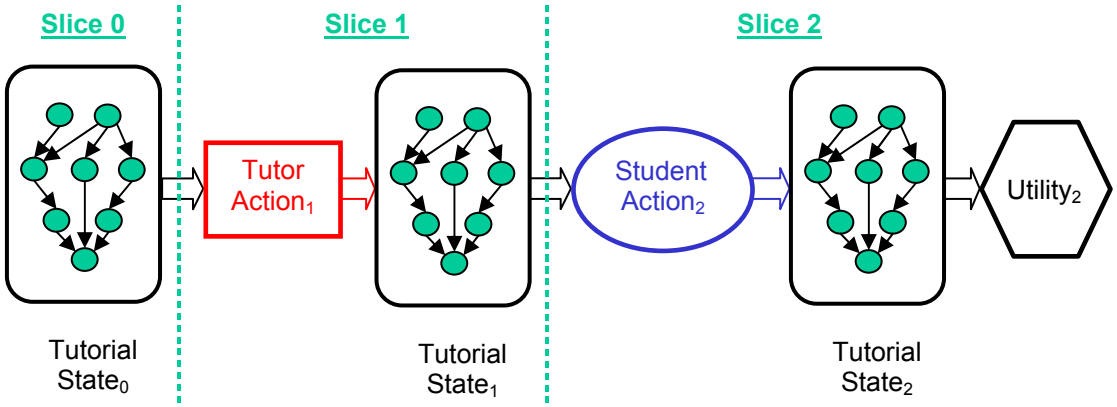


Figure 1.1: Tutor action cycle network, high-level overview

1.2.2.1 High-level overview of DT Tutor's architecture

Each TACN consists of three slices, as illustrated in Figure 1.1[1]. The *Tutorial State*$_s$ subnetwork in each slice is a belief (sub)network representing the student's state and all other attributes of the tutorial state, such as the discourse state and the state of the tutorial task (e.g., solving problems). The *Tutor Action*$_1$ decision node and the *Student Action*$_2$ chance node represent tutor and student actions, respectively. The *Utility*$_2$ node is a high-level representation of multiple utility nodes that together represent the tutor's preference structure regarding the various possible outcomes of the tutor's action for the current TACN.

TACNs are used both for deciding the tutor's action and for updating the tutorial state. Let us first consider how a TACN is used for deciding the tutor's action. During this phase, *Slice 0* represents the tutor's current beliefs about the tutorial state, *Slice 1* represents the tutor's possible actions and predictions about their influence on the tutorial state, and *Slice 2* represents a prediction about the student's next action, its influence on the tutorial state, and the utility of the resulting tutorial state outcomes. The decision network inference algorithm calculates the action with maximum expected utility and the tutor selects that action. This ends the decision-making phase. The tutor executes the action. After the tutor has observed the student's action or decided that the student is at an impasse, the update phase begins.

[1] In figures in this report, decision nodes are represented by rectangles, chance nodes are represented by ovals, utility nodes are represented by hexagons, and subnetworks are represented by rounded rectangles. Each arc into or out of a subnetwork actually represents multiple arcs to and from various subnetwork nodes. For subnetwork and node names, a subscript of *0*, *1*, or *2* refers to the slice number of the component. A subscript of *s* refers to any slice in which the component appears.

The tutor enters the student action as evidence in *Slice 2* and updates the network. At this point, the posterior probabilities in *Tutorial State*$_2$ represent the tutor's current beliefs. Since it is now time for another tutorial action selection, another TACN is created and the dynamic network is rolled forward: posterior probabilities from *Tutorial State*$_2$ of TACN$_i$ are copied as prior probabilities to *Tutorial State*$_0$ of TACN$_{i+1}$, where they represent the tutor's current beliefs[2]. This initializes the new TACN. The old TACN is discarded. This ends the update phase. The tutor is ready to begin the next phase, deciding what action to take next.

With this architecture, the tutor both reacts to past student actions (e.g., for corrective feedback), whose effects are summarized by the beliefs in *Tutorial State*$_0$, and anticipates future student actions and their ramifications (e.g., to provide proactive help) as represented by the beliefs in *Tutorial State*$_2$. In principle, the tutor can look ahead any number of slices without waiting to observe student actions in order to consider the long-term effects of its action alternatives. The tutor simply predicts probability distributions for the next student action and the resulting *Tutorial State*$_2$, rolls the dynamic decision network forward, predicts the tutor's next action and the following student action, and so on. However, a large amount of lookahead can be computationally prohibitive, so DT Tutor currently looks ahead only as far as the student's next action and the resulting tutorial state.

[2] This is a naïve network rollup scheme that neglects additional dependencies between nodes in the new slice (slice 0 of TACN$_{i+1}$) that are induced by shared dependence on nodes in previous time slices. Future work includes refining this rollup using an algorithm for approximate summarization of past dependencies (e.g., Boyen & Koller, 1998).

2.0 SCIENTIFIC CONTRIBUTIONS AND RELATED WORK

This section discusses the scientific contributions of this work in the context of related work. The scientific contributions of this work include (1) a non-trivial comparison of a decision-theoretic tutoring system to a competing approach, and (2) a novel computational architecture for tutorial action selection.

2.1 COMPARATIVELY EVALUATE DECISION-THEORETIC TUTORING

CAPIT (Mayo & Mitrovic, 2001) and iTutor (Pek, 2003) appear to be the only other decision-theoretic tutors that have been implemented and evaluated. However, these tutors were compared only with no tutoring at all (CAPIT: Mayo & Mitrovic, 2001), with "self-learning and consulting the teacher when required" (iTutor: Pek, 2003, p. 136), and with randomized action selection (CAPIT: Mayo & Mitrovic, 2001). This work does not directly assess effectiveness with students, but it does comparatively assess decision-theoretic tutoring against a higher standard: a Fixed-Policy Tutor that selects tutorial actions by emulating the fixed policy employed by the Cognitive Tutors (Anderson et al., 1995), which are theory-based (Anderson & Lebiere, 1998), widely-used and highly effective (Koedinger et al., 1997).

2.2 A NOVEL ARCHITECTURE FOR TUTORIAL ACTION SELECTION

Related work sometimes extends beyond ITSs to include user-modeling research because many systems that are not explicitly educational model the user (for an ITS, the student) to inform decisions about what actions to take in order to facilitate the interaction. Below, DT Tutor's architectural contributions are described in terms of important elements of the design space for a user modeling system: deciding what actions to take, modeling change in the user and the situation over time, deciding the *type* as well as the

topic of tutorial actions, modeling only observable or also unobservable attributes, modeling the user's focus of attention, modeling the user's affective state, and predicting the user's next action.

2.2.1 Making decisions

Applications that use a belief network representation often resort to heuristics to decide which action to take. For instance, Andes1, the first version of a physics ITS from which DT Tutor is descended, used heuristics to decide the topic of *what-next?* help (Gertner et al., 1998). Like many other belief network applications, Andes1 incorporated no explicit notion of the utilities of the possible outcomes of its actions. Such applications cannot integrate considerations regarding the probabilities and utilities of the possible action outcomes. Instead, Andes1 selected actions using probability thresholds and rules which reflect implicit priorities.

Some applications take outcome probabilities computed by a belief network and multiply them by their associated utilities outside the network to compute expected utilities for decision-theoretic action selection. These include an ITS for English capitalization and punctuation (CAPIT, Mayo & Mitrovic, 2001) and various other user modeling applications (e.g., Horvitz et al., 1999). An advantage of computing expected utility outside the network is potentially faster inference due to a smaller network (no decision or utility nodes with associated arcs) and the flexibility to consider only subsets of actions or outcomes in the expected utility calculations. However, the potential speedup is mitigated by (1) forgoing the option to use specialized decision network algorithms such as those that find the decision with maximum expected utility without computing exact expected utility values for all alternatives (e.g., Shachter & Peot, 1992), and (2) the potential to miss less obvious decision alternatives or outcome combinations with higher expected utility, resulting in decisions with less than maximum expected utility.

A few user modeling applications use decision network or equivalent representations to directly compute the decision with maximum expected utility. DT Tutor and Conati's representation (2002) for an educational game use DDN architectures to select actions for helping a user with a task. iTutor (Pek, 2003) uses a DDN for deciding actions at a different grain size: pre-computing a policy for selecting curriculum topics such as which problems to present to a student. Jameson and colleagues (2001) use a decision network to decide whether to present instructions individually or several at a time.

One benefit of decision-theoretic representations is support for value of information computations to guide user queries and other information-seeking behaviors. Applications that utilize value of information include those of Horvitz and colleagues (Horvitz et al., 1998; Paek & Horvitz, 2000) and iTutor (Pek, 2003). DT Tutor does not currently query the user or make decisions about other information-seeking behaviors and so it does not utilize value of information at this time.

2.2.2 Deciding the type of tutorial action as well as the topic

Many probabilistic ITSs decide the *topic* of tutorial discourse actions in real time as they tutor the student, but use a fixed, predetermined policy to decide the *type* of tutorial discourse action, such as *hint*ing at various levels of detail or explicitly telling the student how to *do* the next task step. For example, both Andes1 (Conati et al., 2002) and the model-tracing tutors (e.g., Anderson et al., 1995; Koedinger et al., 1997) dynamically decide which problem step to give the student help on (the *topic* of the tutorial action) and then use a fixed sequence of progressively more explicit hints (the *type* of the tutorial action), bottoming out with a hint that tells the student exactly how to do the step. Similarly, the decision-theoretic CAPIT ITS (Mayo & Mitrovic, 2001) uses decision theory to decide which problem-solving constraint to give feedback on (the *topic* of the tutorial action), but the content and style of the feedback messages seem to be determined in advance. The only other decision-theoretic ITS, iTutor (Pek, 2003), uses decision theory to precompute curriculum topics but then uses heuristics to decide which hint to give the student.

However, dynamically deciding which type of tutorial help to provide is important too. Reye's (1995) proposal apparently envisioned using decision theory to decide the tutorial action type as well as the topic, since it included examples of the tutor deciding whether to simply present a topic or to first ask the student about her knowledge of it. Moreover, the tutorial action type influences both the student's cognitive and affective states. For instance, a vague initial hint provides a small amount of cognitive information that may be sufficient to remind a student of what she already knows with a relatively small negative influence on her feeling of independence (del Soldato & du Boulay, 1995). But, at least with the model-tracing tutors, "… students are often annoyed with the vague initial messages and decide there is no point in using the help facility at all" (Anderson et al., 1995, p.199). Conversely, an explicit bottom-out hint provides a large amount of cognitive information that a student may require to complete a task step. Accomplishing task steps with extensive tutorial help may decrease the student's feeling of independence while increasing the student's confidence (del Soldato & du Boulay, 1995). But the cognitive information available in extensive tutorial help may be overused by students who do not really need it – "hint abusers" – causing them to learn little (Anderson et al., 1995, p.198). Thus, the type of tutorial action influences both the student's cognitive and affective states, and this influence depends at least in part on the student's prior mental state. Since prior mental state varies by student and even time for an individual student, it is not possible to determine in advance the best type of tutorial action to provide.

DT Tutor dynamically decides both the type of tutorial discourse action to provide – currently: *prompt*, *hint*, *teach*, *do* (tell the student exactly how to do a step), or *null* (no tutorial action) – and the

tutorial action topic (e.g., a particular problem step). It does this by looking ahead to predict the influence of the tutorial action type and topic on tutorial state attributes such as the student's knowledge, the student's affective state, and task progress (among other attributes). The possible tutorial action types have differing effects on the various tutorial state attributes, with none dominant for maximizing all attributes. By modeling tradeoffs among the expected outcomes of the tutorial action type alternatives relative to the tutor's objectives and priorities, DT Tutor can decide not only to provide vague initial help (e.g., a *prompt* or a *hint*) but it can also, for instance, decide instead to progress directly to *teach*ing a task-related rule or even to *do*ing a step for the student. Moreover, DT Tutor uses the same set of considerations to decide whether to provide proactive help (and if so, the topic and type of help to provide), which Andes, CAPIT, iTutor and the model-tracing tutors do not provide.

2.2.3 Modeling change over time

Systems that have used probabilistic networks to model change over time include POLA and Andes1, ancestors of DT Tutor, which employ a static atemporal belief network for each problem. POLA avoided temporal representation by dynamically adding nodes to represent problem-solving actions as the student completed them, along with nodes to represent the student's related physics knowledge (Conati & VanLehn, 1996). In effect, the semantics of each version of the incrementally-built networks changed with each time step to represent the tutorial state at the current point in time (Schäfer & Weyrath, 1997). Because POLA built its networks incrementally, it could not use them to model student knowledge related to uncompleted steps or to predict which action the student was most likely to attempt next (Conati et al., 2002). Andes1's networks do include nodes to represent uncompleted problem-solving actions and related knowledge, but the semantics of these nodes does not distinguish between steps that have already been completed and steps that Andes1 believes the student can complete (Conati et al., 2002). Thus, Andes1's networks cannot track the student's most recent action or current focus of attention (Conati et al., 2002). Andes1 models the evolution of a student's knowledge at a high level by copying updated beliefs about the student's knowledge between the atemporal networks for each successive problem, but this modeling is at too coarse a grain size to influence tutorial actions while the student is working on any particular problem.

Horvitz and colleagues have modeled change over time with a set of single-slice network models by embedding the notion of time within variable definitions (e.g., "attribute *a* at time *t*") (Horvitz et al., 1998) or by encoding time-dependent conditional probabilities (Horvitz et al., 1998) or utilities (e.g., Horvitz & Barry, 1995). Usually, each successive network represents the current point in time. The state evolution model is specified externally to the networks and is implicit in the changing variable

definitions, conditional probabilities and utilities. Without arcs across slices or an equivalent mechanism, many temporal dependencies may be neglected, such as the conditional dependence of attributes on their previous values. Without nodes to represent beliefs in more than one slice, a network cannot model changes in beliefs about the present through evidence-based revision of beliefs about the past.

CAPIT (Mayo & Mitrovic, 2001) uses a two-slice static temporal belief network to predict student problem-solving actions in terms of constraints. It adapts conditional probabilities to the current student while she works, using an algorithm for atemporal models heuristically modified to give greater weight to more recent events. Thus, CAPIT adapts its static temporal belief network to reflect changes in the tutorial state beyond its two-slice limit. However, the network does not track the order in which constraints have been attempted or feedback has been given, so it cannot track the student's focus of attention or make a more specific prediction about the student's next action. CAPIT's student model is limited to observable constraints, so it cannot model the evolution of the student's knowledge or other unobservable tutorial state attributes.

Dynamic belief network representations can model the temporal evolution of the model's state over any number of slices, including projections about future slices (Russell & Norvig, 1995), by dynamically creating new slices and removing old slices as they are no longer needed. Reye (1996) proposed dynamic belief network representations for ITSs to model the evolution of the student's knowledge over time and showed (Reye, 1998; Reye, 2004) how two probabilistic ITSs (Corbett & Anderson, 1992; Shute, 1995) can be characterized as special cases of a dynamic belief network approach. Other user modeling applications include a game (Albrecht et al., 1998) and office productivity tools (e.g., Horvitz et al., 1999), among others. Dynamic belief networks share with static belief networks the lack of an integrated provision for decision-making.

A DDN extends a dynamic belief network representation to include decision-making capability. iTutor (Pek, 2003) uses a DDN to pre-compute which curriculum topics to present to the student but then uses a dynamic belief network to track the student's knowledge as she progresses through the curriculum. Both DT Tutor and Conati (2002) employ DDNs both for decision-making and for modeling observable and unobservable attributes as they change over time, combining all these capabilities within integrated DDN architectures.

2.2.4 Which attributes to model

The set of attributes that an application considers should naturally influence the actions that it selects. For instance, if a help or tutoring application does not consider the user's focus of attention, its help is liable to be directed towards a topic that the user is not concerned about, which may confuse the user (e.g.,

Gertner et al., 1998). Many ITSs consider only one or two sets of attributes, such as the student's knowledge and task progress. A strength of decision-theoretic approaches is the ability to smoothly integrate considerations involving multiple sets of attributes. Described below is research related to modeling some of the more important attributes that DT Tutor can model.

2.2.4.1 Modeling observable and unobservable attributes

Some applications have used statistical methods to probabilistically model only observable user attributes. These include a machine learning system for predicting the details of subtraction errors (Chiu & Webb, 1998), CAPIT (Mayo & Mitrovic, 2001), and ADVISOR (Beck & Woolf, 2000), an ITS for grade school arithmetic. Limiting modeling to observable user attributes affords the considerable advantage of simplifying machine learning efforts (e.g., Jameson et al., 2001). All required data can be gathered from log files and other readily observable sources that record values for the attributes of interest (e.g., Horvitz et al., 1998). Data that the system can observe (e.g., keystrokes, mouse actions and timing data in context) can even be used to adjust prior and conditional probabilities while the system is in use in order to further adapt to specific users or populations (e.g., Horvitz et al., 1998; Mayo & Mitrovic, 2001).

However, there are also important advantages to modeling unobservable attributes (Jameson et al., 2001). Perhaps foremost among these for ITSs is that they are usually concerned with the student's knowledge – often to influence and sometimes to assess – which is unobservable. An application must model attributes if it is to reason about them (Grossmann-Hutter et al., 1999). Second, unobservable attributes often influence observable attributes. For instance, a student's knowledge influences the correctness of her problem-solving actions. So even if an application is concerned only with observable outcomes, it may be advantageous to consider its influence on unobservable attributes as well. In particular, ITSs often influence their students' observable behaviors through discourse and other actions intended to influence the student's mental state. Modeling conditional dependencies between observable and mental attributes allows one to leverage and even to test research from such fields as education and psychology (Grossmann-Hutter et al., 1999). Finally, networks with hidden variables representing unobservable attributes can be more concise (e.g., Heckerman, 1995), making them faster to learn (Binder et al., 1997) and to update (Martin & VanLehn, 1995), with a structure that is easier to elicit from experts (Binder et al., 1997) and more amenable to interpretation in terms of theoretical and empirical knowledge (e.g., Binder et al., 1997; e.g., Grossmann-Hutter et al., 1999). DT Tutor, like many other ITSs and other user modeling systems, models both observable and unobservable attributes.

2.2.4.2 Modeling the user's focus of attention

Identifying the user's focus of attention can be critical to providing assistance that is timely and relevant to the user's needs (e.g., Horvitz et al., 1999). No other decision-theoretic ITS currently models the user's focus of attention. According to Grosz and Sidner (e.g., 1986), knowledge of focus of attention as well as task structure is necessary for understanding and generating task-oriented discourse. DT Tutor follows Grosz in modeling focus of attention relative to a hierarchical task structure. However, instead of modeling focus with a stack as in the work of Grosz and colleagues (e.g., Grosz & Sidner, 1986), DT Tutor's probabilistic approach has more in common with Walker's (1996) cache model of attentional state. The cache model accounts for phenomena such as the influence of the recency of discourse content as well as the influence of the hierarchy of intentions related to the task. The cache model is also consistent with Albrecht and colleagues' (1998) observation that users may interleave actions to achieve multiple goals. Reye (1995) criticizes the stack model's inflexibility regarding the order in which goals may be pursued within an ITS. DT Tutor also models *focus aging*, or decreasing probability of focus on task elements that were in focus at earlier times, which is consistent with both the cache model and Horvitz and colleagues' (1998) approach of associating observations seen at earlier times with decreased relevance to the user's current goals.

Andes1 uses a hierarchically-structured atemporal belief network to narrow in on a set of task steps that may be in the student's task-related focus of attention when she requests *what-next?* help. However, Andes1's network does not distinguish completed steps and cannot track the student's most recent action, so Andes1 uses a heuristic procedure to guess the student's specific focus of attention (Conati et al., 2002; Gertner et al., 1998). The Adele ITS for medical diagnosis (Ganeshan et al., 2000) likewise models focus of attention relative to a hierarchically-structured atemporal belief network. However, Adele does not model uncertainty about the student's focus of attention probabilistically, instead directing the discourse and asking disambiguating questions to limit the possibilities. The Lumière Project's help systems for office productivity programs probabilistically model focus of attention for non-ITS applications, but at least initially avoided detailed modeling of domain-specific content (Horvitz et al., 1998). Some other applications by Horvitz and colleagues (e.g., Horvitz et al., 1999; Paek & Horvitz, 2000) model focus of attention at mostly a coarser level, such as which agent or application program the user is attending to.

2.2.4.3 Modeling the user's affective state

Considering the student's affective or motivational state can be vital for effective tutoring. Lepper and colleagues (1993) observed that their expert human tutors appeared to give as much weight to affective and motivational outcomes as to informational and cognitive outcomes, knowing that a negative affective state can interfere with learning (Goleman, 1995). Many ITSs consider the student's affective state at most implicitly, with corresponding effects on the affective sensitivity of the tutoring that they provide. Most ITSs and other user modeling applications that do consider the student's affective state pay relatively scant attention to other considerations.

For ITSs, detailed models of the student's affective state have been implemented by, for example, del Soldato and du Boulay (1995) and de Vicente and Pain (e.g., 2002). However, these models have at least two shortcomings. First, they do not model the ITS's uncertainty about the student's affective state. Arroyo and Woolf (2001) address this issue with a statistical approach for predicting the student's behavior and affective state. Second, they do not satisfactorily resolve what the tutor should do when there is a conflict between the best tutorial action based on affective outcomes and the best tutorial action based on cognitive or other outcomes.

Decision-theoretic approaches provide a way to take into account the tutor's uncertainty about the student's affective state while balancing considerations regarding affective and other outcomes. DT Tutor uses a DDN to weigh uncertain beliefs and multiple objectives regarding the student's changing affective state along with other tutorial outcomes. Conati (2002) likewise proposes a DDN representation to consider both the user's affective state and "learning state" for an educational game, employing a detailed model of the user's affective state but leaving the model of the user's learning state unspecified. DT Tutor sports a relatively impoverished model of the student's affective state. DT Tutor's main contribution in this area is providing a framework for weighing uncertain, changing beliefs and priorities regarding any number of outcomes, including the user's affective state, at various levels of detail, depending on the needs and capabilities of the application.

2.2.4.4 Predicting and learning from the user's actions

ITSs and other user modeling applications often choose actions, at least implicitly, on the basis of beliefs about how they will influence the user's performance. Conversely, the user's performance can be used as evidence to update the application's user model. Therefore, it can be important for a user modeling application to predict the user's performance and to learn from the user's actual performance.

An application's prediction capabilities depend in part on the factors that it considers. For instance, Chiu and Webb (1998) consider the student's past subtraction performance in detail to arrive at detailed predictions about future subtraction performance, but they do not consider the influence of help. ADVISOR (Beck & Woolf, 2000), on the other hand, models many other factors, including the help provided, to predict the time required for a student to solve an arithmetic problem and whether she will be correct, but does not model or predict the student's performance on problem subskills. Jameson and colleagues (2001) likewise predict a user's execution time and errors based in part on the system's delivery of instructions. CAPIT (Mayo & Mitrovic, 2001) models and makes predictions about student performance in terms of 25 constraints.

All of the systems above model the user and make predictions strictly in terms of observable attributes, which facilitates empirical learning both prior to and during interaction with the user. However, modeling relationships between unobservable attributes, such as the user's knowledge and focus of attention, and observable user actions can help in predicting observable user actions. Furthermore, such models can be used for diagnostic learning about unobservable attributes based on observed user actions.

Albrecht and colleagues (1998) model an unobservable attribute, the user's quest in a game, as part of predicting the user's next action and location within the game space. Horvitz and colleagues (1999) and DT Tutor both model the user's focus of attention as part of predicting the user's next action. DT Tutor models focus of attention along with student knowledge at a finer grain size – particular task steps and rules within the tutorial domain – to predict the topic and the correctness of, but not the time required for, the student's next action.

ADVISOR (Beck & Woolf, 2000) and the systems that use probabilistic networks (e.g., Albrecht et al., 1998; Horvitz et al., 1999; Jameson et al., 2001; Mayo & Mitrovic, 2001; Murray & VanLehn, 2000) model the system's inherent uncertainty by predicting the user's next action probabilistically. The systems that use probabilistic networks also have the capability to learn diagnostically about unobserved attributes (e.g., the user's goal, knowledge, focus of attention, and even potentially observable attributes) based on observed user actions.

3.0 TECHNICAL APPROACH

This section describes the technical approach for developing DT Tutor into a full-fledged ITS, the *Calculus Related Rates Tutor* (*Calculus Tutor* for short). The components include a domain expert, a decision-theoretic action selection engine, and a student interface. Appendix A provides an introduction to the Calculus Tutor and its domain from a student's point of view, including a screen shot of the interface in Figure A1. Appendices C through E and G provide additional information about the materials used with the Calculus Tutor, a variety of screen shots, and sample help messages.

3.1 THE DOMAIN EXPERT

The domain expert performs several functions for DT Tutor. First, it solves problems in the domain while creating for each problem a *problem solution graph* structure which is the basis for (1) DT Tutor's dynamic decision networks and the (2) the goal reification in the Goals Window of the student interface. These are the domain expert's fundamental capabilities that will be described here. The domain expert also checks the correctness of student equations and acts as log server for the web-based version of the student interface. To describe the problem solver, it is first necessary to describe the problem domain.

3.1.1 The calculus related-rates problem domain

A sample word problem for this domain follows:

> The economy of the newly–founded republic of San Pedro is growing such that, in any year y, the level m of the money supply in billion dollars is 2 times the square of the number of years elapsed. The gross national product g of the economy is 4 times the money supply. How fast is the gross national product growing when y equals 2 years? (Singley, 1986, p.8)

In equation form, the givens are $m = 2y^2$, $g = 4m$, and $y = 2$, and the goal is to find dg/dy when y equals 2. Singley (1986) developed a model-tracing tutoring system for 32 types of problems in this domain. Quoting Singley (1986, pp.9-10), the problems have the following features:

- Three variables, referred to generically as *x, y,* and *z*. The value of the *z* variable is always given.
- Two relations, one between *x* and *y* and the other between *y* and *z*. Each relation could be stated either as a regular equation (e.g., $x = 3y^2$) or as a derivative (e.g., $dx/dy = 6y$). Furthermore, these relations could be stated either with *x* in terms of *y* (forward direction) or with *y* in terms of *x* (backward direction). By crossing these two binary features, each relation could take on four possible forms. Given two relations, this meant a total of sixteen possible "initial states" for the problems.
- A goal, either to find the value of the *x* variable (an "integration" goal) or the value of dx/dz (a "differentiation" goal) for a particular value of *z*. These goals were so named because, in most cases, finding a value for *x* involved integration and finding a value for dx/dz involved differentiation. Crossing the 16 initial states by these two goals yields a total of 32 problems.

In addition, Singley's specification seems to implicitly include the following restrictions:

1. For integration operations, the arbitrary constant of integration is neglected. For instance, $p = \int 32t\,dt = 16t^2$ rather than $p = \int 32t\,dt = 16t^2 \underline{+ c}$ (Singley, 1986, p.151).
2. For the "differentiation" problems, the relationship between the *x* and *y* variables is always such that the first derivative is a constant (e.g., $dx/dy = c$, where *c* is an arbitrary constant) (Singley, 1986, p.100).
3. Again for the "differentiation" problems, the relationship between the *y* and *z* variables is always such that the first derivative is not a constant (e.g., $dy/dz = f(z)$).

Restriction (1) above is mathematically incorrect. It simplifies the integration operation and probably more importantly simplifies combining the resulting equation with other equations, facilitating students' movement through the resulting problem space by simplifying calculus operations and algebraic manipulations. For the purposes of this study, the domain-specific content of the tutoring is much less important than its effectiveness, so restrictions of this sort can be tolerated.

Restrictions (2) and (3) are restrictions on the problems presented to students. They simplify applying the chain rule to achieve the "differentiation" goal of finding the value of *dx/dz* for a particular value of *z* (i.e., $dx/dz = dx/dy * dy/dz = c * f(z) = f'(z)$).

To solve these problems, the following calculus and algebra operators are supported as quoted from (Singley, 1986, pp.10-11):

- *Differentiate*. Takes a regular equation stating *x* in terms of *y* and produces the derivative *dx/dy*.
- *Integrate*. Takes the derivative *dx/dy* and produces a regular equation stating *x* in terms of *y*.
- *Apply chain rule*. Takes two derivatives, *dx/dy* and *dy/dz*, and produces a third, *dx/dz*.
- *Substitute equations*. Takes two regular equations, *x* in terms of *y*, *y* in terms of *z*, and produces a new equation stating *x* in terms of *z*.
- *Flip derivative*. Takes the derivative *dx/dy* and produces the derivative *dy/dx*.
- *Restate equation*. Takes a regular equation stating *x* in terms of *y* and transforms it into an equation stating *y* in terms of *x*.
- *Evaluate*. Given an equation stating either *x* or *dx/dz* in terms of *z*, and a value for *z*, returns the value of *x* or *dx/dz* respectively.

Apply chain rule and *substitute equations* are the only "combining operator[s]" (Singley, 1990, p.110), so called because they put "previously unassociated variables in direct relation to one another" (Singley, 1986, p.11). Since the initial relations are between (1) *x* and *y*, and (2) *y* and *z*, and the goal is to find the value of *x* or *dx/dz* in terms of the value of *z*, a combining operator must be applied in each problem in order to relate the variables *x* and *z*. The operators *differentiate*, *integrate*, *flip derivative*, and *restate* are unary since they take a single operand.

Singley (1986) also imposes a few restrictions on problem solutions, apparently to reduce floundering:

- Illegal operator applications are blocked – i.e., applying an operator to an invalid operand is not allowed.
- Higher-order differentiation is blocked. It is not taught in Singley's tutor's supporting material and it is not needed to solve the problems that are presented to students.
- Previously performed operator applications may not be repeated.

- After the student has derived the sought equation for the problem, which states the value of the sought variable (generically, x or dx/dz) in terms of the variable whose value is known (z), further exploration of the problem space is "restricted."
- Evaluation seems to be blocked except on the problem's target equation with the value $z=c$.

Singley (1986) also developed a goal-posting version of his tutor which, for each problem, (1) tells the student a top-level goal of which combining operator to apply, (2) tutors the student to set the subgoals required to apply the combining operator, (3) directs the student to pursue these subgoals and then apply the combining operator in a prescribed order, and (4) blocks operator selections that do nothing to satisfy the current problem-solving goal. However, Singley (1986) does not clearly specify which combining operator should be applied given the initial problem state. Singley (1986, p.28) states that

> It's generally the case that those initial states composed of regular equations favor use of the *substitute equations* operator and those composed of derivatives favor use of *apply chain rule*. Of course, this bias is somewhat modulated by the goal type of the problem, with integration goals favoring *substitute equations* and *differentiation* goals favoring *apply chain rule*.

Even less clear is what to do for initial states composed of one regular equation and one derivative. Perhaps because of lack of clear direction on which combining operator to apply, Singley's goal-posting tutor simply *tells* the student which combining operator to apply instead of helping the student select the combining operator herself.

Singley (1986) specifies a means-ends analysis approach for solving problems in this domain and designates an "optimal" solution for 12 of the 32 problem types, but does not specify how these solutions are culled from the large number of possible solutions or how to find an "optimal" solution without an exhaustive search of all possible solutions.

3.1.2 Problem solutions generated by the domain expert

The domain expert can solve all of the types of calculus related-rates problems with which students may be presented (and others as well). Depending on the problem-solving rules and heuristics that it employs, the problem solver can produce either forward- or backward-chaining solutions, including multiple solutions for each problem type. The problems and the operations used to solve them conform to the domain specification of Singley (1986) except that students will be given problems in equation rather than word form to avoid the complex task of modeling and tutoring the process of translating word problems into equations. Singley (1986) likewise does not model the translation task.

A fundamental characteristic of backward-chaining solutions that distinguishes them from forward-chaining solutions is decomposition of the problem into goals and subgoals, along with related rules, that can be used to direct and constrain student problem-solving actions to effective actions within the problem space. A core principle of the ACT theories (e.g., Anderson, 1983; Anderson, 1993; Anderson & Lebiere, 1998) upon which the original Cognitive Tutors are based is that problem solving involves decomposing the problem into goals and subgoals. Furthermore, the ACT theories assume that students can convert their declarative knowledge (e.g., knowing the chain rule) into procedural knowledge (e.g., knowing how to apply the chain rule) only by relating it to task goals (Anderson et al., 1995). According to the theories, the student's procedural knowledge consists of a set of production rules that specify which problem-solving actions to apply (including setting subgoals) in service of the student's problem-solving goals. Therefore, developers of the Cognitive Tutors attempt to create interfaces that make explicit, or *reify* (Collins & Brown, 1988) the goal structure underlying the problem solving (Anderson et al., 1995). To this end, an important contribution of Singley's (1986) work on a Cognitive Tutor for calculus related-rates problems was developing and demonstrating the effectiveness of a user interface which reified – to a limited extent – each problem's goal structure (Anderson et al., 1995). Using backward-chaining solutions as a basis for tutoring is thus consistent with a Cognitive Tutor approach, which should facilitate a fair comparison between the decision-theoretic version and fixed-policy version (which emulates Cognitive Tutors) of the tutor. Consequently, the current version of the problem solver uses backward-chaining to produce a single partial-order solution for each problem using the heuristics defined in the Calculus Tutor Tutorial which is listed in Appendix A.

Figure 3.1 illustrates the problem solution graph for a simple problem, Problem P1, created by the problem solver and used as input for both the student interface and the decision-theoretic action selection engine. The problem statement is "Transform the given equations and evaluate to find dq/ds=<number> when s=2" and the given equations are dq/dr=3, dr/ds=$10*s^2$ and s=2. Nodes in the top two rows of the graph with crosshatch filling represent rule nodes. The remaining nodes consist of goal and equation nodes. Nodes with solid shading represent given nodes. The shaded goal node "Eval dq/ds=num" represents the given goal "evaluate to find dq/ds=<number>." The shaded goal nodes represent the given equations. The domain expert solves the problem by using rule *evaluate operand 1* (rule node "Eval Op1") to set a goal to find an equation of form dq/ds=f(s), which corresponds to node "Find dq/ds=f(s)." Given this goal, rule *select chain rule* (node "Select Chain") is used to set the goal *apply chain rule* (node "Apply Chain"). Given this goal, rules *evaluate operand 1* (rule node "Eval Op1") and *evaluate operand 2* (rule node "Eval Op2") are used to set goals *find dq/dr=<number>* and *find dr/ds=f(s)*. Equations of the desired equation forms were given (*dq/dr=3* and *dr/ds=10*s²*) and so a *find equation form* rule (node "Find Eq Form") is used to find them. Once the desired equations are found, rule *execute chain rule*

(node "Exec Chain") is used to apply the chain rule to the given equations to establish equation $dq/ds=30*s^2$. Finally rule *execute evaluation* is applied to equations $dq/ds=30*s^2$ and $s=2$ to derive the answer, $dq/ds=120$.

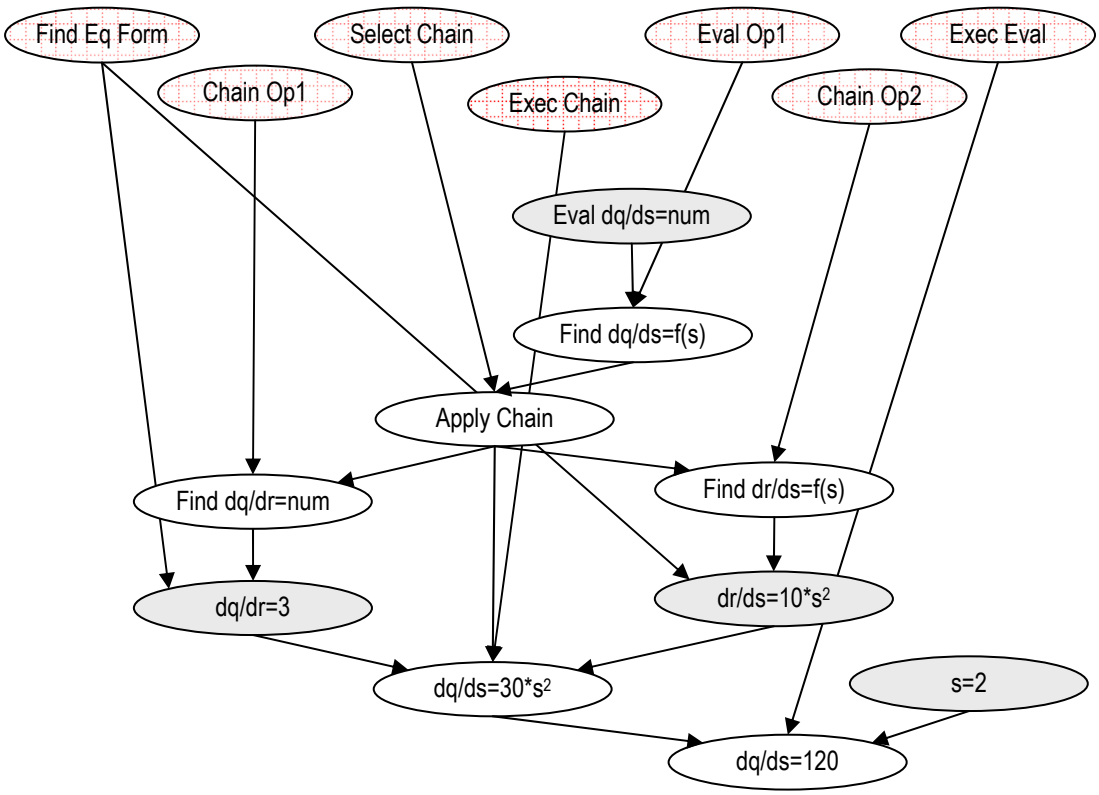


Figure 3.1: Problem solution graph for problem P1

3.2 DECISION-THEORETIC ACTION SELECTION ENGINE

This section builds on section 1.2 above, General Approach, providing more detail about the structure, mechanism and capabilities of DT Tutor's tutorial action selection engine. For a still more detailed description, see (Murray et al., 2004)

3.2.1 Tutor Action Cycle Network in more detail

Figure 3.2 shows DT Tutor's TACN architecture in more detail. Each of the elements depicted will be described below. The *Tutorial State*$_s$ subnetwork in each slice is further divided into subnetworks to model various tutorial state attributes. In Figure 3.2, *Student Model*$_s$ is composed of the *Student Knowledge*$_s$ subnetworks to model the student's task-related knowledge, the *Student Focus*$_s$ subnetworks to model the student's task-related focus of attention, *Student Help Style*$_s$ nodes to model the student's style of using help, and *Student Independence*$_s$ nodes to model the student's affective feeling of independence. Outside the student model are the *Discourse State*$_s$ subnetworks to model the state of the discourse between student and tutor. The *Tutor Action*$_1$ and *Student Action*$_s$ representations, shown in Figure 1.1 as single nodes, consist here of more than one node, as described below.

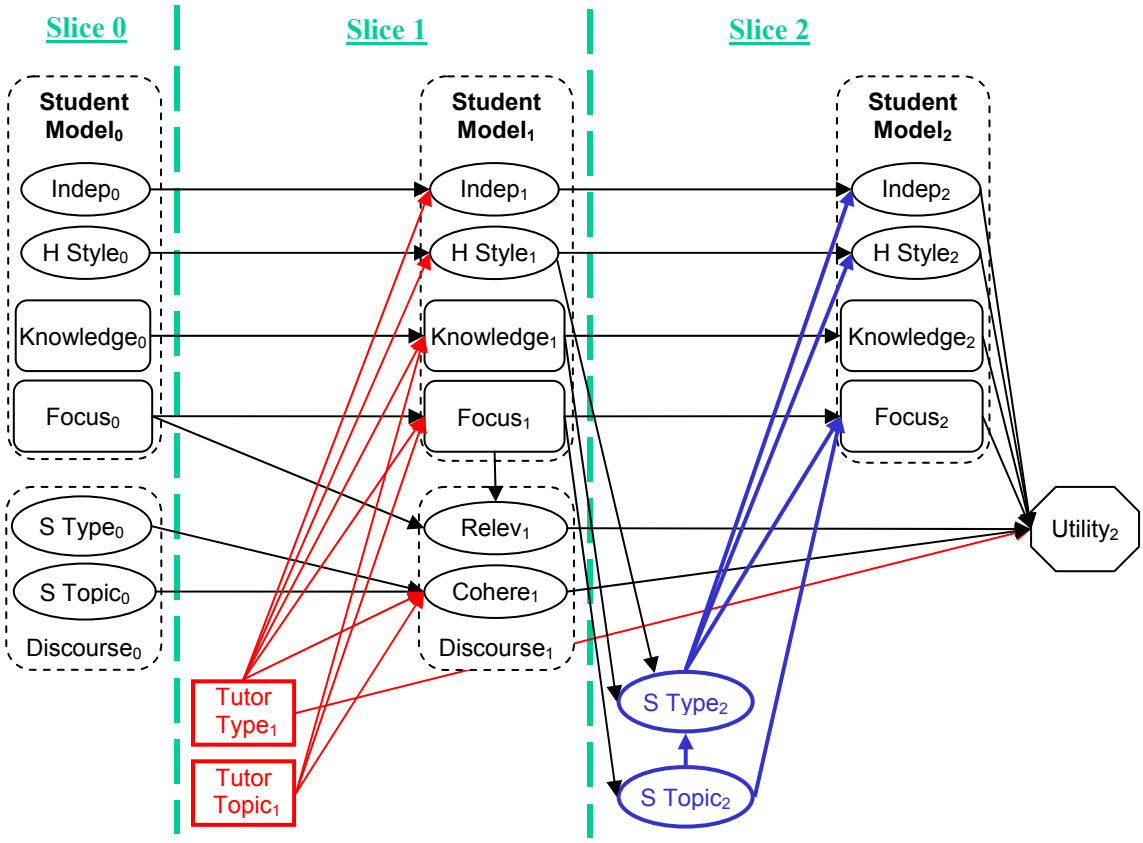


Figure 3.2: DT Tutor's Tutor Action Cycle Network

3.2.2 Problem solution graph structure

The *Student Knowledge*$_s$ and *Student Focus*$_s$ subnetworks, which are the problem-specific part of each Tutor Action Cycle Network (TACN), are based on the problem solution graph for each problem created by the domain expert. For each problem, DT Tutor simply reads in a file created by the domain expert.

3.2.3 Tutor Action Nodes

DT Tutor addresses the tutor action *topic* in the manner specified by the tutor action *type*. These action components are represented by the decision nodes *Tutor Action Topic*$_1$ and *Tutor Action Type*$_1$. The *Tutor Action Type*$_1$ alternatives are currently *prompt*, *hint*, *teach*, *do* (tell the student exactly how to do a step), and *null* (no tutor action).

The *Tutor Action Topic*$_1$ alternatives may consist of any problem step (fact or goal) or related rule in the problem solution graph. However, students rarely repeat steps that they have already completed successfully, and they are unlikely to be able to complete steps for which prerequisites have not been completed. Accordingly, tutors are less likely to address such steps. Therefore, for faster response time, DT Tutor normally considers as the tutor action topic only uncompleted steps for which prerequisites have been completed and related rules. A tutor action topic of *null* is also supported to model no tutor action.

The *Tutor Action*$_1$ nodes may influence the student's affect, knowledge, and focus of attention, as well as the discourse state. *Tutor Action Type*$_1$ also influences *Utility*$_2$ in order to model individual differences in tutoring styles among tutors.

3.2.4 Student Action Nodes

Like *Tutor Action*$_1$, DT Tutor's student action representation consists of nodes to model the student's problem-solving action *topic* and *type*: *Student Action Topic*$_s$ and *Student Action Type*$_s$ respectively (*S Topic*$_s$ and *S Type*$_s$ in Figure 3.2). *Student Action Topic*$_s$ may be any step in the problem solution graph or *null* to model either no action at all or an action with no specific topic (such as a general help request). *Student Action Type*$_s$ may be *correct*, *error*, *impasse*, or *null*. A *correct* action matches a step in the problem solution graph. An action type of *impasse* models either a help request on a specific topic (specified by the *Student Action Topic*$_s$ value in the same slice) or a general help request such as "*What should I do next?*" *Null* means no student action. All other student actions are of type *error*.

The *Student Action*$_0$ nodes (in *Slice 0*) in a TACN represent the most recent student action while the *Student Action*$_2$ nodes represent the student action following the tutor's action in the current TACN. In the initial TACN, the *Student Action*$_0$ nodes have the value *null* since there is no previous student action.

3.2.5 Student Focus Subnetworks

The *Student Focus*$_s$ subnetworks represent the tutor's beliefs about two components of the tutorial state: (1) the student's focus of attention within the current problem, and (2) the student's problem-solving progress. Figure 3.3 illustrates the *Student Focus*$_s$ subnetworks for a simple problem with just five steps (facts or goals), the first of which (*Step 1*$_s$) is given. The student's focus of attention is modeled relative to the problem steps, so the *Student Focus*$_s$ subnetworks consist of just the step nodes in the problem solution graph.

Student Focus$_s$ step nodes have four possible values: *not_ready, ready, in_focus,* and *complete*. The student is unlikely to be able to successfully complete problem steps for which prerequisites have not been completed, and is therefore less likely to attempt them. Such steps have the value *not_ready*. The student is also unlikely to repeat problem-solving steps that have already been completed successfully. These steps have the value *complete*. The remaining steps are uncompleted steps that the student could productively attempt next since all prerequisite steps have been completed, and thus are more likely to be in the student's focus of attention. They have some distribution over the values *ready* and *in_focus*, with *ready* meaning that the student is ready to attempt the step next, and *in_focus* meaning that the step is also in the student's focus of attention. In Figure 3.3, the probability distribution between *ready* and *in_focus* is depicted by the density of the dots shading the nodes, with denser dots meaning that the node is more likely to be *in_focus*.

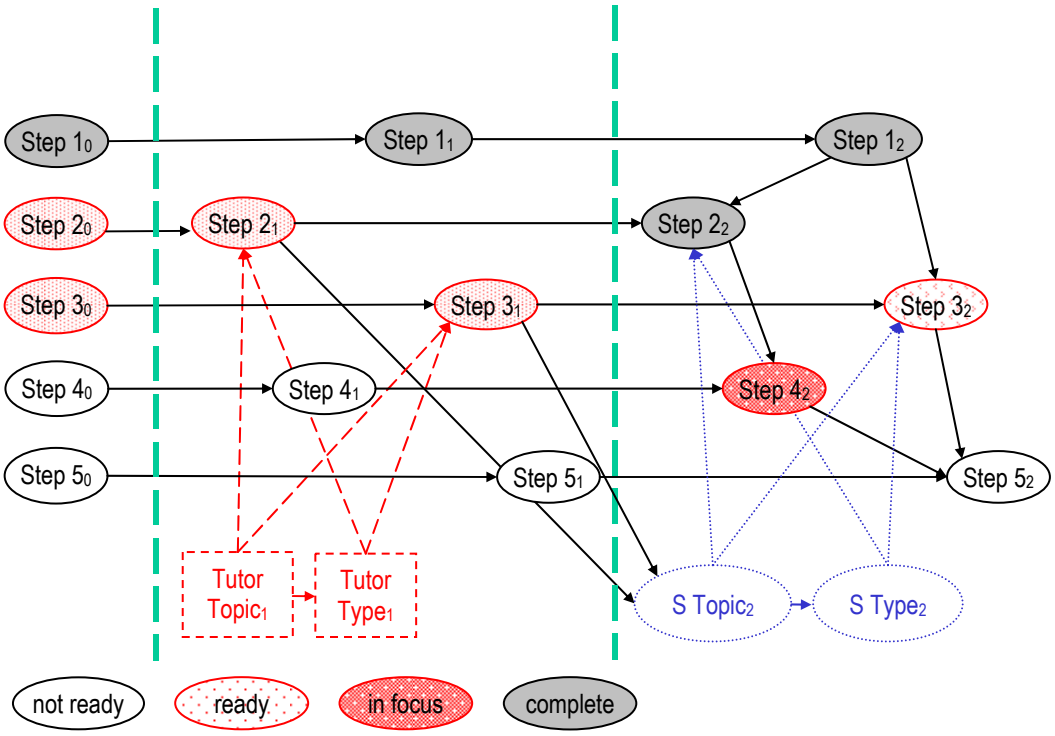


Figure 3.3: Student Focus subnetworks in TACN

Nodes in slice 0 represent the tutor's prior beliefs about the tutorial state and are disconnected except for arcs to slice 1. In *Student Focus*$_0$ of the first TACN for a tutorial session, prior probabilities for the given steps (the problem goal and facts) are set to *complete* with probability *1.0*. Steps with uncompleted precedent steps are set to *not_ready* with probability *1.0*. Prior probabilities for the remaining steps are set to a distribution over the values *ready* and *in_focus*, with a probability mass of *1.0* for *in_focus* divided equally among these steps. For subsequent TACNs, prior probabilities for slice 0 are copied from posterior priorities in slice 2 of the previous TACN. Slice 0 in Figure 3.3 depicts a situation in which *Step 1*$_0$ was given, *Step 2*$_0$ and *Step 3*$_0$ have equal probabilities of being *in_focus*, and *Step 4*$_0$ and *Step 5*$_0$ are *not_ready*.

Student Focus$_1$ represents the influence of the tutor's action on the student's focus of attention. The tutor normally considers addressing only steps that are *ready* or *in_focus*, so in Figure 3.3 there are arcs from the tutor action nodes to *Step 2*$_1$ and *Step 3*$_1$. *Student Focus*$_1$ influences the *topic* of the student's next action, *Student Action Topic*$_2$, which may be any problem step.

The student action nodes can in turn influence the *Student Focus*$_2$ step nodes. In slice 2 of Figure 3.3, the student has just completed *Step 2*, so it is *complete*. *Student Focus*$_2$ step nodes are also influenced

by their prerequisite steps. In Figure 3.3, when *Step* 2_2 becomes *complete*, its child, *Step* 4_2, has a distribution over the values *ready* and *in_focus* since all of its prerequisite steps (just *Step* 2_2) are now *complete*.

3.2.5.1 Focus evolution and aging

Temporal arcs between *Student Focus*$_s$ step nodes model the persistence of the student's focus of attention and task progress over time. For instance, steps that are *not_ready* remain so until either all of their parent steps are *complete* or the student completes the step (e.g., by guessing). In contrast, steps that are *in_focus* at some point in time become a little less likely to be *in_focus* with each passing slice. This is to model focus aging: steps that were *in_focus* slowly become less *in_focus* over time as the student moves on to other topics. In Figure 3.3, *Step 3*'s probability of being *in_focus* decreases from slice 1 to slice 2 as the student completes *Step* 2_2 instead.

When there are multiple steps that could be *in_focus* because they are the next *ready* step along some portion of a solution path, DT Tutor needs some way to decide how likely the various steps are to be *in_focus*. To do this, DT Tutor, like Andes1 (Gertner et al., 1998), assumes a depth-first bias: Students usually prefer to complete work on one portion of a solution path before starting to work on another. A depth-first bias in problem solving corresponds to a depth-first traversal of the problem solution graph. Such a bias is consistent with activation-based theories of human working memory (e.g., Anderson, 1993) and observations of human problem solvers (e.g., Newell & Simon, 1972). However, depth-first bias is not absolute (VanLehn et al., 1989): at any given step, there is some probability that a student will not continue depth-first.

To model depth-first bias, when a step first becomes *ready* or *in_focus* because all of its parent steps have become *complete*, that step has a high probability of being *in_focus*. This is because the student, having just completed the last of the step's parents, is likely to continue working with the step itself. In Figure 3.3, *Step* 4_2 is highly likely to be *in_focus* since *Step* 2_2 has just been completed. Focus aging helps to model another aspect of depth-first bias: preferring to backtrack to more recently *in_focus* steps. When the student completes or abandons a portion of the solution path, steps that were recently *in_focus* but that are still not *complete* have had less focus aging than steps that were *in_focus* in the more distant past, so the more recently raised steps remain more likely to be *in_focus*.

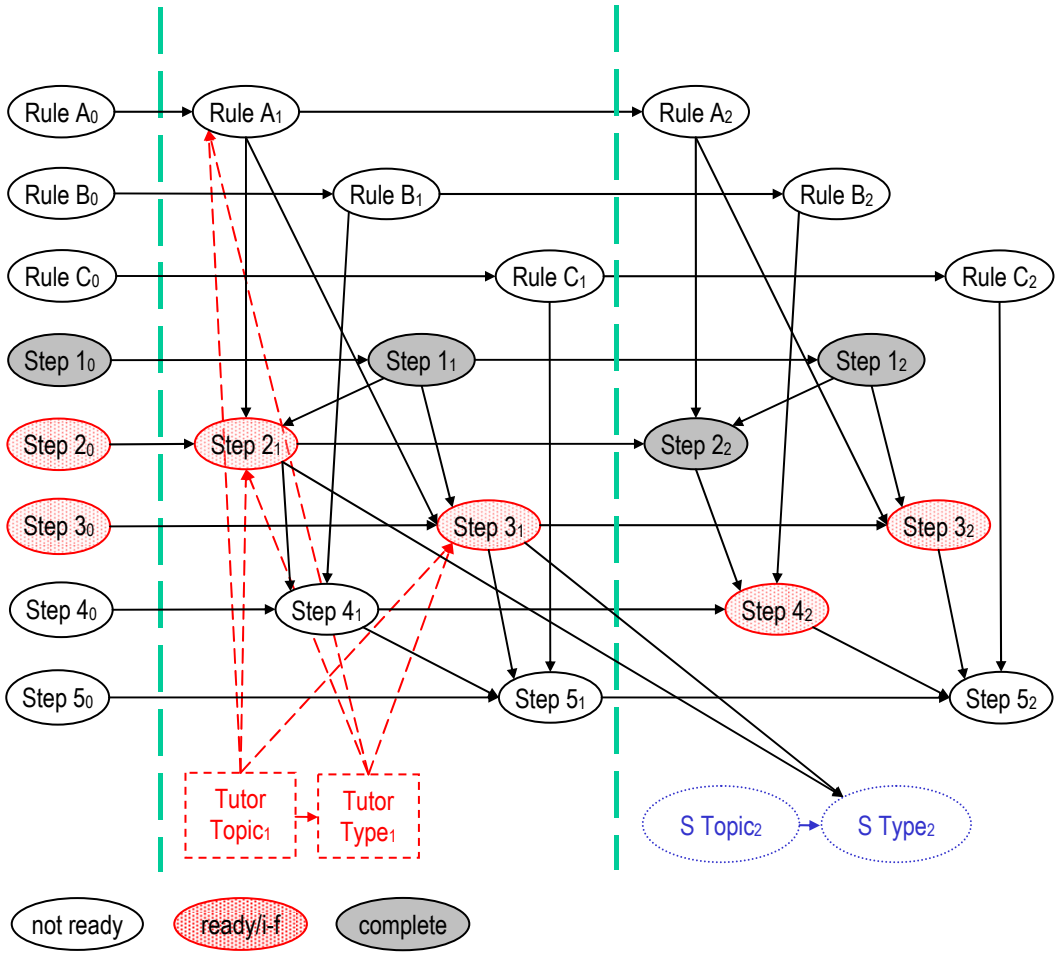


Figure 3.4: Student Knowledge subnetworks in TACN

3.2.6 Student Knowledge Subnetworks

The *Student Knowledge$_s$* subnetworks represent the tutor's beliefs about the student's problem-related knowledge. Figure 3.4 provides an illustration for the same problem that was described in the previous subsection. To create these subnetworks, the problem solution graph is converted into a belief network, associating each node with a probability distribution for the values *known* and *unknown*. Rule nodes represent the tutor's belief about the student's knowledge of the corresponding rule. Step nodes represent the tutor's beliefs about the student's capability to derive the corresponding fact or goal given the student's rule knowledge. In Figure 3.4, the step nodes are shaded according to whether their *Student Focus$_0$* subnetwork values are *not_ready*, *ready* or *in_focus* ("*ready/i-f*"), or *complete*. This shading is

intended to illustrate why the tutor action nodes influence some nodes (the *ready* or *in_focus* nodes and their rule parents) and not others, as explained below.

In the first TACN for a tutorial session, prior probabilities for the *Student Knowledge*$_0$ rule nodes are based on the best information available, such as pretest data for a particular student or statistical data for a student population. Prior probabilities for the given steps (the problem goal and the given facts) are set to *known* with value *1.0*. Prior probabilities for the remaining steps are set to *unknown* with probability *1.0*. For subsequent TACNs, prior probabilities for slice 0 are copied from posterior priorities in slice 2 of the previous TACN.

Within slices 1 and 2, the *Student Knowledge*$_s$ subnetworks have the same basic structure as the problem solution graph: atemporal arcs from rule nodes model the influence of rule knowledge on the student's ability to derive related steps, and atemporal arcs between step (fact or goal) nodes model prerequisite relations. Temporal arcs between corresponding nodes in adjacent slices model the persistence of the student's knowledge over time.

Student Knowledge$_1$ represents the influence of the tutor's action on the student's knowledge. The tutor normally considers addressing only steps that are *ready* or *in_focus* – this is the reason for the shading in Figure 3.4. The tutor also considers tutoring on rules related to steps that are *ready* or *in_focus*, since (1) these rules are more likely to be in the student's focus of attention, and (2) tutoring on them may provide the knowledge necessary for the student to complete the corresponding step. Therefore, in Figure 3.4 there are arcs from the tutor action nodes to *Step 2*$_1$ and *Step 3*$_1$ (there is not an arc to *Step 1*$_1$ since it was given) and to *Rule A*$_1$ since it is the parent of both *Step 2*$_1$ and *Step 3*$_1$.

Given the topic of a student action (*Student Action Topic*$_2$), the student's knowledge of the topic influences the student action type (e.g., *correct*, *error*, or *impasse*), *Student Action Type*$_2$. Therefore, *Student Knowledge*$_1$ step nodes influence *Student Action Type*$_2$.

In slice 2, the *Student Action*$_2$ nodes do not directly influence *Student Knowledge*$_2$ nodes. This is because a student action does not influence the student's knowledge without feedback (e.g., from the tutor), which is not modeled until the next TACN. Rather, once *Student Action*$_2$ has been observed, it influences *Student Knowledge*$_1$ nodes diagnostically, which in turn influence the corresponding *Student Knowledge*$_2$ nodes.

3.2.7 Discourse State subnetworks

DT Tutor employs a relatively simple model of the discourse state as illustrated in Figure 3.2. *Discourse State*$_0$ is simply the most recent student action (*Student Action Topic*$_0$ and *Student Action Type*$_0$, which have the value *null* for the initial TACN). *Discourse State*$_1$ models the influence of the tutor's action on

the discourse state. There is no separate representation for *Discourse State*$_2$ because (1) the utility of the tutor's contribution to the discourse state is based on *Discourse State*$_1$, and (2) the contribution of the *Slice 2* student action to the discourse state is represented by the student action itself as *S Type*$_0$ and *S Topic*$_0$ of the next TACN.

In *Discourse State*$_1$, the *Coherence*$_1$ node ("*Cohere*$_1$" in Figure 3.2) represents the coherence of the tutor's action in response to the previous student action as either *coherent* or *incoherent*. For instance, *negative feedback* in response to a student *error* is *coherent*. It is important to note that the tutor is not obligated to select a *coherent* action because such actions may not have maximum expected utility.

The *Relevance*$_1$ node ("*Relev*$_1$" in Figure 3.2) in *Discourse State*$_1$, with values *high* and *low*, models how well the tutor cooperates with the student's focus of attention by assessing the extent to which the same problem steps are *in focus* before and after the tutor's action. Problem steps that are in the student's focus of attention are likely to be *in focus* in *Student Focus*$_0$. A tutorial action that addresses a problem step or related rule that is in the student's focus of attention will further increase the probability that the problem step is *in focus* in *Student Focus*$_1$. Therefore, if the same problem steps are most likely *in focus* in *Student Focus*$_0$ and *Student Focus*$_1$, *Relevance*$_1$ is most likely *high*.

3.2.8 Student Independence nodes to model affect

The *Student Independence*$_s$ nodes shown in Figure 3.2 represent the tutor's beliefs about the student's feeling of independence, or self-efficacy within the domain (e.g., whether she feels like she can solve problems without the tutor's help). Independence is one of the attributes modeled in del Soldato and du Boulay's (1995) seminal work for affective modeling within ITSs, and is related to the attributes of *challenge* and *confidence* suggested by Lepper and colleagues (1993). The *Independence*$_s$ nodes have five possible values, *level 0* through *level 4*, with higher levels representing greater independence. Both tutor and student actions influence the *Independence*$_s$ nodes.

Tutor Action Type$_1$'s influence on *Independence*$_1$ is in inverse relation to the explicitness of the help provided because explicit help prevents the student from increasing her feeling of independence by achieving successes on her own. For instance, a *Tutor Action Type*$_1$ value of *null* (no help action) slightly increases the student's feeling of independence, a value of *prompt* (the least explicit help action) slightly decreases the student's feeling of independence, and a value of *do* (the most explicit help action) sharply decreases the student's feeling of independence.

Similarly, successful student actions increase the student's feeling of independence while unsuccessful actions decrease it. For instance, a *Student Action Type*$_2$ value of *correct* boosts *Independence*$_2$ while a *Student Action Type*$_2$ values of *error* or *impasse* have the opposite effect. Arcs

between corresponding *Independence*$_s$ nodes in adjacent slices model the persistence of the student's affective state over time.

3.2.9 Student Help Style nodes

The *Student Help Style*$_s$ nodes ("*H Style*$_s$" in Figure 3.2) represent the way that the student uses help in the current tutorial session as either *neutral* or *abuse*. *Student Help Style*$_0$ influences *Student Knowledge*$_1$ because student help style influences the way that the student uses the tutor's help (the influence from the *Tutor Action*$_1$ nodes). *Student Help Style*$_1$ in turn influences *Student Action Type*$_2$ – e.g., even if the student knows how to do a particular step, if her help style is *abuse* she may, e.g., request help instead of attempting the step on her own. *Student Action Type*$_2$ can in turn influence *Student Help Style*$_1$ diagnostically. Temporal arcs between slices represent the relative persistence of the student's help style over the course of a tutoring session, although it can be influenced by tutor actions (e.g., refusing to provide explicit help) and student actions (e.g., correct problem-solving actions that increase the student's confidence in being able to complete problem steps without the tutor's help.

3.2.10 Utility subnetwork

Node *Utility*$_2$ in Figure 3.2 is actually a number of utility nodes in a structured utility model representing tutor preferences regarding the following outcomes:

1. Student rule knowledge in slice 2 (rule nodes in *Student Knowledge*$_2$)
2. Student problem solving progress in slice 2 (step nodes in *Student Focus*$_2$)
3. Student independence in slice 2 (*Independence*$_2$)
4. Student help style in slice 2 (*Student Help Style*$_2$)
5. Tutor action type in slice 1 (*Tutor Action Type*$_1$)
6. Discourse state coherence in slice 1 (*Coherence*$_1$)
7. Discourse state relevance in slice 1 (*Relevance*$_1$)

DT Tutor uses linearly-additive multiattribute utility functions to combine subutilities for the outcomes above: Subutilities are combined by assigning a weight to each subutility, multiplying each subutility value by its weight, and summing the weighted subutility values. These functions make it easy to change DT Tutor's behavior by simply changing the weights. For instance, DT Tutor will focus on student rule knowledge at the expense of problem-solving progress if a high weight is assigned to the former and a low weight is assigned to the latter.

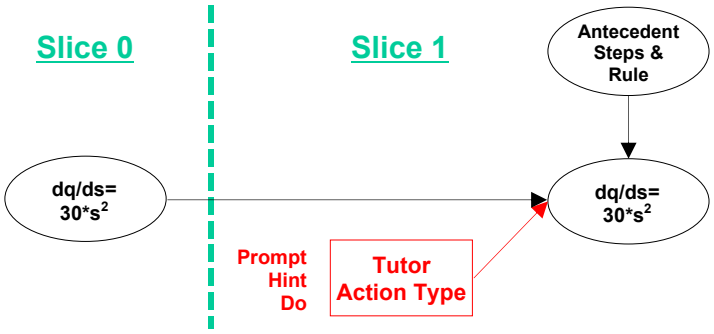
3.2.11 Filter nodes

Filter nodes are used to reduce the sizes of conditional probability tables. Many nodes in the *Student Knowledge*$_s$ and *Student Focus*$_s$ subnetworks are influenced by the *topic* and *type* of either the tutor's action (in slice 1) or the student's action (in slice 2), but only if the action topic (*Tutor Action Topic*$_1$ or *Student Action Topic*$_2$) corresponds to the node. For instance, in Figure 3.3, node *Step 2*$_1$ in the *Student Focus*$_s$ subnetwork is influenced by the tutor's action only if *Tutor Action Topic*$_1$ (abbreviated to *Tutor Topic*$_1$ in the figure) has the value *Step 2*. *Tutor Action Topic*$_1$ always has other possible values as well, including *null* and often values corresponding to other step nodes. Without a filter node, the conditional probability table for *Step 2*$_1$ would require a complete set of entries for every combination of values for *Tutor Action Topic*$_1$ and *Tutor Action Type*$_1$ even though *Step 2*$_1$ is influenced by the tutor action only if *Tutor Action Topic*$_1$ has the value *Step 2*. Instead, deterministic filter nodes are inserted between the *Tutor Action Topic*$_1$ and *Student Action Topic*$_2$ nodes and the nodes that they influence (herein called *target nodes*). Each filter node reduces the influence of the tutor or student action *topic* node to a binary distinction: either the topic node directly influences the target node or it does not. Since filter nodes have only one input (an action topic node) and a binary output, their conditional probability tables remain relatively small while they limit the increase in the size of their target nodes' conditional probability tables to a factor of two. Filter nodes wherever they are useful throughout TACNs but do not clutter network diagrams because they have no effect on network semantics.

3.2.12 Rule-based conditional probability table creation

Conditional probability table (CPT) entries make up the vast majority of a TACN's numeric entries. They often follow patterns so that the CPTs of different nodes may be similar in many respects even if the nodes have different numbers of influences from different parent nodes. For instance, consider the CPTs of two different *Student Knowledge*$_1$ subnetwork step nodes, one of which has one parent (antecedent)

step, and the other of which has two parent steps (both steps also have rule node parents). For both nodes, if any of the parent nodes is *unknown*, then the student does not have the information to complete the step successfully, so the step is likely to be *unknown* unless the student guesses correctly. Conversely, for both nodes, if all of the parent nodes are *known*, then the step is likely to be *known* unless the student somehow slips in making the inference required for the step. The probability of a correct guess or a slip is likely to be about the same for both steps, at least in the absence of knowledge about special circumstances related to either node – probabilities summarize uncertainty due to such ignorance anyway (Russell & Norvig, 1995). Similar levels of tutor help may also have similar probabilities of helping a student know different steps, again in the absence of more specific knowledge, and especially in extreme cases, such as when the tutor simply prompts the student (providing little or no information) or when the tutor tells the student exactly how to do a step.



Conditional Probability Table for *dq/ds = 30*s²* in Slice 1

Step, Slice 0	Unknown						Known					
Antecedents	Unknown			Known			Unknown			Known		
T Action Type	Prompt	Hint	Do	Prompt	Hint	Do	Prompt	Hint	Do	Prompt	Hint	Do
Unknown	1 – g	1 – h	s	s	s	s	f	f	f	f	f	f
Known	g	h	1 –s	1 – s	1 - s	1- s	1 - f	1 - f	1 - f	1 - f	1 - f	1 - f

g= probability of a correct **guess**
h= probability that a **hint** about an unknown step will be successful
s= probability of a **slip** on a known step
f= probability of **forgetting** a step known in the previous slice

Figure 3.5: Simplified CPT for a Student Knowledge step node

DT Tutor uses rules to specify such patterns along with numeric parameters representing the probability of a correct guess, a slip, success after a specific level of tutor help, etc. Figure 3.5, along with the rules listed below, provides an example of automated CPT creation for *Student Knowledge*$_1$ subnetwork step nodes. The table in Figure 3.5 is the CPT for the slice 1 node *dq/ds=30*s²* (shown in Figure 3.1) except

that it has been simplified as follows: (1) The step's slice 1 parents – its antecedent steps and related rule – are simplified to a single node, *Antecedent Steps & Rule*, which has value *unknown* if any of the parents are *unknown* and value *known* otherwise. (2) The *Tutor Action Type*$_1$ decision node has just 3 alternatives – *prompt*, *hint*, and *do* – which can be extended as described below. (3) The influence of *Tutor Action Topic*$_1$ (through a filter node) is not shown. If the value of *Tutor Action Topic*$_1$ does not correspond to this node (i.e., if the tutor addresses some other knowledge element), then *Tutor Action Type*$_1$ does not directly influence the student's knowledge of this node. For these cases, the table entries are the same as the table entries for the content-free *Tutor Action Type*$_1$ value of *prompt*, which also does not influence the student's knowledge of this node.

The rules and parameters follow[3]. Each rule specifies only the conditional probability that the node is *known*, *p(known)*, since *p(unknown)* is simply *1 – p(known)*.

1. If a step is *known* in slice 0, then the step is *known* with probability *1 – f*, where *f* is a parameter representing the probability of forgetting a step known in the previous slice.
2. Otherwise, if all of the step's parents are *known*, then the step is *known* with probability *1 – s*, where *s* is a parameter representing the probability of a slip on a known step.
3. Otherwise, the probability that the step is *known* depends on *Tutor Action Type*$_1$.
    - If *Tutor Action Type*$_1$ is content-free, such as *prompt*, then the step is *known* with probability *g*, where *g* is a parameter representing the probability of guessing an unknown step correctly.
    - If *Tutor Action Type*$_1$ is *do* (tell the student exactly how to do a step), then the step is *known* with probability *1 – s*.
    - If *Tutor Action Type*$_1$ is neither content-free nor as explicit as *do*, the step is *known* with a probability corresponding to the efficacy of *Tutor Action Type*$_1$ at conveying the information. In Figure 3.5, this probability is *h*, a parameter representing the probability that a *hint* about an unknown step will be successful. This schema is easily extended to various levels of *hint* efficacy and to other *Tutor Action Type*$_1$ values such as *teach*.

[3] The parameters *g*, *h* and *s* represent subjective parameters before DT Tutor began learning conditional probabilities empirically. With empirically learned probabilities, all of the entries on the left side of the table (for *Step, Slice 0 = unknown*) are copied from Table 4.6.

3.3 THE STUDENT INTERFACE

A Java-based student interface was created to be used with all three tutorial action selection engines using a common pool of help messages. Appendix A provides an introduction to the Calculus Tutor's student interface and its domain from a student's point of view, including a screen shot of the interface in Figure A1, and Appendices D, E and G provide a variety of screen shots and sample help messages.

3.3.1 Reification of goal structure in the Goals Window

For each problem, the problem solution graph created by the domain expert (see Figure 3.1 for an example) is used as the basis for a reification of the problem's goal and solution structure in the Goals Window. The Goals Window is an extended and customized implementation of Singley's (1986) goal reification for a Cognitive Tutor in the same domain, intended to communicate the goal structure underlying the problem solving as required for Cognitive Tutor implementations (Anderson et al., 1995), Appendix D shows screen shots for each of the 5 problems used in this study after all problem steps have been successfully completed with the goals in the Goals Window fully expanded. (In these screen shots, the Goals Window has goal numbers added in brackets to the right of each goal that are not actually displayed in the interface.) The list of goals is not normally fully expanded but instead expands as subgoals are created and shrink as subgoals are completed as illustrated in the following example.

3.3.1.1 An extended example of student interface displays

At the start of problem P1, whose problem solution graph is shown in Figure 3.1[4], the Goals Window contains the following:

find equation form **dq/ds=<number>** using **evaluate**
evaluate operand 1: find equation form _____
evaluate operand 2: s=2

and the Accepted Equations Window contains the 3 given equations:

[4] It might be instructive to trace this extended example through Figure 3.1 as well.

(1) dq/dr=3 given

(2) dr/ds=10*s^2 given

(3) s=2 given

The "_____" at the end of the second line is highlighted in blue as a prompt to indicate that the student should click on it to work on a problem step. When the student clicks on it, a "Select Equation Form" Dialog Window (see the Dialog Windows in Appendix E) is displayed. The Calculus Tutor Tutorial covered how to select the correct equation form. If the student selects correct equation form *dq/ds=f(s)*, the following would be displayed in the Goals Window (the Accepted Equations Window would remain unchanged for now):

find equation form **dq/ds=<number>** using **evaluate**

evaluate operand 1: find equation form dq/ds=f(s) using equation form / operator ____

evaluate operand 2: s=2

When the student clicks on the "equation form / operator _____" prompt, the "Use Accepted Equation or Operator" Dialog Window (once again, see Appendix E) is displayed. For this step, the student should click on "Operator(s)" since none of the equations in the Accepted Equations Window has equation form *dq/ds=f(s)*, so an operator must be applied to create an equation of the desired form. When the student clicks "Operator(s)," the "Select Operator(s)" Dialog Window is displayed. If the student correctly uses the heuristics supplied in the Calculus Tutor Tutorial, she will select "chain rule." If she does, the following will be displayed in the Goals Window:

find equation form **dq/ds=<number>** using **evaluate**

evaluate operand 1: find equation form dq/ds=f(s) using chain rule

chain rule operand 1: find equation form _____

chain rule operand 2: find equation form _____

evaluate operand 2: s=2

As shown above, deciding to use operator *chain rule* has caused two subgoals to be set and displayed, corresponding to finding the two operands required to apply the chain rule. If the student clicks on the *chain rule operand 1* prompt, the "Select Equation Form" Dialog Window is displayed. If the student correctly selects equation form *dq/dr=<number>*, the following will be displayed in the Goals Window:

find equation form **dq/ds=<number>** using **evaluate**

evaluate operand 1: find equation form dq/ds=f(s) using chain rule

chain rule operand 1: find equation form dq/dr=<number> using accepted equation / operator _____

chain rule operand 2: find equation form _____

evaluate operand 2: s=2

If now the student clicks on the "equation form / operator _____" prompt, the "Use Accepted Equation or Operator" Dialog Window is displayed. For this step, the student should click on "Accepted Equation" since one of the equations in the Accepted Equations Window has equation form *dq/dr=<number>*. At this point, the "Select Accepted Equation Window" is displayed. To select the correct equation, the student should click on the box to the left of the equation in the Accepted Equations Window. If the student selects the correct equation, *dq/dr=3*, the following is displayed in the Goals Window:

find equation form **dq/ds=<number>** using **evaluate**

evaluate operand 1: find equation form dq/ds=f(s) using chain rule

chain rule operand 1: dq/dr=3

chain rule operand 2: find equation form _____

evaluate operand 2: s=2

As shown, the display of the steps to *find chain rule operand 1* has been collapsed to just the resulting equation, *dq/dr=3*. If the student wants to, she can click on any completed goal to toggle back and forth between the short and long forms of the goal display:

chain rule operand 1: dq/dr=3

… or …

chain rule operand 1: find equation form dq/dr=<number> using given equation – Result: dq/dr=3

The student should then use a similar procedure to find chain rule operand 2 (another given equation). At this point, the Goals Window will look like this:

find equation form **dq/ds=<number>** using **evaluate**

evaluate operand 1: find equation form dq/ds=f(s) using chain rule – Result _____

chain rule operand 1: dq/dr=3

chain rule operand 2: dr/ds=$10*s^2$

evaluate operand 2: s=2

Note that a "Result _____" prompt has been added to the end of goal line 2, indicating that it is now time to apply the chain rule to its two operands. If the student clicks on this prompt, the "Enter Equation" dialog window is displayed, directing the student to enter the result of applying the chain rule in the Equation Entry Window. If the student enters the correct equation, a new equation will be added to the Accepted Equations Window:

(1)	dq/dr=3	given
(2)	$dr/ds=10*s^2$	given
(3)	s=2	given
(4)	$dq/ds=30*s^2$	chain rule(1,2)

The new equation (4) lists as its derivation "chain rule(1,2)," indicating that the chain rule was applied to equations (1) and (2). The Goals Window will now look like this:

find equation form **dq/ds=<number>** using **evaluate** – Result _____

evaluate operand 1: $dq/ds=30*s^2$

evaluate operand 2: s=2

Note that since the operands for the chain rule are no longer needed, they have been collapsed in the Goals Window. If the student would like to see subgoals that have been completed and collapsed, she can click on a little handle to the left of their parent goal to display them. Again, now that the *evaluate operand 1* subgoal has been completed, just the resulting equation is displayed in the short form of the goal display, but the student can click on it to toggle between the short and long form display. When the student clicks on the first goal, the "Enter Equation" dialog window is displayed, directing the student to enter the result of applying evaluate to its operands in the Equation Entry Window. If the student enters the correct equation, the Accepted Equations window will be updated as follows:

(1)	dq/dr=3	given
(2)	dr/ds=10*s^2	given
(3)	s=2	given
(4)	dq/ds=30*s^2	chain rule(1,2)
(5)	dq/ds=120	evaluate(3,4)

The Goals Window will now display just the final answer in goal line 1:

dq/ds=120

As illustrated above, there are always one or two prompts displayed until the problem is solved. The student may click on any prompt to work on the corresponding step.

3.3.1.2 Immediate flag feedback

Whenever the student makes an entry in the interface, it is immediately flagged as either an error, by highlighting the entry in red, or as correct, by highlighting the entry in green. Besides clicking on blue prompts, the student may click on entry that is highlighted in red to attempt the step again. Green highlighting for correct steps reverts to a black font when the student clicks on another goal so that only the latest correct entry will be highlighted in green.

3.3.1.3 Correspondence between dialog windows and types of rules

There are five different types of Dialog Windows (listed in Appendix E). The "Use Accepted Equation or Operator" window is simply a disambiguation dialog to find out whether the student wants to select an accepted equation or select operator(s). If the student asks for help on the "Use Accepted Equation or Operator" window, she will get help on either selecting an accepted equation or selecting operator(s), whichever is correct for the current problem state. The remaining four types of Dialog Windows correspond to the four different types of steps and related rules. This correspondence is described in further detail in the following section about help messages.

3.3.1.4 Help messages

Help messages are always displayed in the lower portion of the Dialog Window that corresponds to the problem step for which help is being provided. Even if the student clicks the general *Help!* button, the tutor selects a step to provide help for and displays the help in the corresponding Dialog Window.

For each type of step and related rule, four different kinds of help messages are provided: *prompt, hint, teach,* and *do*. The *prompt* message is intended to point out pertinent information that is already available in the interface but not to provide any new information. The *hint* message is intended to provide partial information about the step – not enough to teach the student how to do the step but perhaps enough to either remind the student or help the student figure out how to do the step. The *teach* message is supposed to provide all the information that the student needs to understand the rule related to the step, including at least one example, and thus to help the student complete the step correctly by learning the rule. *Teach* messages were not supposed to tell the student exactly what to enter for a step. However, for Select Operator(s) help messages, this researcher could not think of a way to provide complete information to *teach* the student which operator to select without naming the operator, so the *teach* messages for Select Operator(s) steps do give the answer away although the student must find it amongst a relatively large amount of text. The *do* message was intended to tell the student exactly what to enter for the current step without teaching her anything about the related rule.

The sets of *prompt, hint, teach* and *do* messages for different steps that use the same Dialog Window were all very similar. Appendix G lists sample help messages for each Dialog Window.

4.0 EVALUATION: DATA COLLECTION PHASE

An evaluation was conducted in two phases: (1) data collection and tuning, and (2) assessment. This chapter describes the data collection and tuning phase.

4.1 GOALS OF THE DATA COLLECTION AND TUNING PHASE

The data collection and tuning phase of the evaluation had three purposes. First was to provide empirical data for learning many of DT Tutor's (DT's) probabilities. DT uses its probabilistic model of the tutorial state to compute probabilities for the various possible outcomes of its tutorial action alternatives. The probabilities of the possible outcomes, combined with their utilities, are the basis of DT's tutorial action selections (see section 1.2.1 for a more in-depth explanation). Thus, the probabilities in DT's probabilistic model are a core influence on which tutorial actions it selects. A decision-theoretic representation supports obtaining probabilities and utilities from any combination of the best sources available. For instance, they can be based on (1) subjective beliefs, (2) logic, (3) pedagogical, cognitive, and psychological theory, and (4) empirical data such as results from pretests, logged student interactions with the system, and posttests. Obtaining key probabilities from empirical data was the first purpose of this phase.

The second purpose of the data collection phase was tuning DT's utilities. DT, like human tutors (Merrill et al., 1992), undertakes a delicate balance, considering multiple attributes of the tutorial state in order to decide which tutorial action to select. If one or more probabilities or utilities are significantly out of balance – for instance, if the utility for one tutorial attribute is given too much weight – then this delicate balance can be upset. To avoid this situation, the experimenter performed minor tuning of DT's utilities so that it seemed to behave reasonably according to that experimenter's subjective opinion, not knowing how it would perform on the collected scenarios or how the judges' opinions would compare.

The final purpose of the data collection phase was to collect scenarios from tutorial interactions with real students as the basis for the second phase, assessment. Each scenario was a point in an

interaction between a computer tutor and a student where the computer tutor must decide what tutorial help action to select, if any. The scenarios were collected in order to compare the tutorial help actions selected by different computer tutors in identical situations.

4.2 DESIGN OF THE DATA COLLECTION EXPERIMENT

4.2.1 Subjects

The subjects, or students, were required to be at least 18 years old and to have taken algebra but not calculus. They were recruited from ads posted around the University of Pittsburgh campus and in the University of Pittsburgh student newspaper. All students who met the requirements and passed the pretest (described below) were accepted, so their demographics varied; not all of them were University of Pittsburgh students. The students were paid $7/hour for their participation. 60 students completed the procedure in its entirety; their data was used for the remainder of the evaluation.

4.2.1.1 Printed materials

The printed materials consisted of the following:

1. A questionnaire to collect demographic information: age, gender, native language, high school grade point average, SAT scores, college/university attendance, and math classes taken in high school and college.
2. A 12-page tutorial on solving calculus related-rates problems and using the Calculus Related Rates Problem Tutor (*Calculus Tutor* for short) interface. Assuming an understanding of basic algebra, the tutorial covered everything the students needed to know in order to use the tutor. This tutorial is included as Appendix A.
3. A pretest and posttest consisting of multiple-choice and short-answer questions that tested each of 28 concepts needed to solve the problems that students would face using the Calculus Tutor. The pretest and posttest problems were different but isomorphic both in surface and solution structure. The orders of equivalent problems in the two tests were the same except for one set of problems – selecting which operator(s) to use – for which using the same order might have given the answers away. A copy of the posttest is included as Appendix B.
4. A tip sheet that students read and then used as they wished while they were using the Calculus Tutor. The tip sheet included instructions for using the Calculus Tutor interface and a list of

symbols and definitions. 27 of the 60 students used a tip sheet which had an additional instruction, listed in Figure 4.1, intended to dissuade them from using help when they didn't really need it (Murray & VanLehn, 2005). For these *help-dissuaded* students, the Calculus Tutor delayed for 10 seconds after the student clicked the help button before the tutor provided help[5]. The remaining 33 students were not given any instructions about whether and when to request help, and they did not experience a help delay from the Calculus Tutor.

Do not request help unless you need it. If you need help, click on either the Help! button at the bottom of the screen or a help button displayed in the Prompt Window.

- ***There will be a substantial delay before help is presented*** to discourage students from requesting help when they don't really need it. During this delay, the help button will darken but nothing else will happen.
- ***Do not click any other buttons while waiting for help*** or you may lose the help you are waiting for.

Figure 4.1: Text discouraging help requests on 27 of 60 tip sheets

4.2.1.2 The Random Tutor

A version of the Calculus Related Rates Problem Tutor, the *Random Tutor*, was created especially for the data collection phase of the experiment. The Random Tutor used the same student interface as DT Tutor, described in section 3.3, with the same pool of help messages so that only the method used for selecting tutorial actions was different. For each opportunity to provide tutorial help, the Random Tutor used *random selection among relevant actions*: For proactive help opportunities, the Random Tutor decided randomly, with a 50% probability, whether to provide proactive help. Proactive help opportunities occurred (1) whenever a student clicked on a prompt within the Goals Window of the interface, and (2) whenever a student made an error. For reactive help opportunities (in response to a student's help request), the Random Tutor always provided help. If the student clicked the interface's general *Help!* button, the Random Tutor randomly decided which step to provide help for (the tutorial action *topic*) from

[5] The commercial version of Cognitive Tutor Algebra, a model-tracing tutor, delays successive levels of help to prevent rapid-fire help requests (Baker et al., 2004).

among the steps with prompts in the Goals Window. Otherwise, the tutorial action topic had already been determined by either (1) the step that the student had clicked within the Goals Window, or (2) the step that the student was working on when she clicked the help button within the Prompt Window. If the Random Tutor decided to provide help, it selected from among the tutorial action *types* of *prompt*, *hint*, *teach* and *do*; otherwise, it selected the *null* action type. To avoid excessive repetition of help types for repeated help opportunities, the Random Tutor decided in advance a random order for the response types *prompt*, *hint*, *teach* and *do*, and then returned response types in that order, repeating the order cyclically if necessary.

The Random Tutor selected tutorial actions randomly for two purposes: First, random selection of help served as a control condition during the assessment phase versus the more principled fixed-policy and decision-theoretic methods. Second, random selection was used to collect data about the effectiveness of individual help actions during tutoring: A potential confound when assessing the effectiveness of an individual help action in a series of help actions is that the apparent effectiveness of the individual help action may be due to the accumulated effectiveness of a sequence of help actions. Since sequences of help actions cannot be avoided (students sometimes need more than one incidence of help to complete a step), instead the effects of sequences of help actions were controlled statistically by randomizing over the sequences in which individual help actions occurred.

Students solved five multi-step problems using the Random Tutor. The five problems were set up so that students would encounter each of the 28 covered domain concepts at least twice during problem solving. Students were allowed as much time as they needed to complete the problems and most took about an hour.

4.2.2 Procedure

Each student carried out a fixed procedure, during which time they were allowed to take as much time as they wanted – i.e., the task was fixed but time on task was not. Students could complete the procedure over one or two sessions and take breaks as they liked. After completing a consent form, the students filled out a brief demographic questionnaire. They then studied a 12-page tutorial on solving calculus related-rates problems and using the Calculus Related Rates Problem Tutor (*Calculus Tutor* for short), which took an average of about 45 minutes. Assuming an understanding of basic algebra, the tutorial covered everything the students needed to know in order to use the Calculus Tutor. After completing the tutorial, the students could end their first session if they wished. Students who divided their participation into two sessions were allowed to review the tutorial at the start of the second session. Next, students turned in the tutorial – they were not allowed to look at it again – and took the pretest, which took an

average of about 25 minutes. Students who did not score too low or too high on the pretest were allowed to continue. For students who were allowed to continue, the lowest score was 8 out of 28, the highest score was 25, the mean score was 18.5, and the median score was 19. Next, students solved five multi-step problems using the Random Tutor. After students finished solving the five problems using the Random Tutor, they took the posttest (which was isomorphic to the pretest), completing their participation. Posttest scores were a low of 5 (unlike on the pretest, students had no external motivation not to score too low), a high of 27, a mean of 20.9, and a median of 23.

4.3 PARTITIONING INTO TRAINING AND TEST DATA SETS

The student data was partitioned into training and test sets of 30 students each, which were matched according to pretest scores. The training and test sets were established by repeatedly randomly partitioning the 60 students into two groups and then using a t-test to compare the two groups' pretest scores until a pair of partitions was found for which t=0.0. Students who were dissuaded from using help unless they really needed it (see section 4.2.1.1, item 4) were split as evenly as possible between the groups, with 14 in the training set and 13 in the test set. Subsequently, the training set was used for all procedures prior to the second phase of the evaluation, assessment, for which the test set was used.

4.4 LEARNING PROBABILITIES EMPIRICALLY

The structures of DT Tutor's (DT's) probabilistic networks were determined in advance, as described in section 3.2. The overarching structure of DT's 3-slice dynamic decision network architecture, which is common to all of DT's networks, was determined by task analysis; indeed, it is one of DT's contributions to the field of intelligent tutoring systems (see section 6.7.1). Within each network are problem-specific subnetworks, the *Student Knowledge* and *Student Focus* subnetworks in each slice, whose structure is computed by DT's domain expert as summarized in section 3.1.2. With the structures of DT's networks thus postulated, there was no need to learn them.

The parameterization of DT's networks – learning prior and conditional probabilities – was another story. Determining prior and conditional probabilities is a fundamental part of constructing any probabilistic network. DT's probabilities are a core influence on its tutoring behavior, as discussed in sections 1.2.1 and 4.1. Data collected from pretests, logged student interactions with the Random Tutor,

and posttests was used to learn many of DT Tutor's (DT's) key probabilities empirically. To learn DT's probabilities, a number of challenges had to be faced:

1. Learning about students in the presence of help abuse
2. Unobservable variables such as student rule knowledge
3. Learning with sparse data
4. Variables that change over time

The methods used to address these challenges are described in the following subsections. Section 6.1 further discusses the methods used, some surprises in the learned probabilities, and related work. Section 6.6.3.2 discusses future directions.

Many of DT's conditional probability tables were constructed using rule-based techniques for principled creation of thousands of conditional probability entries from a relatively small number of seed probabilities, as described in section 3.2.12. 158 freely determined probabilities (not counting probabilities that were constrained to a specific value because probabilities must sum to 1) were learned from the training set of student data. 117 freely determined probabilities were specified subjectively but 84 of these were for just the three *Independence* nodes (1 in each of the 3 network slices), leaving 33 freely determined probabilities specified subjectively for the rest of the network. These remaining subjectively-specified probabilities were all for unobservable variables: the evolution of the student's help style (this variable is discussed in section 4.4.1), the influence of the student action type (e.g., *correct*, *error* or *impasse*) on the student's focus of attention (modeled by the *Student Focus*$_s$ networks, described in section 3.2.5), focus aging in the *Student Focus*$_s$ networks, and a frame axiom that a student's knowledge remains unchanged in the absence of other influences.

Only basic techniques were used for learning DT's probabilities. The probability for each outcome of a distribution was calculated simply as the ratio of events with that outcome to the total number of like events. For conditional probabilities, the denominator of this ratio was the number of events with the same values for the conditioning variables. Section 6.1.1 briefly discusses more advanced techniques.

Most of the conditional probabilities that were learned concerned the influence of the various tutorial help actions on the student. The influence of the tutor's help depends largely on the amount of attention and effort the student devotes towards trying to understand and use the help. A conditional probability representing the likelihood that a student will learn from a particular help action should likewise depend upon the way the student uses the help – the student's *help style*. DT Tutor models the

student's help style (see section 3.2.9) for precisely this reason. Accordingly, before learning conditional probabilities, the training set was partitioned according to the students' apparent help styles.

4.4.1 Identifying student help style, including help abuse

Unfortunately, sometimes students misuse help. For example, when bottom-out help is available that tells a student exactly what to do (such as the Calculus Tutor's *do* help), as many as 82-89% of students who request help continue to request it until they get to the bottom-out help (Aleven & Koedinger, 2000). In another study (Aleven et al., 2004), a large percentage of students who abused help did so by "clicking through" intermediate level or less explicit help, not spending enough time viewing the help to use it. Bottom-out help is sometimes necessary to prevent students from getting stuck when they don't understand the tutor's help (Anderson et al., 1995), but it can be susceptible to abuse. Students who abuse help by ignoring less explicit help are probably less likely to learn from less explicit help than students who pay it careful attention.

There are also other ways of using help unproductively or "gaming the system" (Baker et al., 2004). For example, sometimes students appear to attempt steps without taking enough time to think (Aleven & Koedinger, 2000), perhaps engaging in a trial-and-error approach and perhaps intentionally eliciting proactive help after their errors. A potentially new way of eliciting proactive help was observed in this study (Murray & VanLehn, 2005): some students repeatedly began a step by clicking on a goal and then canceled the ensuing prompt until they received proactive help. This pattern of behavior was likely caused by a unique set of circumstances. First, the Random Tutor (described in section 4.2.1.2) provided proactive help about 50% of the time when students clicked on a goal (before the student had a chance to attempt the step). Second, some students were discouraged from asking for help unless they really needed it, as described in section 4.2.1.1, item 4. Due to this combination of circumstances, students may have been motivated to elicit help without asking for it, and with the Random Tutor's random provision of proactive help, they could. Similar patterns might be observed with other computer tutors that provide proactive help if there is any motivation to avoid requesting help – for instance, many model-tracing tutors link progress through the tutor with help-seeking behavior (Anderson et al., 1995).

Besides abusing help, sometimes students do not ask for help even when they need it (Aleven et al., 2004). Among the 30 students in the training set, 7 students never requested help. Only 4 of these students were help-dissuaded (see section 4.2.1.1, item 4), so some students did not request help out of their own volition. (Interestingly, 2 other help-dissuaded students in the training set made 59 and 60 help requests respectively, the second- and third-most in the training set, so not all help-dissuaded students avoided requesting help.) Some students who did not request help may not have needed it, as evidenced

by one who made only 17 errors, the least among training set students and well below the median of 45.5. But another student who never requested help made 132 errors and elicited 119 instances of proactive help – both counts by far the highest among training set students. Other students who avoided asking for help may have paid careful attention to the proactive help that they did receive in order to avoid needing more help.

For the purpose of learning empirical probabilities for DT Tutor (DT), it does no good to learn conditional probabilities based on distinctions that DT does not model. DT does model the student's help style as described in section 3.2.9, currently as either *abuse*, which means that the student solicits or elicits help that she does not need, or *neutral* for all other students. DT does not model help avoidance at this time for two reasons. The main reason is that many of the students who avoid requesting help use the help that they do receive (e.g., proactive help) effectively, much like help-neutral students who use help appropriately. Therefore, conditional probabilities representing the effectiveness of tutorial help for these students should be about the same as for help-neutral students. Avoiding requesting help is not necessarily a problem, especially when proactive help is available, as with the student who made only 17 errors. Help-avoiding students who make excessive errors and elicit excessive proactive help, like the student who made 132 errors and elicited 119 instances of proactive help, need to be treated differently not because they do not ask for help, but because they elicit help that they may not need. Such students are more like help abusers in obtaining an excessive amount of help but not using it effectively, so they are classified as help abusers.

The second reason that DT does not model help avoidance at this time is that it would not treat help avoiders differently even if it could detect them. Help avoiders who also abuse help are classified as help abusers, as discussed above. For the remaining help avoiders, their essential difference from help-neutral students is that when they don't know how to complete a step, they are more likely to make an error than to request help. But currently help requests and errors have the same effect on DT's utility for the tutorial situation – they are both just evidence that the student does not know the required knowledge element[6] – so DT has no preference about whether the student requests help or makes an error. Anecdotal evidence from this study (see the judge's comment at section 5.5.1, item 2.b) suggests that the judges might prefer that the tutor provide proactive help when a student avoids using help, so perhaps DT will be modified in the future to treat help avoiders differently.

So far in this section, a student's help style has been discussed as if it were a static property of the student. But it is more likely that a student's help style can change dynamically according to the mix of motivators present in the tutorial situation. For instance, a help-dissuaded student might start a tutorial

[6] Model tracing tutors such as the PACT Geometry Tutor also treat help requests and errors the same for knowledge tracing (Aleven & Koedinger, 2000).

session by avoiding asking for help but after she finds out that the "substantial delay" before help is presented (see section 4.2.1.1, item 4) is only 10 seconds, she might begin to ask for help much more often. However, it is difficult to detect minute-by-minute changes to a student's help style because interpretation of a student's help-seeking behavior depends partly on her knowledge and intentions, which are unobservable. Instead, for the purposes of this study, a student's help style is defined as the dominant help style apparent over the course of a tutorial session.

As described above, help abuse can be manifested differently by different students. Examples included continually clicking all the way through to bottom-out help, not spending enough time with less explicit help to learn from it, requesting help excessively, or eliciting excessive proactive help (even with no help requests) either by making excessive errors or by clicking on goals and canceling prompts until help is received. Therefore, there was no one measure of student behavior that would capture all instances of help abuse. Instead, help abusers were identified subjectively by considering a mix of behavior measures:

1. Number of help requests
2. Total help received (to detect students who elicited help excessively by whatever means)
3. Number of errors
4. Net pretest to posttest gain (to detect students who did not learn much from the help that they received)
5. Ratio: correct answers after bottom-out help / correct answers after all kinds of help (to detect students who were rarely helped except by bottom-out help)
6. Number of help messages that were viewed for 2 seconds or less

Table 4.1 lists 7 students (by subject ID) who were identified as help abusers along with their scores on the behavior measures listed above. For comparison purposes, Table 4.1 also lists the median value on each of these measures for all students in the training set (median is used instead of mean because the help abusers were outliers who skewed the means of the distributions). As shown in the table, all of the students who were identified as help abusers performed worse than the median on at least 4 of 6 measures.

It is interesting to note that the students identified as help abusers performed at least as well on the pretest as their help-neutral peers, with a mean of 19.6 and a median of 20 versus a mean of 18.2 and a median of 18 for the help-neutral students, although this difference in pretest scores was not statistically significant. However, the help-abusing students did not learn nearly as much as the help-neutral students

did, with a mean net gain of -2.9 and a median of -4 versus a mean of 3.7 and a median of 3 for the help-neutral students. This difference in net gain scores was significant, $t(28)=4.130$, $p<.001$.

Since 7 of 30 students, or approximately 25%, were classified as help abusers, the prior probabilities for DT's model of student help style, *Student Help Style*$_0$, were set to .25 for *abuse* and .75 for *neutral*.

Table 4.1: Help abusers and their scores on help abuse measures

Student	Help Requests	Total Help	Errors	Net Gain	Success After *do*	Help Msgs Clicked Thru
s10	59	135	68	-4	.78	10
s20	43	94	28	1	.86	3
s43	52	120	47	-6	.76	0
s67	0	119	132	-5	.68	1
s73	47	108	40	-2	.74	15
s80	109	197	86	-5	.78	77
s88	60	151	70	1	.58	11
Training Set Median	5.5	83.5	45.5	1	.55	0

4.4.2 Learning prior probabilities

Besides prior probabilities for student help style, prior probabilities were learned for each of 28 key domain concepts, or rules, required to successfully use the Calculus Related Rates Problem Tutor (*Calculus Tutor* for short). Each of these rules was tested on the pretest. Pretest performance should not have been directly influenced by student help style because there was no help given until after the pretest and because students were motivated to do as well as they could on the pretest in order to continue participating in the study. 30 data points were available for each rule with no missing data since each rule was tested individually on the pretest and all 30 students in the training set completed the pretest. The learned prior probability that a rule was *known* was simply the ratio of students who got the pretest problem right to the number of students assessed (30). Of course, the probability for *unknown* was 1 – p(*known*). The learned prior probabilities are listed in .Table 4.2

Table 4.2: Learned prior probabilities for Calculus Tutor domain rules

Rule	Probability
1. Select equation form	
chain rule operand 1	.43
chain rule operand 2	.67
differentiate operand	.73
evaluate operand 1	.40
flip derivative operand	.93
integrate operand	.70
restate operand	.93
substitute operand 1	.57
substitute operand 2	.73
2. Apply operator	
execute chain rule	.80
execute differentiate	.70
execute evaluate	.87
execute flip derivative	.97
execute integrate	.60
execute restate with exponent	.83
execute restate with multiplication	.97
execute substitute	.83
3. Find equation form	
find equation form - function	.90
find equation form - number	.87
4. Select operator	
select chain rule	.27
select differentiate	.47
select differentiate after substitute	.50
select flip derivative	.70
select integrate	.33
select integrate after chain rule	.57
select restate with exponent	.63
select restate with multiplication	.50
select substitute	.13

The first set of domain rules in Table 4.2, *select equation form*, concern specifying the equation form for each operator's operand(s) as explained in the Equation Forms section of the Calculus Tutor Tutorial in Appendix A. The second set of rules, *apply operator*, concerns how to apply each of the operators to its operands as explained in the Operators section of the Calculus Tutor Tutorial. The third set of rules, *find equation form*, concerns finding an equation from the list of Accepted Equations that matches an equation form, as explained in the Equation Forms and The Calculus Tutor's Problem-Solving Procedure sections of the tutorial. The last set of rules, *select operator*, involves selecting operators as explained in the Heuristics (Rules of Thumb) for Selecting Operators section of the Calculus Tutor Tutorial (included as Appendix A).

As Table 4.2 shows, prior probabilities varied significantly for different rules, ranging from a low of .13 for selecting operator *substitute* to a high of .97 for executing the flip derivative operator, which should have been familiar from the students' backgrounds in basic algebra. Prior probabilities also varied significantly across sets of rules as shown in Table 4.3, which lists for each set of rules the number of rules and descriptive statistics for that set's prior probabilities: mean, minimum, maximum and standard deviation. Judging by both mean and minimum prior probabilities, the *select operator* rules seemed to be most difficult, followed by *select equation form*, *apply operator*, and *find equation form* in that order.

Table 4.3: Learned prior probabilities by rule type

Rule Type	N	Mean	Min	Max	Std Dev
Select equation form	9	.68	.40	.93	.19
Apply Operator	8	.82	.60	.97	.13
Find Equation Form	2	.89	.87	.90	.02
Select Operator	9	.46	.13	.70	.18

4.4.3 Learning conditional probabilities

As described in section 4.2.1.2, the Random Tutor was used during the data collection phase partly to collect data about the effects of *individual* tutorial actions by statistically controlling for the effects of *sequences* of tutorial actions by randomization. CAPIT, one of few other decision-theoretic tutors, likewise used a random data collection strategy (Mayo & Mitrovic, 2001).

4.4.3.1 Learning conditional probabilities related to unobservable variables

The bulk of the key conditional probabilities to be learned were for the effects of tutorial help actions on student rule knowledge and student problem-solving progress. A student's rule knowledge at one point in time (time slice) is strongly influenced by her rule knowledge at previous points in time (previous time slices). A student's problem-solving progress is strongly influenced by her rule knowledge as well. But a student's rule knowledge is unobservable and so must be estimated based on observable tutorial state attributes. So the bulk of the key conditional probabilities to be learned had to do with unobservable attributes: either they were for an unobservable variable or they were partly conditioned on an unobservable variable.

The Random Tutor's problems were set up so that students would encounter each of 28 rules at least twice during problem solving. The pretest assessed the students' knowledge of the rules before tutoring and the posttest assessed their knowledge after tutoring. Thus, the students' activities were arranged to reveal, for each rule, whether they knew the rule before tutoring, and if not, whether and when they learned it during tutoring. Evidence, either from problem solving or from the posttest, that a student had learned a rule after a tutorial help action was interpreted as evidence that the tutorial action had helped her learn the rule. This evidence is uncertain (and accordingly beliefs are expressed probabilistically), but one can rarely be certain about the details of another person's mental state anyway. The following sections describe how the values of these variables were estimated based on the observable data of pretests, posttests, and logged interactions with the Random Tutor.

4.4.3.2 Learning conditional probabilities with sparse data

While the students' activities were arranged to reveal the evolution of their knowledge of each rule during tutoring, the data gathered during tutoring was often sparse relative to the information needs. For example, the conditional probability table that a particular tutorial help action will help a particular

student know the result of a particular step within a problem solution, conditioned on the student's rule knowledge and the student's help style, requires conditional probabilities for 16 combinations of conditioning variables: 4 different non-*null* tutorial actions (*prompt*, *hint*, *teach*, *do*), times 2 states for the student's rule knowledge (*known*, *unknown*), times 2 states for the student's help style (*neutral*, *abuse*). And that is assuming that the probabilities regarding the effects of tutorial help are the same for different steps involving the same rule, that they do not change over time, and that the effects of tutorial help are the same whether the student asked for it or not (i.e., whether the help is reactive or proactive) – assumptions that were all made due to the sparsity of the data. 16 conditional probabilities would thus be needed for each of 28 rules, for a total of 448 conditional probabilities. Ideally, 20 or 30 sample events would be available for each conditional probability. With just 20 sample events each, 8,960 events would be needed. Since the training set was already partitioned into help-abusing and help-neutral students, only half of the 8,960 help events, or 4,480, would be needed for the 7 help-abusing students. But the help-abusing students had only 1,294 help events in total. In addition, there were times when random proactive help obfuscated whether the student knew a rule without help, as shall be described below, so not all of the help events were usable. And the help events were not evenly distributed either, since some rules were used more than twice and more difficult rules generally had more help events associated with them.

Since there was not enough data to learn all of the conditional probabilities desired, fewer probabilities were learned by combining conditional probabilities for sets of similar events. To illustrate, take the example from the preceding paragraph of a conditional probability that a particular tutorial help action will help a particular student know the result of a particular problem-solving step, conditioned on the student's rule knowledge and the student's help style. Probabilities for different tutorial action types (*prompt*, *hint*, *teach*, *do*) could not be combined because a primary reason that the probabilities were being learned was to decide which tutorial help type to provide. Events where the student does not know the relevant rule and events where the student does know the relevant rule were too dissimilar, as were events involving help-neutral and help-abusing students. Events were more similar for different rules within the 4 sets of rules in Table 4.2: (1) *select equation form*, (2) *apply operator*, (3) *find equation form*, and (4) *select operator*. While prior probabilities varied for the rules within each set, they also varied across sets of rules, with each set of different difficulty as discussed in section 4.4.2 and shown in Table 4.3. Furthermore, all rules within a set involved similar concepts and somewhat similar template-based help messages (help messages are described in section 3.3.1.4). Therefore, for students who did not know a rule when help was given, the height of the conceptual hurdle relative to the scaffolding provided by the help messages was more the same within a set of rules than between sets of rules. So conditional probabilities were combined for rules within each set by aggregating the help events for all rules within a

rule set and performing the conditional probability calculations upon the aggregated events. Other aggregations were performed for other types of conditional probabilities as described in the following sections.

4.4.3.3 Estimating rule knowledge as it changes over time

A student's rule knowledge is unobservable, so it must be estimated by observable events. Evidence from both the pretest and previous interactions with the tutor was used to determine a current estimate of student rule knowledge at any point in time. This current estimate, which was used only for learning probabilities, was binary and deterministic with values *known* or *unknown*. (The tutor's student model, on the other hand, modeled the student's rule knowledge probabilistically.) The current estimate was determined by the student's most recent performance related to the rule. The initial current estimate for each rule was based on pretest performance (*known* if the student got the associated pretest item right; otherwise *unknown*). Thereafter, the current estimate was updated to *known* after a successful problem-solving step without help that told the student exactly what to do on that step (without *do* help or some kinds of *teach* help, as described in section 3.3.1.4). Conversely, the current estimate was updated to *unknown* after a student help request, error, or cancellation of a step attempt.

Estimating rule knowledge proved to be difficult for the 7 of 30 students in the training set who abused help because their performance while using the tutor did not provide reliable evidence (see section 4.4.1 for a discussion). Many of the students who abused help seemed to ask for help in order to avoid applying their rule knowledge to the problem, preferring instead to ask for help until the tutor told them exactly what to do (with bottom-out help), so requesting help was not reliable evidence that they needed it. Some other students seemed to attempt to elicit proactive help by beginning steps and then canceling the ensuing prompts until proactive help was received, so canceling a step was not reliable evidence that they didn't know how to do it. Still others attempted steps without taking much time to think, which oftentimes resulted in errors, so errors were likewise unreliable as evidence that a help-abusing student did not know a rule. In other words, evidence that a student was a help abuser often overwhelmed evidence about whether a student knew individual rules.

For the 23 of 30 students who did not abuse help, these estimates of whether rules were known seemed to correlate more closely with student rule knowledge. Evidence for this comes from much higher empirical probabilities that steps were *known* when the related rule was estimated to be *known*, as shown in Table 4.6.

4.4.3.4 Estimating p(guess) and p(slip)

P(guess) was defined as the probability that a student will know the correct result for a problem step when she does not know the related rule. P(slip) was defined as the probability that a student will not know the correct result for a problem step when she knows the related rule. These probabilities are used in the conditional probability tables for *Student Knowledge*$_1$ step nodes for step attempts when the tutor does not provide help. P(slip) is also used to calculate the conditional probability table entries for *Student Knowledge*$_1$ step nodes when the student knows the related rule and the tutor provides help, as described in section 4.4.3.6.

Both p(guess) and p(slip) were estimated by student performance on step attempts without help (either proactive or reactive) because the presence of help could change the student's knowledge state as she attempted the step. P(guess) was the ratio of correct step attempts to all step attempts when the student did not know the related rule (according to the estimate of student rule knowledge defined in section 4.4.3.4). P(slip) was the ratio of incorrect step attempts to all step attempts when the student did know the related rule. Probabilities were conditioned on student help style (*neutral* or *abuse*) and estimated for all four rule types as shown in Table 4.4. P(guess) and p(slip) appear to be inversely correlated. Especially for help-neutral students, p(guess) was lower and p(slip) was higher for the more difficult rule types, where difficulty is estimated by the prior probabilities shown in Table 4.3. This pattern is not quite as evident for help-abusing students, whose actions were not as reliable as evidence of their knowledge, although the most difficult rule type, *select operator*, and the least difficult, *find equation form*, clearly show the pattern. Also, help-abusing students were much less likely to guess successfully and more likely to slip in their application of a rule when they did not have help, consistent with their tendency to rely on explicit help to progress through problems.

Table 4.4: P(guess) and p(slip) by rule type and help style

	p(guess)		p(slip)	
Rule type	**neutral**	**abuse**	**neutral**	**abuse**
Select equation form	.33	.18	.48	.68
Apply operator	.61	.17	.28	.52
Find equation form	.82	.25	.11	.15
Select operator	.17	.10	.62	.88

4.4.3.5 Estimating effects of help on student rule knowledge

Probabilities for the effects of tutorial help on student rule knowledge were learned for use as conditional probability table entries for the *Student Knowledge*$_1$ rule nodes. A student's rule knowledge at any point in time (slice) is influenced by her rule knowledge at previous time points (slices), by the tutor's help (if any), and by the way that the student uses tutorial help (student help style). Students who know a rule are assumed not to forget it over the course of a tutoring session, consistent with the assumption of model-tracing tutors (Corbett et al., 2000). It was also assumed that unknown rules would remain unknown without the tutor's help (students had no access to other instruction during tutoring).

Therefore, it was necessary only to learn probabilities that the various tutorial help types would help a student learn an unknown rule. Evidence that a student had learned a rule due to a particular tutorial help type required three components: (1) the student did not know the rule before the help event, (2) the student's action after receiving the help was completing the associated problem step successfully, and (3) the student's next action involving the same rule, either while interacting with the tutor or on the posttest, was successful without tutorial help. For component (1), the method described in section 4.4.3.3 was used to estimate whether the student knew a rule before the help event. Component (2) was readily assessed as a correct student problem-solving action. Component (3) was likewise readily assessed as a correct student problem-solving action without tutorial help. Component (3) was used to distinguish between a lucky guess on one step (in which case the student's next problem-solving action involving the same rule was not as likely to be successful) and learning a rule so that it could be applied consistently across steps.

One difficulty was that component (3) required that the student's next problem-solving action be successful *without help* and the Random Tutor randomly provided proactive help on about 50% of its opportunities to do so (whether or not the student needed it). This meant that about 50% of the help events that satisfied components (1) and (2) were ineligible to satisfy component (3) even if the student could have succeeded without proactive help. This would have led to over-sampling failures for component (3) if no correction were made. Instead, the percentage of students who did not receive proactive help and satisfied component (3) was used to estimate the percentage of students who satisfied components (1) and (2) that would have satisfied component (3) if they had not received proactive help. This estimated percentage was used to extrapolate the number of students who would have satisfied all three components. This extrapolated number of students who learned the rule with tutorial help was divided by the total number of students who satisfied component (1) to estimate the probability that the tutorial help would help a student learn the rule.

One other adjustment was required for the *find equation form* rules: Because these rules were usually already known with high probability (e.g., Table 4.2 shows prior probabilities of .87 and .90), there were not enough samples for when the student do not already know the rule (i.e., when component (1) was not satisfied). Instead, conditional probabilities for these rules were estimated using the same calculation as above for all rule types combined.

The methods above were used to determine conditional probabilities for help-neutral students. For help-abusing students, the estimates of components (1) and (3), whether the student knew the rule before and after tutorial help, proved to be too unreliable. Instead, it was noted that the average number of rules gained (based on pretest to posttest performance) of help-abusing students, 1.86, was about 1/3 the average gains of help-neutral students, 5.48, meaning that help-abusing students were about 1/3 as likely to learn a rule as help-neutral students. Therefore, conditional probabilities that help-abusing students would learn a rule were set to 1/3 the corresponding probabilities that help-neutral students would learn a rule.

Conditional probabilities thus derived about whether students will learn a rule are listed in Table 4.5 by rule type and student help type. The probabilities listed are for p(*known*). P(*unknown*) is simply 1 – p(*known*). The results listed in the table are somewhat surprising: First, the *prompt*, *hint* and *teach* help types were all of about the same effectiveness with none of them dominant for all rule types. Second, the *do* action was most effective at getting students to learn a rule even though it was designed to tell the student only what to enter for the current step and not to tell the student anything about the associated rule.

Table 4.5: P(rule known) by rule type, help type and student help style

Rule type: Help type	Student help style: Help-neutral	Student help style: Help-abusing
Select equation form:		
prompt	.13	.05
hint	.18	.06
teach	.21	.07
do	.41	.14
Apply operator:		
prompt	.35	.12
hint	.33	.11
teach	.32	.11
do	.49	.16
Find equation form:		
prompt	.22	.07
hint	.26	.09
teach	.26	.09
do	.40	.13
Select operator:		
prompt	.20	.07
hint	.31	.10
teach	.31	.10
do	.33	.11

4.4.3.6 Estimating effects of help on student step knowledge when rule unknown

Probabilities for the effects of tutorial help on student step knowledge were learned for use as conditional probability table entries for the *Student Knowledge*$_1$ step nodes. It is assumed that students do not forget known steps because the result of every completed step remains displayed in the Calculus Tutor's interface. It is also assumed that the results of problem steps remain unknown until the student attempts them. Therefore, what is left to learn are the probabilities that the various tutorial help types (*prompt*,

hint, *teach*, *do*) will help the student to know a step result conditioned on (1) the student's knowledge of the related rule, and (2) the student's help style (the way the student uses the tutor's help). This section describes the calculation of conditional probabilities for cases where the student does *not* know or learn the related rule, or p(guess | help type). The calculation for when the student does know or learn the related rule is described in the next section.

For help-neutral students, evidence that a tutorial action helped a student know a problem step *without* learning the related rule is the same as evidence that a student learned the related rule (see section 4.4.3.5) except for component (3). The three required components are: (1) the student did not know the rule before the help event, (2) the student's action after receiving the help was completing the associated problem step successfully, and (3) the student's next action involving the same rule, either while interacting with the tutor or on the posttest, did not receive tutorial help and was *not* successful. For component (1), the method described in section 4.4.3.3 was used to estimate whether the student knew a rule before the help event. Component (2) was again readily assessed as a correct student problem-solving action. Component (3) was likewise readily assessed as an *incorrect* student problem-solving action without tutorial help. An incorrect problem-solving action for component (3) was evidence that the student did not really know the rule even though she got the step for component (2) right.

Just as in section 4.4.3.5, the difficulty in assessing component (3) was that the Random Tutor randomly provided proactive help on about 50% of its opportunities to do so, and when it did, it was impossible to assess whether the student would have completed the related step successfully without tutorial help. To resolve this difficulty, a similar extrapolation scheme was used based on the students who did not receive tutorial help on the next step.

Again as described in section 4.4.3.5, there were not enough help events for *find equation form* rules when the rule was not already known (because students usually knew the *find equation form* rules). Just as before, conditional probabilities for these rules were estimated using the calculation above for all rule types combined.

For help-abusing students, the estimates of components (1) and (3), whether the student knew the rule before and tutorial help, proved to be too unreliable. One of the defining characteristics of help-abusers is that they rely on tutorial help rather than their knowledge to progress through problems. Instead, for help abusers the probability that tutorial help would help the student to know the result of a step was computed without regard as to whether the student knew the related rule or not – i.e., the conditional probabilities are the same whether the rule is (estimated to be) *known* or *unknown*.

The conditional probabilities thus computed are listed in Table 4.6, which lists only p(*known*) for the step. Once again, p(*unknown*) is 1 – p(*known*). Table 4.6 also includes probabilities for when the related rule is known, for which the calculation is described in the next section. As the table shows, when

the rule is unknown, *teach* seems to be more effective than *prompt* and *hint* for step knowledge (these 3 help types were of about the same effectiveness for rule knowledge, as discussed in section 4.4.3.5). For help abusers the probability that help type *do* will help the student know the step is quite a bit higher than the probabilities for the other help types, which is consistent with help abuse behavior patterns.

Table 4.6: P(step known) by rule type, help type, rule known, and student help style

	Rule known		Rule unknown	
Rule type: **Help type**	**Help-neutral**	**Help-abusing**	**Help-neutral**	**Help-abusing**
Select equation form:				
prompt	.62	.09	.10	.09
hint	.58	.20	.06	.20
teach	.70	.21	.18	.21
do	.91	.74	.39	.74
Apply operator:				
prompt	.77	.20	.05	.20
hint	.78	.23	.06	.23
teach	.91	.35	.19	.35
do	.95	.87	.24	.87
Find equation form:				
prompt	.99	.67	.11	.67
hint	.99	.59	.11	.59
teach	.99	.56	.26	.56
do	.99	.94	.35	.94
Select operator:				
prompt	.56	.14	.18	.14
hint	.59	.24	.21	.24
teach	.80	.52	.42	.52
do	.71	.72	.33	.72

4.4.3.7 Estimating effects of help on student step knowledge when rule known

For help abusing students, the probability that the student will know a step with tutorial help is estimated to be the same regardless of whether the student knows the related rule, as discussed in section 4.4.3.6. These values are listed in the *Help-abusing* columns of Table 4.6.

For help-neutral students, the student's rule knowledge was considered when calculating probabilities. When the student knows the rule required to complete a step successfully but the tutor gives her help anyway, there are two ways that she can know the step result: (1) by applying the rule, and (2) by guessing the correct answer based on the tutor's help. The probability of applying the rule correctly is 1 – p(slip), where p(slip) is the probability that a student will not know the correct result for a problem step when she knows the related rule (defined in section 4.4.3.4 with values for each rule listed in Table 4.4). The probability of guessing the correct answer based on the tutor's help is estimated to be the same as the probability that the student will know the step based on the tutor's help without knowing the corresponding rule. This is p(guess | help type) as calculated in the previous section (0) and listed in Table 4.6 in the column *Rule unknown, Help-neutral*. Therefore, for help-neutral students, the probability that a student will know a step when she knows the rule and the tutor helps her anyway is 1 – p(slip) + p(guess | help type).

The calculated probability values are listed in Table 4.6 in the column *Rule known, Help-neutral*. As the table shows, help-neutral students are much more likely to know a step result when they know the related rule. The likelihood that help-neutral students will know a step when they know the related rule seems to be correlated with the rule's difficulty as discussed in section 4.4.2 and shown in Table 4.3. Help type *teach* again seems to be more effective than *prompt* and *hint*. For the *find equation form* rules when the rule is known, the step is known with almost certainty regardless of the tutor's help type because these rules are relatively easy.

4.4.3.8 Estimating conditional probabilities for Student Action Topic

As described in section 3.2.4, the student action topic and the student action type together form the representation of student actions, *Student Action Topic*$_s$ and *Student Action Type*$_s$ in TACN slice 0 and slice 2. *Student Action Topic*$_s$ is either one of the problem steps (when the student is working on a specific problem step) or *null*. The value *null*, in combination with *Student Action Type*$_s$, represents either the student clicking on the general *Help!* button (*Student Action Type*$_s$ = *impasse*) when she is not working on a step, or clicking the *Cancel* button (*Student Action Type*$_s$ = *null*) when she is working on a step, possibly with help provided. At any point in time, only one or the other of the *Help!* and *Cancel*

buttons is available to the student, depending on whether she is working on a step. To create the conditional probability table to predict *Student Action Topic*$_1$, first the probability that the student will click *Help!* or *Cancel* (depending on the situation) is calculated. The remaining probability, which is the bulk of it, is divided up amongst the currently possible problem steps (depending on the status of the problem solution) according to how *in_focus* they are in the *Student Focus*$_1$ subnetwork (see section 3.2.5).

At times when a student can click the *Help!* button, her choices are between clicking *Help!* and clicking on a step in the Goals Window to start working on it. So a training set estimate of the probability that a student will click *Help!* is the number of *Help!* clicks divided by the total number of *Help!* clicks and step clicks together. Separate probabilities were calculated conditioned on student help style (*neutral* or *abuse*) because it was thought that help-abusing students would be more likely to click *Help!* (indeed they were). The calculated probabilities are shown in Table 4.7 in the p(*Help!*) column.

The *Cancel* button is only available when a student is working on a problem step. As described in section 4.4.1, some help abusers repeatedly started working on a problem step and then clicked *Cancel* until they received proactive help, with the effect that students were more likely to click *Cancel* when help was not provided. Therefore, the probability that a student would click *Cancel* was conditioned on both the student's help style (*neutral* or *abuse*) and whether help was currently being provided. The calculated probabilities are shown in the p(*Cancel*) columns of Table 4.7.

As the table shows, help-abusing students were more likely than help-neutral students to click *Help!*. Surprisingly, help-neutral students were more likely to click *Cancel*, although the probabilities for both help-neutral and help-abusing students were small and so could have been heavily influenced by just a few unusual samples. For both help styles, students much more likely to click *Cancel* when no help was provided.

Table 4.7: P(Cancel) and p(Help!) for Student Action Topic

	p(*Cancel*)		
Student Help Style	**Help Provided**	**No Help Provided**	**p(*Help!*)**
neutral	.008	.04	.01
abuse	.003	.03	.07

4.4.3.9 Estimating conditional probabilities for student action type

If the student does not know the result of a problem step (knowledge of problem steps is modeled in the *Student Knowledge*$_1$ nodes), then *Student Action Type*$_1$ will be either a help request or an error. As discussed in section 4.4.1, help-abusing students often request more help than their help-neutral peers, and sometimes make more errors. Therefore, conditional probabilities that students will make a help request or an error are conditioned on the student's help style. The probability of a help request is simply the number of help requests divided by the number of help requests and errors combined. The probability of an error is calculated just the same except with errors in the numerator. Results are shown in Table 4.8. As the table shows, help-neutral students were significantly more likely to make an error than to request help. Help-abusing students were almost as likely to request help as to make an error, and they were significantly more likely than help-neutral students to request help.

Table 4.8: P(Help Request) and p(Error) for Student Action Type

Student Help Style	p(Help Request)	p(Error)
neutral	.13	.87
abuse	.44	.56

4.5 TUNING UTILITIES

With DT Tutor's (DT's) probabilities specified, it was time to specify the remainder of DT's numerical parameters – its utilities. This is relatively uncharted territory for intelligent tutoring systems. Most probabilistic ITSs use neither decision theory nor utilities, and only two other decision-theoretic intelligent tutoring systems have been implemented. One, CAPIT (Mayo & Mitrovic, 2001), considers only a single attribute at a time and so any utility function will do (CAPIT uses a utility of 1.0 for desired states and a utility of 0.0 for other states). The only other decision-theoretic tutoring system, iTutor (Pek, 2003), does consider 3 tutorial state attributes. iTutor, like DT, uses a linearly-additive multiattribute utility function (discussed in section 4.5.2 below) but it is unclear how iTutor obtains the weights for its linear function.

An important consideration was that there was no empirically verifiable way to determine DT's utilities. In contrast to a human tutor or perhaps a more advanced intelligent agent, DT currently has no internal state (i.e., preferences) to which to adapt its utility function. Indeed, DT's utility function, once specified, becomes its preferences (rather than just a mathematical representation of some other internal preference structure). The only applicable criteria for which to adapt DT's utility function are external to DT, such as (1) maximizing effectiveness with students or (2) maximizing human judges' ratings of the tutor. Even these latter criteria do not constrain DT's utilities to empirically verifiable values. In the first case (1), effectiveness with students can be measured, for example, (1a) for any combination of various attributes, such as knowledge gain, affective state (e.g., motivation, feeling of independence), and problem-solving progress, as measured by various instruments; (1b) for any student population; and (1c) over any period of time (e.g., a tutor that is more motivationally effective but less cognitively effective in the short term may prove to be more cognitively effective in the long term by increasing student interest and persistence). In the second case (2), human judges' ratings of the tutor can likewise be for any combination of attributes, for any populations of judges and students, over any period of time, etc. At any rate, attempting to maximize effectiveness with students was not an option for the current study because the actual student-tutor interactions were between students and the Random Tutor. Attempting to maximize judges' ratings was not an option because tuning was permitted only on the training set (the judges rated only the test set) in order to preserve the integrity of the assessment phase. Instead, minor tuning was performed until DT seemed to perform reasonably on several representative scenarios according to the experimenter's subjective preferences.

4.5.1 Utilities for each tutorial state attribute

DT uses a multiattribute utility function to compute the utility for each combination of tutorial state attributes that could be an outcome of the tutor's action. The multiattribute function, which is described in section 4.5.2, combines into an overall utility the individual utilities for each of the attributes that DT considers. This section describes each of the individual utilities as background for an understanding of tuning DT's multiattribute utility function. Only the last of these, tutor response preferences, was tuned, as described below.

4.5.1.1 Discourse coherence

The *Discourse State*$_1$ node models a tutorial discourse action (i.e., response) as *coherent* if *Tutor Action Type*$_1$ abides by the following constraints:

1. **When help is provided:**
   a. The tutor must respond to any student help request with a non-*null* response type.

2. **<u>Weak</u> *successive explicitness constraint*:**

   a. The available help messages are ranked in order of the explicitness of the help that they provide, from least to most explicit. For this study, the sequence is *prompt*, *hint*, *teach*, *do*.

   b. When help is provided (i.e., for non-*null* responses), it must be more explicit than the help previously provided. If the bottom-out help message (*do* for the Calculus Tutor) has already been provided, it may be repeated.

Constraint (1), when help is provided, was implemented to supply a cooperative and understandable interface (e.g., if the tutor didn't respond to a help request at all, the student might think the tutor had crashed). The Fixed-Policy Tutor, with which DT is compared during the assessment phase, follows the same constraint for when help is provided plus a couple more, as described in section 5.2.2.3. Constraint (2) is a weaker version of a similar constraint followed by the Fixed-Policy Tutor, which requires that the tutor's response be <u>minimally</u> more explicit than any previous help message provided.

For tutorial actions that satisfy these constraints, the utility when *Discourse State*$_1$ is coherent, or U(*Discourse State*$_1$=*coherent*), is 1.0. Otherwise, U(*Discourse State*$_1$=*incoherent*) = 0.0.

4.5.1.2 Discourse relevance

The *Discourse Relevance*$_1$ node, described in section 3.2.7, measures how relevant the *topic* of the tutor's response is to topic(s) in the student's focus of attention (e.g., a problem step), as modeled by the *Student Focus*$_s$ subnetworks (described in section 3.2.5). A simple example is that if the student requests help while she is working on a particular problem step, help related to that topic step is relevant to the student's focus of attention. When more than one topic may be relevant, the *Student Focus*$_s$ subnetworks model the relative strength of each topic's relevance. For discourse relevance, U(*Discourse Relevance*$_1$=*relevant*) = 1.0 and U(*Discourse Relevance*$_1$=*irrelevant*) = 0.0.

4.5.1.3 Student rule knowledge

The *Student Knowledge*$_s$ rule nodes, described in section 3.2.6, model the student's knowledge of each domain rule that is applicable to the current calculus related-rates problem. For each rule *rule*$_r$, U(*Student Knowledge*$_2$ *rule*$_r$=*known*) = 1.0 and U(*Student Knowledge*$_2$ *rule*$_r$=*unknown*) = 0.0. Since multiple rules are applicable to each problem, DT sums their utilities[7] to come up with an overall utility for the student's rule knowledge.

4.5.1.4 Student problem-solving progress

The *Student Focus*$_s$ step nodes, described in section 3.2.5, model the student's progress in completing each step in addition to the student's focus of attention. For each step *step*$_s$, U(*Student Focus*$_2$ *step*$_s$ = *complete*) = 1.0 and U(*Student Focus*$_2$ *step*$_s \neq$ *complete*) = 0.0. Since multiple rules are applicable to each problem, DT sums their utilities (see footnote 7) to come with an overall utility for the student's problem-solving progress.

4.5.1.5 Student help style

The *Student Help Style*$_s$ nodes model the student's help style (discussed in section 4.4.1) as either *neutral* or *abuse*. U(*Student Help Style*$_2$=*neutral*) = 1.0 and U(*Student Help Style*$_2$=*abuse*) = 0.0.

4.5.1.6 Student independence

The *Student Independence*$_s$ nodes, described in section 3.2.8, model the student's feeling of independence in terms of 5 levels, level 0 through level 4. Utilities for level 0 through 4 are 0, 0.25, 0.5, 0.75 and 1.0, respectively.

[7] Utilities for student rule knowledge and problem-solving progress are simply summed, but technically the mechanism for doing this is a linearly-additive multiattribute function with all weights equal to 1.0. Such functions are described in more detail in section 0.

4.5.1.7 Tutor response preferences

The utility for tutor response preference is based on *Tutor Action Type*$_1$, which can have value *null*, *prompt*, *hint*, *teach*, or *do*. This utility was originally intended to model differences observed between human tutors in previous studies (e.g., VanLehn et al., 2003) where, with the same student population and subject domain, some tutors tended to be less verbal and more reticent with help while others tended to be more verbal and proactive in leading students through the problem space. Less verbal tutors could be modeled as preferring response types like *null* and *prompt* while more verbal tutors could be modeled as preferring longer, more explicit response types like *teach*.

More generally, utility for tutor response type can be used to at least coarsely model preferences among tutorial actions that are not based on tutorial state attributes that DT models. For instance, the apparent tutor response type preferences discussed above might have been based on differences in the human tutors' beliefs about how best to tutor. If so, and if DT modeled those beliefs, then DT could model the differences in the tutors' response preferences based on differences in their beliefs. Since DT doesn't model the tutors' beliefs about how best to tutor, it can instead model just the *results* of differences in their beliefs as differences in their response preferences. This method is coarser, since it may miss changes to response preferences that result from changes in the tutors' beliefs, but it may provide results that are good enough since modeling of human preferences is necessarily approximate anyway.

The experimenter used the same method to begin to tune DT's behavior so that its responses corresponded more to the experimenter's beliefs about how best to tutor. A problem with DT's responses observed immediately after learning key probabilities was that it selected the *do* response too often. This was due to a surprise in learning conditional probabilities, discussed in sections 4.4.3.5 and 6.1.4, that the *do* response often was most effective not only at getting the student to learn the step that was the topic of the tutor's response, but also at getting the student to learn the related rule. A consequence was that DT began to select *do* more often even though *do* had the most negative influence on the student's feeling of independence. The student's independence changed rather slowly in DT's model, too slowly to counterbalance response type *do*'s more positive influence on both the student's task progress and rule knowledge. For future work, DT's model of the student's affective state needs to be improved (see section 6.6.2). In the meantime, tutor response type preferences were changed to strongly disfavor response type *do*. In addition, the experimenter found that the *null* response type was being selected a little too often and that the *teach* response was not being selected quite often enough (during training), so tutor response type preferences were changed to slightly disfavor *null* and slightly favor *teach*. Table 4.9 provides the complete list of utilities for response type preferences.

Table 4.9: Utilities for tutor response preference

Response Type	Utility
null	6
prompt	7
hint	7
teach	8
do	1

4.5.2 Multiattribute utility function

To model the utility of combinations of multiple attributes (e.g. student knowledge, student affect, problem-solving progress, etc.), DT currently uses a linearly-additive multiattribute utility function, which is the only type of multiattribute utility function currently supported by DT's underlying probabilistic inference engine, SMILE[8]. Use of this type of utility function requires additive independence among preferences for multiple attributes: In brief, changes in lotteries for one attribute do not affect preferences for lotteries in other attributes. In other words, there must be no interaction in preferences among the attributes (Clemen, 1996). von Winterfeldt and Edwards (1986) report that additive independence usually does not hold for human preferences, and indeed it does not for the experimenter's preferences: For instance, comparing Lottery A and Lottery B below in regard to student outcomes, the experimenter would prefer Lottery A in order to avoid risking the situation where the student's outcome is to have both low knowledge and low affect.

A (low knowledge, high affect) with probability 0.5
(high knowledge, low affect) with probability 0.5

B (low knowledge, low affect) with probability 0.5
(high knowledge, high affect) with probability 0.5

However, Clemen (1996, p. 585) retains the perspective that the objective is to create a reasonable representation – an approximate model – of the decision maker's value structure, stating that "[i]f minimal interactions exist among the attributes, then the additive utility function is appropriate."

[8] The core of the decision-theoretic portion of DT's implementation is based on the SMILE reasoning engine for graphical probabilistic models contributed to the community by the Decision Systems Laboratory, University of Pittsburgh (http://www.sis.pitt.edu/~dsll).

With this perspective, and given the available options, the experimenter's utility function was modeled as a linearly-additive multiattribute utility function,

$$U(x_1,\ldots,x_m) = \sum_1^m w_i U_i(x_i)$$

where $U_i(x_i)$ is the utility of attribute x_i and w_i is the weight allotted to attribute x_i in the linearly-additive function. The tutorial state attributes (x_i) for which utility was modeled, along with their weights (w_i), are listed in Table 4.10.

Table 4.10: Weights for linearly-additive multiattribute utility function

Attribute	**Weight**
***First priority*:**	
Discourse Coherence	100,000
***Second priority*:**	
Discourse Relevance	10,000
***Third priority*:**	
Student Knowledge	400
Student Independence	200
Student Help Style	200
Problem-Solving Progress	40
Tutor Response Preference	10

At first, the differences in the magnitudes of the weights in Table 4.10 may seem shocking, but there is a simple explanation. The main reason is that a system of priorities amongst the utilities was implemented. This could have been accomplished with smaller differences between the weights[9], but the orders of magnitude approach shown makes it easy to discern at a glance when higher priority utilities are not at their maximum value.

The first priority was discourse coherence so that DT would always respond appropriately as described in section 4.5.1.1. The second priority was discourse relevance, described in section 4.5.1.2, so that DT would always respond with a topic that was relevant to the student's focus of attention.

[9] Utility rankings are unchanged by a linear transformation of multiplying by a non-negative number and adding a constant.

Assigning these two attributes the highest order of magnitude weights guaranteed that DT's response would maximize utilities for these attributes if at all possible, virtually ensuring that DT would respond appropriately within the discourse context and with a relevant response topic.

The third priority was to enable DT, like human tutors, to decide the tutor's response *type* while maintaining a delicate balance (Merrill et al., 1992) among the remaining considerations of student knowledge, student independence, student help style, problem-solving progress, and tutor response preferences. As shown in the bottom section of Table 4.10, different weights are assigned to different third-priority attributes, making it appear that there are still more priorities at work, but this is mostly an illusion, as described below.

The first goal in assigning weights to the third-priority attributes was to adjust the ranges of the utilities for the different attributes to be about the same. For example, at one point during tuning, the weighted expected utility for Student Knowledge (which had the highest weight of 400) varied from 2084 for the *null* response to 2146 for the *do* response[10], for a range of 62. At the same time, the weighted expected utility for Tutor Response Preference (which had the lowest weight of 10) ranged from 10 for the *do* response to 80 for the *teach* response, for a range of 70. It is the *changes* in utilities for different decision alternatives rather than the absolute values of the utilities that influence which alternative the tutor selects. With similar ranges for the different attributes' utilities, the pros and cons of the tutor's decision alternatives, reflected in the weighted expected utilities for different attributes, can be traded off in the desired delicate balance without the expected utility for one attribute overwhelming the others. The reason that different attributes require different multipliers to have approximately the same range of weighted expected utilities is that the tutor's actions have more influence on the probabilities of some attributes than others, as reflected in their conditional probability table entries. For the example above, the *un*weighted utilities for Student Knowledge varied from 5.21 (for *null*) to 5.365 (for *do*), a range of 0.155. At the same time, the unweighted utilities for Tutor Response Preference (which is deterministic) varied from 1 to 8, a range of 7, which is about 45 times greater than the range of 0.155. But by weighting Student Knowledge 40 times greater than Tutor Response Preference, 400 to 10, the range for Student Knowledge was brought up to be just about equivalent to the range for Tutor Response Preference. Since Student Knowledge was at its highest for response type *do* when Tutor Response Preference was at its lowest, the two attributes were approximately counterbalanced.

With multiattribute utility weights thus assigned so that the various attributes would counterbalance each other appropriately, a few small adjustments were made to the weights while the

[10] Weighted expected utilities for Student Knowledge and Problem-Solving Progress can be much greater than their multiattribute utility weights because the utilities were summed for multiple rules and multiple steps, respectively. Utilities for rule knowledge were usually 4 digits – i.e., greater than 1,000 but less than 10,000 – which is why the next higher magnitude weight (for Discourse Relevance) was 5 digits (10,000).

experimenter simulated different response patterns for a student until DT seemed to behave appropriately in a few representative situations. For instance, the experimenter verified that the priority system described above worked as designed, that the tutor did not intervene with proactive help at every opportunity, and that the tutor tended to provide less explicit help (e.g., *null* or *prompt*) when the student was less likely to need help and more explicit help (e.g., *teach*) when the student was more likely to need it. A great deal more attention could have been paid to this phase, with hopefully even better results, if not for time limitations for completing the study.

5.0 EVALUATION: ASSESSMENT PHASE

This chapter describes work on the primary focus of this study, a comparative assessment of the tutorial action selection capabilities of DT Tutor, the Fixed-Policy Tutor and the Random Tutor.

5.1 GOALS OF THE ASSESSMENT PHASE

The primary purpose of the assessment phase was to compare the judges' ratings of DT Tutor's tutorial action selections with their ratings of the action selections of the Random Tutor and the Fixed-Policy Tutor. This comparison followed a traditional hypothesis testing approach for the following hypotheses:

> **Hypothesis 1:** According to ratings by skilled human tutors, tutorial action selections by decision-theoretic methods can be better than selections made randomly among relevant tutorial actions.

> **Hypothesis 2:** According to ratings by skilled human tutors, tutorial action selections by decision-theoretic methods can be better than selections made by a fixed policy that emulates the fixed policies of theory-based, widely accepted and highly effective computer tutors.

A secondary purpose was to learn details about what skilled tutors think about the best tutorial responses to provide for various types of tutorial situations, and also what they think about the pool of responses currently available from the student interface. This information, combined with empirical data about DT Tutor's patterns of responses in different situations, might be used to improve both the student interface and DT Tutor's action selections. Such information can even be used to inform a fixed policy (or other

methods of selecting tutorial actions) for improved tutorial action selections (at least according to the judges' ratings), as indeed it was as described in section 5.7.

5.2 DESIGN OF THE ASSESSMENT PHASE EXPERIMENT

5.2.1 Subjects

Three judges were recruited from the population of graduate students in mathematics at the University of Pittsburgh. All had extensive experience tutoring calculus as well as other mathematics to college and high school students. The first three applicants who met the requirements were accepted. They consisted of two males and one female, ranging in age from early-20s to mid-30s. One was from the USA and two were from other countries (specifics not provided here to protect their anonymity). They were paid $10/hour for training (described below) and thereafter paid $1/scenario (tutoring situation) that they evaluated for a total of 350 scenarios. These were considered skilled tutors, although not necessarily expert tutors, because of their extensive mathematical knowledge and experience teaching calculus.

5.2.2 Materials

5.2.2.1 Printed materials

The printed materials included all of the materials used by the students in the data collection and tuning phase of the experiment (see section 4.2.1.1) because, before providing the assessments, the judges first went through the same procedure that the students went through in order to develop the judges' intuitions for the scenarios that they would be rating. The remaining printed materials were used only while the judges performed scenario assessments. They consisted of the following:

1. Screen shots of the Calculus Tutor interface for all 5 problems that students worked on (see Appendix D). The screen shots were of the interface after all problem steps had been successfully completed. The Accepted Equations Window listed all of the equations in a successful solution. The Goals Window showed all problem goals expanded to show the structure of the completed problem solution along with every correct problem step entry. The Goals Window had goal numbers added in brackets

to the right of each goal to correspond to the goal numbers listed in the scenarios that the judges evaluated (described in item 3 below).

2. Screen shots of the Calculus Tutor interface for all Dialog Windows that are displayed (see Appendix E). These screen shots show the Dialog Window displayed for every type of step. They were provided to the evaluators so that the *Relevant Action Histories* (see item 3.f below) in the scenarios that the judges evaluated could just refer to the name of the Dialog Window instead of showing a screen shot every time.

3. Scenarios descriptions depicting tutorial situations for judges' ratings and comments (see Appendix F for a sample). 350 scenario descriptions were created. Details about scenarios selection are described in section 5.2.2.2 below. Here the elements of each scenario description are described in order:

    a. Screen shot showing the Calculus Tutor interface at the current moment.
    b. Description specific to the scenario type (see section 5.2.2.2 below for a description of scenario types):
        - Step-specific help request: The problem, goal number, correct entry, and Goals Window prompt for the student's next action, the help request.
        - General Help Request: The same information as for a step-specific help request, listed for all possible next steps (at most 2). By the way, only 5 of 350 scenarios turned out to be general help requests, so they played an insignificant role in the analyses.
        - Error: The problem, goal number, correct entry, and Goals Window prompt for the student's last action, the error.
        - Step start: The problem, goal number, correct entry, and Goals Window prompt for the student's next action, clicking on a prompt in the Goals Window.
    c. General student history: Number of correct entries, number of errors, and number of help requests.
    d. Whether this student was discouraged from requesting help.
    e. The student's performance on the pretest problem for the rule related to this problem step (*correct* or *error*).
    f. *Relevant Action History*: This step lists previous student-tutor interactions on (1) any previous steps that use the same rule that is related to the current step, and (2) the current step.

g. A page for tutorial response ratings for each of the 5 possible tutorial response types to the current situation. These are listed in random order with just the text for the corresponding help message (without the response type name of *null*, *prompt*, *hint*, *teach*, or *do*). For the *null* response type, the text displayed is "(<window name> dialog window with no message)," where <window name> is either "Blank" or one of the Dialog Windows listed in Appendix E. Judges were asked to rate each response (whether or not any of the tutors provided it) on a scale of 1 (worst) to 5 (best), and told that the ratings were not comparisons – e.g., they could give more than one response the same rating.

h. Space for any comments about the listed tutorial responses.

i. Space asking for the best single tutorial response, if any, to display for the current scenario. This could be one of the listed tutorial responses, a response that the judge creates, or "none."

j. The responses to General Help! Request scenarios were usually situations where the tutor had a choice about which problem step to provide help for. Consequently, these scenarios had a few more elements:

- A history of the most recent student-tutor interaction, provided to give the judge a feel for the student's current focus of attention.
- Relevant action histories for the rules related to each of the steps the tutor could provide help for.
- A question about which of the currently possible steps the tutor should provide help for, including a choice of "The tutor should not respond."
- Separate pages for tutorial response ratings and comments for the help messages related to each of the possible next steps.

5.2.2.2 Scenario types and stratified sampling

Scenarios are classified into three types according to the type of tutorial situation they represent:

1. Help requests: These are scenarios where the student has requested help. Help can be either step-specific or general. Step-specific help occurs when the student presses the *Help* button within a Dialog Window while she is working on a step – for these help requests, the student's focus of attention at the time she requests help is assumed to the step she is working on. General help occurs when the student presses the *Help!* button at the lower left side of the interface. This button is only available when the student is not working on a specific step, so the tutor must decide which step to provide help for. Responses to help requests are reactive help.

2. Errors: These are scenarios where the student has just made an error. Help responses to errors are proactive help (sometimes called unsolicited help).

3. Step starts: These are scenarios where the student has just clicked on one of the goals with an open prompt in the Goals Window to begin working on the corresponding step. Help responses to step starts are proactive help.

For assessing the performance of the different tutorial action selection methods, 350 scenarios were to be selected from the test set. The intention was to select the scenarios randomly so as not to introduce any bias. However, a completely random selection would have produced a highly skewed sampling among the scenario types listed above. Of 5009 scenarios in the test set, 2837 were step starts, 1754 were errors, and only 418 were help requests, so a random sampling would have collected about 57% or 200 step start scenarios, 35% or 122 errors, and 8% or 28 help requests. This is just the opposite of what was desired for assessing help selection, for which the help provided for help requests is arguably most important, the help provided for errors is probably next most important, and it is debatable whether help should be provided for step starts at all. With a completely random distribution, the judges' ratings of the tutors would be dominated by their ratings for step start scenarios and only weakly influenced by their ratings for help request scenarios. Therefore, a stratified sample was selected with the sample for each stratum randomly selected from among all the scenarios in that stratum: 175 help requests, 100 errors and 75 step starts.

One additional criteria was employed for selecting scenarios: Since the Random Tutor selected actions randomly, it might for a specific step select, say, *do* help (the most explicit help message), and then if the student was not successful, follow the *do* help with a *prompt* (the least explicit non-*null* help message). Such sequences of help messages violated both the weak *successive explicitness constraint* followed by DT Tutor (see section 4.5.1.1) and the strong successive explicitness constraint followed by the Fixed-Policy Tutor (see the next section, 5.2.2.3), which prevent those tutors from providing less explicit help messages after more explicit help messages for the same step. It was unclear just how DT Tutor or the Fixed-Policy Tutor should respond following sequences of help messages that violated even the weaker successive explicitness constraint, since providing successively more explicit messages would no longer be possible. A related concern was that it was unclear just how such seemingly odd (indeed, random) sequences of help messages would affect the judges' intuitions about what kind of help to provide next. Therefore, scenarios whose *Relevant Action History* (described in section 5.2.2.1, item 3.f above) included sequences of tutorial actions that violated the weaker successive explicitness constraint were excluded from the sample.

5.2.2.3 The Fixed-Policy Tutor

A Fixed-Policy Tutor was designed as a stiff test for comparison with DT Tutor (see Research Hypothesis 2 in section 1.1.2). The Fixed-Policy Tutor used the same student interface as DT Tutor, described in section 5.2.2.3, with the same pool of help messages so that only the method used for selecting tutorial actions was different. The Fixed-Policy Tutor's policy for providing help messages emulates the tutorial action selection policy of the Cognitive Tutors (Anderson et al., 1995), which are theory-based (e.g., Anderson & Lebiere, 1998), widely used and highly effective (e.g., Koedinger et al., 1997). The policy consists of the following constraints:

1. **When help is provided:**
    a. The tutor must respond to any student help request with a non-*null* response type.
    b. Proactive help is not provided after the student's first error on a step, but it is provided after subsequent errors (Aleven & Koedinger, 2000). (Other fixed-policy tutors, including Cognitive Tutors, may not ever provide proactive help after errors, or may provide proactive help after a different number of errors. The policy for earlier Cognitive Tutors was to "never volunteer help" (Anderson et al., 1995, p. 199), and Aleven and Koedinger's (2000, p. 198) paper states that "… we have modified the PACT Geometry tutor so that it initiates help after two errors.")
    c. Proactive help is never provided for step start scenarios.
2. **<u>Strong</u> *successive explicitness constraint*:**
    a. The available help messages are ranked in order of the explicitness of the help that they provide, from least to most explicit. For this study, the sequence is *prompt*, *hint*, *teach*, *do*. (Other fixed-policy tutors, including Cognitive Tutors, may have a sequence of different length, and there may be more than one message per level of explicitness. The point is that there are successive levels of explicitness (Anderson et al., 1995).)
    b. When help is provided (i.e., for non-*null* responses), it must be <u>minimally</u> more explicit than the help previously provided. If the bottom-out help message (*do* for the Calculus Tutor) has already been provided, it may be repeated.

DT Tutor's discourse coherence model (see section 4.5.1.1) abides by the same constraints as the Fixed-Policy Tutor's except in two respects: First, DT Tutor does not follow constraints (1b) and (1c). Second, DT Tutor's constraint (2b) does not include the term "minimally."

It is important to note all of the tutors – DT Tutor, the Random Tutor, and the Fixed-Policy Tutors – employ the student interface's capability to automatically provide immediate "flag" feedback after errors: the student's entry is displayed in red to indicate that it is an error (correct actions get immediate flag feedback in green). The Cognitive Tutors typically provide immediate flag feedback as well (Koedinger et al., 1997). In addition, for a commonly occurring slip or misconception, Cognitive Tutors may also provide an error or "bug" message "that indicates what is wrong with the answer or suggests a better alternative" (Koedinger et al., 1997, p. 35). The Calculus Tutor's student interface has not yet implemented any bug messages.

The fixed policy above is also very similar to the fixed policy of Andes1 (Conati et al., 2002) except for a difference in responses to student errors (Gertner & VanLehn, 2000): Andes1 also provides flag feedback, but it provides a hint or an error message only for simple syntactic errors.

5.2.3 Procedure

First, a replay mechanism developed for this study was used to replay the logs of the interactions between students and the Random Tutor. While the logs were replayed, the actions that DT Tutor would have selected for the same tutorial situations were recorded. When the actions selected by the Random Tutor and DT Tutor differed, the action originally selected by the Random Tutor was provided in order to preserve the fidelity of the replay, and DT Tutor updated its model of the tutorial state to include the action actually provided by the Random Tutor. A similar process was undertaken to record the actions that the Fixed-Policy Tutor would have taken for the same situations. These replays recorded the actions that DT Tutor and the Fixed-Policy Tutor would have taken for every scenario, not just the sample of scenarios used for assessment.

Next, the judges rated all possible responses for 350 scenarios. Note that with this design it is possible to use the same judges' ratings to assess the tutorial action selections of still more tutors, or updated versions of the same tutors, as long as they select from the same pool of help messages. The judges were told that they were rating scenarios in order to provide information about what help messages would be best to provide for various situations. They had no idea which tutor provided which responses or that their ratings would be used to compare tutors.

5.3 FAST RESPONSE TIME BY LIMITING PROBLEM SOLUTIONS

Fast response time by DT Tutor (DT) was not required for the success of this study. As described in section 5.2.3, the responses of DT and the Fixed-Policy Tutor were computed offline based on logs created from interactions between the Random Tutor and real students. The fact that the Random Tutor's responses were nearly instantaneous sufficed to create pleasantly snappy interactions with students. However, DT Tutor's response time had previously been a problem, as described in (Murray & VanLehn, 2000) and noted in the literature (e.g., Conati et al., 2002), although response time has been improving (Murray et al., 2004). Therefore, significant further improvement is noteworthy.

No formal study of DT's response times was conducted, but from informal observations of DT computing its own responses to logged scenarios, it appeared that DT provided sub-second response time for almost all scenarios, if not all scenarios, of the 30 test set students each working on the same 5 problems. It did this while running concurrently with several other applications on a PC[11], including the domain expert and the student interface described in sections 3.1 and 3.3 respectively, using the SMILE[12] probabilistic reasoning engine and a clustering algorithm (Huang & Darwiche, 1996) for exact inference. An additional challenge was that the logged interactions were replayed at 15 times original speed with all delays over 10 seconds removed (i.e., instead of reducing a 10-second delay to 10/15 = 2/3 seconds, delays of 10 seconds or more were simply skipped during the replay), so the system did not have the respites it had during the original interactions while waiting for the next student input.

Probabilistic inference is NP-hard in the worst case for both exact (Cooper, 1990) and approximate (Dagum & Luby, 1993) algorithms. DT's networks possess several characteristics that can make inference challenging:

1. Multiply-connected, with some network nodes having three or more parents, for which exact inference can be NP-hard (Cooper, 1990). Multiply-connected networks seem to be necessary to represent many complex, real-world domains (Cooper, 1990).
2. Large (Cheng & Druzdzel, 2000; Russell & Norvig, 1995), also as seems to be necessary for many complex, real-world domains (e.g., Cooper, 1990).
3. Temporal (Cooper et al., 1989), increasing both the number of nodes (for multiple slices) and connectivity, with temporal as well as atemporal arcs (Ngo et al., 1997)

[11] 667-MHz Pentium III, 512-Mb RAM, running Windows 98 – the same system used for (Murray et al., 2004)
[12] The core of the decision-theoretic portion of DT's implementation is based on the SMILE reasoning engine for graphical probabilistic models contributed to the community by the Decision Systems Laboratory, University of Pittsburgh (http://www.sis.pitt.edu/~dsl).

4. Large conditional probability tables (e.g., Cheng & Druzdzel, 2000)
5. Decision networks, requiring an update for each alternative (Russell & Norvig, 1995)

While some characteristics can be addressed by specialized inference algorithms (e.g., Lin & Druzdzel, 1999; Shachter & Peot, 1992), DT's combination of characteristics pose a stiff test.

The key to DT's improved performance was a reduction in the both the number of response *topics* that it considers (i.e., values for *Tutor Action Topic*$_1$) and the number of response *types* that it considers (i.e., values for *Tutor Action Type*$_1$). Previously, DT had considered providing 7 different help *types* for every step (i.e., *topic*) in the problem. For instance, for a 15-step problem, the decision network algorithm that DT used required 15*7 = 105 network updates to compute exact expected utilities for every combination of decision alternatives (not all algorithms require computing exact expected utilities for every alternative – e.g., (Shachter & Peot, 1992)). For the current version of DT, only 5 different response types were considered for each step (*null, prompt, hint, teach, do*) instead of 7.

Reducing the number of topics that DT considers had an even bigger effect. The domain expert described in section 3.1 implemented the heuristics described in the Calculus Tutor Tutorial (Appendix A) to limit to one the number of problem solution paths for each problem. These student interface described in section 3.3 both enforced the heuristics and reified student problem-solving goals that might otherwise be invisible (Singley, 1990). The combination of limiting the number of solution paths and making all problem-solving steps (Singley, 1990) visible[13] allowed DT to limit the number of user interface problem steps that could be within the student's focus of attention to two[14]. This meant that DT considered at most 2 response topics times 5 response types, or at most 10 network updates for each decision, an order of magnitude reduction.

Another factor that has surely helped to improve DT's response time is continuing improvements to SMILE, the reasoning engine for graphical probabilistic networks that DT employs.

5.4 DISTRIBUTIONS OF RESPONSES SELECTED BY THE TUTORS

As a first step in assessing the tutors' performance on the test set of scenarios, their response selection patterns were analyzed. The following subsections describe and compare the selections of the tutors,

[13] All problem-solving steps at the grainsize considered by the Calculus Tutor's user interface were visible. It would be possible to decompose these problem steps into smaller components.

[14] The fact that at most 2 problem steps were possible in the Calculus Tutor at any one time, as opposed to 1 or 3 or more, is an artifact of the problem solver's heuristics and the domain operators. The point is that the number of possible next problem-solving steps was sharply reduced.

focusing on the Fixed-Policy Tutor (FT) and DT Tutor (DT) since the Random Tutor's (RT) selections were random and so mostly uninteresting.

The details in this section may seem excessive without an understanding of their purposes, which are as follows:

1) To provide a feel for how FT and DT behaved in various situations
2) To demonstrate that FT and DT differed from one another in their responses
3) To demonstrate that DT's responses varied by situation
4) To identify some failings in DT's response selections
5) To provide a basis for explaining differences in the judges' ratings of FT and DT

Table 5.1 shows the distribution of response types (*null*, *prompt*, *hint*, *teach*, *do*) across all scenarios for the Random Tutor (RT), the Fixed-Policy Tutor (FT) and DT Tutor (DT). A Pearson's chi-square test of association found the differences in response patterns to be significant, $\chi^2(8)=149.1$, $p<.001$.

Table 5.1: Distributions of response types for all scenarios, percentages

	null	**prompt**	**hint**	**teach**	**do**
RT	23	18	15	22	21
FT	41	23	11	14	11
DT	27	14	3	47	9

RT's responses were fairly evenly distributed across all of the response types, which was expected since RT selected response type randomly. The distribution of RT's responses is otherwise unremarkable and so won't be discussed much further.

5.4.1 The Fixed-Policy Tutor's overall distribution of response selections

FT selected more *null* responses than any other type. This was expected since FT's policy is not to respond (i.e., to respond *null*) when the student selects a step to start working on it (start step scenarios) and when the student makes an error for the first time on a step. In addition, FT exhibited a general trend to select more of the less explicit response types (e.g., *null*, *prompt*) and fewer of the more explicit

response types (e.g., *teach*, *do*). This is consistent with FT's policy of selecting successively more explicit help: FT selects less explicit help for most steps and only for a few steps does it progress all the way to selecting the most explicit help. An exception to this trend occurred, however, when RT had already provided somewhat explicit help, in which case FT's policy was to select the help alternative that was minimally more explicit; this is how FT managed to select more *teach* than *hint* responses.

5.4.2 DT Tutor's overall distribution of response selections

DT's responses were significantly different from FT's, $\chi^2(4)=96.0$, $p<.001$. Particularly remarkable were a large number of *teach* responses and a small number of *hint* responses.

5.4.2.1 DT Tutor's large number of teach responses

A surprise in DT's distribution of response selections was that it selected significantly more *teach* responses than any other type. There appear to be a few reasons for this. First, DT's discourse coherence model, in concert with a low utility for incoherent responses, followed a constraint of selecting successively more explicit help responses (see section 4.5.1.1). FT's policy followed a similar but stronger constraint (see section 5.2.2.3) which limited it to selecting help messages that were minimally more explicit. Therefore, DT, like FT (which also selected more *teach* than *hint* responses, although by a much smaller margin), was constrained to select more explicit help responses than RT had already provided (or else to reply *null* or repeat the *do* response). So DT and FT sometimes selected *teach* just because RT had most recently selected *hint*. When RT had most recently selected *prompt*, FT was constrained to select *hint* (or no help at all) because *hint* is minimally more explicit than *prompt*. DT was free to select *hint* as well in such situations but, since *prompt* and *hint* were usually of similar effectiveness (see sections 4.4.3.5, 4.4.3.6, 4.4.3.7, 5.5.1 and 6.1.4) and usually less effective than *teach*, DT usually selected *teach* over *hint*.

Second, even though the *do* message (telling the student exactly how to *do* a step without teaching the rationale for the action) was usually more effective than *teach*, *do* sharply decreased the student's feeling of independence (see section 3.2.8). So *teach* was sometimes preferred over *do* because its influence on the student's affective state was less deleterious.

Third, one of DT's utilities was set to slightly favor the *teach* response over other responses as described in section 4.5.1.7 and shown in Table 4.9. This was done as part of tuning during the data collection phase because the experimenter believed that DT was not responding with *teach* often enough.

The adjustment was based on training set performance before assessment with the test data. Perhaps this was an overadjustment, upsetting DT's delicate balance of multiple considerations (considering the student's knowledge, problem-solving progress and feeling of independence, etc. – see section 4.5) so that it selected *teach* too many times and *hint* too few. A change to this utility would reduce the number of *teach* responses.

5.4.2.2 DT Tutor's small number of hint responses

The other main surprise in DT's distribution of response selections is that it selected very few *hint* responses. Part of this must simply be due to an imbalance between the *hint* and *teach* responses, as mentioned in section 5.4.2.1 above. But another reason is that the empirically-learned conditional probabilities for the teaching effectiveness of the *prompt* and *hint* responses were often about the same, as discussed in sections 4.4.3.5, 4.4.3.6, 4.4.3.7, and 6.1.4. In fact, the judges sometimes considered the *prompt* and *hint* messages to be equivalent (even though they had different content) – anecdotal evidence for this is listed in section 5.5.1, item 6. But *hint* had a more negative impact than *prompt* on the student's feeling of independence, according to DT's model (see section 3.2.8). And both *prompt* and *hint* were usually less effective teaching responses than *teach*. So, all else being equal, DT preferred *prompt* over *hint* when the student was less likely to need explicit help, and it preferred *teach* over *hint* when the student was more likely to need explicit help.

5.4.3 First-message-opportunity scenarios: pretest-wrong, pretest-right

A subset of the scenarios will often be important for understanding the contrasts between the tutorial responses selected by FT and DT. This subset, which includes 188-190[15] of the 350 scenarios, involves problem-solving steps for which the Random Tutor (RT) had not yet given help to the student. These are called *first-message-opportunity scenarios* (FMOs) because the tutor has the opportunity to select the first help message to be displayed for the current step. For these scenarios, DT had free reign over which tutorial response to select (if any) while FT adhered to its fixed policy. For other scenarios, involving problem steps for which RT had already given help to the student, both FT and DT abided by a constraint (described in Section 4.5.1.1) to select a tutorial response (if any) that was more explicit than the help that RT had already given, or else to repeat the *do* response if RT had already given it. In the most

[15] The number of first-message-opportunity scenarios faced by a tutor depended on which topics the tutor selected for the 5 general impasse scenarios, for each of which the tutor had two topic choices.

constraining cases, when RT had already given either *teach* or *do* help, both FT and DT were constrained to give *do* help (if any). Thus, first-message-opportunity scenarios were less constrained and so had more potential for revealing differences in the tutoring behaviors of FT and DT. First-message-opportunity scenarios cut across the three subsets of scenarios described in section 5.2.2.2 (help requests, errors and step starts) because any of these types of scenarios may involve a first-message-opportunity.

First-message-opportunity scenarios were sometimes partitioned according to student performance on the pretest problem that corresponded to the rule required to complete the current problem step: *pretest-wrong* and *pretest-right*. The idea behind this partitioning is that students who get a pretest problem wrong are more likely than those who get it right to need explicit help during tutoring on steps that require knowledge of the rule tested by the pretest problem. This is by no means a perfect test – e.g., the student might have merely slipped on the pretest problem or the student might have learned the rule since the pretest – but one advantage is that it does not require subjective judgments by the experimenter.

With the pretest-wrong/right partitioning, it must be noted that DT was not given information about the pretest performance of students in the test set. However, DT could glean information about the likelihood that a particular student in the test set knew a rule in two ways: (1) by the percentage of the training set students who got the corresponding pretest problem correct (recorded as prior probabilities as described in section 4.4.2), and (2) by the student's actions during tutoring on steps related to the rule such as correct actions, help requests and errors.

Table 5.2 displays FT's and DT's number of response types for the first-message-opportunity subsets pretest-wrong and pretest-right. FT's and DT's responses are significantly different, $\chi^2(2)=91.0$, $p<.001$. FT's response pattern is the same for both pretest-wrong and pretest-right, with 56-58% of responses *null* and the remainder *prompt*. DT's response patterns significantly differ between pretest-wrong and pretest-right: $\chi^2(2)=7.59$, $p=.023$. For the pretest-wrong scenarios, DT selected relatively more *teach* responses (51%), and for the pretest-right scenarios, DT selected relatively more *null* and *prompt* responses (69%). DT thus tends to provide more explicit help when students are more likely to need it, and to provide less explicit help, if any, when students are less likely to need it.

Table 5.2: FMO responses for all scenarios, percentages: pretest wrong & right, FT & DT

Subset	Response Type: Null	Prompt	Hint	Teach	Do
Pretest-wrong					
FT	56	44	0	0	0
DT	27	23	0	51	0
Pretest-right					
FT	58	42	0	0	0
DT	41	28	0	31	0

5.4.4 The tutors' response distributions for help requests

Table 5.3: Distributions of response types for help requests, percentages

	null	prompt	hint	teach	do
RT	0	26	19	26	29
FT	0	43	17	22	18
DT	0	17	5	60	18

Table 5.3 displays the tutors' distributions of response types for help requests. None of the tutors selected the *null* response for help requests because not responding to a help request was considered bad user interface design. FT displayed the same general trend described in section 5.4.1 to select more of the less explicit response types (e.g., *prompt*) and fewer of the more explicit response types (e.g., *teach*, *do*). DT likewise exhibited its tendency to select many *teach* responses and few *hint* responses (discussed in sections 5.4.2.1 and 5.4.2.2). Indeed, nearly 2/3 of the *teach* responses that DT issued were for help requests, probably because the very fact that the student requested help (in the absence of help abuse) increased the likelihood that the student needed more explicit help. DT's responses were significantly different from FT's, $\chi^2(3)=60.8$, $p<.001$.

5.4.4.1 FT and DT response distributions for FMO help requests

Table 5.4: FMO responses for help requests, percentages: pretest wrong & right, FT & DT

	Response Type				
Subset	**Null**	**Prompt**	**Hint**	**Teach**	**Do**
Pretest-wrong					
FT	0	100	0	0	0
DT	0	34	0	66	0
Pretest-right					
FT	0	100	0	0	0
DT	0	44	0	56	0

Table 5.4 shows FT's and DT's distributions of responses for first-message-opportunity help requests. FT always selected *prompt* while DT split its responses between *prompt* and *teach*, and this difference between FT and DT was significant, $\chi^2(1)=63.3$, $p<.001$. DT selected *teach* 66% of the time for the pretest-wrong scenarios and a little less often, 56% of the time, for the pretest-right scenarios, again exhibiting a tendency to provide more explicit help when the student is more likely to need it, but this difference was not significant, $\chi^2(1)=.726$, $p=.394$.

5.4.5 The tutors' response distributions for errors

Table 5.5: Distributions of response types for errors, percentages

	null	**prompt**	**hint**	**teach**	**do**
RT	45	13	7	21	14
FT	68	6	10	10	6
DT	39	12	3	45	1

Table 5.5 shows the tutors' distributions of response types for errors. Non-*null* responses to errors are considered proactive (i.e., unsolicited) help because the student has not asked for help. RT returned

significantly more *null* responses than any other response type because it randomly decided to provide proactive help about 50% of the time, so about 50% of the time it decided not to provide proactive help – i.e., to return a *null* response. FT returned an even larger number of *null* responses because its policy is not to provide proactive help for first errors and 68 out of the 100 error scenarios were first errors. DT tended to select *null* or *teach* responses. DT's responses were significantly different from FT's, $\chi^2(4)=39.5$, $p<.001$.

5.4.5.1 FT and DT response distributions for FMO errors

Table 5.6: FMO responses for errors, percentages: pretest wrong & right, FT & DT

	Response Type				
Subset	**Null**	**Prompt**	**Hint**	**Teach**	**Do**
Pretest-wrong					
FT	82	18	0	0	0
DT	41	14	0	45	0
Pretest-right					
FT	92	8	0	0	0
DT	38	35	0	27	0

For first-message-opportunity errors, FT always selected *null* for first errors and *prompt* for second errors. Most first-message-opportunity scenarios were first errors (for second errors that remained first-message-opportunities, RT must have selected *null* for the first error). FT's response selections differed significantly from DT's, $\chi^2(2)=27.7$, $p<.001$. DT selected from among *null*, *prompt* and *teach*, tending to select relatively more *teach* responses for pretest-wrong scenarios (when students were more likely to need explicit help) and more *null* and *prompt* responses for pretest-right scenarios, but this difference was not significant: $\chi^2(2)=3.27$, $p=.195$.

5.4.6 The tutors' response distributions for step starts

Table 5.7: Distributions of response types for step starts, percentages

	null	prompt	hint	teach	do
RT	49	7	15	16	13
FT	100	0	0	0	0
DT	72	9	0	19	0

Table 5.7 shows the tutors' distributions of response types for step start scenarios (when a student selects a step to start working on it). Any help provided for these scenarios is considered proactive (i.e., unsolicited) because the student is not asking for help at the time that it is provided. As with error scenarios, RT returned significantly more *null* responses than any other response type because it randomly decided to provide proactive help about 50% of the time, so about 50% of the time it decided not to provide proactive help – i.e., to return a *null* response. FT followed its policy to never respond (i.e., to always respond *null*) to step start scenarios. DT likewise selected mostly *null* responses but also a few *prompt* and *teach* responses, and this made DT's responses significantly different from FT's, $\chi^2(2)=24.4$, $p<.001$.

5.4.6.1 FT and DT response distributions for FMO step starts

Table 5.8: FMO responses for step starts, percentages: pretest wrong & right, FT & DT

	Response Type				
Subset	**Null**	**Prompt**	**Hint**	**Teach**	**Do**
Pretest-wrong					
FT	100	0	0	0	0
DT	46	17	0	37	0
Pretest-right					
FT	100	0	0	0	0
DT	86	7	0	7	0

66 of 75 step start scenarios were also first-message-opportunity scenarios. Not all step start scenarios are first-message-opportunity scenarios because sometimes a student selects a step to resume work on it after previously receiving help but not successfully completing it. Again, FT always selected *null* for these scenarios regardless of the situation and, not surprisingly, its responses were significantly different from DT's: $\chi^2(2)=22.2$, $p<.001$. DT chose mostly the *null* response for pretest-right scenarios but chose relatively more *teach* and *prompt* responses for pretest-wrong scenarios (when students were more likely to need explicit help), and this difference was significant: $\chi^2(2)=12.5$, $p=.002$.

5.5 THE JUDGES' EVALUATIONS

5.5.1 The judges' comments

As part of their evaluations of each scenario, the judges were asked to write down two types of freeform entries: (1) optional comments about any of the scenario's response options, and (2) a required comment about the single best response, which could have been "none" (no response), one of the response options given for the scenario, or a response that the judge made up.

This study focuses on the judges' numeric ratings of the scenario response options rather than their comments. Nevertheless, many of their comments were informative. They will be used within this text to help illustrate and interpret some of the numeric results. Furthermore, some of the comments may be considered surprising and so may inspire further investigation.

Some of the interesting points made by the judges along with representative quotes are listed below. In the list, the notation "Best response" indicates that the comment was made about the single best response for a scenario.

1. Regarding selecting successively more explicit help (FT's policy of selecting help in order of increasing explicitness: *prompt*, *hint*, *teach*, *do*):
    a. **Successively explicit help is often preferred:** Best response – "C [*prompt*] is the best, although it might be too difficult. If it is, use A [*hint*] next time."
    b. **Don't always start out with the least explicit help:** Best response – "Since the student didn't solve the pretest problem, [help] must be more clear than A [*hint*] or C [*prompt*]. E [*teach*]."

c. **Don't always progress from less to more explicit help:** Best response – "If D [*teach*] doesn't help, next time give E [*prompt*] as a hint ""

d. **Don't always get more explicit a step at a time:** Best response – "There is no history. If A [*prompt*] doesn't work, next time use B [*teach*]."

e. **It's okay to repeat a help message**: Best response – "B [*teach*] didn't help last time [on the current step], but it's still the best tip. If it doesn't help, next time give the exact answer."

2. **Judges react to help abuse or avoidance:**
   a. Help abuse: Best response – "The student abuses help -- he/she is only waiting for the exact answer. A [*null*]."
   b. Help avoidance: Best response – "E [*do*] because he has to at least try using help more"
3. **Consider whether a response worked before:**
   a. Responses that worked before: Best response – "Well, since B [*hint*] worked before, let's choose B, although I don't know how this tip is easier or more clear than the others. B [*hint*]."
   b. Responses that didn't work before: Best response – "If E [*teach*] doesn't help (it didn't before), use the empty message next time."

4. **No help in response to a help request is okay[16]:** Best response – "The student seems to make lots of mistakes and ask for a lot of help: it might be that he doesn't take the time to think, so no help will trigger that, hopefully."

5. **Sometimes just provide an example:** Best response – "Taking the first response [*teach*] and using just the example is better."

6. **The tutors' *prompt* and *hint* messages are sometimes similar:** Best response – **"**A [*hint*] and E [*prompt*] are equally good. E [*prompt*]."

5.5.2 The judges' individual ratings

Comparisons were conducted between DT Tutor (DT) and the other two tutors: the Fixed-Policy Tutor (FT) and the Random Tutor (RT). Table 5.9 summarizes the results of a repeated-measures ANOVA with

[16] It is likely that the judge assumes that the tutor would somehow let the student know, implicitly or explicitly, that it has received the help request, for to do otherwise would probably be bad user interface design – e.g., the student might think the tutor has crashed. The judge probably just doesn't want the tutor to provide help.

tutor ratings as the dependent variable, tutor (DT, FT and RT) and judge (judge 1, 2 and 3) as within-subjects variables, and scenario type (help request, error or step start) as the between-subjects variable. As the table shows, there were main effects for tutor, judge, and scenario type at level $p<.001$, and the interactions between these variables were significant as well. In other words, ratings were significantly influenced by the tutor being rated, the judge doing the rating, and the type of scenario. The focus here is on effects for tutor, both the main effect and the two-way interactions for Judge x Tutor and Scenario Type x Tutor. These interactions are described in the following subsections.

Table 5.9: Tutor x Judge x Scenario Type, repeated-measures ANOVA

Source	df	F
Between subjects		
Scenario Type (S)	2	36.33**
Within subjects		
Tutor (T)	2	21.42**
Judge (J)	2	52.99**
J x T	4	18.28**
S x T	4	5.49**
J x S	4	26.25**
T x J x S	8	3.53*

*p = .001. **p < .001

5.5.2.1 Judge x Tutor ratings

Table 5.10 shows the mean rating for each tutor over all scenarios, both by each judge and the overall mean rating for all judges combined. Figure 5.1 shows the same information graphically. The ratings for DT are higher than the ratings for FT and RT, both for each judge and for the mean of all judges. Pairwise tests of statistical significance comparing DT and the other tutors using composite judges' ratings will be described in section 5.6.

The ratings for FT are higher than the ratings for RT except from Judge 3. The exception for Judge 3 reflects an interaction between tutor and judge – i.e., the influence of the different tutors on the ratings depends in part on which judge is doing the rating. This interaction is discussed in section 5.5.2.5 below.

Table 5.10: Tutor x Judge, mean ratings: RT vs. FT vs. DT

Rater	Tutor RT	FT	DT
Judge 1	3.14	3.35	3.44
Judge 2	2.89	3.19	3.39
Judge 3	2.75	2.55	3.22
All judges	**2.93**	**3.03**	**3.35**

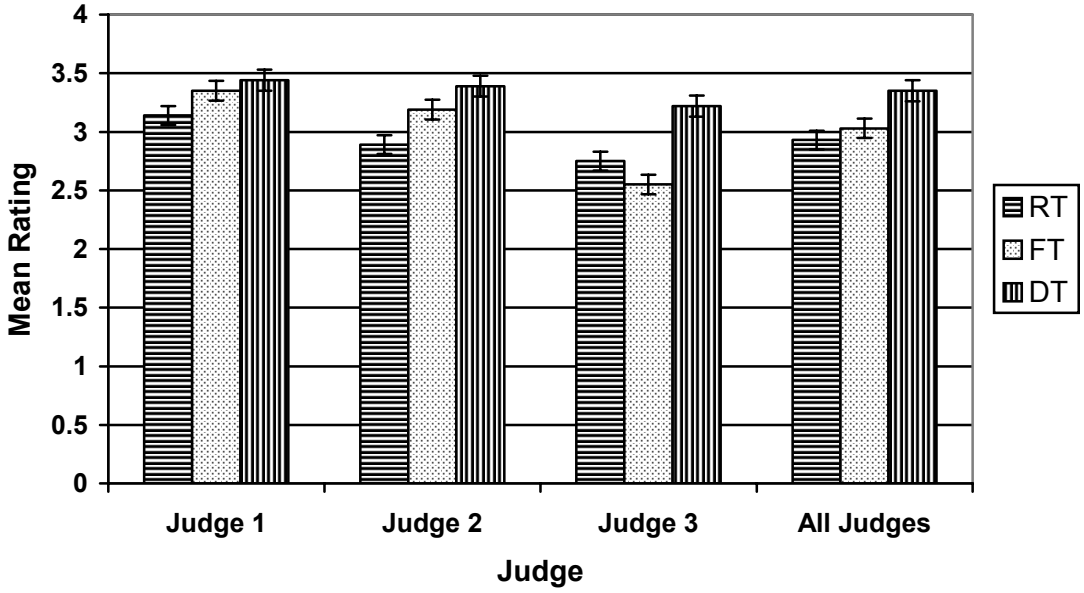


Figure 5.1: Tutor x Judge, mean ratings: RT vs. FT vs. DT[17]

[17] The error bars in Figure 5.1 and in all other figures within this thesis represent the standard error of the mean. At the time of this writing, the Microsoft Excel software used for these figures supports only one size of error bar per column type (e.g., in this figure, one error bar size for RT, one for FT, and one for DT). Therefore, the error bars shown are the largest standard error of the mean applicable for the column type. This is a conservative representation – the errors are generally smaller except on one column of each type.

5.5.2.2 Scenario Type x Tutor Ratings

Table 5.11 shows the mean rating for each tutor by all judges for each scenario type (help requests, errors, step starts and first message opportunities) and overall. Figure 5.2 shows the same information graphically. The ratings for DT are higher than the ratings for FT and RT for each scenario type and overall ("All Scenarios"), although just barely so in the case of help requests (pairwise tests of significance are described in section 5.6).

FT is rated higher than RT overall as well as for first message opportunities and help requests. However, it is rated lower for both errors and step starts. Thus, this view of the data also shows an interaction, this time between scenario type and tutor – i.e., the influence of the different tutors on the ratings depends in part on the scenario type. These interactions are discussed in the following sections.

Table 5.11: Tutor x Scenario Type, mean ratings: RT vs. FT vs. DT

	Tutor		
Scenario Subset	**RT**	**FT**	**DT**
Help Requests	3.18	3.51	3.56
Errors	2.40	2.24	2.99
Step Starts	3.04	2.97	3.35
First Message Opportunities	2.97	3.06	3.44
All Scenarios	2.93	3.03	3.35

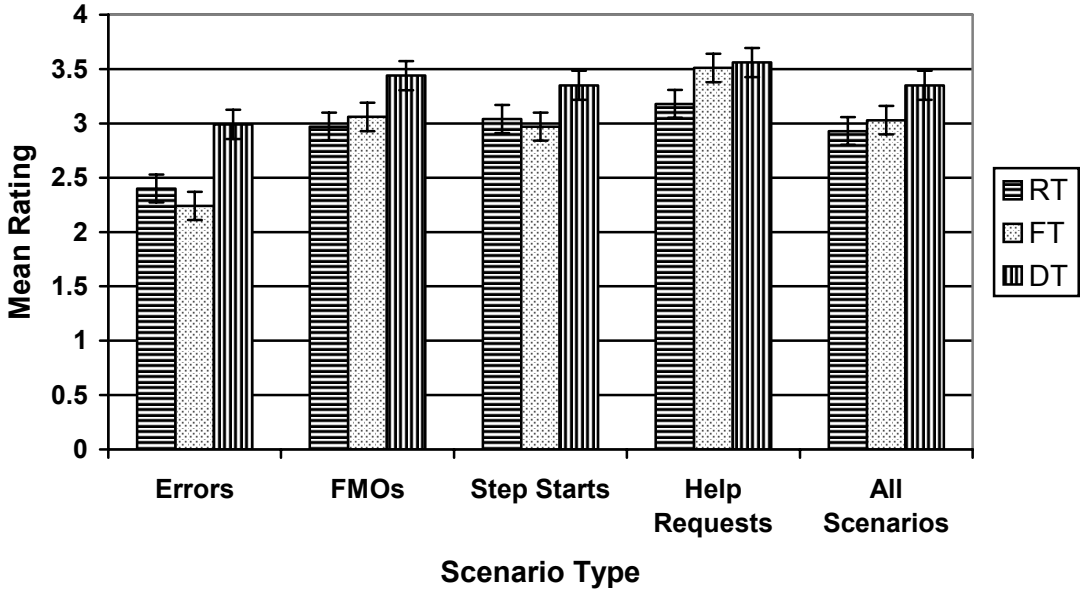


Figure 5.2: Tutor x Scenario Type, mean ratings: RT vs. FT vs. DT

5.5.2.3 Scenario Type x Tutor Interaction for Errors

FT was rated lower than RT for errors because FT always selects a *null* response (i.e., no response) the first time the student makes an error, and for error scenarios the judges rated *null* responses the lowest as shown in Table 5.12. Consequently, on the 68 of 100 error scenarios that involved the student's first error, FT received low ratings for its response while RT received a variety of ratings for its randomly selected responses. Figure 5.3 and Figure 5.4 show the differences in judges' ratings of RT and FT for first errors and subsequent errors, respectively. For the 32 error scenarios that involved second or subsequent errors, FT was rated more highly than RT.

Table 5.12: Error scenario ratings, means by each judge and overall

	Tutorial Response Type				
Rater	**Null**	**Prompt**	**Hint**	**Teach**	**Do**
Judge 1	1.50	2.82	3.76	3.89	2.46
Judge 2	1.79	3.52	3.66	3.85	1.21
Judge 3	1.08	3.34	3.23	4.28	2.13
All Judges	**1.46**	**3.23**	**3.55**	**4.01**	**1.93**

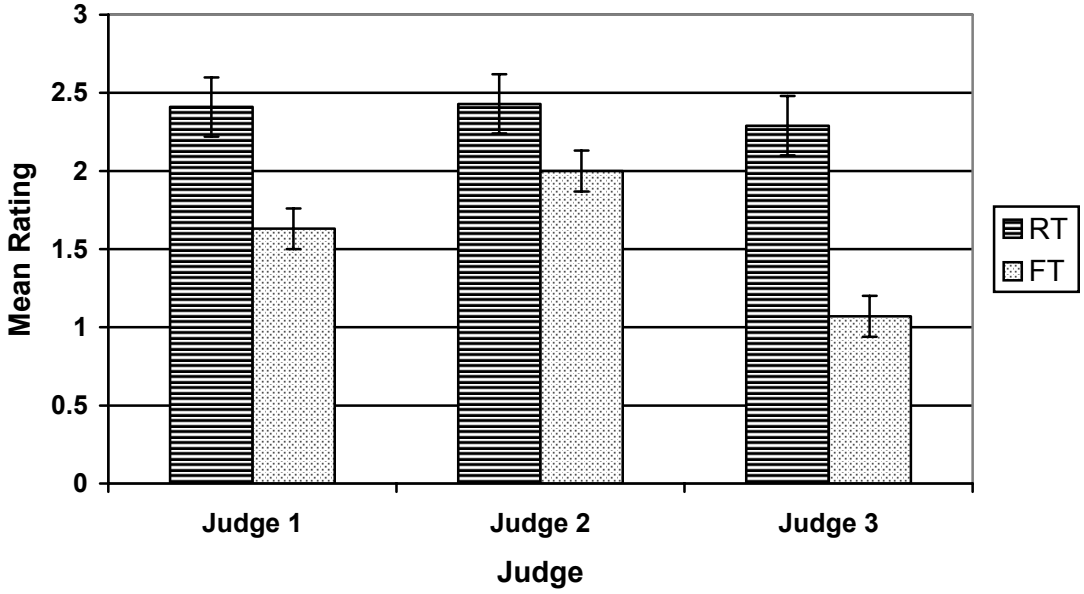


Figure 5.3: First error scenario ratings by each judge: RT vs. FT

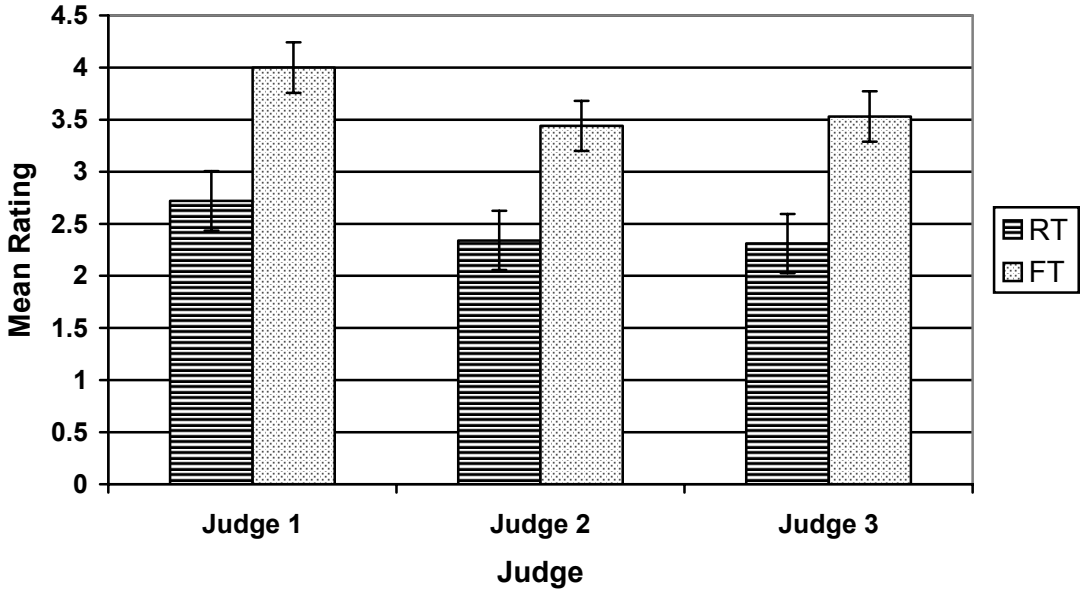


Figure 5.4: Subsequent error scenario ratings by each judge: RT vs. FT

5.5.2.4 Scenario Type x Tutor Interaction for Step Starts

FT was rated lower than RT for step starts because FT always selects a *null* response (i.e., no response) for step starts and the mean judges' rating for the *null* response was the second lowest of all the tutorial response options, after *do* responses, as shown in Table 5.13. However, notice that the low mean rating for *null* responses is due to Judge 3, who rated *null* much lower than the other judges did, at 1.59 versus 3.91 and 3.43. Figure 5.5 shows that Judge 3 was the only judge who rated FT lower than RT for step starts, but it was by a large enough amount to affect the relative means of FT and RT over all judges.

Table 5.13: Step start scenario ratings, means by each judge and overall

	Tutorial Response Type				
Rater	**Null**	**Prompt**	**Hint**	**Teach**	**Do**
Judge 1	3.91	3.59	3.80	2.95	2.01
Judge 2	3.43	3.92	3.33	3.03	1.16
Judge 3	1.59	3.67	3.37	4.13	1.96
All Judges	**2.97**	**3.72**	**3.50**	**3.37**	**1.71**

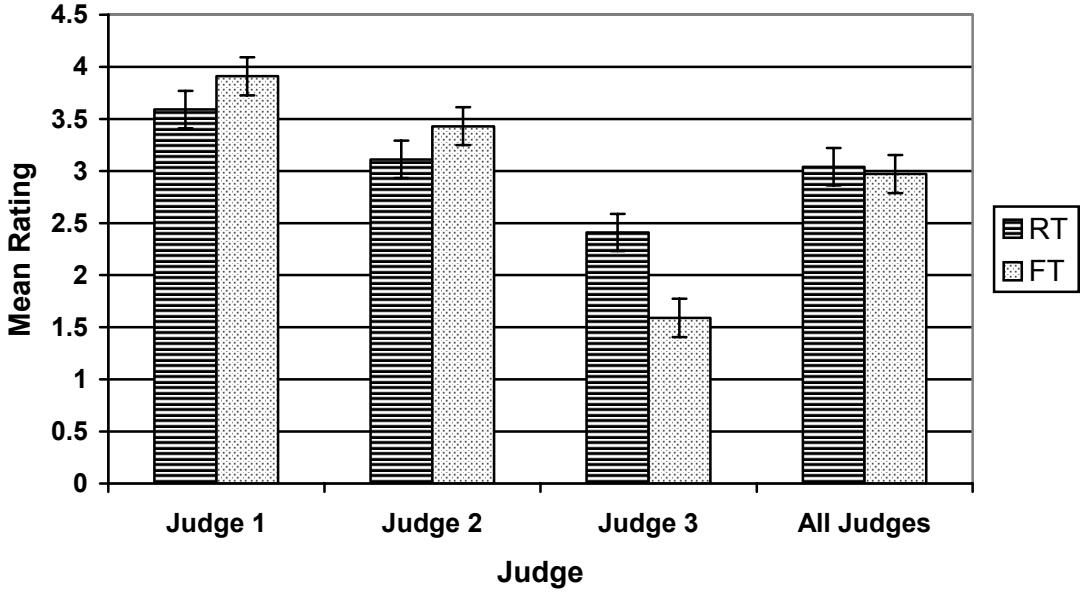


Figure 5.5: Step start scenario ratings by each judge: RT vs. FT

5.5.2.5 Judge x Tutor Interaction for Judge 3

As mentioned in section 5.5.2.1, there is an interaction between tutor (RT, FT and DT) and judge (Judge 1, 2 and 3), particularly for Judge 3, who rated FT lower than RT overall while the other judges rated FT higher than RT overall (see Figure 5.1).

Judge 3 rated FT higher than RT for help requests but lower than RT for errors and step starts, as shown in Figure 5.6. For errors, as described in section 5.5.2.3, Judge 3 rated FT lower than RT because FT selected only *null* responses on first error scenarios (68 of 100 error scenarios) and all of the judges – particularly Judge 3 – rated *null* responses lower than other responses for error scenarios (see Table 5.12 and Figure 5.3).

For step starts, Judge 3 was the only judge who rated FT lower than RT, as described in section 5.5.2.4 and shown in Figure 5.5. Again, this was because FT selected only *null* responses for step starts and Judge 3 rated these responses particularly low (see Table 5.13).

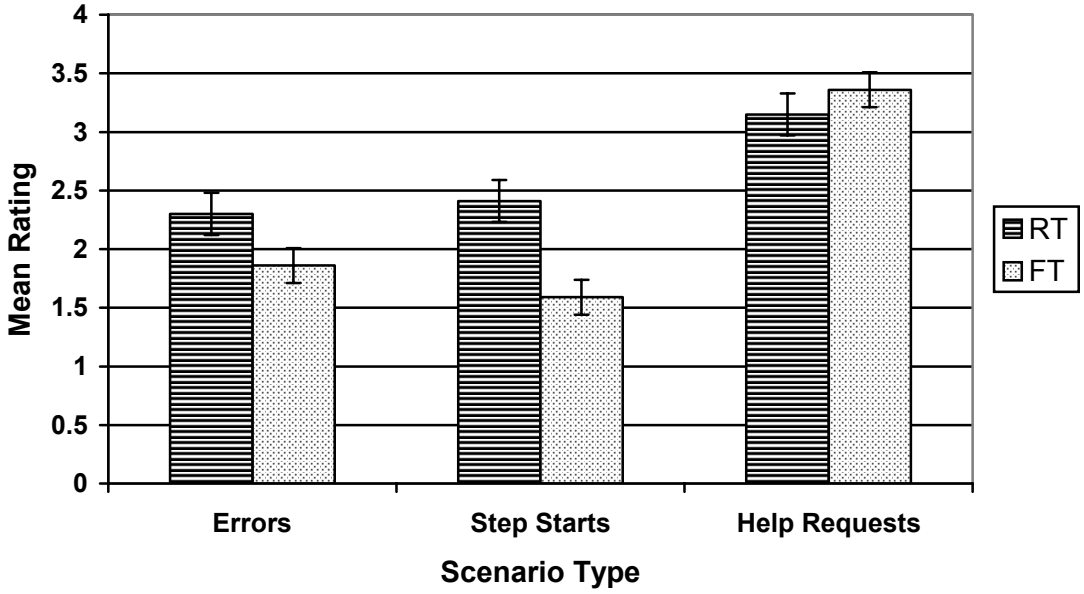


Figure 5.6: Tutor x Scenario Type, Judge 3: RT vs. FT

5.5.3 Composite judges' ratings

A composite set of judges' ratings, properly constructed from the ratings of the three judges, has the potential to represent the population of skilled tutors (the judges were all skilled tutors, as described in section 5.2.1) better than any one of the three skilled tutors who participated in this study. This section will motivate, justify and describe the method used to construct the composite judges' ratings used in this evaluation.

5.5.3.1 Similarities among judges' ratings for all responses

Agreement between the three judges' ratings is shown in Table 5.14. Agreement was calculated in two ways. First, Pearson correlation coefficients were computed for each of the judge's ratings for all three tutors' responses on all scenarios. The coefficients are significant but not high, revealing a relatively large amount of variation in opinions among these judges, who were intentionally uncoached in order to leave their intuitions undisturbed. Second, for each judge's preferred tutorial response for each scenario, the mean rating (on a scale of 1 to 5) of that tutorial response by the other two judges was computed.

Table 5.14 shows the mean value of this measure over all scenarios. This measure reflects agreement about each judge's most preferred response for each scenario, which were not necessarily the responses that any of the tutors selected. None of the agreement measures was low enough to warrant throwing out all of the ratings of any of the judges for being unreasonably outside the norm.

Table 5.14: Agreement among judges, all scenarios

Comparison		Pearson's correlation coefficient	Rating by the Other Judges
Judge 1	**vs. Judge 2**	r=.526**	3.38
	vs. Judge 3	r=.360**	
Judge 2	**vs. Judge 1**	r=.526**	3.72
	vs. Judge 3	r=.447**	
Judge 3	**vs. Judge 1**	r=.360**	3.71
	vs. Judge 2	r=.447**	
** $p < .001$			

5.5.3.2 Contrasts in ratings for subsets of scenarios

We have already seen some systematic contrasts in the judges' ratings for subsets of the scenarios: (1) Figure 5.1 and section 5.5.2.5 showed that Judge 3 rated FT lower than RT overall, in contrast to Judges 1 and 2; and (2) section 5.5.2.4 described how FT was rated lower than RT for step start scenarios due to the influence of Judge 3, who rated the *null* response much lower than did the other judges (1.59 versus 3.91 and 3.43 – see Table 5.13).

For step start scenarios, which are opportunities to provide proactive help when the student first selects a step to start working on it, *null* responses are widely accepted. Forgoing providing unsolicited help gives the student a chance to complete the step on her own, potentially reaping the benefits of knowledge construction and promoting her feeling of independence. In fact, FT, gives only *null*

responses for these scenarios, and indeed even considering whether to provide proactive help in such situations is one of the major differences between FT and DT. Therefore, it appears that Judge 3 may be outside the tutor norm for step start scenarios. This disparity in the judges' ratings is reflected in low Pearson's correlation coefficients between Judge 3 and the other judges on step start scenarios, shown in Table 5.15.

Table 5.15: Agreement among judges, step start scenarios

Comparison		**Pearson's correlation coefficient**	**Significance**	**Rating by the Other Judges**
Judge 1	**vs. Judge 2**	r=.545	p<.001	3.23
	vs. Judge 3	**r=.089**	p=.183	
Judge 2	**vs. Judge 1**	r=.545	p<.001	3.55
	vs. Judge 3	**r=.200**	p=.003	
Judge 3	**vs. Judge 1**	**r=.089**	p=.183	3.59
	vs. Judge 2	**r=.200**	p=.003	

But Judge 3 was not the only judge who may have been outside the tutor norm for a subset of scenarios. For instance, for the 75 first-message-opportunity help requests, Judge 1 gave *null* responses relatively high ratings, averaging 3.34, while Judge 3 gave *null* responses relatively low ratings, averaging 1.25. Most tutors, including computer tutors, do respond in some way (i.e., with a non-*null* response) to help requests – to do otherwise would probably be bad user interface design. (However, it is likely that Judge 1 anticipated that the tutor would provide some sort of response – just not help – as mentioned in section 5.5.1 and discussed in section 6.6.3.1). For these scenarios, it may be Judge 1's ratings that are outside the tutor norm. This disparity in the judges' ratings shows up in the agreement measures for first-message-opportunity help requests, shown in Table 5.16: (1) a Pearson's correlation coefficient of only r=.177 between Judge 1 and Judge 3, and (2) a mean rating by the other judges of only 2.76 for Judge 1.

Table 5.16: Agreement among judges, first-message-opportunity help requests

Comparison		Pearson's correlation coefficient	Significance	Rating by the Other Judges
Judge 1	**vs. Judge 2**	r=.377	p<.001	**2.76**
	vs. Judge 3	**r=.177**	p=.008	
Judge 2	**vs. Judge 1**	r=.377	p<.001	3.66
	vs. Judge 3	r=.306	p<.001	
Judge 3	**vs. Judge 1**	**r=.177**	p=.008	3.64
	vs. Judge 2	r=.306	p<.001	

5.5.3.3 Composite judges' ratings use the median rating for each response

For constructing composite ratings from the ratings of our three judges to represent the population of skilled tutors, the goal was to discount ratings that were outside the norm without coarsely excluding any of the judges' ratings. To this end, the median rating for each response was used rather than the more commonly applied mean. The median discounts the effect of the magnitude of outlying ratings while still taking their existence into account. For instance, for the three ratings 1, 4, and 5, the median, which is 4, reflects the majority consensus of a rating toward the top end of the 1 to 5 scale, while discounting the magnitude of the outlying rating of 1 (in fact, it is unaffected by the magnitude of ratings at or below the median value). The mean for this set of values, 3.33, is affected by the magnitude of the outlying rating. Furthermore, the median discounts the magnitude of any outlying rating, regardless of the source, so it functions just as well regardless of which judge's ratings are outliers for a particular subset of scenarios. With outlying ratings for individual responses thus discounted by using the median of the three judges' ratings, composite ratings for *sets* of responses were computed as the *mean* of the median ratings for each response.

5.6 COMPARING COMPOSITE RATINGS OF THE TUTORS

Using composite judges' ratings constructed as described in section 5.5.3.3, DT Tutor (DT) was compared with the other two tutors: the Fixed-Policy Tutor (FT) and the Random Tutor (RT). Table 5.17 summarizes the results of a repeated-measures ANOVA with composite judges' ratings as the dependent variable, tutor (DT, FT and RT) as the within-subjects variable, and scenario type (help request, error or step start) as the between-subjects variable. As the table shows, there were main effects for tutor and scenario type at level $p<.001$, and the interaction between tutor and scenario type was significant as well, $p=.001$.

Table 5.17: Tutor x Scenario Type, repeated-measures ANOVA: RT vs. FT vs. DT

Source	df	F
Between subjects		
Scenario Type (S)	2	32.86**
Within subjects		
Tutor (T)	2	20.77**
T x S	4	4.79*

*p = .001. **p < .001

Table 5.18 shows each tutor's mean composite rating for each scenario type (help requests, errors, step starts and first message opportunities) and for all scenarios. Figure 5.7 shows the same information graphically. As with the individual judges' ratings, the composite ratings for DT are higher than the ratings for FT and RT for each scenario type, although just barely so in the case of help requests. Pairwise tests of significance will be described shortly

Table 5.18: Tutor x Scenario Type, composite ratings: RT vs. FT vs. DT

Scenario Subset	Tutor RT	FT	DT
Help Requests	3.23	3.59	3.66
Errors	2.31	2.10	2.95
Step Starts	3.11	3.19	3.55
First Message Opportunities	2.99	3.12	3.54
All Scenarios	2.94	3.08	3.43

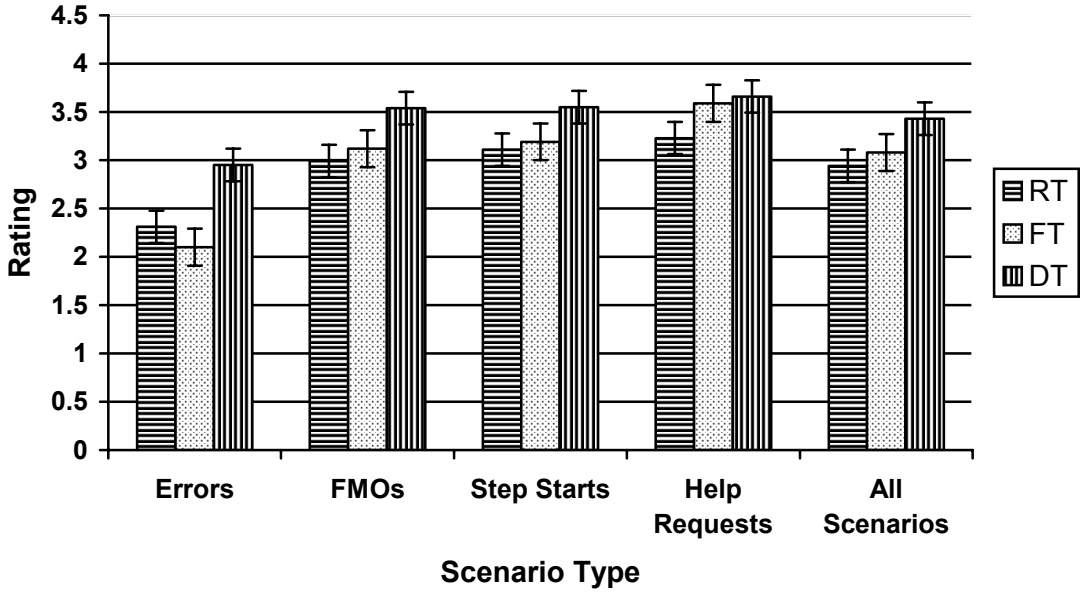


Figure 5.7: Tutor x Scenario Type, composite ratings: RT vs. FT vs. DT

The interaction(s) between scenario type and tutor discussed in sections 5.5.2.2 through 5.5.2.4 still exist with the composite ratings, but they are smaller. In particular, FT is no longer rated lower than RT for step starts. This is because using the median judges' rating discounted the influence of the outlying judge's low ratings of FT's *null* responses (discussed in section 5.5.2.4). FT is still rated lower than RT for errors because the judges were unanimous in giving low ratings to *null* responses to errors, as discussed in section 5.5.2.3.

Table 5.19 displays results of paired-sample t-tests comparing RT vs. DT and FT vs. RT for all scenarios and for each scenario type, along with effect sizes and mean composite ratings. Effect sizes were calculated as the difference in means divided by the standard deviation of the control group: either RT or FT as applicable in their comparisons with DT.

Table 5.19: Tutor x Scenario Type, composite ratings, paired t-tests: RT vs. DT, FT vs. DT

Comparison			df	t	Sig.	Bonferroni Sig.*	Effect Size
RT vs. DT	**RT Mean**	**DT Mean**					
All Scenarios	2.94	3.43	349	5.746	<.001	<.01	.35
Help Requests	3.23	3.66	174	3.937	<.001	<.01	.33
Errors	2.31	2.95	99	3.324	.001	.010	.49
Step Starts	3.11	3.55	74	2.572	.012	.120	.30
FMOs	2.99	3.54	187	5.057	<.001	<.01	.40
FT vs. DT	**FT Mean**	**DT Mean**					
All Scenarios	3.08	3.43	349	5.251	<.001	<.01	.24
Help Requests	3.59	3.66	174	1.078	.282	1.0	.06
Errors	2.10	2.95	99	4.693	<.001	<.01	.61
Step Starts	3.19	3.55	74	3.222	.002	.020	.22
FMOs	3.12	3.54	187	4.351	<.001	<.01	.28

* Significance with Bonferroni correction for 10 t-tests (Sig. x 10)

5.6.1 Composite ratings: Random Tutor vs. Decision-Theoretic Tutor

As Table 5.19 shows, the composite judges' rating for DT was higher than the composite rating for RT overall and for help requests, errors and first message opportunities, significant at level $p<.01$ with effect sizes ranging from .33 to .49. Only for step start scenarios was DT not rated significantly higher than RT after the Bonferroni correction for multiple comparisons. However, the significance before the Bonferroni correction was $p=.012$ and the Bonferroni correction is known to be very conservative to protect against Type I errors (Corston & Colman, 2003). The effect size for step starts was still a healthy .30.

5.6.2 Composite ratings: Fixed-Policy Tutor vs. Decision-Theoretic Tutor

Referring again to Table 5.19, the composite judges' rating for DT was higher than the composite rating for FT overall and for the subsets help requests, errors, step starts and first message opportunities, all with significance p=.02 or less and with effect sizes ranging from .22 to .61. For help requests, however, DT, with mean 3.66, and FT, with mean 3.59, were rated approximately equivalently with a .06 effect size and a significance level (with Bonferroni correction) of approximately p=1.0. Results for subsets of scenarios are further discussed in the following sections.

5.6.2.1 Decision-Theoretic Tutor vs. Fixed-Policy Tutor: Help requests

Since DT's and FT's ratings were approximately the same for help requests, one might expect that DT and FT selected mostly the same tutorial responses in the same situations. However, their patterns of responses were significantly different, as shown in Table 5.3 and confirmed with a Pearson's chi-square test of association, $\chi^2(3)=60.8$, $p<.001$.

As discussed in section 5.4.4.1, FT and DT also behaved significantly differently for the subset of help requests that were also first-message-opportunity scenarios, for which FT always selected the *prompt* response according to its fixed-policy. DT's response selections varied: For the pretest-wrong scenarios, DT selected *prompt* only 34% of the time and *teach* 66% of the time, receiving a mean composite rating of 4.00 while FT received a mean composite rating of 3.55 for its *prompt* responses. A paired-samples t-test found this difference in mean composite ratings to be significant, $t(28) = 2.218$, $p=.035$. For the pretest-right scenarios, DT selected *prompt* slightly more often, 44% of the time, and received a mean composite rating of 3.80 while FT's responses (again, all *prompt*) received a higher mean composite rating, 4.02, although this difference just failed to reach marginal significance, $t(44) = 1.634$, $p=.109$. Apparently, the judges generally preferred the *teach* response when the student was more likely to need explicit help and the *prompt* response when the student was less likely to need explicit help. DT adjusted its response selections according to the same preference structure but did not adjust them enough when the student was less likely to need explicit help.

5.6.2.2 Decision-Theoretic Tutor vs. Fixed-Policy Tutor: Errors

As discussed in section 5.5.2.3, FT received lower ratings than RT (and so also lower than DT) for error scenarios because it always selects a *null* response the first time the student makes an error. 68 out of the

100 error scenarios involved the student's first error, and all of the judges gave low ratings to *null* responses after errors (see Table 5.12), so FT responded *null* and received a low rating for a majority of the error scenarios. For first errors, DT's mean composite rating, 2.88, was significantly higher than FT's rating of 1.35, according to a paired-samples t-test, $t(67)=8.516$, $p<.001$, with a large effect size of 2.58.

On the 32 error scenarios that did not involve the student's first error, FT, with a mean composite rating of 3.69, was rated higher than DT, which had a mean of 3.09, $t(31) = 2.094$, $p=.044$. This was in turn due to DT replying *null* on 13 of these 32 scenarios, for which it received a mean rating of only 1.23 compared to FT's mean of 3.10. The bottom line is that our judges did not like *null* responses to errors.

5.6.2.3 Decision-Theoretic Tutor vs. Fixed-Policy Tutor: Step starts

As with error scenarios, FT received lower ratings than DT for step start scenarios because of *null* responses. Per its fixed policy, FT always selected *null* responses for step start scenarios, which are scenarios when a student selects a step to start working on it (before the student has had a chance to complete the step – correctly or in error – or to request help). DT also selected *null* on 54 of the 75 step start scenarios, and for these scenarios, DT received the same ratings as FT. On the 21 step start scenarios for which DT did not reply *null*, DT's mean composite rating, 3.67, was significantly higher than FT's mean composite rating of 2.38, $t(20) = 3.959$, $p=.001$, effect size .92. DT's significant advantage in ratings when it did not reply *null* led to a significant advantage over FT in ratings for step scenarios overall, 3.55 versus 3.19, as shown in Figure 5.7 and Table 5.19.

Sections 5.5.2.4 and 5.5.2.5 describe how Judge 3's particularly low ratings of FT's *null* responses on start steps resulted in a lower *mean* judges' rating than even RT's, at 2.97 versus 3.04. Using the *median* composite rating described in section 5.5.3.3, FT's composite rating is no longer lower than RT's, at 3.19 versus 3.11. However, even the median composite rating did not rank *null* responses as high as response types *prompt*, *hint* and *teach*, as shown in Table 5.20. Once again, our judges did not favor *null* responses.

Table 5.20: Step start scenario composite ratings by response type

	Tutorial Response Type				
	Null	**Prompt**	**Hint**	**Teach**	**Do**
Composite (median) rating	3.19	3.75	3.53	3.25	1.24

5.6.2.4 Decision-Theoretic Tutor vs. Fixed-Policy Tutor: FMO scenarios

First-message-opportunity (FMO) scenarios, described in section 5.4.3, have particular potential for revealing differences in the behaviors of FT and DT because when the tutors have an opportunity to provide the first help message for a step, they are not constrained to provide a message that is more explicit than the message(s) already provided. DT's responses were significantly different from FT's responses for these scenarios, for which FT always provided either the *null* or the *prompt* response. DT included the *teach* response in addition to *null* and *prompt*, and varied its responses according to the likelihood that the student needed explicit help. These differences paid off in terms of the judges' composite ratings. As shown in Table 5.19, DT was rated significantly higher than FT for first-message-opportunity scenarios, $p<.01$, effect size .28. Looking more closely at first-message-opportunity scenarios, DT is rated more highly than FT both for pretest-wrong and for pretest-right scenarios as shown in Table 5.21. For pretest-wrong scenarios, DT's mean composite rating is significantly higher, $p<.01$, with effect size .55. For pretest-right scenarios, DT's mean composite rating is not significantly higher after the Bonferroni correction, $p=.158$.

Table 5.21: FMO scenarios, composite ratings, paired t-tests: FT vs. DT

Comparison	**FT**	**DT**	**df**	**t**	**Sig.**	**Bonferroni Sig.***	**Effect Size**
Pretest wrong	2.51	3.24	74	4.606	p<.001	p<.002	.55
Pretest right	3.53	3.73	112	1.776	p=.079	p=.158	.14

* Significance with Bonferroni correction for the 2 t-tests (Sig. x 2)

5.7 COMPARING ENHANCED VERSIONS OF THE TUTORS: DTe vs. FTe

The Fixed-Policy Tutor's ratings were hurt a great deal by its *null* responses to first errors and step starts, as described in the preceding sections. The question naturally arises as to how much better FT would be rated with a simple change to its fixed policy to not select *null* responses for these types of scenarios (and consequently to never select *null* responses, since these are the only types of scenarios for which FT selects them). Consequently, an enhanced[18] version of FT, FT*e*, was developed which never selects a *null* response. Instead, for each scenario, FTe simply gives the next hint in its hint sequence. For instance, if the student has not yet received any help for the current step, FTe would respond to a step start scenario with a *prompt* response.

It must be emphasized that calling this version of FT "enhanced" is not a claim that FTe is a more effective tutor than FT; rather, "enhanced" refers to the anticipated result that FTe will receive higher ratings than FT from this study's judges. Section 6.6.3.1 discusses this distinction.

To the extent that our judges did not favor *null* responses, DT was also hurt by its *null* responses. Therefore, in order to more fairly compare enhanced fixed-policy tutoring with the current decision-theoretic tutor according to the judges' apparent preferences, DT was likewise modified to never return *null* responses. This modification to DT could have been accomplished in a number of ways. The method used was a minor change to DT's discourse coherence model, the same model used to emulate FT's preference for successively more explicit tutorial responses (described in section 4.5.1.1), to consider *null* tutorial responses incoherent (just as *null* responses to help requests were already considered incoherent). Another alternative would have been to modify DT's utility model of tutorial response type preferences to assign *null* tutorial responses extremely low utility relative to the other tutorial response types. No other aspect of DT was changed, with the result that the modified DT, called DT*e* for the purposes of this study, selected exactly the same tutorial responses as DT except in situations where DT would have selected a *null* response.

[18] Enhanced according to our judges' ratings

Table 5.22: Tutor x Scenario Type, repeated-measures ANOVA: FTe vs. DTe

Source	df	F	p
	Between subjects		
Scenario Type (S)	2	.554	.575
	Within subjects		
Tutor (T)	1	2.939	.087
S x T	4	1.291	.276

Table 5.22 summarizes the results of a repeated-measures ANOVA comparing FTe and DTe with composite judges' ratings as the dependent variable, tutor (FTe and DTe) as the within-subjects variable, and scenario type (help request, error or step start) as the between-subjects variable. As the table shows, the differences in composite tutor ratings are only marginal, p=.087, and the other differences are insignificant.

Table 5.23 lists composite ratings for FTe and DTe and repeats the composite ratings of RT, FT and DT for comparison purposes, including ratings for each scenario type (help requests, errors, step starts and first message opportunities) and for all scenarios. Figure 5.8 shows the same information graphically. Ratings for FTe and DTe are significantly higher than the ratings for FT and DT except for help request ratings, which are unchanged (FTe and DTe respond to help requests the same as FT and DT, respectively). Among the enhanced tutors, the Fixed-Policy Tutor (FTe) is much closer to DT Tutor (DTe) than FT was to DT. DTe holds a slight advantage over FTe both overall and for all scenario subsets except step starts, for which FTe's mean rating is .01 larger than DTe's. Tests of significance will be discussed below.

Table 5.23: Tutor x Scenario Type, composite ratings: RT vs. FT vs. DT vs. FTe vs. DTe

	Tutor				
Scenario Subset	**RT**	**FT**	**DT**	**FTe**	**DTe**
Help Requests	3.23	3.59	3.66	3.59	3.66
Errors	2.31	2.10	2.95	3.54	3.72
Step Starts	3.11	3.19	3.55	3.77	3.76
First Message Opportunities	2.99	3.12	3.54	3.74	3.77
All Scenarios	2.94	3.08	3.43	3.62	3.70

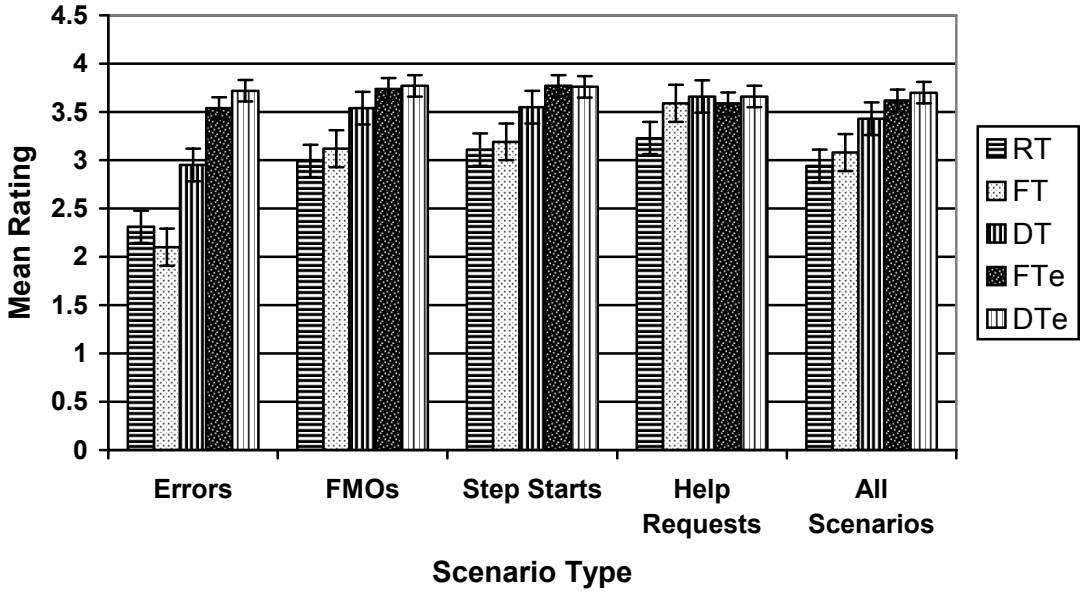


Figure 5.8: Tutor x Scenario Type, composite ratings: RT vs. FT vs. DT vs. FTe vs. DTe

Table 5.24 displays results of paired-sample t-tests comparing FTe vs. DT and FTe vs. DTe for all scenarios and for subsets of scenarios, along with effect sizes and mean composite ratings. Effect sizes were calculated as the difference in means divided by the standard deviation of FTe's ratings. The first set of rows shows that FTe's ratings are nominally higher than DT's ratings both overall and for all scenario subsets except for help requests, with effect sizes ranging from .17 to .53 (the negative signs in the table simply indicate that FTe's mean was subtracted from DT's mean). For help requests, FTe is

unchanged from FT and so DT maintains its insignificant advantage. Using paired-sample t-tests, the differences in ratings are marginally significant overall and significant for errors, $p<.01$, even with the Bonferroni correction for multiple t-tests, which is a conservative correction to protect against Type I errors (Corston & Colman, 2003).

The second set of rows shows that FTe's and DTe's ratings are almost equivalent, with small effect sizes ranging from -.02 to .16. The difference between FTe and DTe in means for all scenarios is marginally significant, $p=.055$, and the difference in means for errors is significant at $p=.026$. However, taking into account the Bonferroni correction for multiple t-tests, these differences evaporate.

Table 5.24: Tutor x Scenario Type, composite ratings, t-tests: FTe vs. DT, FTe vs. DTe

Comparison			df	t	Sig.	Bonferroni Sig.*	Effect Size
FTe vs. DT	**FTe Mean**	**DT Mean**					
All Scenarios	3.62	3.43	349	2.755	.006	.060	-.17
Help Requests	3.59	3.66	174	1.078	.282	1.000	.06
Errors	3.54	2.95	99	4.049	<.001	<.01	-.53
Step Starts	3.77	3.55	74	1.269	.209	1.000	-.32
FMOs	3.74	3.54	187	2.060	.041	.410	-.25
FTe vs. DTe	**FTe Mean**	**DTe Mean**					
All Scenarios	3.62	3.70	349	1.924	.055	.550	.08
Help Requests	3.59	3.66	174	1.078	.282	1.000	.06
Errors	3.54	3.72	99	2.261	.026	.260	.16
Step Starts	3.77	3.76	74	.159	.874	1.000	-.02
FMOs	3.74	3.77	187	.414	.679	1.000	.04

* Significance with Bonferroni correction for the 10 t-tests (Sig. x 10)

5.7.1 DTe vs. FTe: first-message-opportunity scenarios

DTe's and FTe's response selections and performance were compared on first-message-opportunity scenarios partitioned into pretest-wrong and pretest-right subsets (see section 5.4.3 for background about this method). For these scenarios, FTe always selected the *prompt* response. DTe's response selections, on the other hand, varied, and the difference between the two tutors' response selections was significant:

$\chi^2(1)=117.1$, $p<.001$. For the 75 pretest-wrong scenarios, DTe selected the *teach* response 60% of the time and the *prompt* response 40% of the time. For the pretest-right scenarios, DTe's tendencies were reversed: it selected *prompt* 61% of the time and *teach* 39% of the time. This difference among DTe's selections was also significant: $\chi^2(1)=8.02$, $p=.005$.

The differences in the two tutors' response selections affected the ratings. For pretest-wrong scenarios, DTe's mean composite rating of 3.84 was significantly higher than FTe's mean composite rating of 3.60, $p=.035$, effect size = .29. For the pretest-right scenarios, FTe's mean composite rating of 3.82 was higher than DTe's rating of 3.73, effect size .13, but this difference was not quite significant, $p=.123$.

This tendency with first-message-opportunity scenarios was previously observed for help requests in section 5.6.2.1 when comparing FT and DT (which selected the same responses as FTe and DTe, respectively, for help requests). A similar pattern occurred for first-message-opportunity errors: FTe always selected *prompt*. For pretest-wrong error scenarios, DTe selected *teach* 64% of the time and its mean composite rating of 3.64 was marginally higher than FTe's rating of 3.32, $p=.09$, effect size .38. For pretest-right error scenarios, DTe selected *prompt* 62% of the time and its mean composite rating of 3.65 appeared to be slightly higher than FTe's rating of 3.58, effect size .10, but this difference was not significant, $p=.603$. The difference among DTe's selections for pretest-wrong versus pretest-right scenarios was marginally significant, $\chi^2(1)=3.02$, $p=.082$.

6.0 DISCUSSION

6.1 LEARNING PROBABILITIES

Learning probabilities empirically was of lesser priority in the current study, for which the primary focus was a comparative assessment of DT Tutor's (DT's) tutorial action selection capabilities. Still, prior and conditional probabilities are a fundamental any probabilistic network and a core influence on DT's behavior. The data collection phase of this study was designed specifically to facilitate learning probabilities empirically (in addition to providing data for the assessment phase). The data collected for learning included (1) pretests and posttests covering every essential domain rule, and (2) 9,872 scenarios of student-tutor interactions, of which 5,287 were help events. Only about half of the data collected from 60 students was used for training (from the 30 students in the training set) in order to reserve some unseen data for the assessment phase. The bulk of DT's probabilities were learned empirically, including all of the key probabilities regarding student knowledge, the effects of the tutor's actions on student rule knowledge and problem-solving progress, the student's help style, and student performance.

6.1.1 Techniques for learning probabilities

Only basic techniques were used for learning DT's probabilities. The probability for each outcome of a distribution was simply calculated as the ratio of events with that outcome to the total number of like events. Probability distributions were calculated independently, which, while taking into account direct conditional dependence between variables (as indicated by arcs from one node to another in the network structure), did not take into account indirect dependence between variables. No prior knowledge of the probabilities was assumed[19], which is equivalent to using the uniform density function (all values equally likely) to represent prior beliefs in a Bayesian approach. Indeed, the method used for calculating probabilities empirically followed a strict frequentist approach rather than initializing probabilities based

[19] This can be considered either objectivity or prior ignorance about these values. In fact, some of the learned conditional probabilities were surprising, as discussed in this section.

on subjective beliefs and then updating them as in a Bayesian approach. (However, some of DT's prior and conditional probabilities are still specified subjectively.)

Many more advanced techniques exist. Heckerman (1995) and Neapolitan (2004) provide explanations and overviews. Heckerman (1995, p. 17) folds such methods under the umbrella of probabilistic classification or regression functions, commenting that "… a Bayesian network can be viewed as a collection of probabilistic classification/regression models, organized by conditional-independence relationships." Corbett (2000), in work for model-tracing tutors, learns parameters for Bayesian "knowledge tracing" equations by fitting equation parameters to model the performance of a representative group of students as they work through a curriculum; these parameters are periodically adjusted to fit individual students by means of regression equations. A more traditionally Bayesian approach to learning parameters is to augment the Bayesian network to be learned (this can be extended to decision networks) with nodes to represent the uncertain parameter values to be learned. CAPIT (Mayo & Mitrovic, 2001), one of few decision-theoretic tutors, uses such an approach by Cheng and colleagues (Cheng et al., 1998) to learn parameters for its 2-slice Bayesian networks which have only observable variables. iTutor, the only other implemented decision-theoretic tutor (besides DT) of which the author is aware (Pek, 2003), uses methods similar to DT's for learning probabilities empirically. None of these probabilistic tutors appear to attempt to learn the effectiveness of the various tutorial response types (e.g., *prompt*, *hint*, *teach*, *do*) to use as a partial basis for deciding how to respond to the student.

Learning DT's parameters presents an additional challenge because most of the entities represented by the nodes within its large dynamic decision network change as the student interacts with the tutor[20]. Learning DT's probabilities must also cope with hidden (unobservable) variables (which CAPIT does not have) and a training set with missing samples for some combinations of events. While techniques exist to overcome these hurdles (see, e.g., Heckerman, 1995; Neapolitan, 2004), using advanced learning techniques would have been a relatively large additional undertaking for the current study. Improved learning remains a high priority for future work.

Another version of DT's underlying decision-theoretic engine for selecting tutorial actions, a prototype for Project Listen's Reading Tutor (Murray et al., 2001b), does already employ in a limited fashion the more advanced technique of augmenting the Bayesian network: It includes *Tutor Efficacy*$_s$ subnetworks with separate nodes to model the effectiveness of each tutorial action alternative. The *Tutor Efficacy*$_s$ subnetworks tune the model to the particular student, reducing the need to learn accurate conditional probabilities regarding the effects of tutorial actions on student knowledge. It would be

[20] Even most of the entities represented by nodes with the same name change because they represent the tutorial situation at different points in time. An exception is the *Student Help Style*$_s$ nodes which are currently modeled as static over the course of a tutoring session.

straightforward to add *Tutor Efficacy* networks to DT and thus to begin learning online many of the key probabilities that were learned offline in this study.

6.1.2 Learning about students' rule knowledge in the presence of help abuse

In the current study, learning related to the unobservable variable of students' rule knowledge proved to be one of the biggest challenges. Rule knowledge was estimated based on student performance on the pretest, during tutoring, and on the posttest. For the 23 of 30 students who were not help abusers (help-neutral students), these estimates seemed to provide reasonable results. Estimating rule knowledge for help-abusing students, however, proved to be impracticable because their problem-solving and help-seeking actions did not reliably reflect their knowledge. Instead, other techniques were used: In section 4.4.3.5, the effects of help on help-abusing students' rule knowledge were estimated as a percentage of the effects on help-neutral students' rule knowledge. In section 4.4.3.6, the effects of help on help-abusers' knowledge of problem-solving steps were estimated based on their problem-solving performance without regard to the (unreliable) estimate of their rule knowledge. Thus, the inability to accurately trace help-abusers' rule knowledge resulted in less precise modeling of help abusers.

The basic reason for this imprecision in modeling help abusers' rule knowledge was that their actions during tutoring did not reliably reflect their knowledge. This obfuscation of observable evidence for unobservable variables cannot be cleared up simply by applying more advanced learning techniques. Instead, at least two lines of attack can be pursued.

The first and preferred method is investigating methods to decrease help abuse and other under desirable help-seeking behaviors in the first place (e.g., Aleven et al., 2004; Baker et al., 2004). Within this study, 27 of the 60 students were dissuaded from requesting help as described in section 4.2.1.1, item 4. As described in (Murray & VanLehn, 2005), help-dissuaded students requested help less often (however, recall from section 4.4.1 that not all help abusers requested help excessively), students who requested less help scored higher on the posttest, and help-dissuaded students marginally gained more than their non-dissuaded counterparts. However, 3 of 7 help abusers in the training set were dissuaded from requesting help (one was the student who never requested help but made 132 errors), so more must be done. Anderson and colleagues (1995, p. 198) found that linking progress through the tutor with help-seeking behavior "is an effective way of dealing with hint abusers." This strategy seems to have had at most limited success as evidenced by observations of continuing problems with help misuse (e.g., Aleven & Koedinger, 2000; Aleven et al., 2004). Aleven and colleagues (2004), and Baker and colleagues (e.g., 2004), among others, are actively investigating alternatives, including developing an agent to tutor students about help use.

Failing prevention of help abuse, more accurate assessment of help-abusers' rule knowledge would help to build more accurate models. While help abusers performed relatively poorly during tutoring (judging by their numbers of help requests and errors as shown in Table 4.1), their pretest scores, which assessed their rule knowledge, were nominally higher and statistically equivalent compared to help-neutral students. Therefore, this study's pretest seemed to be a more accurate indicator of help-abusers' rule knowledge than their performance while using the tutor. The reason for the pretest's accuracy was probably that there was no help available for it and students were motivated to do their best in order to be able to continue participating in the study. Designing additional assessment tools with comparable accuracy (which includes arranging student motivations to encourage them to do their best) to be interleaved with, or part and parcel of, tutoring could help to obtain more accurate assessment of help-abusers' rule knowledge and how it is influenced by a computer tutor. In addition, a probabilistic tutor like DT can be initialized with student-specific prior probabilities from more accurate assessment tools. DT used population parameters learned from the training set for the assessment phase of this study, but it could just as easily use student-specific prior probabilities obtained from the pretest.

6.1.3 Learning with sparse data

Despite collecting 9,872 scenarios, including 5,287 help events, during the data collection phase, this study still encountered problems of learning conditional probabilities with sparse data (refer to section 4.4.3.2). Only about half of the data available was used for learning (the training set) in order to reserve data for the assessment phase, so more data is available. Still, an even larger number of events would be needed to have adequate samples to learn more than 448 conditional probabilities using a frequentist approach without prior beliefs (with reasonably accurate prior beliefs, probabilities could be learned with less data), particularly for combinations of events that occurred only rarely or not at all during the data collection phase.

The solution used in this study was to aggregate subsets of similar events to learn probabilities at a coarser grain size. As discussed in section 4.4.3.2, events were combined for each of the 4 sets of similar rules: (1) *select equation form*, (2) *apply operator*, (3) *find equation form*, and (4) *select operator*. All the rules within each set shared a general difficulty level, similar concepts and somewhat similar (template-based) help messages. For rare types of events, such as help events when a *find equation form* rule was unknown, sparse data even for the set of rules made it necessary to compute conditional probabilities by aggregating across all sets of rules the events that satisfied the other criteria for the event type. Such approximations for rare events should not have much effect on DT's performance since the

events are rare by definition and the maximum effect of an incorrect estimate on DT's behavior would only be a change in the type of tutorial response that DT provides (*prompt*, *hint*, *teach* or *do*).

6.1.4 Some surprises in the learned probabilities

There were surprises in the learned probabilities regarding the effectiveness of the various tutorial help types (*prompt*, *hint*, *teach*, *do*). First, help types *prompt* and *hint* seemed to be about equally effective for helping students learn rules (section 4.4.3.5 and Table 4.5) and complete steps (sections 4.4.3.6 and 4.4.3.7, Table 4.6). This is consistent with several comments by judges that the *prompt* and *hint* messages were sometimes similar (section 5.5.1, item 6). The information content of *prompt* and *hint* messages was supposed to be different: *Prompt* messages were supposed to point out relevant information that was already present in the interface but not to provide any new information. *Hint* messages were supposed to offer some information that was not already present in the interface, but not necessarily to point out relevant information that was already present. *Hint* messages were modeled as having a more negative impact on the student's feeling of independence because the student may not feel that she can succeed independently if she needs substantive help from the tutor. Since the effectiveness of *prompt* and *hint* messages turned out to be similar but *hint* had a more negative impact on the student's feeling of independence, DT was more likely to select *prompt* and so it selected only a small number of *hint* responses, as discussed in section 5.4.2.2. Apparently, the *prompt* and *hint* messages should be either more clearly differentiated or merged.

Second, help type *teach* was only about as effective as help types *prompt* and *hint* for helping students to learn rules. *Teach* messages were intended to provide all the information necessary to understand the rule related to the current step and thus to help the student complete the step successfully. At the other extreme, *prompt* messages were not supposed to provide any explicit information about the rule related to the current step. It is possible that the *teach* messages were too long. Anderson and colleagues (1995) advise minimizing presentation of instruction while problem solving and to make help messages "as short and to the point as possible" (p.198). Anecdotal evidence that the *teach* messages may have been too long is that the judges sometimes preferred as a help message just the example(s) from *teach* messages (section 5.5.1, item 5). Perhaps students didn't want to read about rules, preferring instead learning by doing – indeed, help type *do* was particularly effective, as discussed below.

For helping students with problem steps (as opposed to learning rules), help type *teach* was more effective than *prompt* and *hint*, as expected (sections 4.4.3.6 and 4.4.3.7, Table 4.6). Perhaps this is because the *teach* messages were so explicit that students could use them as a recipe instead of understanding the underlying concepts. Indeed, this researcher found no way to provide complete *teach*

messages for the rules of type *select operator* without actually giving the answers away (although the answers were embedded within relatively long messages), as discussed in section 3.3.1.4. For *select operator* rules, *teach* was even more effective than *do* at helping students complete the step (see Table 4.6).

Third, the biggest surprise was that *do* help messages were most effective for helping students to learn rules (section 4.4.3.5 and Table 4.5). *Do* messages were designed to tell the student exactly what to enter to complete the current step but not to provide any information about the related rule. The fact that *do* messages were most effective for learning rules is evidence that students were assembling information that was not available in the interface or in the help message. A likely explanation, consistent with research about how students learn, is that students "self-explained" (Chi et al., 1989), or explained to themselves, the example provided by the current problem situation and its solution as presented in the *do* help message, inferring the missing information. Alternatively, they might have remembered rule information that was presented in the tutorial that they studied before using the tutor. However students assembled the rule information, their process seemed to be more effective for learning and retention than receiving the information in a *teach* help message. The surprising finding that *do* was most effective should be verified in other situations in order to more exactly characterize when and why it is most effective.

These surprises about the effectiveness of the various tutorial help types illustrate the importance of extensively testing a computer tutor with students to learn how its actual effects may differ from the anticipated effects.

6.1.5 Expected patterns in the learned probabilities

Many of the patterns in the learned probabilities were not surprises. Finding anticipated patterns is evidence that even the Random Tutor (which was used to interact with students for learning probabilities) functioned as planned in many respects and that many of the learned probabilities are reasonable. Some examples follow:

1. Help-abusing students were much more likely to click the general *Help!* button (section 4.4.3.8 and Table 4.7) and to request help of all kinds (section 4.4.3.9 and Table 4.8).
2. Help-abusing students were much more likely to click *Cancel* when help was not provided, probably because they did not get proactive help (section 4.4.3.8 and Table 4.7).
3. Help-abusing students were much more likely to know a step result when given bottom-out *do* help than when given other types of help like *teach* (section 4.4.3.6 and Table 4.6).

4. Help-abusing students were much less likely to guess successfully and more likely to slip (section 4.4.3.4 and Table 4.4).
5. Help-neutral students were more likely to make an error than to make a help request (section 4.4.3.9 and Table 4.8).
6. Help-neutral students were much more likely to know a step when they knew the related rule (section 4.4.3.7 and Table 4.6).
7. The probabilities of guessing and slipping were inversely correlated (section 4.4.3.4 and Table 4.4).
8. Some rules were more difficult than others, as assessed on the pretest for prior probabilities (section 4.4.2, Table 4.2, Table 4.3), for students to guess and to apply without slipping (section 4.4.3.4 and Table 4.4), and for students to apply even with the tutor's help (section 4.4.3.7 and Table 4.6).
9. For the easiest rules (the *find equation form* rules), help-neutral students who knew the rule almost never failed on related steps regardless of the tutor's help type (section 4.4.3.7 and Table 4.6).

6.2 TUNING UTILITIES

As discussed in section 4.5, there was no empirically verifiable way to determine DT's utilities for the current study. Therefore, the experimenter performed minor tuning until DT seemed to perform reasonably on several representative scenarios according to the experimenter's subjective preferences.

DT attempts a delicate balance of considerations regarding multiple competing objectives. For the current study, DT considered the tutorial state attributes of discourse coherence and discourse relevance, tutor response preferences, and the student's knowledge, help style, feeling of independence (part of the student's affective state), and problem-solving progress. While there is evidence that human tutors consider multiple tutorial state attributes in deciding how to respond to the student (e.g., Lepper et al., 1993; Merrill et al., 1992), few other computer tutors consider attributes other than student knowledge probabilistically, and none consider so many attributes in combination or use decision theory to do it. Another decision-theoretic tutoring system, CAPIT (Mayo & Mitrovic, 2001), considers only a single

attribute for its utility and so has no need for a multiattribute utility function. The only other implemented decision-theoretic tutoring system, iTutor (Pek, 2003), does consider 3 tutorial state attributes: (1) value of information for student assessment, (2) distance between domain concepts, and (3) whether to end the tutoring session. iTutor, like DT, uses a linearly-additive multiattribute utility function (discussed in section 4.5.2 and below) but it is unclear how iTutor obtains the weights for each subutility.

Defining a suitable multiattribute utility function is essential for DT to consider multiple attributes of the tutorial state. As discussed in section 4.5.2, the additive independence condition for using a linearly-additive multiattribute utility function was technically not satisfied because there were interactions between preferences for different attributes. But a linearly-additive multiattribute utility function was the only available option and according to Clemen (1996, p. 585) such a function may still be suitable for modeling purposes "[i]f minimal interactions exist." With this perspective, and given the available options, the experimenter's utility function was modeled as a linearly-additive multiattribute utility function, $U(x_1,...,x_m) = \sum_1^m w_i U_i(x_i)$, where $U_i(x_i)$ is the utility of attribute x_i and w_i is the weight allotted to attribute x_i. Two tasks were then required for tuning DT's utilities: (1) determining each subutility $U_i(x_i)$, and (2) defining the weights w_i for the linearly-additive multiattribute utility function.

All of DT's subutilities were left untuned except for the utility of tutor response preferences, which was tuned primarily to counteract DT's tendency to select response *do* (which in turn stemmed from the surprising finding while learning probabilities that the *do* response was most effective at getting the student to learn both the step and the related rule). As discussed in section 4.5.2, it would probably be more accurate to model the underlying reasons why the experimenter's response type preferences differed from DT's actual responses (before tuning), but modeling is an approximate task and so tuning the utility of tutor response preferences satisficed for the current study. However, the underlying reasons need to be investigated as part of future work. First, as discussed in section 6.1.4, the surprising finding that *do* was most effective should be verified in other situations in order to more exactly characterize when and why it is most effective. This information will help to model underlying reasons for selecting response type *do* or not. Second, for any situations where the *do* response turns out to be most effective in terms of the student's cognitive state but is still not preferred, the reasons why it is still not preferred need to be further investigated. One likely possibility is influences of the *do* response on the student's affective state such as the student's feeling of independence (because the student may feel like she cannot function independently without the tutor telling her exactly what to do). If this is the case, DT's model of student independence needs to be further refined, probably as part of a more sophisticated model of several aspects of the student's affective state.

Utility for tutor response preferences was also tuned to slightly favor *teach* responses, as described in section 4.5.1.7. One side effect of this seems to be that DT selected the *teach* response more

often than the other responses – probably too often, as discussed in section 5.4.2.1. This was probably an overadjustment in tuning that could be rectified simply by changing the utility for the *teach* response to be the same as the utilities for *prompt* and *hint*.

Section 4.5.2 described the weighting system developed for the weights w_i in the linearly-additive multiattribute utility function. The weights were arranged to both (1) implement a priority system among sets of attributes, and (2) to facilitate a delicate balance among attributes of the same priority. This system seemed to work well for the study's purposes. But there are more possibilities for tuning the weights that deserve mention. First, it is easily possible to turn on or off entire submodels of DT's model of the tutorial state. For instance, DT has in the past (e.g., Murray et al., 2004) incorporated a model of the student's morale in addition to the student's independence as part of its model of the student's affective state. For the current study, the model of morale, which was primitive just like the model of independence, was regarded as insufficiently differentiated from the model of independence to be worth including. So student morale was at first eliminated from DT's consideration by simply giving it a multiattribute utility weight of zero (later, the morale submodel was removed as unnecessary). As another example, DT could easily be made to behave just like the Fixed-Policy Tutor by zeroing out all weights except the weight for discourse coherence, along with small changes to the *Discourse Coherence*$_1$ model to exactly match the Fixed-Policy Tutor's slightly stronger constraints (see sections 4.5.1.1 and 5.2.2.3) – in fact, this capability proves that DT has a superset of the Fixed-Policy Tutor's capabilities. Zeroing out of weights can also be used to test subsets of DT's components in isolation, as was done in (Murray et al., 2004). Finally, and less drastically than zeroing out weights, DT's behavior can easily be adjusted to favor the student's cognitive state over the student's affective state, or the student's rule knowledge over the student's problem-solving progress, etc., simply by changing the relative strengths of the corresponding weights.

Future work includes implementing a more accurate multiattribute utility function, of course, but perhaps more importantly seeing what can be done to further improve the fidelity of the current linear function. The importance of refining DT's supporting models, such as its model of the student's affective state, has already been mentioned. In addition, with completion of the current study, the test data can now be used for tuning DT's utilities to the judges' preferences. In future studies, experiments can be conducted to learn how to tune DT's performance to improve its effectiveness with students.

6.3 RANDOM VS. DECISION-THEORETIC: SUPPORT FOR HYPOTHESIS 1

The judges, who were skilled tutors, clearly rated DT Tutor (DT) higher than the Random Tutor (RT). Based on the composite ratings of the judges, DT's ratings were higher both overall and for the subsets of help requests, errors, and first message opportunities, significant at the level $p<.01$ (see Table 5.19). For one subset, step starts, was DT's mean composite rating was not higher enough to be significant, $p=.120$, although that was using a Bonferroni correction for 10 t-tests (multiplying the paired-sample t-test significance by a factor of 10), which is known to be very conservative to protect against Type I errors (Corston & Colman, 2003). Effect sizes ranged from .30 to .49, meaning that the differences between DT and RT were not just statistically significant but also large enough to make an impact. DT's advantage in ratings was robust among all three judges' individual ratings (e.g., see Table 5.10 and Figure 5.1).

These results support Hypothesis 1: According to ratings by skilled human tutors, tutorial action selections by decision-theoretic methods can be better than selections made randomly among relevant tutorial actions.

6.4 FIXED-POLICY VS. DECISION-THEORETIC TUTORING

6.4.1 Fixed-Policy Tutor vs. Decision-Theoretic Tutor: Support for Hypothesis 2

The judges rated DT Tutor (DT) higher than the Fixed-Policy Tutor (FT) overall and for the scenario subsets of errors, start steps and first-message-opportunity scenarios with substantial effect sizes ranging from .22 to .61, all with significance $p<.02$ or better (see Table 5.19). DT was not rated higher than FT only for help requests – for these, DT's mean composite rating of 3.66 was nominally higher than FT's rating of 3.59 with an effect size of only .06.

With DT significantly surpassing FT both overall and for all major subsets of scenarios other than help requests, and with DT rated nominally but insignificantly higher than FT for help requests, these results support Hypothesis 2: According to ratings by skilled human tutors, tutorial action selections by decision-theoretic methods can be better than selections made by a fixed policy that emulates the fixed policies of theory-based, widely accepted and highly effective computer tutors.

6.4.2 FT vs. DT: Adapting the tutor's response type to the situation

Fixed-policy tutors such as FT use a time-tested and proven, even theoretically-based (e.g., Anderson et al., 1995) policy for selecting the response type for tutorial actions. However, this policy considers very few attributes of the tutorial situation. FT and similar tutors (see, e.g., Anderson et al., 1995) consider only (1) whether the student has just made a help request or the *n*th error, where *n* is the policy's threshold number of errors for providing help, and (2) the most recent response type for the current step (in order to select the response type that is one level more explicit or else to repeat the most explicit response type, *do*). The result is set of response selections that are all the same regardless of other attributes of the tutorial situation such as the student's knowledge (and the associated likelihood that the student needs explicit help), the student's affective state, and whether the student misuses help (e.g., abuse or avoidance). As discussed in section 5.4, FT's response is always *null* for first error and step start scenarios. For first-message-opportunity scenarios, FT's response is always either *null* (for first errors and step starts) or *prompt* (for subsequent errors and help requests). After its first response, FT always follows in lock step its sequence of increasingly explicit help.

FT's simple policy was designed to emulate the policies of model-tracing tutors, which do not volunteer help unless the student appears to be floundering (e.g., making multiple errors in a row) (Anderson et al., 1995). This policy, along with the policy of providing successively more explicit hints, was designed to motivate students to do as much of the work as possible themselves, based on psychological research showing that students remember material better when then they generate it themselves. However, even the architects of the model-tracing tutors and the theory behind them admit that "these may not be the best choices" since, for example, "[s]ome students stubbornly refuse to seek help even when they need it" and "students are often annoyed with the vague initial messages and decide there is no point in using the help facility at all" (Anderson et al., 1995, p. 199). Once students begin clicking past vague initial help messages, as many as 82-89% of students using one model-tracing tutor click all the way through to bottom-out help, which explicitly tells the student exactly what to do (Aleven & Koedinger, 2000).

Human tutors, on the other hand, "are capable of taking a variety of events and conditions into account" (McArthur et al., 1990, p. 231) in deciding when and how to provide help. As Lepper and colleagues (1993, p. 85) observed, human tutors "sometimes seek to forestall errors, sometimes intervene as soon as errors occur; at other times they may allow errors to occur." These decisions involve "complicated tradeoffs about when and how to provide new information and assistance" (Lepper et al., 1993, p. 85) as they "pay simultaneous and continuous attention to the both the cognitive and affective state of the learner" (p. 78). Merrill and colleagues (1992) found that expert human tutors maintain a

"delicate balance" (p. 280) between allowing students freedom and giving them sufficient guidance, and that the "content and timing of feedback appear to depend critically on the consequences of the particular error or impasse encountered" (p. 283). The very effectiveness of tutorial help "may arise because of the contingency of feedback style and content on the nature of the student's error" (Merrill et al., 1995, p. 346). Since human tutors' decisions are tied so closely to the cognitive and affective state of the learner, Lepper and colleagues (Lepper et al., 1993, p. 100) expect that expert tutors will employ different strategies for different students, particularly for students "… who differ widely in their abilities or their motivations."

DT, like human tutors, considers multiple tutorial state attributes to decide when and how to provide help. These attributes include the student's knowledge, affective state, help style, problem-solving progress, and focus of attention. The result is that DT's responses likewise vary based on tutorial state attributes, as described in section 5.4. For first-message-opportunity scenarios, a particularly revealing set of 188-190 of the 350 scenarios (see section 5.4.3), DT's responses varied based on the likelihood that the student would need explicit help (using the metric of whether the student got the corresponding pretest item right or wrong, as discussed in section 5.4.3) both overall and for the scenario subsets of help requests, errors, and step start scenarios. DT provides proactive help for both errors (including first errors) and for step start scenarios, but not always. For step start scenarios, the likelihood that DT will provide proactive help and the help that it provides vary significantly according to how likely the student is to need explicit help. DT's use of decision theory to balance multiple, potentially competing considerations is designed to help it respond reasonably to an unlimited variety of situations – even unanticipated situations – without having to come up with a fixed-policy for every combination of probabilistic beliefs about tutorial state attributes.

DT's variations in responses in accordance with multiple tutorial state attributes paid off in generally higher ratings from the human judges. In addition to significantly higher ratings than FT both overall and for the major subsets of scenarios other than help requests (for which DT's ratings advantage was insignificant), a particularly telling result was for the first-message-opportunity subsets of pretest-wrong and pretest-right: Not only did DT's responses vary significantly for these two subsets, but its ratings were significantly higher for the pretest-wrong subset with a substantial effect size and also marginally higher for the pretest-right subset before the significance was diluted by the conservative Bonferroni correction for multiple t-tests. Even for help requests (for which DT was not rated significantly higher than FT), DT was rated significantly higher for the pretest-wrong subset of first-message-opportunity scenarios with a substantial effect size. For the pretest-right subsets of first-message-opportunity scenarios, FT's performance was generally improved relative to DT's (although FT never quite held a significant advantage), probably because of a better fit between FT's policy of

providing either *null* or *prompt* help (depending on the scenario type) and the fact that pretest-right students were less likely to need explicit help.

The advantages that DT holds in situations that expose the variability of its responses and its sensitivity to the tutorial context are evidence that no single tutorial response, or even fixed series of responses, is best for every student in every tutorial situation.

6.4.3 Examples of judges' preferences for more explicit help than FT would select

For many scenarios, DT's responses were similar to FT's, offering either no help or one of the less explicit help messages when DT thought the student was less likely to need explicit help, and for these scenarios DT and FT usually received similar ratings from the judges. However, for scenarios where DT thought that the student was more likely to need explicit help, DT's more explicit response selections were often quite different from FT's selections, with correspondingly different ratings. The subsections below provide examples of such scenarios for help requests, errors, and step starts.

6.4.3.1 Example of preferences for more explicit help for a help request

In this scenario, the student was working on the problem step of integrating equation $dh/di=6*i^2$ with respect to i, for which the correct entry is $h=2*i^3$ (integration operations neglected the arbitrary constant of integration in order to simplify the problem space for students, as described in section 3.1.1, item 1). The student had gotten the integration item correct on the pretest, and before the current problem step had made 33 correct entries, 14 errors, and 16 help requests. On this step, however, the student had made 3 errors in a row without any help from the tutor, with the successive incorrect entries $h=i^3$, $h=12*i^3$, and $h=18*i^3$. The student had then asked for help and the tutor had responded with a *prompt* message, "Apply *integrate* to $dh/di=6*i^2$, then simplify the resulting equation." The student had then requested more help by clicking "Explain Further." The ratings for this scenario were based on the various tutors' responses to this last help request.

FT, following its policy, selected the next least explicit help, the *hint* message:

1. Use the reverse of the power rule to transform $dh/di=6*i^2$ into a regular equation
2. Then simplify the resulting equation

DT, on the other hand, selected the *teach* message:

> *integrate* transforms a derivative equation into a regular equation using the reverse of the power rule
>
> Example – *operand*: da/db = 12 * b^3 → a = 12/4 * b^4 → *result*: a = 3 * b^4
>
> (1) Change the LHS into the variable from the numerator of the derivative expression
> (2) Add 1 to the exponent
> (3) Divide the RHS by the new exponent
> (4) Simplify the resulting equation

The only other options for help messages, besides the previously provided *prompt* message, were no message at all or the *do* message: "Enter equation h=2*i^3."

The judges unanimously gave FT's response a rating of 3 (the middle rating) and DT's response a rating of 5 (the highest rating). Judge 2, in selecting the *teach* response as best, commented, "It looks like the student doesn't know how to integrate, although he got it right on pretest." Judge 3 commented that "A [teach] is the best, since the student simply need to refresh how to integrate." (Judge 1 made no comment.) Apparently, the judges felt that more explicit help was preferable given the student's recent performance.

6.4.3.2 Example of preferences for more explicit help for an error

In this scenario, the student was working on the problem step of selecting the equation form for *substitute operand 1*. Before the current problem step, the student had made 39 correct entries, 48 errors and 4 help requests. The student had an extensive history related to the rule for required for this problem step. First, the student had gotten the related item wrong on the pretest. Then, on a previous problem for which the correct entry was g=f(h), the student had selected equation form dh/di=f(i) in error, received the proactive prompt "To use substitute to create an equation of form g=f(i), *substitute operand 1* must be in what form?" The student then made three successive errors – dg/dh=f(h), dh/di=f(i) [again] and dh/dg=f(g) – before receiving a proactive *hint* message:

> *substitute operand 1* must be a *regular* equation involving:
> … (1) one variable that is in g=f(i)
> … (2) one variable that is not in g=f(i)

The student then made another incorrect entry, g=f(i), before receiving the *do* help message proactively – "Select equation form g=f(h)" – and entering the correct equation.

On the current step, for which the correct entry was t=f(u), the student started out by making an error, v=f(u). The ratings for this scenario were based on the various tutors' responses to this error.

FT selected the *null* response (no help message) for this, the student's first error on the current step. DT, on the other hand, selected the *teach* help message:

> when *substitute's* operands have the following forms:
> … *operand 1*: <variable 1> = f(<variable 2>)
> … *operand 2*: <variable 2> = f(<variable 3>)
>
> the resulting equation will have this form:
> … *result*: <variable 1> = f(<variable 3>)
>
> Example: To create an equation of the form a=f(c) when the other variable in the Accepted Equations is b, *operand 1*'s equation form must be a=f(b)

Judges 1 through 3 rated FT's response 1, 2, and 1 respectively (a median rating of 1) while rating DT's response 5, 5, and 3 (a median rating of 5). Judge 3, in selecting the *do* help message as the best response, commented, "Neither of the tips helped this student before. Let's try to give the exact answer. D [do]." (Judges 1 and 2 did not comment.) For this scenario, it appears that the judges considered events that occurred prior to the current step (FT considers only events for the current step) to decide to provide explicit help immediately after the student's first error. Judge 3's comment provides direct evidence for this interpretation.

6.4.3.3 Example of preferences for more explicit help for a start step scenario

In this scenario, the student was working on the problem step of selecting an operator, for which the correct entry was *chain rule*. Prior to the current step, the student had made 9 correct entries, 13 errors and 2 help requests. The student had gotten the related pretest item correct. On a previous step for which the correct entry was also *chain rule*, the student had received a proactive *prompt* message when she clicked on the step in the interface's Goals Window: "Select operator(s) that will *efficiently* transform Accepted Equation(s) into an equation of the form dq/ds=f(s)." But the student had made an error by selecting operator *substitute*. The next time the student clicked on the step in the goals window, she received a proactive *teach* message:

When:
(1) at least 2 out of 3 of the first two given equations and *evaluate operand 1*'s equation form are in derivative form
… and …
(2) *evaluate operand 1*'s equation form, dq/ds=f(s) is in derivative form

operator *chain rule* minimizes transforming equations between regular and derivative form

(Note that this *teach* message gives the correct entry away as discussed in section 3.3.1.4.) The student's next entry was correct.

On the current step, for which the correct entry was *chain rule*, the student again received the *prompt* help message when she clicked on the step in the Goals Window: "Select operator(s) that will *efficiently* transform Accepted Equation(s) into an equation of the form dx/dz=f(z)." She then made two consecutive errors, selecting *substitute* and then *differentiate after substitute*. The student's next action was once again to click on the step in the Goals Window to resume working on it. The ratings for this scenario were based on the various tutors' responses to this start step scenario.

FT selected the *null* response, as it always does for start step scenarios. DT selected the *teach* response:

When:
(3) at least 2 out of 3 of the first two given equations and *evaluate operand 1*'s equation form are in derivative form
… and …
(4) *evaluate operand 1*'s equation form, dx/dz=f(z) is in derivative form

operator *chain rule* minimizes transforming equations between regular and derivative form

Judges 1 through 3 rated FT's response 1, 2, and 1 respectively (a median rating of 1) while rating DT's response 5, 3, and 5 (a median rating of 5). Judge 3, in selecting the *teach* help message as best, commented "Neither C [prompt] nor D [hint] worked before, while E [teach] did. So, E [teach]." (Judges 1 and 2 did not comment.) All of the judges' ratings make it clear that they prefer for the tutor to provide help for such step start scenarios. In addition, Judge 3's comment once again provides evidence that the judges (who were skilled tutors) consider aspects of the tutorial state beyond the student's performance on the current step.

6.4.4 FT vs. DT: The role of proactive help

Clearly, a major reason why DT surpassed FT in the judges' ratings was DT's use of proactive help (i.e., non-*null* responses), which FT never uses for step start and first error scenarios. DT does not have a

policy about whether to provide proactive help for such scenarios (DT bases its decisions on underlying factors such as the student's knowledge and affective state), but it did provide proactive help on 62% of the first error scenarios and on 28% of the step start scenarios. The only subset of scenarios for which DT's ratings were not substantially higher than FT's ratings was for help requests, the one major subset for which proactive help does not apply (because the help is by definition reactive to the student's help request).

For first error scenarios (see section 5.6.2.2), the judges rated DT's mixture of responses significantly higher than FT's *null* responses with a large effect size. But for scenarios involving subsequent errors, for which FT always provided proactive help but DT sometimes did not, FT was actually rated higher than DT, this time because of DT's decisions not to provide proactive help.

For start step scenarios (see section 5.6.2.3), DT did not provide proactive help for 72% of the scenarios, and for these scenarios it received the same rating as FT. But DT's proactive help for 28% of the step start scenarios was rated enough higher than FT's *null* responses that DT was rated significantly higher for step start scenarios overall.

6.4.4.1 Effects of enhancing a fixed policy

The considerable impact of proactive help on the ratings inspired a test to see if a simple change to FT's policy would enable it to perform as well as DT according to the judges' ratings. An enhanced[21] FT, FTe, was created with a policy identical to FT's except it always provided proactive help. Like FT, when FTe did provide help (proactive or reactive), it selected its responses in order of successive explicitness.

FTe was rated at least nominally higher than DT except for help requests, for which FTe was unchanged from FT because help requests do not involve proactive help. Even after the conservative Bonferroni correction for multiple t-tests, FTe's overall rating was marginally significantly higher than DT's and FTe's rating for errors was significantly higher than DT's, with substantial effect sizes. This comparison showed that the judges preferred even more proactive help than DT was giving.

The change to FT to create FTe was created post hoc, after observations of the judges' ratings made it clear that such a change would probably lead to an improvement over FT's ratings. Since the post hoc change led to a large improvement, it seemed that the only way to fairly compare the improved fixed-policy tutor to an equivalent decision-theoretic tutor would be to give DT the benefit of the same change. DT's design gives it a superset of FT capabilities, as discussed in sections 6.2. Changes to DT are usually made subtly since it chooses responses based on the underlying reasons for choosing one response over

[21] Again, enhanced according to the judges' ratings

another such as effects on the student's knowledge and affective state. But subtle changes to DT's model of underlying tutorial state attributes would seem to invite post hoc changes that could go beyond the simple change made to FT's policy. So instead, a simple change was made to DT's discourse coherence model so that DT would always provide proactive help but otherwise would behave exactly as it had before.

The ratings for the resulting "enhanced" DT, DTe, appear to be slightly higher than the ratings for FTe both overall and for the major scenario subsets except step start scenarios, for which DTe's ratings were .01 less. An ANOVA showed a marginal difference between the tutors. According to paired-sample t-tests, DTe's ratings were marginally higher overall and significantly higher for errors, but the conservative Bonferroni correction diluted the differences to insignificance and effect sizes were small. According to these results, DTe's advantage over FTe is small if it exists at all.

However, a closer look at first-message-opportunity scenarios (see section 5.4.3 for background) revealed some advantages in the variability of DTe's responses. For all first-message-opportunity scenarios, FTe selected response *prompt*, the least explicit of the non-*null* help types. DTe's response distribution was significantly different from FTe's and it varied significantly between the pretest-right and pretest-wrong subsets. For pretest-wrong scenarios, DTe's ratings were marginally higher than FTe's with a healthy effect size. For first-message-opportunity errors, DTe's ratings for pretest-wrong scenarios were marginally higher than FTe's and its ratings for pretest-right scenarios were nominally higher but not significant.

Comparisons with FTe show that a fixed-policy selected after seeing the test data (or at least after seeing a representative sample of training data) can do just about as well at selecting tutorial responses as decision-theoretic methods, at least at DT's current stage of development. FT's current policy can easily be enhanced (at least according to the judges' ratings) to provide proactive help and then be competitive with DT's and DTe's performance. FT's and FTe's policies remain relatively simple, although FT is representative of the policies of many highly effective model-tracing tutors, and so they still significantly lag DT and DTe in their sensitivity to the context of the multi-attribute tutorial state. Theoretically, a fixed policy can be used to implement policies of arbitrary complexity – at the extreme, consisting of a table lookup of what to do in each unique situation (although policies about real-numbered attributes like probabilities would have to be discretized). Of course, anything that can be implemented as a fixed policy can also be implemented in DT, since DT's action selection capabilities are a superset of FT's. And DT is still at a relatively early stage of development as well (see section 6.2 about tuning DT), so its capabilities can also be further improved.

6.4.5 Should you choose fixed-policy or decision-theoretic tutoring?

The bottom line in choosing a method for making tutorial decisions is bang for the buck: which technology delivers the desired capabilities for the least development and maintenance costs (in time and money). These are software engineering issues that must be quantified for a sufficient resolution, and comparing time and costs are not the focus of this thesis. Still, several person-months spent developing DT and a day or two spent developing FT have qualified the author to make a few comments.

Clearly, if the desired behavior of the tutor is unambiguously defined, only simple capabilities are required, and only simple changes to the tutor's behavior are anticipated over the life of the tutor, fixed policy is best. A simple fixed policy is more predictable in its behavior, easier to implement, and easier to make simple changes to. It's not even close.

However, as the desired behavior of the tutor becomes more ambiguous, the required capabilities increase, or as the need for flexibility in the tutor's behavior increases, decision-theoretic tutoring becomes more attractive. In situations where it is not clear what the tutor's behavior should be – e.g., when there is a conflict between satisfying the cognitive and affective needs of the student (del Soldato & du Boulay, 1995) – decision theory can be used as a principled way to balance competing objectives by unifying considerations regarding both the probabilities and the utilities of the possible outcomes.

For similar reasons, decision theory is useful when the tutor must be capable of balancing multiple objectives, such as goals regarding the student's knowledge, affective state, and task progress. Human tutors must balance competing objectives (Merrill et al., 1992) and this is natural for a decision-theoretic system. An equivalent (rule-based) fixed-policy tutor requires either a complex set of "non-trivial" rule antecedents or a "quite sophisticated" conflict resolution algorithm (McArthur et al., 1990, p. 232) as the number of combinations of conditions and objectives to consider grows exponentially with each tutorial state attribute modeled.

If the factors or priorities influencing complex tutorial behaviors change over time, it can be easier to change the complex behavior of a decision-theoretic tutor. For example, DT's behavior can easily be entirely changed simply by modifying one or more weights for the utilities that correspond to components of its model of the tutorial state: At the extremes, DT will ignore tutorial attributes if their corresponding utilities are zeroed out, or focus on just one attribute if its utility is much larger than the others. In between, DT will emphasize one attribute (e.g., student domain knowledge) at the expense of another (e.g., task progress such as solving problems) if corresponding changes are made to the attributes' relative weights.

At a more detailed level, a decision-theoretic tutor's behavior in specific situations can be more difficult to predict and control because of the multitude of parameters (e.g., prior and conditional

probabilities, utilities) that determine its behavior. However, DT's capabilities are a superset of FT's, as described in section 6.2. If the tutor will face specific situations for which the desired responses are known in advance, DT can be configured to provide the desired responses in those situations, just as DTe was created by configuring DT to always provide proactive help (see section 6.4.4.1).

For fixed-policy systems, which are usually rule-based, it is usually easy to predict and control behavior when it is controlled by a small set of rules. But rule-based systems can become unwieldy and hard to maintain as the number of conditions (e.g., the number and detail of the attributes in the tutorial state representation) and the number of outcomes (e.g., desired tutorial behaviors) increase. Rules for a tutor designed to approach the complexity of human tutoring can become quite complex:

> … the antecedent conditions of those "if-then" rules are often nontrivial. Rather than associating some relatively simple event with a fixed response, factors such as K goals for the student, inferences about the student's knowledge, overall pedagogical policy, and local history of events appear to modulate the selection of techniques in ways we have only begun to clarify. (McArthur et al., 1990, p. 232)

Finally, it must be noted that cost/benefit tradeoffs change over time as new tools are developed which can lighten the effort of building one kind of tutor or another – tools such as shell systems, graphical network development environments, and tutor development kits – and as the state of the art advances, leading to increased expectations for the capabilities of computer tutors.

6.5 SHOULD COMPUTER TUTORS PROVIDE PROACTIVE HELP?

An inescapable conclusion from the analysis of the judges' ratings is that they generally preferred providing proactive help to not providing it, both for errors (including first errors) and for step start scenarios. Section 6.4.4 discussed the large role of proactive help in DT Tutor's (DT's) ratings advantage over the Fixed-Policy Tutor (FT), and section 6.4.4.1 described large gains in ratings when FT and DT were modified to always provide proactive help.

Not only did this study's judges (who were skilled tutors) indicate that they preferred providing proactive help, but other studies have found that skilled and expert human tutors actually do proactively help students, at least in subtle ways. For instance, McArthur and colleagues (1990) observed that their expert tutors appeared to devote as much effort towards structuring problem-solving tasks for students as they did to critiquing weaknesses in the students' performances. Their tutors seemed to make a priority of structuring problem-solving tasks for students so that they were neither too difficult nor too easy, which in many cases minimized the students' errors. Lepper and colleagues (1993) found that their expert tutors

sometimes endeavored to prevent students from making errors and sometimes intervened as soon as errors occurred. Their tutors also sometimes forewarned their students about the difficulty of upcoming problems, even in cases where the problems weren't actually any more difficult, apparently in order to inoculate their students from the negative affective consequences of failure and to set up opportunities for their students to feel a sense of achievement. Fox (1993, p. 61) noticed that her skilled tutors were more likely than Galdes' tutors (1990) "to step in with some form of assistance" rather than sit back and wait for students to ask for help. Fox observed her tutors asking questions before students got stuck, often to frame the problem and the solution. Fox likened her tutors' framing to *scaffolding* (Vygotsky, 1978), whereby the teacher structures the task so that it is always within the learner's grasp and then gradually fades the assistance away as the learner's abilities increase. Thus, skilled and expert human tutors often help their students – quite subtly in many cases – in order to both (1) minimize student errors, floundering, and the negative affective consequences of failure, and (2) increase the likelihood of student success and the associated positive affective consequences.

The computer tutors in this study, with their limited help response types of *null*, *prompt*, *hint*, *teach*, and *do*, obviously can't match the subtlety of expert tutors. But they can use even their limited repertoire to help prevent student errors and impasses, and conversely increase the likelihood of student successes, along with the associated affective consequences. Our judges preferred that they do. This study was not designed to measure differences in learning depending on whether proactive help is provided, and so it cannot say definitively whether computer tutors should provide it. However, human tutors remain the gold standard for teaching effectiveness and so the study of their actions is widely used to inform the design of computer tutors (e.g., Fox, 1993; Lepper et al., 1993; McArthur et al., 1990; Merrill et al., 1992; Putnam, 1987). For the same reason, their opinions (e.g., the ratings and comments of the judges in this study) carry some weight, particularly when they coincide with actions observed in human-human tutoring. Together, the actions and judgments of skilled and expert human tutors strongly suggest that proactive help by computer tutors may be fertile ground for filling the gap between computer and human tutors, with a potential for cognitive and affective benefits with the students they teach.

But if computer tutors should try providing more proactive help, when should they provide it? Despite the preference of this study's judges for proactive help at most opportunities, both the actions of expert tutors and educational theory suggest that tutors should not provide proactive help all the time. Lepper and colleagues (1993, p. 85) found that expert tutors "sometimes seek to forestall errors, sometimes intervene as soon as errors occur; at other times they may allow errors to occur." Their tutors' decisions apparently involved "complicated tradeoffs about when and how to provide new information and assistance" (p. 85). Merrill and colleagues (1992, p. 283) observed that for their expert tutors the timing of feedback appeared "to depend critically on the consequences of the particular error or impasse

encountered," and that the very effectiveness of tutorial help "may arise because of the contingency of feedback style and content on the nature of the student's error" (1995, p. 346). When proactive help is always provided, there is no opportunity to fade the scaffolding (Vygotsky, 1978) and thereby to spur students to become independent practitioners of the skills they are learning.

Therefore, proactive help is more likely to be effective to the extent that it correctly anticipates the cognitive and affective consequences of providing or not providing the help. Obviously, proactive help is unnecessary when a student could with some effort complete a step on her own, at which time it may thwart a chance for learning and for the positive affective consequences of independent achievement. Conversely, proactive help when a student would otherwise flounder can save time, provide valuable information at a time when the student is prepared and motivated to learn it, and prevent the negative affective consequences of frustration and failure. DT Tutor attempts to look ahead for just such purposes: to anticipate the effects of its actions on the tutorial situation, including the student's cognitive and affective states, and to select the tutorial action (including the *null* action) that it probabilistically expects will have the most utility.

6.6 LIMITATIONS AND FUTURE WORK

Limitations fall into three basic categories: the limitations of decision-theoretic approaches, the limitations of DT Tutor (DT) in particular, and the limitations of the current study. These are discussed in separate subsections below in increasing levels of detail corresponding to their relevance to the current study, along with plans for future work to overcome many of these limitations.

6.6.1 Limitations of decision-theoretic approaches

A common criticism of decision-theoretic approaches, or more generally probabilistic approaches, is that they require specification of too many numeric parameters. A variety of methods have been proposed to reduce the number of parameters required, from canonical distributions for conditional probability tables such as noisy-OR and noisy-AND (e.g., Henrion, 1989; Pearl, 1988) that require at most a few parameters, to purely qualitative networks (Wellman, 1990). DT breaks no new ground in this respect, but it does employ rule-based construction of conditional probability tables (see section 3.2.12) to facilitate automatic creation of thousands of conditional probability table entries from a much smaller

number of parameters. Efficiency in specifying DT's parameters may be further improved by, for instance, employing canonical conditional probability tables.

A second common criticism is that probabilistic approaches require too much computational overhead. While all modern probabilistic approaches leverage conditional independence relationships and other structure in the probability space for efficient inference, either exact or approximate, probabilistic approaches still generally require much more computation than, say, a typical fixed-policy approach. This is becoming less of a problem as computational hardware and software continue to improve. The version of DT used in this study seemed to be able to consistently provide sub-second response time as described in section 5.3.

6.6.2 Limitations of DT Tutor

A major reason why DT is able to provide sub-second response time, as described in section 5.3, is that its problem solver and user interface together constrained the number of possible next steps that the student could work on to at most two, sharply reducing the number of decision alternatives faced by the previous version of DT, which considered the possibility of tutoring on each step in the problem solution space – even steps that had already been completed. The previous version of DT may have been under-constrained since students and tutors rarely revisit steps that have already been completed[22]. However, with the current version of DT students occasionally expressed a desire to take problem-solving shortcuts that were not permitted, so it is probably over-constrained. Allowing more flexible problem solutions while still tracking the student's focus of attention probabilistically and maintaining acceptable response time are important topics for future research.

A related limitation, similarly important for future work, is supporting student help requests and topics that aren't strictly related to the next problem-solving step. For instance, a student might want to inquire directly about any rule in the domain rather than limiting queries to topics related to the currently possible problem-solving steps and their related rules.

In addition to supporting a wider variety of student queries, DT's repertoire of actions should be extended to query students. Such a capability could be used, for instance, in tandem with allowing additional flexibility in problem solutions (discussed above) to control the potential explosion in possibilities for the next step by asking disambiguating questions when DT is unsure about the student's focus of attention or problem solution plans. Decision-theoretic approaches like DT's support value of

[22] A prominent exception is post-problem reflection (e.g., Katz & Lesgold, 1994), especially for domains where performance is too time-critical to be interrupted for tutoring until afterwards

information computations (Howard, 1966) to figure out the most what information would be most valuable to obtain from the student.

For efficiency reasons, DT dynamically modifies the arcs and decision alternatives considered in its *Tutor Action Cycle Networks* to correspond to the currently possible tutor and student actions in the current problem solution state, as described in section 3.2. This seems to work well in practice and to be reasonable since, due to the interface and problem solution constraints discussed above, at any point in time, some student actions are simply not possible and so there is no need for the tutor to consider helping on those actions either. But since these dynamic modifications depend on the problem solution state, DT can lookahead only as far as the student's next action because it must wait to observe that action before it can create network slices further in the future. This constraint prevents DT from predicting and planning for multiple student-tutor interactions into the future. Lifting this constraint could permit DT to plan more sophisticated tutorial strategies dynamically.

Another important limitation of DT at present is its modeling of the student's affective state. While DT's tutorial action selection engine implemented a relatively early approach for modeling the student's affective state (Murray & VanLehn, 2000) and it was the first to do so decision-theoretically, this has not been the main thrust of the research to date. The version of DT used for this study models only the student's feeling of independence, and in a relatively primitive way. Previous versions of DT (e.g., Murray & VanLehn, 2000; Murray et al., 2004) also modeled the student's morale, although in a similarly primitive way. By now, much more detailed models of the student's affective state exist, as discussed in section 2.2.4.3. However, most of the models implemented so far neither model uncertainty about the user's affective state nor satisfactorily resolve what the tutor should do when there is a conflict between the best tutorial action based on affective outcomes and the best tutorial action based on cognitive or other outcomes. Conati (e.g., 2002), like DT, uses a dynamic decision network solution for these issues, along with a much more detailed model of the user's affective state, although the models of the user's cognitive state that have been published so far seem to be underspecified. Even with a simple model of the student's affective state as just one of several outcomes considered, DT is able to move beyond simply presenting a kinder, gentler or more entertaining interface to adapting its tutoring based on the anticipated effects of its tutorial actions on the affective and cognitive state of the student, just as expert human tutors appear to do (Lepper et al., 1993). Thus, DT is able to select actions with subtler affective impact, such as proactive help before the student experiences failure. Within DT's framework, models related to various attributes can vary in richness and detail depending on the needs and capabilities of the application. At least for human tutors, the student's affective state can be as least as important a consideration as the student's cognitive state (Lepper et al., 1993), so improving DT's model of the student's affective state will be important work towards improving its capabilities as a tutor.

6.6.3 Limitations of the current study

6.6.3.1 The method of comparing the tutors

Probably the most important limitation of the current study is that its design did not provide solid proof about the most important bottom line: effectiveness at helping students learn. The traditional method of comparing tutors is to compare student gains (e.g., pretest to posttest) when using the tutors to be compared. Let's call this method *gains comparison*. The advantage of gains comparison is that it provides solid information about how much students learn. Let's call this study's method *identical scenarios comparison* because it involved comparing the actions selected by various tutors for identical scenarios.

For identical scenarios comparison, there is some reason to believe that being favored by the judges is not a guarantee that a tutoring approach is most effective at helping students learn. One weakness was that sometimes it placed FT in situations (scenarios) in which it would not have placed itself. The scenarios were created by the interaction of students and the Random Tutor, which selected randomly from the response types. Care was taken to select for evaluation only scenarios whose relevant-action-histories did not violate FT's constraint of never giving a less explicit response after a more explicit response, as discussed in section 5.2.2.2. This was so that FT's policy would still seem rational in context (e.g., FT did not have to decide what response to select after a student had been given a *teach* response followed by a *prompt* response). DT likewise abided by the explicitness constraint, as discussed in section 4.5.1.1, so neither tutor was placed in irrational situations. Still, FT sometimes did have to decide, for instance, what response to provide after the student was provided with an initial *hint* response even though FT would have provided an initial *prompt* response. Even so, evidence from the judges' ratings of first message opportunities indicates that, to do well in the ratings, FT must respond well after an initial help message that it did not provide. The reason is that the judges rated FT's selections for first-message-opportunity scenarios lower than DT's, as discussed in section 5.6.2.4. Therefore, for subsequent responses, the judges must have preferred a tutor that would continue well after the initial responses that they preferred – i.e., after initial responses that FT would not provide.

More evidence that the judges' ratings may not have always favored the most effective tutorial actions comes from their ratings for certain types of scenarios. One judge rated *null* responses to help requests highly, as discussed in section 5.5.3.2, and another judge occasionally concurred in commenting that no help would be the best response for a particular help request (see the judges' comments in section 5.5.1). No response to a help request would probably be bad user interface design since without any response (i.e., with a *null* response) the student might think the tutor has crashed. But it is likely that,

rather than no response at all, what the judges had in mind was some sort of response that either explicitly or implicitly refused to provide help for a student who abuses help. For example, the judge might prefer to provide some sort of meta-help message (see, e.g., Aleven et al., 2004) like "Why don't you try this one on your own before asking for help?" Anomalies in any single judge's ratings were effectively handled by using a median composite rating (section 5.5.3.3).

Another questionable set of ratings could not be handled as an anomaly because of general unanimity among the judges: a preference to provide proactive help at most opportunities, both for error scenarios (section 5.6.2.2) and for step start scenarios (section 5.6.2.3). Whether computer tutors should provide proactive help in such situations remains controversial. FT's policy for proactive help follows the policies of most model-tracing tutors (Anderson et al., 1995), which are theory-based, widely-accepted and highly effective. Most model-tracing tutors do not provide proactive help the first time the student makes an error and never provide proactive help for step start scenarios. Indeed, very few if any ITSs other than DT provide proactive help for step start scenarios. But the decision to endow DT with the capability to provide proactive help is based on the fact that human tutors do sometimes provide proactive help for such situations (e.g., Lepper et al., 1993; Merrill et al., 1995), so it may well be that more computer tutors should provide proactive help – this is discussed in section 6.5.

The issue underlying whether to rely on the judges' assessments is that even if human experts unanimously believe in some tutoring practice, that is not proof that the practice leads to the most learning. For instance, social and psychological factors may favor tutorial actions that make students more comfortable or happy but not necessarily bettered tutored, at least in the short term. Indeed, Graesser and colleagues (1995), in their observations of naturalistic tutoring, noted that tutors' "politeness goals are sometimes incompatible with cognitive pedagogical goals" (p.516), and Lepper and colleagues (1990) found that their tutors used indirect methods "despite their belief that more directive methods may sometimes have clear instructional benefits" (p.234). But attention to the student's affective state may pay off in ways such as student interest and persistence which could lead to greater long term gains, both during the course of the tutoring session and in the future (Aist et al., 2002; Lepper et al., 1990; Lepper et al., 1993). Human tutoring remains the gold standard for educational effectiveness (Bloom, 1984) and so the practices of human tutors are widely researched to inform the design of computer tutors (Fox, 1993; Lepper et al., 1993; McArthur et al., 1990; Merrill et al., 1992; Putnam, 1987).

Finally, for identical scenarios comparison, there is the issue of whether the judges' stated preferences (in terms of their ratings of the scenario responses) reflect their actual tutoring practices. Human beliefs and opinions are notoriously susceptible to inaccuracies and to environmental influences such as framing effects (e.g., Tversky & Kahneman, 1974). There is no getting around this issue; it is a potential problem for any study that seeks to benefit from the nuanced and multi-dimensional yet

potentially faulty considerations reflected in human opinion. The best this study could do was to present the scenarios as accurately as possible and to avoid biasing the tutors in any way.

The scenario descriptions presented to the judges (described in section 5.2.2.1 with a sample in Appendix F) provided a good deal of information about the student's general performance (e.g., number of correct entries, errors and help requests), the student's performance on any previous steps that used the same rule, and about the current tutorial situation. However, the scenario descriptions do not purport to provide whatever information the judges might have considered had they been actually tutoring the students (the information that human tutors consider remains an open research issue, and likely varies by tutor), although they included details that human tutors may not usually track, such as the exact number of previous help requests.

Bias for or against specific tutors was avoided by having the judges rate all possible tutorial responses for each scenario, which were ordered randomly in the scenario descriptions, without telling the judges that ratings for different tutors would be compared. However, it is possible that something about the experimental setup or the presentation of the various possible responses biased the judges to prefer some responses to others.

On the other hand, gains comparison has weaknesses too. By itself, it does not provide detailed information about the advantages or disadvantages of the actions by which the tutors being compared earn their learning gains. For example, Approach A may garner an advantage by rarely providing proactive help, discouraging some students from relying on help that they don't really need – i.e., from abusing help – and thereby eliciting more learning. This strategy might not work as well with students who don't ask for help even when they need it – i.e., students who avoid using help – but as long as there are more help abusers than help avoiders in the student population, Approach A will garner a net gain for the student population. Comparing net gains reveals only the net effect and not the means by which those gains were made. To gather more detailed information, gains comparison could be supplemented with other methods – e.g., detailed log analysis – to try to unravel the micro-effects that sum up to the observed macro-effects in student gains. But even in combination with supplemental methods, gains comparison can rarely compare the behavior of different tutoring approaches in identical tutorial situations with real students. There are just too many combinations of student characteristics and student-tutor interactions for identical tutorial situations to occur naturally more than rarely, especially when different tutoring approaches (which presumably behave differently) are being compared. Identical scenarios comparison is designed to make just such comparisons.

Identical scenarios comparison can also be used to gather detailed information about the effects of tutorial actions for use in adjusting a tutor's behavior. In this study, this information was gathered when students used the Random version of the tutor (RT) during the tutorial data collection phase. RT selected

tutorial actions randomly both to provide a control condition and to support measuring the effects of *individual* tutorial actions while controlling for the accumulated effects of *sequences* of tutorial actions by randomizing over the sequences in which the individual tutorial actions occurred. As described in section 4.4, the effectiveness of tutorial actions was estimated from log data by observing each training set student's immediate and longer-term success after receiving tutorial help. For DT, this data was used to determine conditional probabilities that model the effects of the tutorial action alternatives. Tuning DT's tutorial model also tunes DT's behavior since DT uses this model to decide which tutorial alternative to select. Alternatively, effectiveness data could be used to alter the policy of a fixed-policy tutor to favor the most effective tutorial actions.

In the assessment phase of this study, information was gathered about the judges' preferences among the tutorial action alternatives in the test set scenarios. Their numerically expressed preferences were used to comparatively assess the tutorial action selections of DT, RT and the Fixed-Policy Tutor (FT), both overall and for many types of situations, including help requests, errors, step starts, and first-message-opportunities of various types. For the sake of fairness in the comparative assessment, these preferences were not used to tune the behavior of any of the tutors. Since the assessment, however, the preferences have already been the source of inspiration for developing enhanced versions of FT and DT (FTe and DTe, respectively), described in section 5.7, to test the effects of a simple change to FT's policy intended to improve its ratings.

Besides numeric ratings, the judges made comments (see section 5.5.1) that may provide a rich source of information for future research and development. Among the interesting points made were (1) they often do not follow the constraint to select successively more explicit help, (2) they don't mind repeating a help message, (3) they explicitly consider what help did or not work previously when deciding what kind of help to provide next, and (4) no help in response to a help request is okay.

While identical scenarios comparison cannot definitively determine which tutoring approach leads to more student gains, it can be used to compare the response selections of different tutoring approaches in identical situations based on skilled tutors' (or even experts') opinions. It can compare performance both overall and in a variety of situation types. As in this study, it can also be used to objectively estimate the effectiveness of tutorial action alternatives and to gather information about how skilled tutors (or even expert tutors) think. This information can in turn be used to make further improvements not only to decision-theoretic tutors, but also to fixed-policy and other types of computer tutors.

6.6.3.2 Some other limitations of the current study

In the data collection phase, the methods used to learn DT's probabilities were basic, as discussed in section 6.1.1. Quite a bit of knowledge engineering went into determining the structure of DT's probabilistic networks, as described in section 3.2, so it was not necessary to learn their structure for the purposes of the current study. However, it is likely that Bayesian structure learning would provide valuable insights into which elements of DT's networks are most critical. For learning DT's probabilities, using Bayesian techniques to incorporate priors (even subjective) and take into account indirect as well as direct dependence relationships would be an important improvement. Use of Bayesian priors could help to improve learning with sparse data (even though almost 10,000 scenarios were collected) as well, as long as the priors are reasonable. Now that the assessment phase has been completed, data from both the training and test sets can be combined to partially alleviate the sparseness problem. The study encountered serious problems with determining the rule knowledge of help abusers, as described throughout much of section 4.4. To deal with help abuse, prevention is preferred but additional assessment methods may be required, as discussed in section 6.1.2.

There was no empirically verifiable way to determine DT's utilities for the current study. The methods used to tune DT according to the experimenters' preferences, described in section 4.5 and discussed in section 6.2, proved quite workable both for implementing priorities and for enabling DT to attempt a delicate balance among multiple competing objectives. But still it was necessary to make some adjustments by trial and error, which were necessarily inexact. Imbalances in utilities may have been one reason that DT seemed to respond with *teach* too often and with *hint* too seldom, as described in sections 5.4.2.1 and 5.4.2.2. Certainly, DT's utilities could benefit from more attention. Now that the assessment phase has been completed, DT's utilities could be tuned to the judges' preferences (to the extent that they have a consensus). Future experiments can be conducted to tune DT's utilities to improve its effectiveness with students.

Finally, DT's help messages did not perform as expected. As discussed in section 6.1.4, the *prompt* and *hint* messages appeared to be too similar, both in content and in effectiveness. *Teach* messages were only about as effective as *prompt* and *hint* messages for helping students to learn rules, which was their express purpose. And the *do* messages turned out to be most effective at helping students to learn rules, even though the *do* messages say nothing about rules. The reasons for these inconsistencies must be better understood. Increased understanding would probably improve both the help messages and DT's model of the student.

6.7 CONCLUSIONS

The contributions of this work consist of, first, an innovative design for a decision-theoretic engine to make tutorial action selections. Second, this engine was fleshed out into a complete ITS and its action selections were compared with an important competing technology by a panel of judges in order to begin to evaluate the potential of decision-theoretic tutoring.

6.7.1 A decision-theoretic architecture for making tutorial action selections

DT Tutor (DT) was the first tutoring system to be based upon a dynamic decision network (DDN) and thereby to gain the advantages of such a representation (Murray & VanLehn, 2000). Any tutor is necessarily uncertain about the student's changing knowledge and affective state, as these are unobservable. A DDN's Bayesian network representation (Pearl, 1988) handles uncertainty in a theoretically rigorous manner while supporting relatively efficient computation by capitalizing on structure in probabilistic relationships. The network's probabilities can be obtained from any combination of the best information available, including expert and other subjective opinions, logic (e.g., for deterministic relationships), theoretical results (e.g., from pedagogical, cognitive and psychological theory), and empirical findings. These probabilities can be based on population data, tuned to the individual, and adapted online (e.g., Mayo & Mitrovic, 2001; Murray et al., 2004). The dynamic capability of the DDN representation supports modeling the evolution of the student's changing knowledge and affect as well as other changing elements of the tutorial state (e.g., the state of the discourse between tutor and student).

In addition, a tutor is likely to have multiple objectives – e.g., increasing the student's knowledge, helping the student solve problems, and bolstering the student's affective state (Lepper et al., 1993) – and it may be impossible to achieve all objectives simultaneously (Graesser et al., 1995). A DDN's utility model can be used to balance tradeoffs among any number of competing tutorial objectives, facilitating a rich model of the tutorial state. Thus, a DDN can be used to seamlessly integrate considerations regarding uncertain, changing beliefs and priorities regarding any number of tutorial state attributes. DT's tutorial action selection engine leverages this capability to build a rich model of the tutorial state, including discourse coherence, the tutor's preferences, and the student's knowledge, feeling of independence, problem-solving progress, help style, and focus of attention (through discourse relevance).

Decision-theoretic representations are based on the well-founded theoretic underpinnings of probability and decision theory, give them what has been called a normative basis for making rational decisions (Keeney & Raiffa, 1976). Therefore, a potential benefit of using a decision-theoretic

representation for tutoring is clarifying a rationale for tutorial decisions (Jameson et al., 2001) and thus perhaps coming to a deeper understanding of tutoring.

Although not a requirement of the approach, modeling a spectrum of tutorial state attributes gives DT a flexible basis for making decisions. For instance, DT explicitly models tradeoffs between action types such as *prompt*, *hint*, *teach*, and *do* in terms of their effects on task progress, student knowledge and student affect, among other attributes, with none of the action types dominant along all dimensions. As a result, DT might progress directly from *prompt*ing about a step to *teach*ing the related rule, or possibly instead telling the student exactly how to *do* the step, as described in section 5.4. A fixed-policy tutor can achieve the same behavior by fiat. For instance, the Cognitive Tutors (Anderson et al., 1995) and Andes1 (Conati et al., 2002) always work through a sequence of hints starting with the least specific until they terminate at a bottom-out hint that is equivalent to DT's action type *do*. Because they use a fixed tutorial strategy, they do not need to explicitly represent tutorial state attributes such as student affect, so their representational requirements may be less complex. However, they pay for their simplicity by being less flexible. As illustrated in section 5.4, DT can adapt its behavior to a variety of circumstances. Moreover, DT uses the same sets of considerations to provide proactive as well as reactive help, which Andes1 and the Cognitive Tutors do not.

An important element of DT's design is looking ahead to explicitly predict the effects of the tutor's actions. While this is natural for a decision-theoretic application, it is rare for an intelligent tutoring system. Modeling the tutor's influence on the tutorial state enables the tutor to select the actions that it believes will be most beneficial to the student and to the resulting tutorial state. This requires probabilistically predicting how the tutor's actions will influence, for example, the student's knowledge, affective state, and focus of attention.

A novel component of DT's representation is its model of the student's focus of attention. Separate representations for the student's focus of attention and knowledge allow the system to probabilistically predict both the *topic(s)* of the student's next turn, based on the student's focus of attention, and the *type(s)* of action(s) in the student's next turn (e.g., whether the action(s) will be *correct*), based on the student's knowledge. Modeling the student's focus of attention also enables DT to be a cooperative discourse partner and to address topics at times when the student is likely to be interested.

A decision-theoretic agent automatically adjusts its actions to maximize attainment of its current objectives in light of its current beliefs. This makes it easy to change DT's behavior simply by changing its objectives. An effective way to do this is to change the weights associated with DT's linearly-additive multiattribute function, as discussed in section 6.2. For instance, the weight corresponding to a tutorial state attribute can be zeroed either to permanently remove that attribute from consideration or to remove it

temporarily for ablation testing (Murray et al., 2004). An extreme example is that DT can be made to behave just like the Fixed-Policy Tutor simply by zeroing out all weights except for discourse coherence along with a slight change to DT's *Discourse Coherence*$_1$ model to match exactly the Fixed-Policy Tutor's slightly stronger constraint (see section 4.5.1.1). Less drastically, DT's behavior can easily be adjusted to favor one attribute over another simply by changing the relative strengths of the corresponding weights.

DT's tutorial action selection engine is also flexible enough to be applied to diverse domains with relatively minor modifications. Besides the domain of calculus related-rates problems used for DT, the tutorial action selection engine has also been applied to a prototype for Project Listen's Reading Tutor (Murray et al., 2001a, 2001b), a tutoring system that listens to children as they read aloud.

6.7.2 Development and assessment of a decision-theoretic tutor

Given all the potential capabilities of decision-theoretic tutoring, it was time to put them to the test. Therefore, DT's action selection engine was fleshed out into a complete tutoring system with the development of (1) a student interface (section 3.3) and (2) a domain expert server to create problem solution graphs, solve domain problems, check student equations and log tutor-student interactions (section 3.1). Besides DT, two other tutors were developed for comparison purposes: (1) a Random Tutor (RT), which selected randomly from among relevant tutorial actions, and (2) a Fixed-Policy Tutor (FT), which followed a fixed policy similar to that of the Cognitive Tutors (Anderson et al., 1995) to select tutorial actions. All tutors shared the same student interface and help messages; the only difference between them was their method of selecting tutorial actions. Data was collected from 60 students in the form of pretests, posttests, and logged interactions with the Random Tutor. Data from half the students, the training set, was used to learn key probabilities for DT's dynamic decision networks, and DT utilities were tuned according to the experimenter's subjective preferences. Then three human judges, who were all skilled tutors in DT's domain, rated the responses of all three tutors to a representative sample of 350 identical scenarios from the test set of logged interactions with the Random Tutor. The 350 scenarios consisted of 175 help requests, 100 errors, and 75 *step starts*, which were opportunities to provide proactive help when a student first selects a step to begin working on it (section 5.2.2.2).

The first result, albeit informal, was that DT's response time, which had been problematic for larger networks (Murray & VanLehn, 2000; Murray et al., 2004), was no longer a problem due to limitations on the problem solution space created by the domain expert server and implemented by the student interface (section 5.3).

Next, DT was compared to RT and to FT according to the judges' ratings. The only previous comparisons of decision-theoretic tutoring were with no tutoring at all (CAPIT: Mayo & Mitrovic, 2001), with "self-learning and consulting the teacher when required" (iTutor: Pek, 2003, p. 136), and with randomized action selection (CAPIT: Mayo & Mitrovic, 2001). The Fixed-Policy Tutor's policy was designed to emulate the action selection policy of Cognitive Tutors, which can be considered a fair representative of the state-of-the-art, with documented success (e.g., Anderson et al., 1995; Koedinger et al., 1997) in use by thousands of students in hundreds of schools. Therefore, this study's comparison posed a stiffer test than previous comparisons with decision-theoretic tutoring.

First, DT's ratings were compared to RT's. According to the judges' ratings, both individual and composite, DT's action selections were decidedly better than RT's action selections both overall and for the subsets of help requests, errors, and first message opportunities, significant at the level $p<.01$ (section 6.3). Only for step start scenarios was DT's mean composite rating was not higher enough to be significant, $p=.120$, although that was using the very conservative Bonferroni correction for 10 t-tests which increased the original significance of $p=.012$ by an order of magnitude. Effect sizes ranged from .30 to .49, which is large enough to make an impact. This was solid support for Hypothesis 1: According to skilled tutors, tutorial action selections by decision-theoretic methods can be better than selections made randomly among relevant tutorial actions.

Next, DT's ratings were compared to FT's – the stiffer test. The judges rated DT higher than the Fixed-Policy Tutor (FT) overall and for the scenario subsets of errors, start steps and first-message-opportunity scenarios with substantial effect sizes ranging from .22 to .61, all with significance $p<.02$ or better (see section 6.4.1). DT was not rated higher than FT only for help requests; for these, DT's mean composite rating of 3.66 was nominally higher than FT's rating of 3.59 with an effect size of only .06. With DT significantly surpassing FT both overall and for all major subsets of scenarios other than help requests, and with DT about equal to FT (although nominally higher) for help requests, these results support Hypothesis 2: According to skilled tutors, tutorial action selections by decision-theoretic methods can be better than those of a tutor that emulates the fixed policies of theory-based, widely accepted and highly effective tutors.

In addition, DT adapted its response type to multiple attributes of the tutorial situation (section 6.4.2) while FT adhered to its fixed policy, which considered only the discourse state as follows: (1) never provide proactive help (i.e., responding *null*) for step start and first-error scenarios, and (2) otherwise provide the help message that is minimally more explicit than any help messages previously provided for the same step. DT's variations in responses in accordance with multiple tutorial state attributes paid off in generally higher ratings from the human judges. A particularly telling result was for first-message-opportunity scenarios divided according to whether the student got the corresponding

pretest problem wrong or right: DT's ratings were significantly higher for the pretest-wrong subset with a substantial effect size and also marginally higher for the pretest-right subset before the significance was diluted by the conservative Bonferroni correction.

A large part of DT's success relative to FT can be traced to its provision of proactive help (section 6.4.4). Therefore enhanced versions of DT and FT – *DTe* and *FTe* – which always responded with proactive help, were created to see if a simple change to FT's fixed policy (to always provide proactive help) would cause a fixed policy tutor to be rated as highly as a decision-theoretic tutor (section 6.4.4.1). FTe was in fact rated marginally higher than DT overall and significantly higher than DT for errors, $p<.01$, even after the conservative Bonferroni correction and with an effect size for errors of .53. For help request responses, which are reactive help, results were unchanged (DT nominally but insignificantly higher than FTe) because the change to FTe's and DTe's policies affected only proactive help. These results showed that the judges preferred even more proactive help than DT was giving. Comparing DTe and FTe, DTe was rated nominally but insignificantly higher than FTe overall and for all scenario subsets except step starts (for which FTe had an advantage of .01), with small effective sizes.

However, DTe compared to FTe, like DT compared to FT, showed significantly more variability in responses and sensitivity to the tutorial state (section 6.4.4.1). For example, for all first-message-opportunity scenarios, FTe selected response *prompt*, the least explicit of the non-*null* help types. DTe's response distribution was significantly different from FTe's and it varied significantly between the pretest-right and pretest-wrong subsets. For the pretest-wrong scenarios, DTe's ratings were marginally higher than FTe's with a healthy effect size. Also, for first-message-opportunity errors, DTe's ratings for pretest-wrong scenarios were marginally higher than FTe's and its ratings for pretest-right scenarios were nominally higher but not significant.

Comparisons with FTe show that a fixed-policy selected post hoc can do just about as well at selecting tutorial responses as decision-theoretic methods, at least at DT's current stage of development. FT's current policy can easily be enhanced (at least according to the judges' ratings) to provide proactive help and then be competitive with DT's and DTe's performance. FT's and FTe's policies remain relatively simple and so they still significantly lag DT and DTe in their sensitivity to the context of the multiattribute tutorial state. Theoretically, a fixed policy can be used to implement policies of arbitrary complexity – at the extreme, consisting of a table lookup of what to do in each unique situation. Of course, anything that can be implemented as a fixed policy can also be implemented in DT, since DT's action selection capabilities are a superset of FT's (see section 6.2). And DT is still at a relatively early stage of development as well (see, e.g., sections 6.1 and 6.2 about determining DT's numeric parameters), so its capabilities can also be further improved.

A theme throughout the judges' ratings is that they generally preferred more proactive help to less (see section 6.5). While this result may be partially an artifact of the limitations of the assessment method (e.g., relying on human judgment – see section 6.6.3.1), it was robust across judges, tutors, and scenario types. Furthermore, several studies (e.g., Fox, 1993; Lepper et al., 1993; McArthur et al., 1990) human tutors sometimes provide proactive help, at least when broadly defined to include subtle types of help such as structuring tasks so that their students are less likely to fail. Therefore, the implication that more computer tutors should consider providing proactive help, at least as an important topic for further investigation.

While the method used for assessing the tutors did have limitations, especially for determining the bottom line – effectiveness with students (see section 6.6.3.1) – it did provide information that would not have been obtained in a more traditional study of bottom-line effectiveness. First, it provided raw data for learning many of DT's key probabilities. Second, it can be used to compare the response selections of different tutoring approaches in *identical situations*. Any number of tutors can be assessed this way by running them against the logged data – that is how FTe and DTe were assessed even though their creation had not been anticipated. Perhaps most importantly, it can be used to gather both quantitative and qualitative information about how skilled tutors – or even experts – think. In fact, the same scenarios can still be assessed by other judges. In short, data from both the data collection and assessment phases of the study remain a rich source of information about tutoring. This information can be used to make further improvements not only to decision-theoretic tutors, but also to fixed-policy and other types of computer tutors.

Finally, an issue that may describe the bottom line for many readers of this report: Should you choose fixed-policy or decision-theoretic tutoring? For successful resolution, this question must be quantified in terms of performance needs versus development and maintenance costs – software engineering issues that this study was not designed to investigate. However, this study has produced some useful guidelines (see section 6.4.5 for a more detailed discussion): If the desired behavior of the tutor is unambiguously defined, only simple capabilities are required, and only simple changes to the tutor's behavior are anticipated over the life of the tutor, fixed policy is best, no contest, because of its ease of implementation. However, as the desired behavior of the tutor becomes more ambiguous, the required capabilities increase, or as the need for flexibility in the tutor's behavior increases, decision-theoretic tutoring becomes more attractive. Furthermore, the cost/benefit ratio may change over time as new tools are developed (e.g., probabilistic network development tools such as SMILE, which was used for this study) and as the state of the art advances, leading to increased expectations for the capabilities of computer tutors.

APPENDIX A. Calculus Tutor Tutorial

Calculus Tutor Tutorial

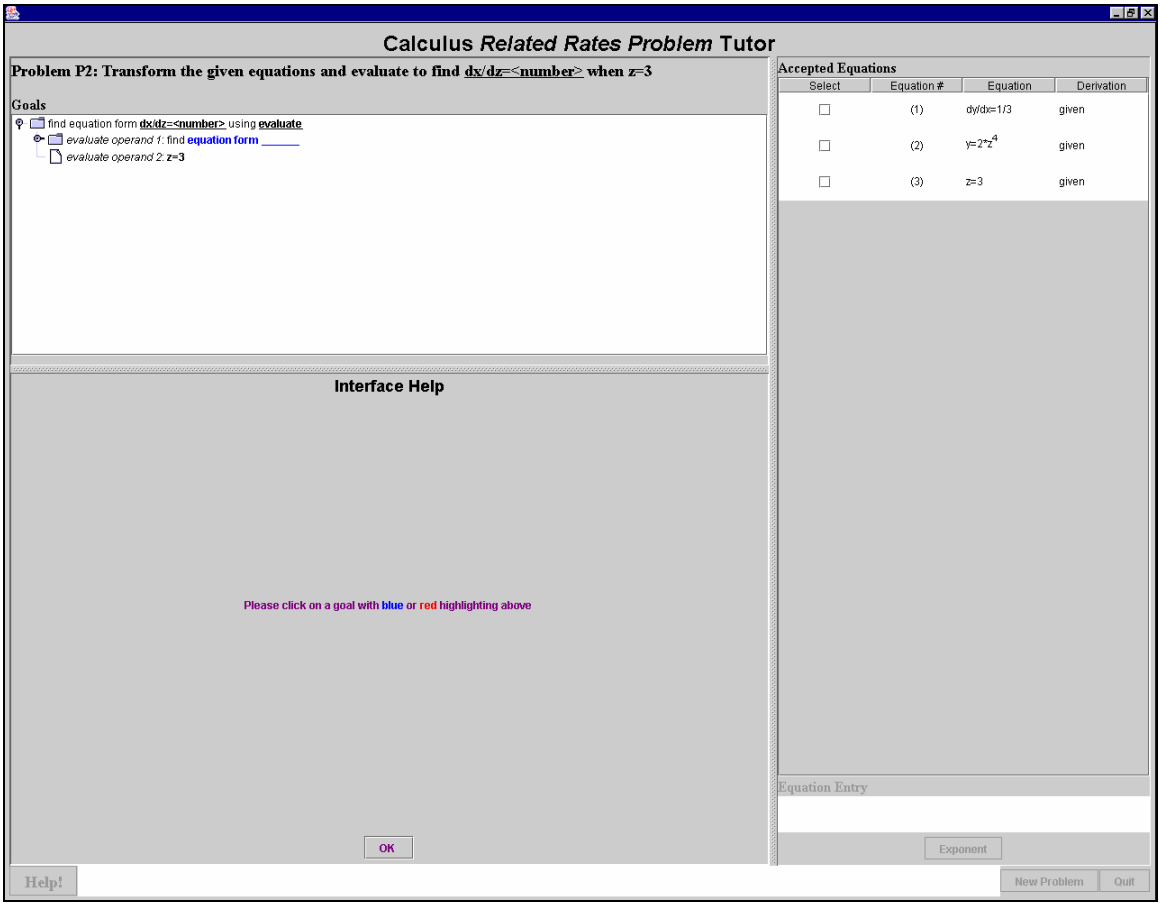


Figure A1: Calculus Tutor screen shot

Introduction

The Calculus *Related Rates Problem* Tutor, or Calculus Tutor for short, helps you solve calculus related-rates problems using calculus, algebra, and a goal-oriented problem-solving procedure. This tutorial explains all that you need to know to begin solving problems using the Calculus Tutor. Try to remember as much of this information as you can, but the Calculus Tutor will help you with things you forget.

A screen shot of the Calculus Tutor's interface is shown in Figure A1. At the top left is the problem number and top-level goal, in this case "Problem P2: Transform the given equations and evaluate to find dx/dz=<number> when z=3." Beneath the problem statement is the **Goals Window**, which you will click on to progress through the problem. The large window beneath the Goals Window that currently says "Interface Help" is the **Dialog Window** where the Calculus Tutor's prompts and help will be displayed. Along the right side of the screen is the **Accepted Equations Window**, which at the beginning of the problem lists two given equations plus an equation that gives a numeric value for one of the variables (z=3). As you create additional equations, they will be added to this list along with the operators and operands used to create them. Beneath the Accepted Equations Window is a small **Equation Entry Window** for entering equations. It includes an **Exponent** button for entering exponents. At the bottom of the screen are buttons to request "**Help!**", select a **New Problem**, or **Quit** the tutor.

Calculus *Related-Rates* Problems

Calculus related-rates problems typically give you equations describing how a set of related variables change with respect to one another and ask you to figure out some other relationship amongst the variables. The problems you will solve all give you two equations and ask you to figure out a third equation. To solve the problems, you will transform the two given equations using mathematical operations, or ***operators***, from calculus and algebra.

We use the symbols "*****" for multiplication and "/" for division.

You must simplify each equation that you create by carrying out all multiplications, divisions, additions, subtractions and exponentiations. For example:

$$x = 10/2 * 3^{3-1} * z^{2+1} \rightarrow x = 5 * 3^2 * z^3 \rightarrow x = 5 * 9 * z^3 \rightarrow x = 45*z^3$$

Operators

The next two sections describe the calculus and algebra operators that you will use with the Calculus Tutor. Each operator transforms one or two operands, which are the inputs to the operation performed by the operator. As we describe the operators, we use the term **LHS** to describe the part of the equation on the left-hand-side of the equals (=) sign, and the term **RHS** to describe the part of the equation on the right-hand-side of the equals sign.

Calculus Operators

These calculus operators all transform or create ***derivative equations***, such as $dx/dy=6*y^2$. Derivative equations are equations that contain a ***derivative expression***, such as dx/dy. We call equations that don't have a derivative expression ***regular equations***. The d in the derivative expression dx/dy stands for "change," so dx/dy stands for "change in x with respect to change in y." Similarly, du/dv stands for "change in u with respect to change in v."

Two important calculus operators are *differentiate* and *integrate*. This section will teach you one rule for each operator. Then you will learn two operators for transforming derivative equations, *chain rule* and *flip derivative*. These operators just involve algebra and so hopefully they won't seem too hard once you read their explanations.

differentiate

The *differentiate* operator transforms a regular equation into a derivative equation. There are many rules for differentiation, but for the Calculus Tutor you just need to learn the **power rule**:

Example 1: operator: *differentiate* operand: $x=2*y^3$

result: $dx/dy = 3 * 2*y^2$. Simplifying, $dx/dy=6*y^2$

(1) **Change the LHS into a derivative expression** – e.g., dx/dy:

- The numerator is d followed by the variable on the LHS of the operand. For the example above, x is the variable on the LHS of operand $x=2*y^3$, so the numerator is dx.
- The denominator is d followed by the variable on the RHS of the operand. For the example above, y is the variable on the RHS of operand $x=2*y^3$, so the denominator is dy.

(2) **Multiply the RHS by the exponent**. For the example above, the exponent is 3, so the RHS becomes $3*2*y^3$.

(3) **Subtract 1 from the exponent**. For the example above, the RHS becomes $3*2*y^{3-1}$, or $3*2*y^2$

(4) Simplify the resulting equation. For the example above, $dx/dy=3*2*y^2$ becomes $dx/dy=6*y^2$.

Example 2: operator: *differentiate* operand: $u=5*v^2$

result: $du/dv=2*5*v^1$. Simplifying (note: $v^1=v$), $du/dv=10*v$

integrate

The *integrate* operator is the reverse of the differentiate operator. It transforms a derivative equation into a regular equation. For the Calculus Tutor, you just need to learn the **reverse of the power rule**.

Example 1: operator: *integrate* operand: $dx/dy=6*y^2$

result: $x=6/3 * y^3$. Simplifying, $x=2*y^3$

(1) **Change the LHS into a single variable** – e.g., x. The variable comes from the numerator of the derivative expression. For the example above, from the numerator of dx/dy we get x.

(2) **Add 1 to the exponent** on the RHS. For the example above, the RHS becomes $6*y^{2+1}$, or $6*y^3$.

(3) **Divide the RHS by the new exponent**. For the example above, the new exponent is 3, so the RHS becomes $6/3*y^3$.

(4) Simplify the resulting equation. For the example above, $x=6/3*y^3$ becomes $x=2*y^3$.

Note: Technically, the integrated equation is $x=2*y^3$**+ c**, where c is any constant number and is called the *arbitrary constant of integration*, but we will not use the "+ c" within Calculus Tutor.

Example 2: operator: *integrate* operand: $dj/dk=2*k$ (note: $k=k^1$)

result: $j=2/2*k^2$. Simplifying, $j=k^2$

chain rule

The *chain rule* operator combines 2 derivative equations to form another derivative equation. To use the chain rule, multiply both sides of the 2 derivative equations together. For example, given equations

Example 1: operator: *chain rule* operand 1: dq/dr=5
operand 2: dr/ds=2*s^3

result: dq/dr * dr/ds = 5 * 2*s^3. Simplifying (the dr's cancel out), dq/ds=10*s^3

(1) **Multiply the two LHSs together**. For the example above, we dq/dr * dr/ds.

(2) **Multiply the two RHSs together**. For the example above, we get 5 * 2*s^3.

(3) Simplify the resulting equation. For the example above, the dr's cancel out, so dq/dr * dr/ds = 5 * 2*s^3 becomes dq/ds=10*s^3.

In the Calculus Tutor, the *chain rule* operands and result will always be in the following forms:

***operand 1*:**	**d<variable 1> / d<variable 2> = <number>**	– e.g., dq/dr=5
***operand 2*:**	**d<variable 2> / d<variable 3> = f(<variable 3>)**	– e.g., dr/ds=2*s^3
***result*:**	**d<variable 1> / d<variable 3> = f(<variable 3>)**	– e.g., dq/ds=10*s^3

where <number> stands for any number (e.g., 5) and f(<variable 3>) stands for *some* function of variable 3. Note that f(<variable 3>) is different in *operand 2* than in *result*, just as 2*s^3 is different than 10*s^3. When *operand 1* and *operand 2* are multiplied together, the d<variable 2> term cancels out.

Using *chain rule*, given a derivative equation relating one variable to a second variable (e.g., q to r in dq/dr=5), and another derivative relating the second variable to a third variable (e.g., r to s in dr/ds=2*s^3), we can combine the derivatives to form a derivative relating the first variable to the third variable (e.g., q to s in dq/ds=10*s^3).

Example 2: operator: *chain rule* operand 1: du/dv = 4
operand 2: dv/dw = w^5

result: du/dv * dv/dw = 4 * w^5. Simplifying, du/dw=4*w^5

flip derivative

The *flip derivative* operator transforms a derivative equation by taking the multiplicative inverse of both sides of the equation.

Example 1: operator: *flip derivative* operand: df/de=5
result: de/df=1/5

(1) **Take the multiplicative inverse of the LHS**. For the example above, df/de becomes de/df.

(2) **Take the multiplicative inverse of the RHS**. For the example above, 5 becomes 1/5.

(3) Simplify the resulting equation if necessary.

In the Calculus Tutor, you will use *flip derivative* only on equations that have a constant number on the RHS, so the algebra for taking the multiplicative inverse should be easy.

Example 2: operator: *flip derivative* operand: da/db = 1/7
result: db/da = 7

Algebra Operators

substitute

The *substitute* operator uses algebra to combine two regular equations that have a variable in common into a third regular equation.

Example 1: operator: *substitute* operand 1: $j=5*k^2$
operand 2: $k=2*m^3$

result: $j = 5 * (2*m^3)^2$. Simplifying, $j = 5*2^2*m^{3*2}$ → $j = 5*4*m^6$ → $j=20*m^6$

(1) **Substitute the variable on the RHS of operand 1 with the expression for it from the RHS of operand 2**. For the example above, we substitute k in operand 1 with the expression $2*m^3$ from operand 2, yielding $j = 5 * (2*m^3)^2$.

(2) Simplify the resulting equation. For the example above, first we carry out the exponentiation $(2*m^3)^2$, yielding $j = 5*2^2*m^{3*2}$, then we carry out the remaining exponentiation and multiplications to get $j=20*m^6$.

In the Calculus Tutor, *substitute*'s operands and result will always be in the following forms:

***operand 1*:**	**<variable 1> = f(<variable 2>)**	– e.g., $j=5*k^2$
***operand 2*:**	**<variable 2> = f(<variable 3>)**	– e.g., $k=2*m^3$
***result*:**	**<variable 1> = f(<variable 3>)**	– e.g., $j=20*m^6$

Using *substitute*, given a regular equation relating one variable to a second variable (e.g., j to k as in $j=5*k^2$), and another regular equation relating the second variable to a third variable (e.g., k to m as in $k=2*m^3$), we can combine the equations to form a regular equation relating the first variable to the third variable (e.g., j to m as in $j=20*m^6$).

Example 2: operator: *substitute* operand 1: $x=3*y$
operand 2: $y=4*z^2$

result: $x=3*(4*z^2)$. Simplifying, $x=12*z^2$

evaluate

The *evaluate* operator is the same as *substitute* except that one equation specifies a numeric value for the variable in common between the two equations.

Example 1: operator: *evaluate* operand 1: $s=4*t^2$
operand 2: $t=3$
result: $s = 4*3^2$. Simplifying, $s = 4*9$ → $s=36$

(1) **Substitute the variable on the RHS of operand 1 with the numeric value for it from the RHS of operand 2**. For the example above, we substitute t in operand 1 with the value 3 from operand 2, yielding $s = 4*3^2$.

(2) Simplify the resulting equation. For the example above, first we carry out the exponentiation 3^2, yielding $s = 4*9$, then we carry out the multiplication to get $s=36$.

In the Calculus Tutor, the operands and result will always be in the following forms:

***operand 1*:**	**<variable 1> = f(<variable 2>)**	– e.g., $s=4*t^2$
***operand 2*:**	**<variable 2> = <number>**	– e.g., $t=3$
***result*:**	**<variable 1> = <number>**	– e.g., $s=36$

OR

***operand 1*:**	**d<variable 1> / d<variable 2> = f(<variable 2>)**	– e.g., $de/df=5*f^3$
***operand 2*:**	**<variable 2> = <number>**	– e.g., $f=2$
***result*:**	**d<variable 1> / d<variable 2> = <number>**	– e.g., $de/df=40$

Example 2: operator: *evaluate* operand 1: $de/df = 5*f^3$
operand 2: $f=2$
result: $de/df=5*2^3$. Simplifying, $de/df = 5*8$ → $de/df=40$

restate

The *restate* operator uses algebra to transform a regular equation by reversing which variables are on the LHS and RHS.

Example 1: operator: *restate* operand: $m=½*n^{1/5}$
result: $n=32*m^5$

(1) **Use algebra to isolate the variable on the RHS**. For the example above:

- Multiply both sides of the equation by 2 to get $2*m=n^{1/5}$
- Take both sides to the 5^{th} power to get $2^5*m^5=n$

(2) **Swap the LHS and RHS**. For the example above, $2^5*m^5=n$ becomes $n=2^5*m^5$.

(3) Simplify the resulting equation. For the example above, we carry out the exponentiation 2^5, yielding $n=32*m^5$.

Example 2: operator: *restate* operand: $a=4*b$
result: $b=1/4*a$

Example 3: operator: *restate* operand: $v=w^3$
result: $w=v^{1/3}$

Equation Forms

In order to talk in a general way about how to do calculus related-rates problems, and in order to use the Calculus Tutor, we need to be able to talk about *types* of equations rather than just particular equations. Instead of saying that the differentiate operator transforms the equation $x=2*y^3$ into $dx/dy=6*y^2$, and separately saying that the differentiate operator transforms the equation $x=y^4$ into $dx/dy=4*y^3$, we would like to be able to say more generally that the differentiate operator transforms the *type* of equation with x on the LHS and an algebraic expression involving y on the RHS into the *type* of equation with dx/dy on the LHS and an algebraic expression involving y on the RHS. And since statements like "an algebraic expression involving y on the RHS" are long-winded, we would like to say that more briefly as well. To speak about *types* of equations, or ***equation forms***, we need just a couple of symbols:

1. Instead of "an algebraic expression involving y on the RHS", we say **f(y)**, which stands for "some function of y." Similarly, some function of z is f(z), and so on. Within the Calculus Tutor, f(y) means y along with perhaps some numbers and arithmetic symbols for multiplication, division, exponentiation, etc., but no other variables. Examples:
   - $2*y^3$ is f(y)
   - $6*y^2$ is f(y)
   - b is f(b)
   - $2*u*v^2$ is not f(u) and not f(v) – it is f(u,v)

   Important Note: Within the Calculus Tutor, f(y) means *some* function of y, but not a *particular* function of y. For example, the operator *differentiate* transforms equation $x=2*y^3$ into equation $dx/dy=6*y^2$. In terms of equation forms, we say that *differentiate* transforms equation form x=f(y) into dx/dy=f(y). But that does not mean that f(y) is the same in x=f(y) and dx/dy=f(y), just as $2*y^3$ is not the same as $6*y^2$.

2. For a type of equation that has **no variables** (like y or z) on the RHS, we say the RHS has some value **<number>**, which stands for any constant number. Examples:
   - 2 is <number>
   - $6*3^2$ is <number>. (Simplifying, $6*3^2 \rightarrow 6*9 \rightarrow 54$, which is a number)
   - 2*b is not <number>

That is all the notation we need to describe ***equation forms*** in the Calculus Tutor. Every equation form will be as follows:

On the LHS:	Either a regular variable, such as **x**, or a derivative expression such as **dx/dy**.
On the RHS:	Either a function of a variable, such as **f(x)**, or **<number>**.

If the LHS of an equation or equation form is a derivative expression, we say that it is in ***derivative form***; otherwise, we say that it is in ***regular form***.

Table A1: Equation Form Examples

Equation Form	Sample Equation	Comments
x=f(y)	$x=2*y^3$	
dx/dy=f(y)	$dx/dy=6*y^2$	Note that the actual function represented by f(y) here is different than the actual function represented by f(y) in the previous example
v=<number>	v=34	
dv/dk=<number>	dv/dk=16	Note that the actual number represented by <number> here is different than the actual number represented by <number> in the previous example

Equation Forms for the Operators

Each operator transforms one or more *operands* from one equation form into another. For instance, *differentiate* transforms an operand with equation form x=f(y) into equation form dx/dy=f(y). Similarly, *chain rule* transforms 2 operands, one with equation form dx/dy=<number> and one with equation form dy/dz=f(z), into equation form dx/dz=f(z).

Of course, **the actual variables in these equation forms may be different**. For instance, *differentiate* can also transform equation form u=f(v) into du/dv=f(v). **It is the patterns in the equation forms that are important.** Regardless of the variables in *differentiate*'s operand, the pattern for *differentiate*'s resulting equation form is:

- d
- the variable on the LHS of the operand – e.g., x in x=f(y) or u in u=f(v)
- /
- d
- the variable on the RHS of the operand – e.g., y in x=f(y) or v in u=f(v)
- =
- f (<variable>), where <variable> is the variable on the RHS of the operand – e.g., y in x=f(y) or v in u=f(v)

For operators with 2 operands (*chain rule*, *substitute* and *evaluate*) **the order of the operands is important to the Calculus Tutor**. For instance, *chain rule* requires the following order for its operands:

operand 1: dx/dy=<number>
operand 2: dy/dz=f(z)

Sample equation forms for the operators are listed in Table A2. Now would be a good time for you to verify that your understanding of the operators fits with the equation forms listed.

Knowing the operators' equation forms will be useful for transforming equations using the Calculus Tutor. We will use them both to select operators and to select equation forms for operators' operands, as described below.

1. **Selecting which operator(s) to use.** For instance, if we need to find an equation of form dx/dy=f(y), we can reason that we must use operator *differentiate* or *chain rule* since these are the only operators that result in an equation form like dx/dy=f(y), as can be seen in the Sample Equation Forms, Result column of Table A2. A later section describes heuristics (rules of thumb) to make selecting operators easier.

2. **Selecting which equation form(s) we need to find.** For instance, if we have decided that we will use operator *differentiate* to find equation form du/dv=f(v), we know that we must find or create an equation corresponding to the equation form for *differentiate*'s operand, u=f(v).

Table A2: Sample equation forms for the operators

	Sample Equation *Forms*		**Sample Equations**	
Operator	**Operand(s)**	**Result**	**Operand(s)**	**Result**
differentiate	**u=f(v)**	**du/dv=f(v)**	$u=3*v^4$	$du/dv=12*v^3$
integrate	**dh/di=f(i)**	**h=f(i)**	$dh/di=6*i^2$	$h=2*i^3$
flip derivative	**dm/dn=<number>**	**dn/dm=<number>**	dm/dn=1/4	dn/dm=4
chain rule	**1. ds/dt=<number>** **2. dt/dv=f(v)**	**ds/dv=f(v)**	1. ds/dt=5 2. $dt/dv=2*v^4$	$ds/dv=10*v^4$
restate	**b=f(c)**	**c=f(b)**	$b=1/8*c^3$	$c=2*b^{1/3}$
substitute	**1. q=f(r)** **2. r=f(s)**	**q=f(s)**	1. $q=4*r^3$ 2. $r=3*s^2$	$q=108*s^6$
evaluate	**1. x=f(z)** **2. z=<number>**	**x=<number>**	1. $x=2*z^4$ 2. z=3	x=162
	1. dp/dq=f(q) **2. q=<number>**	**dp/dq=<number>**	1. $dp/dq=5*q^2$ 2. q=4	dp/dq=80

The Calculus Tutor's Problem-Solving Procedure

The Calculus Tutor uses a quite general goal-oriented problem-solving procedure. The procedure works by setting any subgoals that are necessary to achieve the top-level goal. The subgoals then become the current goals. If any of the current goals are not already achieved, then new subgoals (sub-subgoals) are set to achieve the current goals. This procedure continues, setting further subgoals (sub-sub-subgoals, etc.) as necessary, until all of a goal's subgoals have been achieved, at which point the goal has been achieved. When all of the top-level goal's subgoals have been achieved, the problem has been solved.

For each problem, you will be given a top-level goal of applying the *evaluate* operator to create an equation with a specific form (e.g., x=<number> or dx/dz=<number>) when one of the variables has a numeric value (e.g., when z=3). An example was provided in Figure A1. You will also be given two equations in the Accepted Equations window, plus an equation that states a numeric value for one of the variables (e.g., z=3). *Evaluate operand 2* is always the equation that states a numeric value for one of the variables.

Your first subgoal is deciding the equation form for *evaluate operand 1*. Once that equation form has been decided, your next subgoal becomes deciding whether you can find an equation with that form among the Accepted Equations or whether you must use an operator to create the equation. If you must use an operator, your next subgoal becomes deciding what that operator will be. Once you decide on the operator, your next subgoal becomes deciding the equation form(s) for the operator's operand(s). This process continues, establishing further subgoals if necessary, until you can use an Accepted Equation as an operand. Once you have found the equation(s) for an operator's operand(s), you can apply the operator to create a new equation. As subgoals are achieved, you will eventually find or create *evaluate operand 1*, at which point you can apply the *evaluate* operator to achieve the top-level goal.

This goal-oriented procedure may sound complicated, but after a while its regular pattern becomes evident. Furthermore, **the Calculus Tutor will lead you every step of the way as you click on goals within the Goals Window**. With the Calculus Tutor's help on the problem-solving procedure, there are 4 skills that you will need. Each of these is explained in turn below:

1. **Applying operators.** Rules and examples for applying the 4 calculus and 3 algebra operators are described in the Operators section above.

2. **Selecting an Accepted Equation that matches a sought equation form.** The relationships between equation forms and actual equations are described in the Equation Forms section above. For instance, if the equation form you are looking for is dx/dy=f(y) and the Accepted Equations are $dy/dz=3*z^2$ and $dx/dy=5*y^4$, select equation $dx/dy=5*y^4$.

3. **Specifying equation form(s) for operand(s).** This skill is described in the Equation Forms for the Operators section above. For instance, to apply *differentiate* to create an equation with the form dx/dy=f(y), the equation form for the *differentiate* operand must be x=f(y).

4. **Selecting operators to find equation forms.** An explanation of this final skill is provided in the next section, Heuristics (Rules of Thumb) for Selecting Operators.

Heuristics (Rules of Thumb) for Selecting Operators

With 4 calculus operators, 3 algebra operators, and 2 given equations (3 counting the equation that states a numeric value for one of the variables), you could do a lot of work applying operators and generating new equations without ever achieving the problem goal. That is why the Calculus Tutor uses a goal-oriented problem-solving procedure – to avoid extra work. With the goal-oriented problem-solving procedure and the heuristics listed below, you will be able to solve the problems efficiently with minimal wasted effort. **The Calculus Tutor will require you to follow these heuristics.**

First of all, the operator for the top-level goal, *evaluate*, is given, so you never have to worry about selecting it – the Calculus Tutor won't even give you the choice.

1. **If there is an operator that will achieve a sought equation form in one step, you must use that operator.** For instance, if you need to use an operator to create an equation of form dx/dy=f(y) to achieve the current goal, and one of the Accepted Equations has form x=f(y), then you must use operator *differentiate* to achieve the goal in one step. This heuristic will be used for selecting among the operators *differentiate*, *integrate*, *flip derivative*, and *restate* as summarized in Table A3 below. For each operator, check if you can see how it transforms the Accepted Equation form into the sought equation form in one step.

   Table A3: Operator selection heuristics: differentiate, flip derivative, integrate, & restate

Sought Equation Form	Accepted Equations include equation of form	Use Operator
dp/dq=f(q)	p=f(q)	*differentiate*
du/dv=<number>	dv/du=<number>	*flip derivative*
x=f(y)	dx/dy=f(y)	*integrate*
c=f(d)	d=f(c)	*restate*

2. **The 2 out of 3 heuristic for *evaluate operand 1*:** If at least 2 out of 3 of *evaluate operand 1*'s equation form and the first two given equations (not counting the equation that states a numeric value for one of the variables) are in derivative form, use operator ***chain rule*** as the combining operator for *evaluate operand 1*. Otherwise, use operator ***substitute*** as the combining operator. See Table A4 (next page) for complete details.

   The reason for this heuristic is more complex, plus it involves one more wrinkle, so it is explained in further detail below. If you don't want to bother with the explanation or this explanation isn't helpful to you, you can just skip to the table.

   Explanation: The trick to all the problems is that they all involve 3 variables, the top-level goal is to evaluate an equation involving *variable 1* and *variable 3*, one of the given equations involves *variable 1* and *variable 2*, and the other given equation involves *variable 2* and *variable 3*. To achieve the top-level goal, we must transform the given equations to create for *evaluate operand 1* an equation involving *variable 1* and *variable 3*. Therefore, we must at some point *combine* (1) an

equation involving *variable 1* and *variable 2*, and (2) an equation involving *variable 2* and *variable 3*, to create an equation involving *variable 1* and *variable 3*. The only operators we have that combine variables in this way (so we call them ***combining operators***) are ***chain rule*** and ***substitute***. So we know that at some point we're going to have to use either *chain rule* or *substitute*. In fact, we need to use *chain rule* or *substitute* only once because once we have created an equation involving *variable 1* and *variable 3*, we won't need to combine these variables again. Since we know we're going to have to use either *chain rule* or *substitute* one time, we can simplify our search for a problem solution by deciding to use one of these two operators right away, when we are deciding which operator to use for *evaluate operand 1*.

But which combining operator should we select? Well, since *chain rule* involves only derivative equations (both for its operands and its result), it is usually more efficient when most of our equations are in derivative form, since fewer transformations between regular and derivative form will be required. Similarly, *substitute* is usually more efficient when most of our equations are in regular form. So if at least 2 out of 3 of the two given equations (not counting the equation stating *variable 3*'s numeric value) and *evaluate operand 1*'s equation form are in derivative form, use *chain rule* as the combining operator; otherwise, at least 2 out of 3 of these equations must be in regular form, so use *substitute* as the combining operator.

Here's the wrinkle: If, for example, the two given equations are in derivative form and *evaluate operand 1*'s equation form is in regular form, then our heuristic says to use *chain rule* as the combining operator. This is correct, but the problem is that *chain rule* will result in a derivative equation while the *evaluate operand 1*'s equation form is in regular form. We will need to *integrate* the equation that results from applying *chain rule* in order to create the regular equation form for *evaluate operand 1*. This means we need to plan to use an ***operator sequence***, ***integrate* after *chain rule***. Similarly, if the two given equations are in regular form and *evaluate operand 1*'s equation form is in derivative form, we need to plan to use the ***operator sequence differentiate* after *substitute***.

All of this information is summarized in Table A4 below.

Table A4: 2 out of 3 heuristic for selecting operator(s) for evaluate operand 1

Equation form for first two given eqs & *evaluate operand 1*	***evaluate operand 1's* equation form**	**Operator(s) for *evaluate operand 1***
derivative	derivative	*chain rule*
derivative	regular	*integrate* after *chain rule*
regular	regular	*substitute*
regular	derivative	*differentiate* after *substitute*

APPENDIX B. Posttest

DT04-1 Experiment

Posttest

User ID: _____

Date: _____

Please answer the following questions to the best of your ability.

Your performance will have no impact on your record at the University of Pittsburgh.

If you have any questions, please feel free to ask the experimenter.

This test usually takes approximately 30 minutes to complete.

THANKS FOR PARTICIPATING!

1. Specify the equation form for the differentiate operand by circling one expression for the left-hand-side (LHS) and one expression for the right-hand-side (RHS).

    - find equation form **dx/dy=f(y)** using **differentiate**
        - *differentiate operand*: find **equation form** _____

LHS		**RHS**
_____	=	_____
x		f(x)
y		f(y)
z		f(z)
dx/dy		K
dx/dz		
dy/dx		
dy/dz		
dz/dx		
dz/dy		

2. Specify the equation form for the *integrate* operand by circling one expression for the LHS and one expression for the RHS.

    - find equation form **y=f(z)** using **integrate**
        - *integrate operand*: find **equation form** _____

LHS		**RHS**
_____	=	_____
x		f(x)
y		f(y)
z		f(z)
dx/dy		K
dx/dz		
dy/dx		
dy/dz		
dz/dx		
dz/dy		

3. Specify the equation form for *chain rule operand 1* by circling one expression for the LHS and one expression for the RHS.

    - find equation form **dx/dz=f(z)** using **chain rule**
        - *chain rule operand 1*: find **equation form** _____
        - *chain rule operand 2*:

LHS		**RHS**
_____	=	_____
x		f(x)
y		f(y)
z		f(z)
dx/dy		K
dx/dz		
dy/dx		
dy/dz		
dz/dx		
dz/dy		

4. Specify the equation form for *chain rule operand 2* by circling one expression for the LHS and one expression for the RHS.

    - find equation form **dx/dz=f(z)** using **chain rule**
        - *chain rule operand 1*:
        - *chain rule operand 2*: find **equation form** _____

LHS		**RHS**
_____	=	_____
x		f(x)
y		f(y)
z		f(z)
dx/dy		K
dx/dz		
dy/dx		
dy/dz		
dz/dx		
dz/dy		

5. Specify the equation form for the *flip derivative* operand by circling one expression for the LHS and one expression for the RHS.

    - find equation form **dx/dy=K** using **flip derivative**
        - *flip derivative operand*: find **equation form** _____

LHS		**RHS**
_____	=	_____
x		f(x)
y		f(y)
z		f(z)
dx/dy		K
dx/dz		
dy/dx		
dy/dz		
dz/dx		
dz/dy		

6. Specify the equation form for the *restate* operand by circling one expression for the LHS and one expression for the RHS.

    - find equation form **y=f(z)** using **restate**
        - *restate operand*: find **equation form** _____

LHS		**RHS**
_____	=	_____
x		f(x)
y		f(y)
z		f(z)
dx/dy		K
dx/dz		
dy/dx		
dy/dz		
dz/dx		
dz/dy		

7. Specify the equation form for *substitute operand 1* by circling one expression for the LHS and one expression for the RHS.

    - find equation form **x=f(z)** using **substitute**
        - *substitute operand 1*: find **equation form _____**
        - *substitute operand 2*:

LHS		**RHS**
_____	=	_____
x		f(x)
y		f(y)
z		f(z)
dx/dy		K
dx/dz		
dy/dx		
dy/dz		
dz/dx		
dz/dy		

8. Specify the equation form for *substitute operand 2* by circling one expression for the LHS and one expression for the RHS.

    - find equation form **x=f(z)** using **substitute**
        - *substitute operand 1*:
        - *substitute operand 2*: find **equation form _____**

LHS		**RHS**
_____	=	_____
x		f(x)
y		f(y)
z		f(z)
dx/dy		K
dx/dz		
dy/dx		
dy/dz		
dz/dx		
dz/dy		

9. Specify the equation form for *evaluate operand 1* by circling one expression for the LHS and one expression for the RHS.

    - find equation form **x=K** using **evaluate**
        - *evaluate operand 1*: find **equation form** _____
        - *evaluate operand 2*: z=5

LHS		**RHS**
_____	=	_____
x		f(x)
y		f(y)
z		f(z)
dx/dy		K
dx/dz		
dy/dx		
dy/dz		
dz/dx		
dz/dy		

10. Circle the single most efficient operator or operator sequence (according to the heuristics presented in the tutorial) to derive the goal's equation form from the Accepted Equations.

    Accepted Equations
    1. $dy/dz=8*z^4$
    2. $x=2*y$
    3. $z=2$

    Goal: find equation form **x=f(z)** using **operator(s)** _____

    Operators and operator sequences:

    chain rule
    differentiate
    differentiate after *substitute*
    flip derivative
    integrate
    integrate after *chain rule*
    restate
    substitute

11. Circle the single most efficient operator or operator sequence (according to the heuristics presented in the tutorial) to derive the goal's equation form from the Accepted Equations.

Accepted Equations
1. $z=y^{1/2}$
2. $dx/dy=15*z^2$
3. $x=5*z^3$

Goal: find equation form **y=f(z)** using **operator(s) _____**

Operators and operator sequences:

chain rule
differentiate
differentiate after *substitute*
flip derivative
integrate
integrate after *chain rule*
restate
substitute

12. Circle the single most efficient operator or operator sequence (according to the heuristics presented in the tutorial) to derive the goal's equation form from the Accepted Equations.

Accepted Equations
1. $x=5*y^3$
2. $dx/dy=15*y^2$
3. $dz/dy=1/2$

Goal: find equation form **dy/dz=K** using **operator(s) _____**

Operators and operator sequences:

chain rule
differentiate
differentiate after *substitute*
flip derivative
integrate
integrate after *chain rule*
restate
substitute

13. Circle the single most efficient operator or operator sequence (according to the heuristics presented in the tutorial) to derive the goal's equation form from the Accepted Equations.

Accepted Equations
1. $z=y^{1/4}$
2. $x=5*y^2$
3. $z=4$

Goal: find equation form **dx/dz=f(z)** using **operator(s) _____**

Operators and operator sequences:

chain rule
differentiate
differentiate after *substitute*
flip derivative
integrate
integrate after *chain rule*
restate
substitute

14. Circle the single most efficient operator or operator sequence (according to the heuristics presented in the tutorial) to derive the goal's equation form from the Accepted Equations.

Accepted Equations
1. $dz/dy=4*y$
2. $y=3*x^3$
3. $dx/dy=5*y^2$

Goal: find equation form **x=f(y)** using **operator(s) _____**

Operators and operator sequences:

chain rule
differentiate
differentiate after *substitute*
flip derivative
integrate
integrate after *chain rule*
restate
substitute

15. Circle the single most efficient operator or operator sequence (according to the heuristics presented in the tutorial) to derive the goal's equation form from the Accepted Equations.

Accepted Equations
1. $y=4*z^5$
2. $dy/dx=13*x^2$
3. $z=3$

Goal: find equation form **dx/dz=f(z)** using **operator(s) _____**

Operators and operator sequences:

chain rule
differentiate
differentiate after *substitute*
flip derivative
integrate
integrate after *chain rule*
restate
substitute

16. Circle the single most efficient operator or operator sequence (according to the heuristics presented in the tutorial) to derive the goal's equation form from the Accepted Equations.

Accepted Equations
1. $x=3*y^2$
2. $dz/dy=4*y^3$
3. $z=y^4$

Goal: find equation form **dx/dy=f(y)** using **operator(s) _____**

Operators and operator sequences:

chain rule
differentiate
differentiate after *substitute*
flip derivative
integrate
integrate after *chain rule*
restate
substitute

17. Circle the single most efficient operator or operator sequence (according to the heuristics presented in the tutorial) to derive the goal's equation form from the Accepted Equations.

Accepted Equations
1. $dz/dy=12*y^2$
2. $y=½*x$
3. $z=4*y^3$

Goal: find equation form **x=f(y)** using **operator(s) _____**

Operators and operator sequences:

chain rule
differentiate
differentiate after *substitute*
flip derivative
integrate
integrate after *chain rule*
restate
substitute

18. Circle the single most efficient operator or operator sequence (according to the heuristics presented in the tutorial) to derive the goal's equation form from the Accepted Equations.

Accepted Equations
1. $dy/dx=1/3$
2. $dy/dz=2*z^4$
3. $z=3$

Goal: find equation form **x=f(z)** using **operator(s) _____**

Operators and operator sequences:

chain rule
differentiate
differentiate after *substitute*
flip derivative
integrate
integrate after *chain rule*
restate
substitute

19. Circle the equation in the list of Accepted Equations that corresponds to the goal's equation form.

    Accepted Equations
    1. $x=4*y^5$
    2. $dx/dy=5$
    3. $x=80$

    Goal: find equation form **x=K** using **accepted equation** _____

20. Circle the equation in the list of Accepted Equations that corresponds to the goal's equation form.

    Accepted Equations
    1. $y=3*z^2$
    2. $dy/dz=5*z^3$
    3. $dy/dz=14$

    Goal: find equation form **dy/dz=f(z)** using **accepted equation** _____

21. Apply the operator differentiate to its operand and fill in the result in the space after "**yielding**".

    - find equation form **dx/dz=f(z)** using operator **differentiate** **yielding** _____
      - *differentiate operand*: $x=2*z^4$

22. Apply the operator *integrate* to its operand and fill in the result in the space after "**yielding**".

    - find equation form **dy/dx=f(x)** using operator **integrate** **yielding** _____
      - *integrate operand*: $dz/dy=8*y^3$

23. Apply the operator *chain rule* to its operands and fill in the result in the space after "**yielding**".

    - find equation form **dx/dz=f(z)** using operator **chain rule** **yielding** _____
      - *chain rule operand 1*: $dx/dy=2$
      - *chain rule operand 2*: $dy/dz=5*z^4$

24. Apply the operator *flip derivative* to its operand and fill in the result in the space after "**yielding**".

- find equation form **dy/dz=K** using operator **flip derivative yielding** _____
  - *flip derivative operand*: dz/dy=½

25. Apply the operator *restate* to its operand and fill in the result in the space after "**yielding**".

- find equation form **x=f(y)** using operator **restate yielding** _____
  - *restate operand*: y = ½ * x

26. Apply the operator *restate* to its operand and fill in the result in the space after "**yielding**".

- find equation form **x=f(y)** using operator **restate yielding** _____
  - *restate operand*: $y=x^{1/2}$

27. Apply the operator *substitute* to its operands and fill in the result in the space after "**yielding**".

- find equation form **x=f(z)** using operator **substitute yielding** _____
  - *substitute operand 1*: x=2*y
  - *substitute operand 2*: $y=3*z^3$

28. Apply the operator *evaluate* to its operands and fill in the result in the space after "**yielding**".

- find equation form **dx/dz=K** using operator **evaluate yielding** _____
  - *evaluate operand 1*: $dx/dz=5*z^2$
  - *evaluate operand 2*: z=3

APPENDIX C. Calculus Tutor Tips

Tips for Using the Calculus Tutor

The Calculus Tutor will lead you through the problems using the knowledge you have gained in the tutorial and helping you out as necessary. Some tips for using the Calculus Tutor are listed below.

1. **To progress through the problems, click in the Goals Window on any goal that is highlighted in blue or red**. You may have to click on a goal more than once to get the Calculus Tutor to respond. Then follow the prompts displayed in the Prompt Window.

   Colors used in Goals Window

   ***Blue*:** Click to pursue goal.

   ***Red*:** Error. Click to correct the error.

   ***Black*:** Previously established correct entry.

   ***Green*:** Correct entry. Entry will turn black after next click in Goals Window.

2. When a message is displayed in the Prompt Window, **the corresponding goal will be highlighted in bold type in the Goals Window**.

3. **Simplify new equations** before entering them in the New Equation Window. All equations should have either a single variable or a derivative expression (e.g., x or dx/dz) on the left-hand-side (LHS), and all multiplications, divisions and exponentiations should be carried out. Examples:

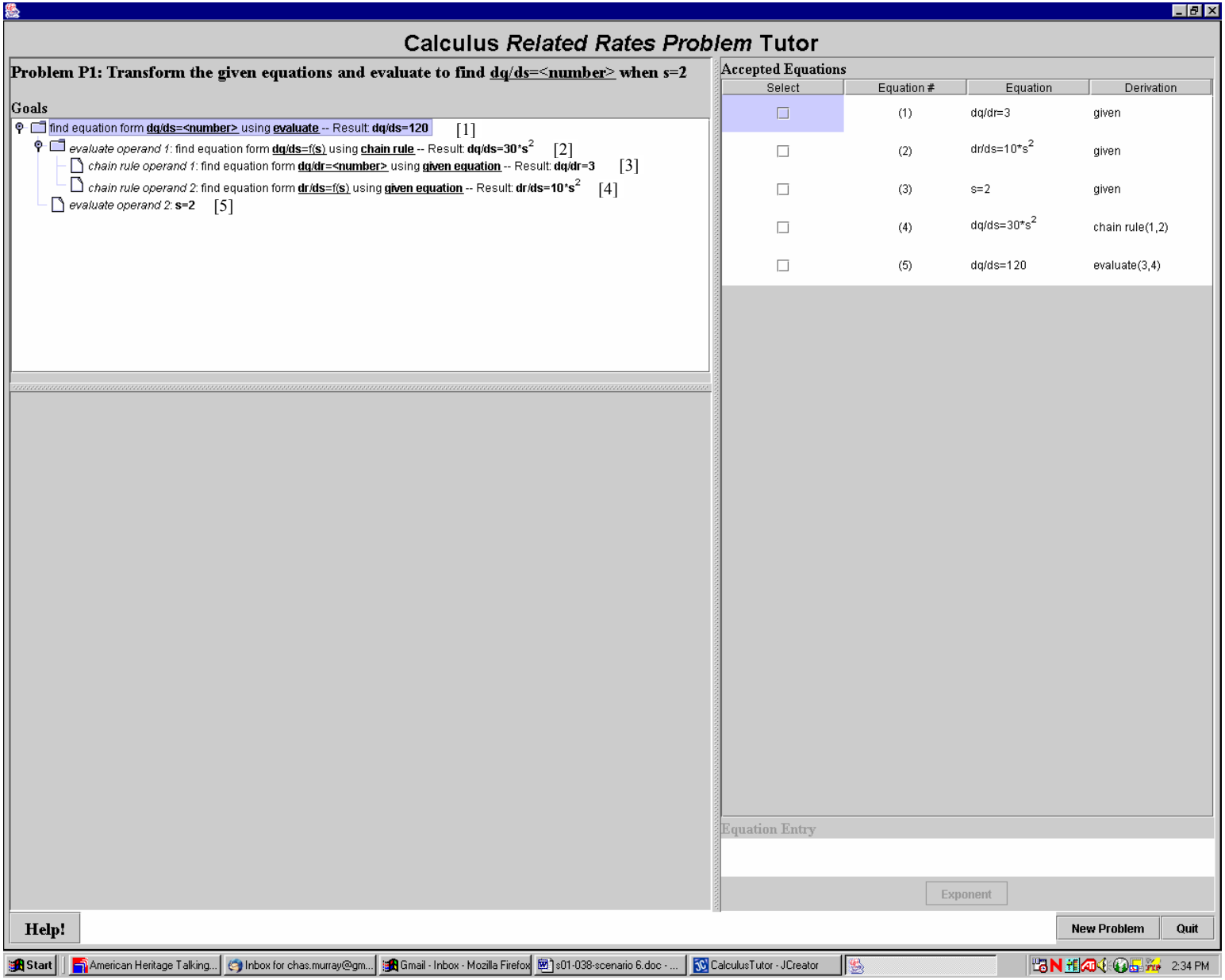
   - Instead of entering 1/2*(3*z^2)=1/6*x, enter x=9*z^2.
   - Instead of entering 1/2*(3*2^2)=dx/dz*1/6, enter dx/dz=36.

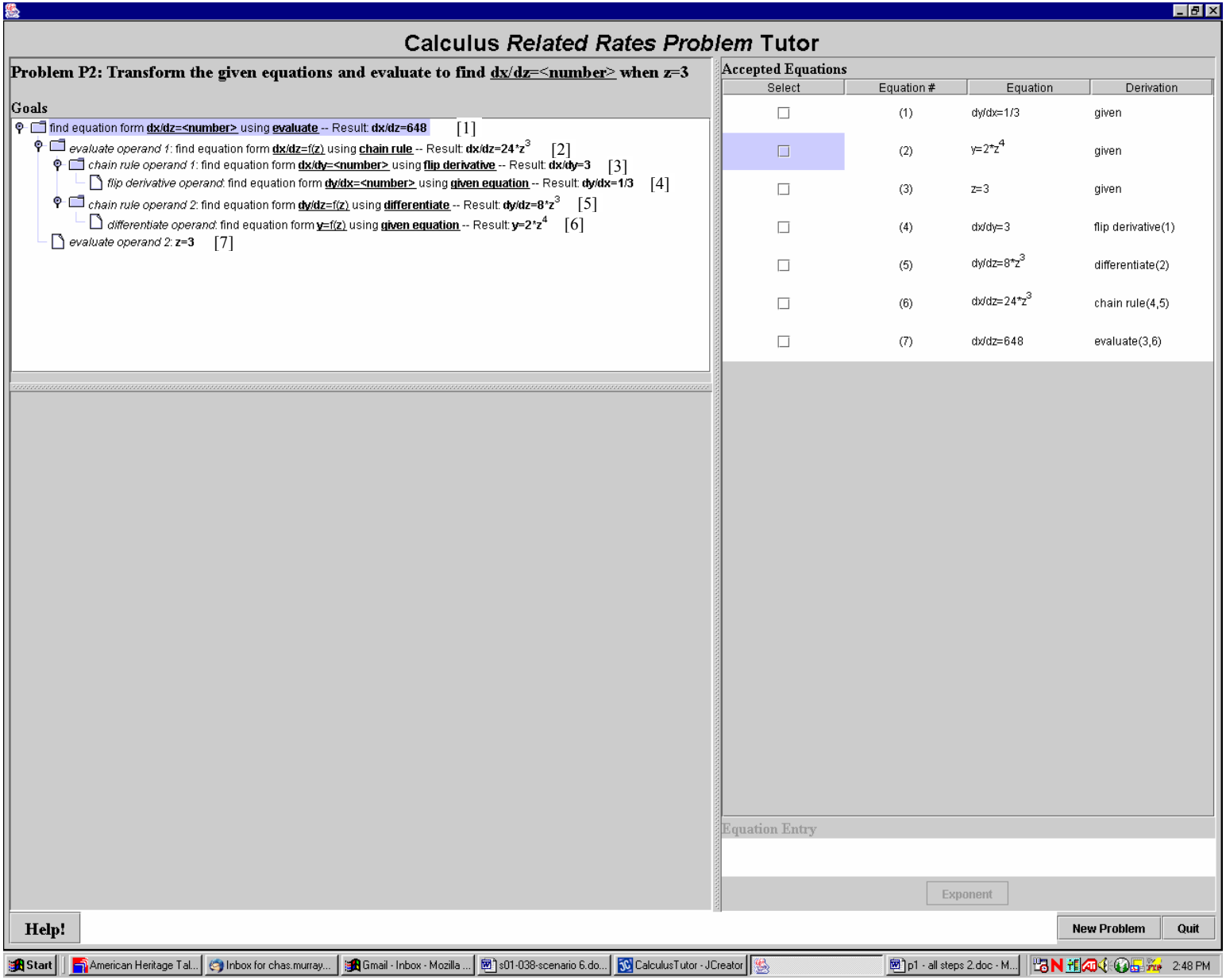
4. **To enter an exponent in the Equation Entry Window** (e.g., the ^{2} in y^2), you can either click on the Exponent button below the window, or enter the symbol ^ followed by the exponent – e.g., x=3*y^2 is automatically formatted as x=3*y^2. When you are through entering an exponent, you can click on the Exponent button to resume entering regular text.

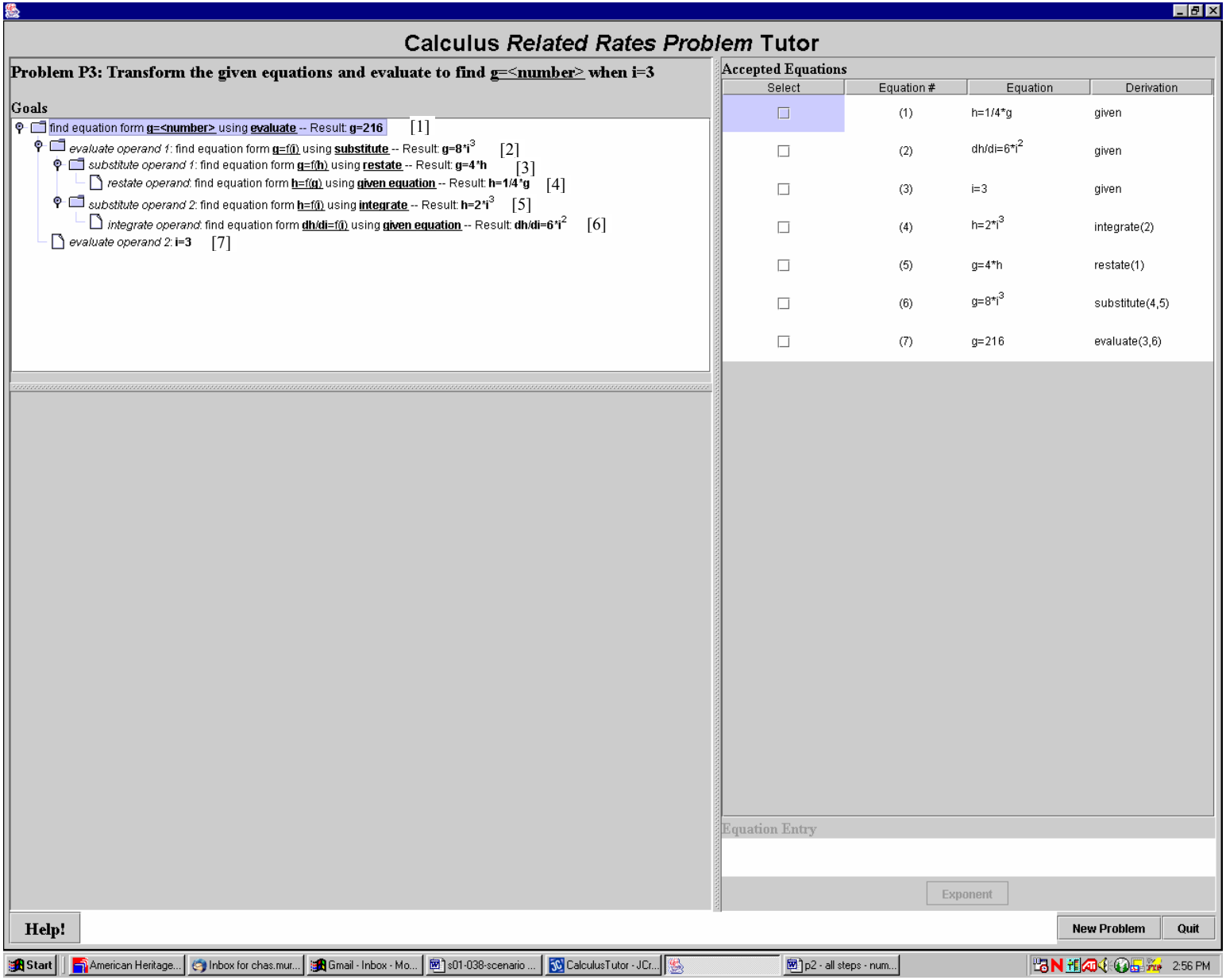
Symbols & Definitions

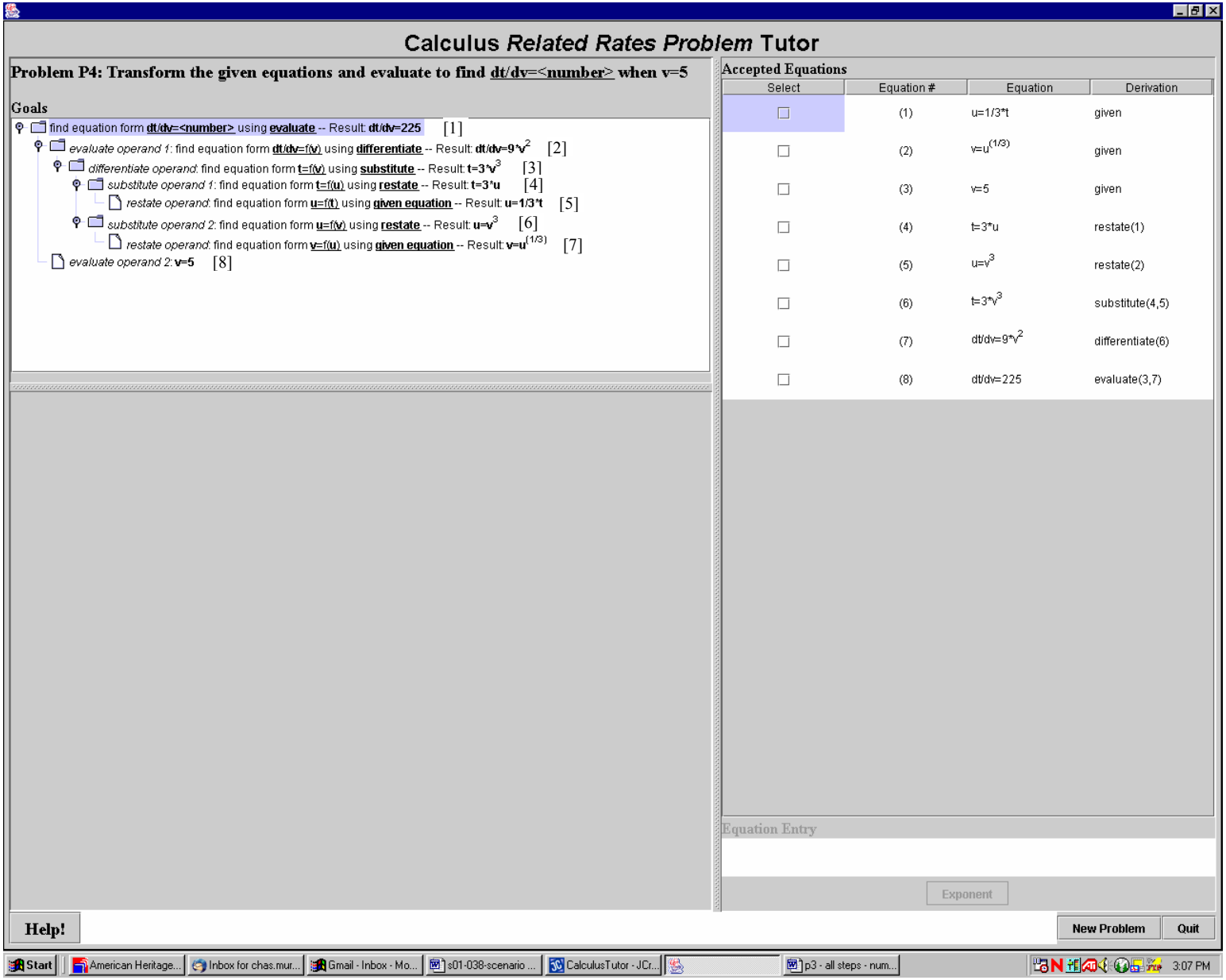
- ***** – Symbol for multiplication. E.g., 2*2=4
- **/** – Symbol for division. E.g., 4/2=2
- **derivative expression** – E.g., dx/dy
- **derivative equation** – An equation with a derivative expression
- **derivative form** – An equation or equation form with a derivative expression
- **equation form** – E.g., for $dx/dy=2*y^3$, the equation form is dx/dy=f(y)
- **f(<variable>)** – A function of <variable>. E.g.: for $x=2*y^3$, x=f(y); for x=2*v, x=f(v)
- **given** – An Accepted Equation that was given along with the problem statement
- **LHS** – The left-hand-side of an equation. E.g., x in $x=2*y^3$
- **multiplicative inverse** – Example 1: dx/dy → dy/dx. Example 2: 5 → 1/5
- **<number>** – In an equation form, <number> represents any number (no variables).
- **operator** – A mathematical operation. E.g., *differentiate*, *chain rule, restate,* etc.
- **operand** – The input equation that is transformed by an operator
- **regular equation** – An equation without a derivative expression
- **regular form** – An equation or equation form without a derivative expression
- **RHS** – The right-hand-side of an equation. E.g., $2*y^3$ in $x=2*y^3$

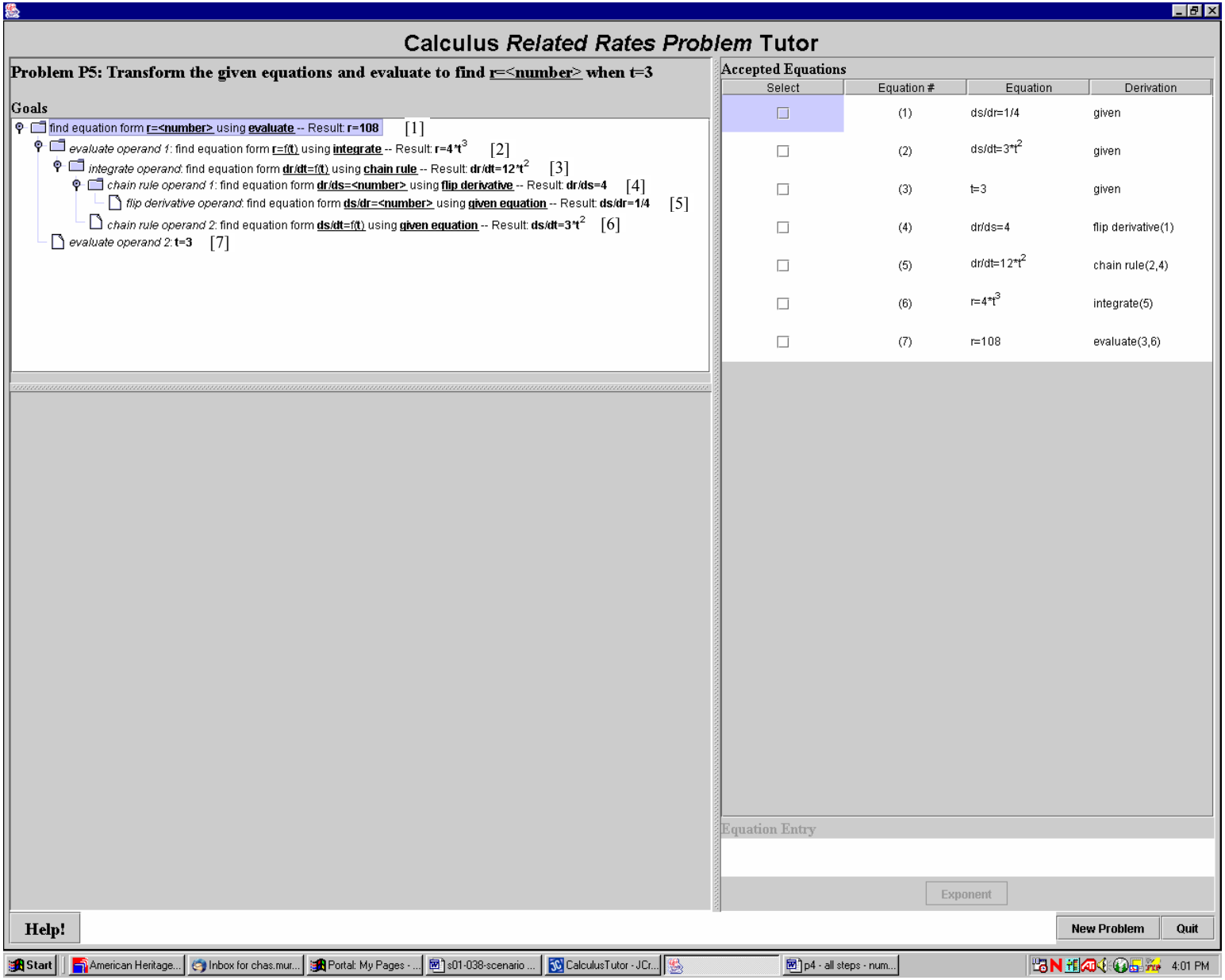
APPENDIX D. Expanded Problem Screen Shots with Goal Numbers


Calculus Related Rates Problem Tutor
Problem P1: Transform the given equations and evaluate to find dq/ds=<number> when s=2
Goals
find equation form dq/ds=<number> using evaluate -- Result: dq/ds=120 [1]
evaluate operand 1: find equation form dq/ds=f(s) using chain rule -- Result: dq/ds=30*s^2 [2]
chain rule operand 1: find equation form dq/dr=<number> using given equation -- Result: dq/dr=3 [3]
chain rule operand 2: find equation form dr/ds=f(s) using given equation -- Result: dr/ds=10*s^2 [4]
evaluate operand 2: s=2 [5]
Accepted Equations
Select
Equation #
Equation
Derivation
(1) dq/dr=3 given
(2) dr/ds=10*s^2 given
(3) s=2 given
(4) dq/ds=30*s^2 chain rule(1,2)
(5) dq/ds=120 evaluate(3,4)
Equation Entry
Exponent
Help!
New Problem
Quit
Start
American Heritage Talking...
Inbox for chas.murray@gm...
Gmail - Inbox - Mozilla Firefox
s01-038-scenario 6.doc - ...
CalculusTutor - JCreator
2:34 PM

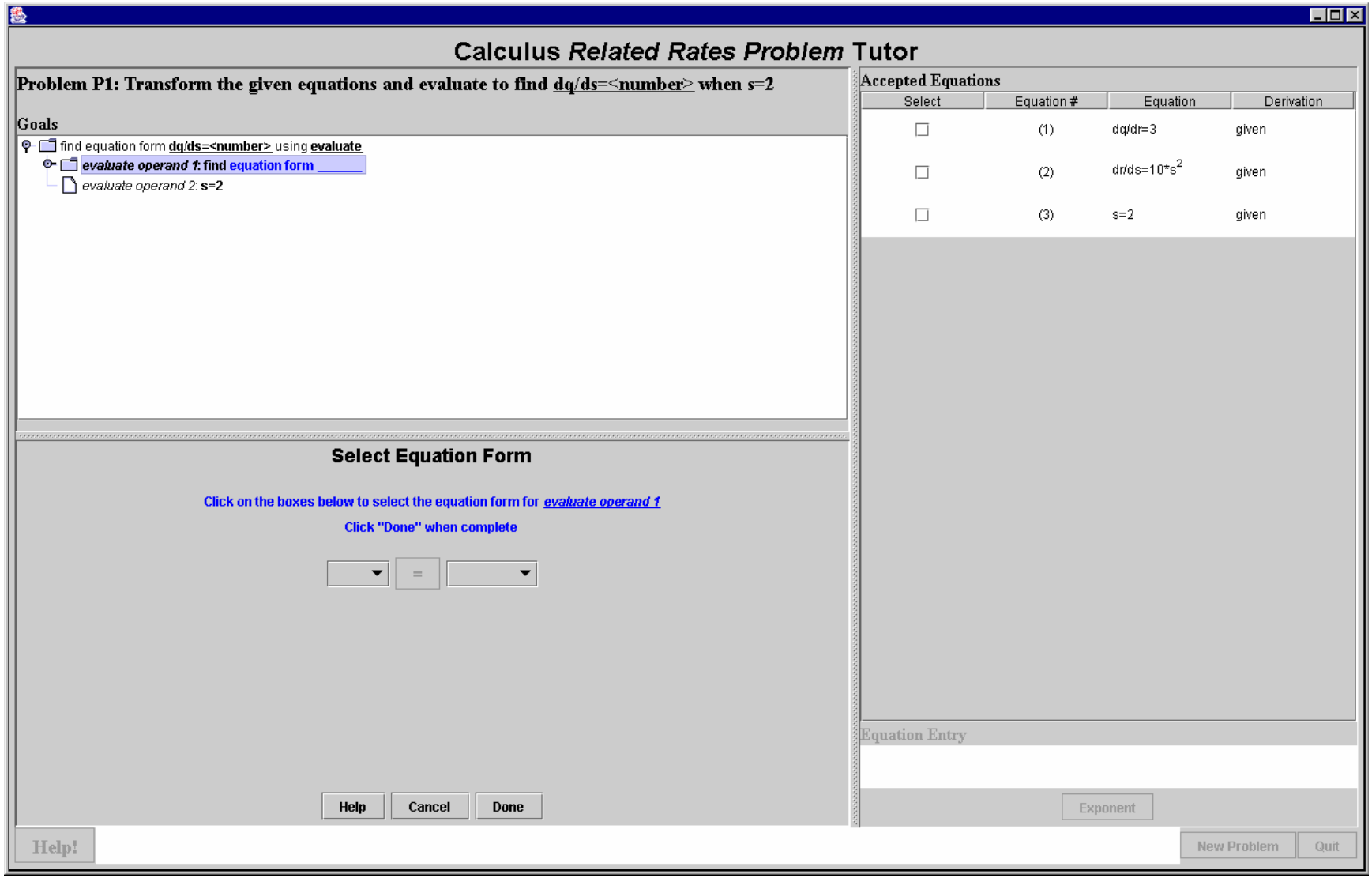

Calculus Related Rates Problem Tutor
Problem P2: Transform the given equations and evaluate to find dx/dz=<number> when z=3
Goals
find equation form dx/dz=<number> using evaluate -- Result: dx/dz=648 [1]
evaluate operand 1: find equation form dx/dz=f(z) using chain rule -- Result: dx/dz=24*z^3 [2]
chain rule operand 1: find equation form dx/dy=<number> using flip derivative -- Result: dx/dy=3 [3]
flip derivative operand: find equation form dy/dx=<number> using given equation -- Result: dy/dx=1/3 [4]
chain rule operand 2: find equation form dy/dz=f(z) using differentiate -- Result: dy/dz=8*z^3 [5]
differentiate operand: find equation form y=f(z) using given equation -- Result: y=2*z^4 [6]
evaluate operand 2: z=3 [7]
Accepted Equations
Select
Equation #
Equation
Derivation
(1) dy/dx=1/3 given
(2) y=2*z^4 given
(3) z=3 given
(4) dx/dy=3 flip derivative(1)
(5) dy/dz=8*z^3 differentiate(2)
(6) dx/dz=24*z^3 chain rule(4,5)
(7) dx/dz=648 evaluate(3,6)
Equation Entry
Exponent
Help!
New Problem
Quit
Start
American Heritage Tal...
Inbox for chas.murray...
Gmail - Inbox - Mozilla ...
s01-038-scenario 6.do...
CalculusTutor - JCreator
p1 - all steps 2.doc - M...
2:48 PM

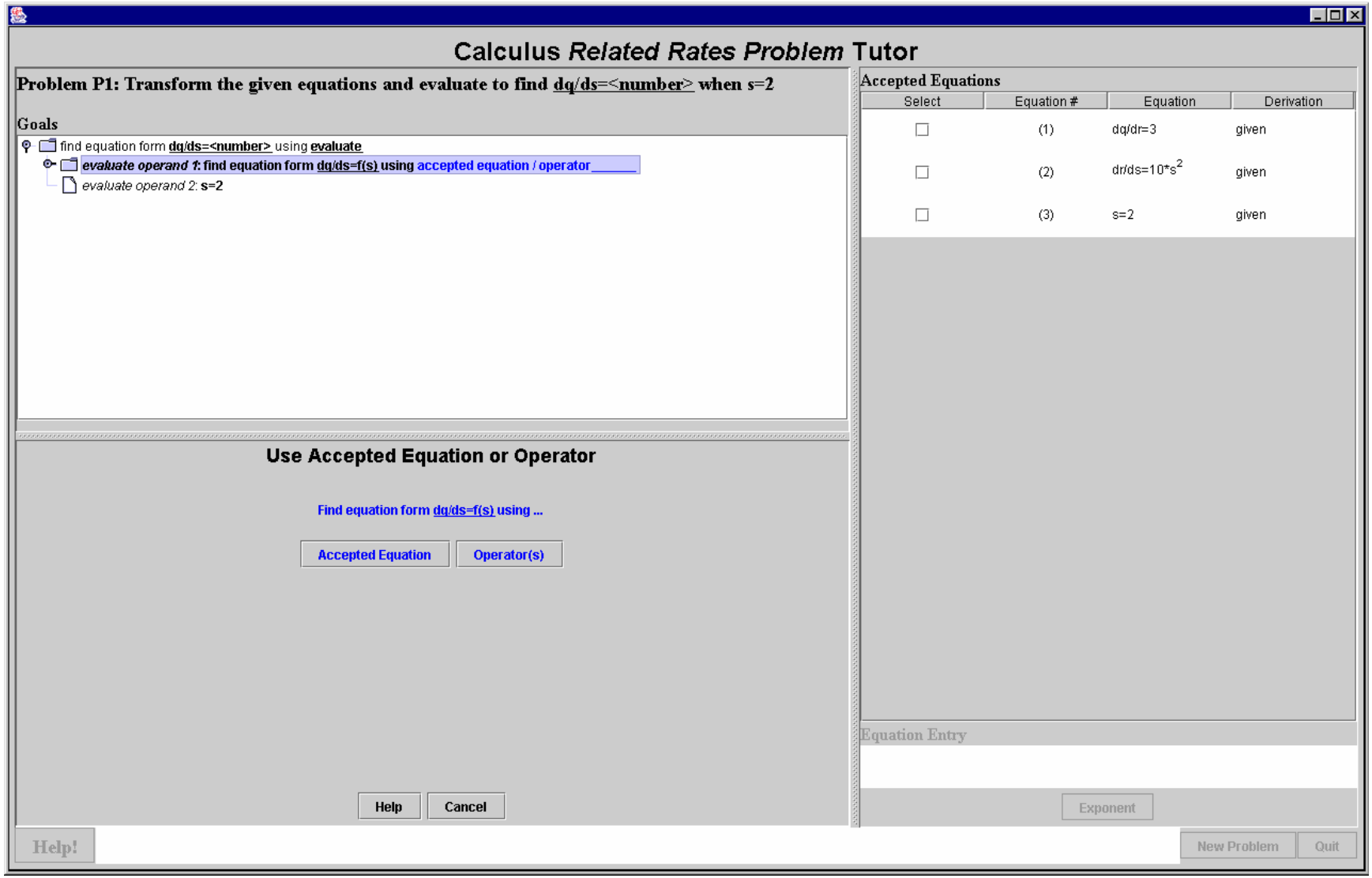

Calculus Related Rates Problem Tutor
Problem P3: Transform the given equations and evaluate to find g=<number> when i=3
Goals
find equation form g=<number> using evaluate -- Result: g=216 [1]
evaluate operand 1: find equation form g=f(i) using substitute -- Result: g=8*i^3 [2]
substitute operand 1: find equation form g=f(h) using restate -- Result: g=4*h [3]
restate operand: find equation form h=f(g) using given equation -- Result: h=1/4*g [4]
substitute operand 2: find equation form h=f(i) using integrate -- Result: h=2*i^3 [5]
integrate operand: find equation form dh/di=f(i) using given equation -- Result: dh/di=6*i^2 [6]
evaluate operand 2: i=3 [7]
Accepted Equations
Select
Equation #
Equation
Derivation
(1) h=1/4*g given
(2) dh/di=6*i^2 given
(3) i=3 given
(4) h=2*i^3 integrate(2)
(5) g=4*h restate(1)
(6) g=8*i^3 substitute(4,5)
(7) g=216 evaluate(3,6)
Equation Entry
Exponent
Help!
New Problem
Quit
Start
American Heritage...
Inbox for chas.mur...
Gmail - Inbox - Mo...
s01-038-scenario ...
CalculusTutor - JCr...
p2 - all steps - num...
2:56 PM

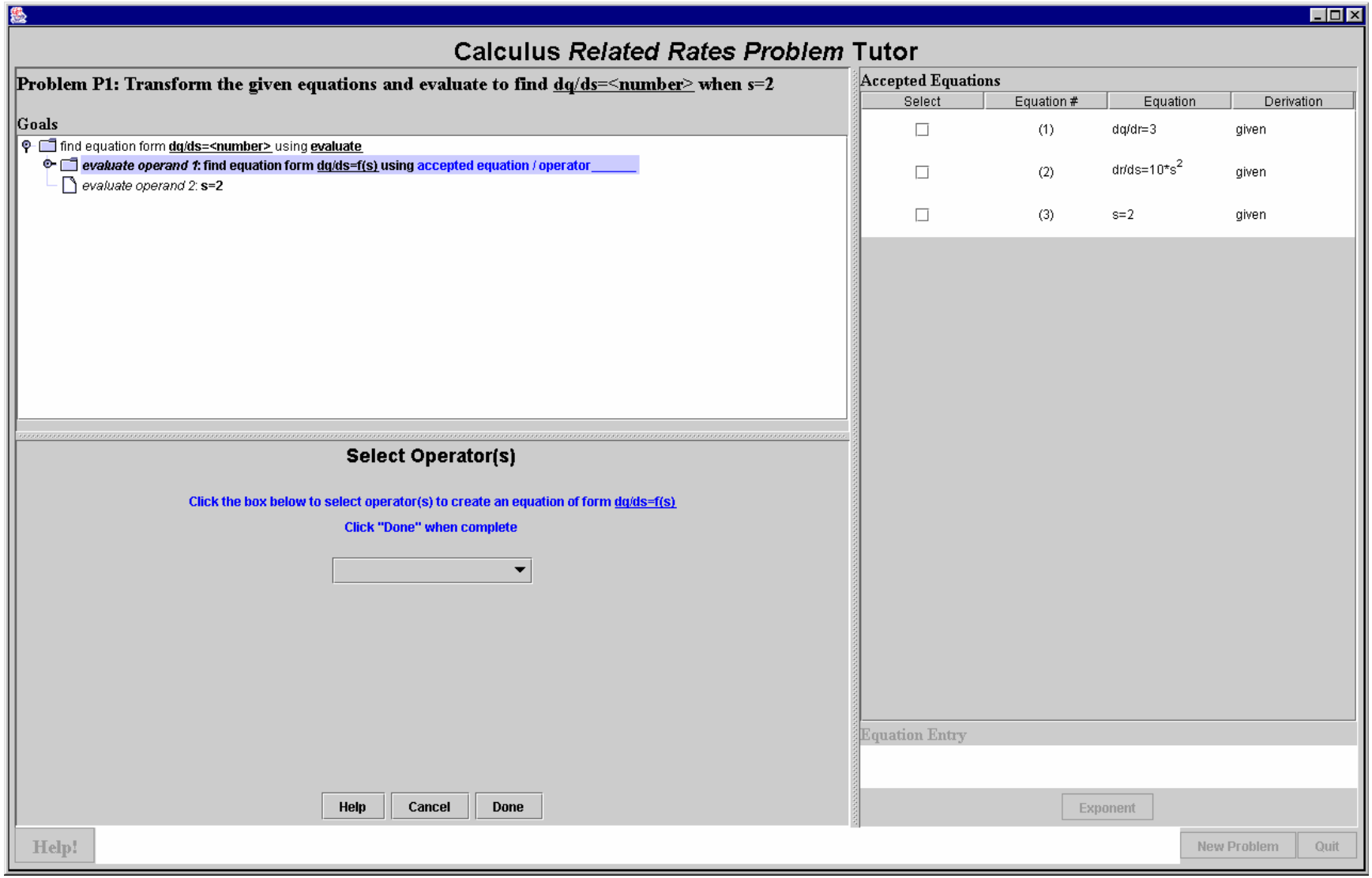

Calculus Related Rates Problem Tutor
Problem P4: Transform the given equations and evaluate to find dt/dv=<number> when v=5
Goals
find equation form dt/dv=<number> using evaluate -- Result: dt/dv=225 [1]
evaluate operand 1: find equation form dt/dv=f(v) using differentiate -- Result: dt/dv=9*v^2 [2]
differentiate operand: find equation form t=f(v) using substitute -- Result: t=3*v^3 [3]
substitute operand 1: find equation form t=f(u) using restate -- Result: t=3*u [4]
restate operand: find equation form u=f(t) using given equation -- Result: u=1/3*t [5]
substitute operand 2: find equation form u=f(v) using restate -- Result: u=v^3 [6]
restate operand: find equation form v=f(u) using given equation -- Result: v=u^(1/3) [7]
evaluate operand 2: v=5 [8]
Accepted Equations
Select
Equation #
Equation
Derivation
(1) u=1/3*t given
(2) v=u^(1/3) given
(3) v=5 given
(4) t=3*u restate(1)
(5) u=v^3 restate(2)
(6) t=3*v^3 substitute(4,5)
(7) dt/dv=9*v^2 differentiate(6)
(8) dt/dv=225 evaluate(3,7)
Equation Entry
Exponent
Help!
New Problem
Quit
Start
American Heritage...
Inbox for chas.mur...
Gmail - Inbox - Mo...
s01-038-scenario ...
CalculusTutor - JCr...
p3 - all steps - num...
3:07 PM

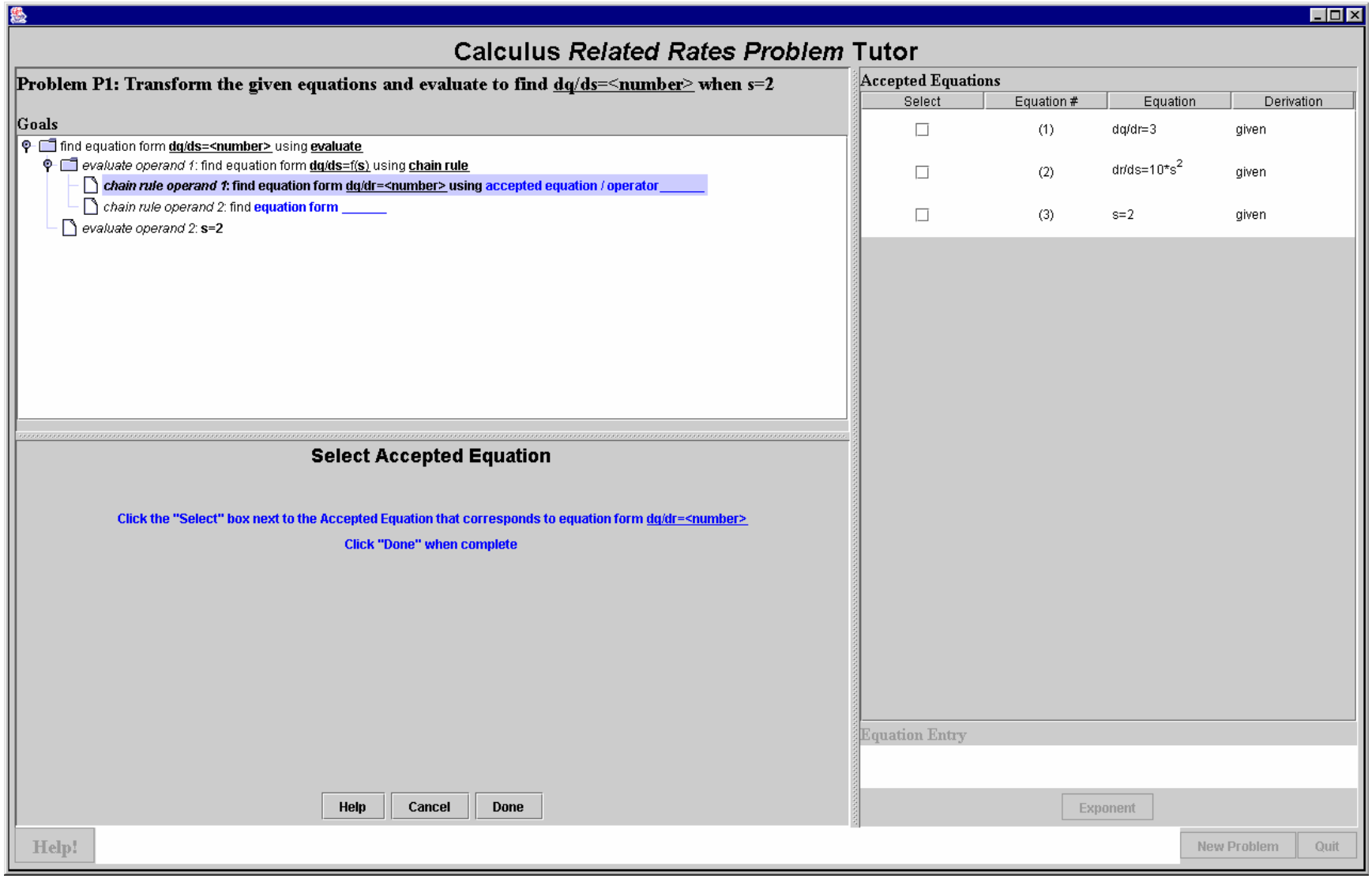

Calculus Related Rates Problem Tutor
Problem P5: Transform the given equations and evaluate to find r=<number> when t=3
Goals
find equation form r=<number> using evaluate -- Result: r=108 [1]
evaluate operand 1: find equation form r=f(t) using integrate -- Result: r=4*t^3 [2]
integrate operand: find equation form dr/dt=f(t) using chain rule -- Result: dr/dt=12*t^2 [3]
chain rule operand 1: find equation form dr/ds=<number> using flip derivative -- Result: dr/ds=4 [4]
flip derivative operand: find equation form ds/dr=<number> using given equation -- Result: ds/dr=1/4 [5]
chain rule operand 2: find equation form ds/dt=f(t) using given equation -- Result: ds/dt=3*t^2 [6]
evaluate operand 2: t=3 [7]
Accepted Equations
Select
Equation #
Equation
Derivation
(1) ds/dr=1/4 given
(2) ds/dt=3*t^2 given
(3) t=3 given
(4) dr/ds=4 flip derivative(1)
(5) dr/dt=12*t^2 chain rule(2,4)
(6) r=4*t^3 integrate(5)
(7) r=108 evaluate(3,6)
Equation Entry
Exponent
Help!
New Problem
Quit
Start
American Heritage...
Inbox for chas.mur...
Portal: My Pages - ...
s01-038-scenario ...
CalculusTutor - JCr...
p4 - all steps - num...
4:01 PM

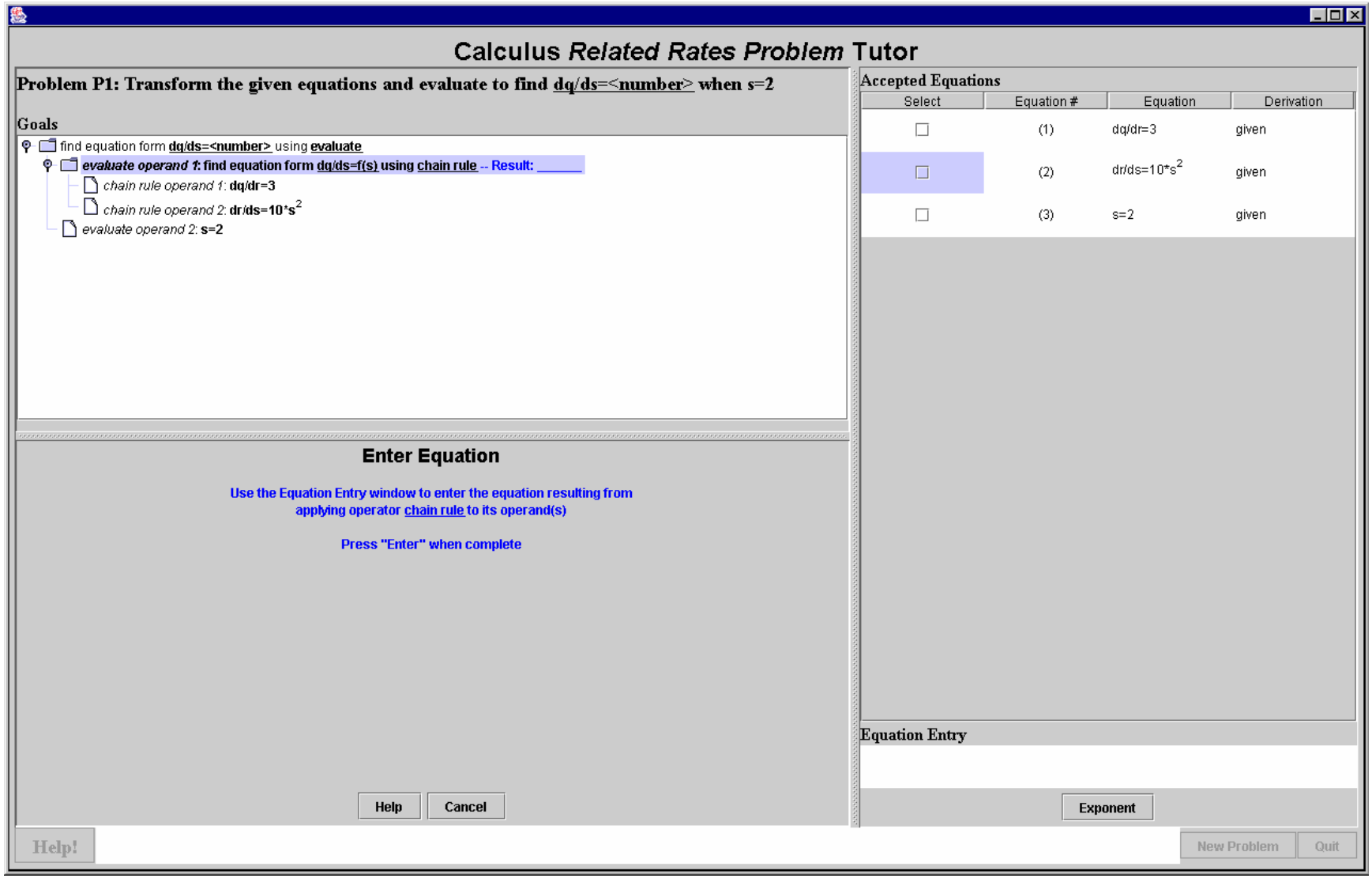
APPENDIX E. Screen Shots of Dialog Windows


Calculus Related Rates Problem Tutor
Problem P1: Transform the given equations and evaluate to find dq/ds=<number> when s=2
Goals
find equation form dq/ds=<number> using evaluate
evaluate operand 1: find equation form _____
evaluate operand 2: s=2
Select Equation Form
Click on the boxes below to select the equation form for evaluate operand 1
Click "Done" when complete
=
Help
Cancel
Done
Accepted Equations
Select
Equation #
Equation
Derivation
(1)
dq/dr=3
given
(2)
dr/ds=10*s^2
given
(3)
s=2
given
Equation Entry
Exponent
Help!
New Problem
Quit

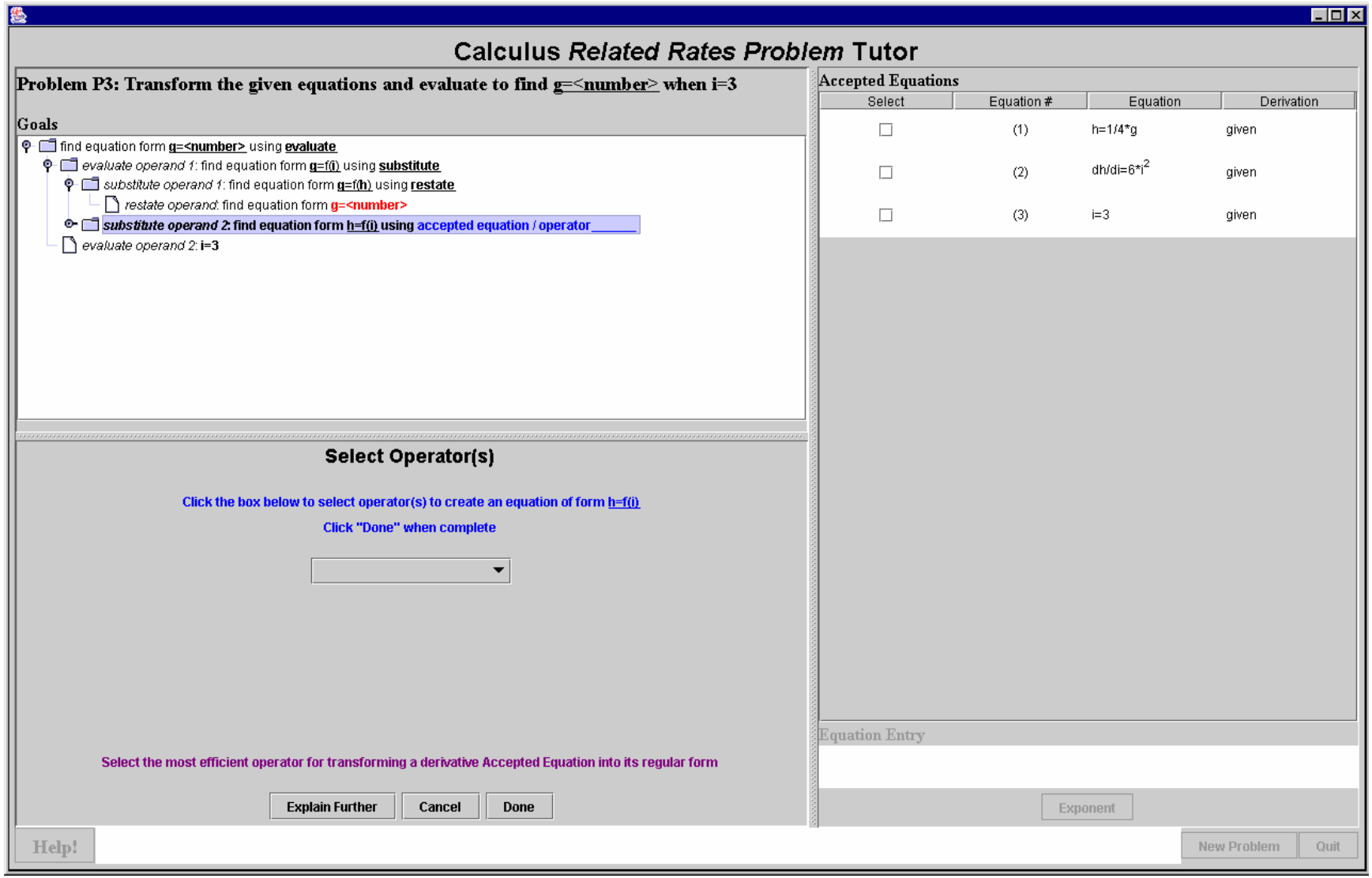

Calculus Related Rates Problem Tutor
Problem P1: Transform the given equations and evaluate to find dq/ds=<number> when s=2
Goals
find equation form dq/ds=<number> using evaluate
evaluate operand 1: find equation form dq/ds=f(s) using accepted equation / operator _____
evaluate operand 2: s=2
Use Accepted Equation or Operator
Find equation form dq/ds=f(s) using ...
Accepted Equation
Operator(s)
Help
Cancel
Accepted Equations
Select
Equation #
Equation
Derivation
(1)
dq/dr=3
given
(2)
dr/ds=10*s^2
given
(3)
s=2
given
Equation Entry
Exponent
Help!
New Problem
Quit


Calculus Related Rates Problem Tutor
Problem P1: Transform the given equations and evaluate to find dq/ds=<number> when s=2
Goals
find equation form dq/ds=<number> using evaluate
evaluate operand 1: find equation form dq/ds=f(s) using accepted equation / operator _____
evaluate operand 2: s=2
Select Operator(s)
Click the box below to select operator(s) to create an equation of form dq/ds=f(s)
Click "Done" when complete
Help
Cancel
Done
Accepted Equations
Select
Equation #
Equation
Derivation
(1)
dq/dr=3
given
(2)
dr/ds=10*s^2
given
(3)
s=2
given
Equation Entry
Exponent
Help!
New Problem
Quit


Calculus Related Rates Problem Tutor
Problem P1: Transform the given equations and evaluate to find dq/ds=<number> when s=2
Goals
find equation form dq/ds=<number> using evaluate
evaluate operand 1: find equation form dq/ds=f(s) using chain rule
chain rule operand 1: find equation form dq/dr=<number> using accepted equation / operator _____
chain rule operand 2: find equation form _____
evaluate operand 2: s=2
Select Accepted Equation
Click the "Select" box next to the Accepted Equation that corresponds to equation form dq/dr=<number>
Click "Done" when complete
Help
Cancel
Done
Accepted Equations
Select
Equation #
Equation
Derivation
(1)
dq/dr=3
given
(2)
dr/ds=10*s^2
given
(3)
s=2
given
Equation Entry
Exponent
Help!
New Problem
Quit


Calculus Related Rates Problem Tutor
Problem P1: Transform the given equations and evaluate to find dq/ds=<number> when s=2
Goals
find equation form dq/ds=<number> using evaluate
evaluate operand 1: find equation form dq/ds=f(s) using chain rule -- Result: _____
chain rule operand 1: dq/dr=3
chain rule operand 2: dr/ds=10*s^2
evaluate operand 2: s=2
Enter Equation
Use the Equation Entry window to enter the equation resulting from applying operator chain rule to its operand(s)
Press "Enter" when complete
Help
Cancel
Accepted Equations
Select
Equation #
Equation
Derivation
(1)
dq/dr=3
given
(2)
dr/ds=10*s^2
given
(3)
s=2
given
Equation Entry
Exponent
Help!
New Problem
Quit

APPENDIX F. Sample Scenario Description



Student's next action – **Help Request** for problem P3, goal 5 – correct entry is Operator integrate

substitute operand 2: find equation form h=f(i) using accepted equation / operator _____

Student History Correct Entries – **27** Errors – **48** Help Requests – **45** **Help Requests *Not* Discouraged**

Student and tutor actions related to the concept required for this step, **heuristic: use operator *integrate***

A. Pretest problem: Correct

B. Previous student and tutor actions in this or other problems:

1. This step: problem P3, goal 5 – correct entry is Operator integrate

   Student: Clicked general **"Help!"** button

   Tutor: **Select Operator window** with message:

   Select operator(s) that will *efficiently* transform Accepted Equation(s) into an equation of the form h=f(i)

   Student: Clicked "Explain Further"

   Tutor: Message:

   Select the most efficient operator for transforming a derivative Accepted Equation into its regular form

Tutorial Response Ratings

Evaluator ID: _____ Date: _____

Rate the following possible tutorial responses to this situation on a scale of 1 to 5, *where* 1 *is your worst rating and* 5 *is your best.*
Note: These ratings are not comparisons among the possible responses. You may give more than one response the same rating.

Select Operator(s) window (unless otherwise specified) with one of the following messages:

	Rating	Response
A.	_____	Select operator *integrate*
B.	_____	(No message)
C.	_____	Operator *integrate* can transform an Accepted Equation that is a derivative version of the goal's regular equation form Example: Operator *integrate* can transform an Accepted Equation of form da/db=f(b) into an equation of form a=f(b)
D.	_____	Select the most efficient operator for transforming a derivative Accepted Equation into its regular form
E.	_____	Select operator(s) that will *efficiently* transform Accepted Equation(s) into an equation of the form h=f(i)

Any Comments About Tutorial Responses on Previous Page

Your Best Tutorial Response

If the tutor should respond, write your best single tutorial response given that the response must:

- *take one turn*
- *be related to a step that the student can take next in the interface*
- *not ask the student to do something outside the interface*

If the tutor should not respond, write "none."

APPENDIX G. Sample Help Messages

Sample Help Messages for Dialog Select Equation Form

Prompt

To use chain rule to create an equation of form dx/dz=f(z), *chain rule operand 1* must be in what form?

Hint

chain rule operand 1 must be a *derivative* equation involving:
... (1) one variable that *is* in dx/dz=f(z)
... (2) one variable that *is not* in dx/dz=f(z)
…(3) a number

Teach

When *chain rule's* operands have the following forms:
... *operand 1*: d<variable 1> / d<variable 2> = <number>
... *operand 2*: d<variable 2> / d<variable 3> = f(<variable 3>)

the resulting equation will have this form:
... *result*: d<variable 1> / d<variable 3> = f(<variable 3>)

Example: To create an equation of the form da/dc=f(c) when the other variable in the Accepted Equations is b, *operand 1's* equation form must be da/db=<number>

Do

Select equation form dx/dy=<number>:

Sample Help Messages for Dialog Select Operator(s)

Prompt

Select operator(s) that will *efficiently* transform Accepted Equation(s) into an equation of the form g=f(i)

Hint

Select the most efficient operator considering:
(1) at least 2 out of 3 of the first two given equations and *evaluate operand 1's* equation form are in regular form
(2) *evaluate operand 1's* equation form, g=f(i), is in regular form

Teach

When:
(1) at least 2 out of 3 of the first two given equations and *evaluate operand 1's* equation form are in regular form
... *and* ...
(2) *evaluate operand 1's* equation form, g=f(i), is in regular form

operator *substitute* minimizes transforming equations between regular and derivative form

Do

Select operator *substitute*

Sample Help Messages for Dialog Select Accepted Equation

Prompt

Match equation form dr/ds=f(s) with one of the accepted equations

Hint

Select an equation that has the same variables as equation form dr/ds=f(s)

Teach

Select an Accepted Equation by matching both the LHS and RHS of equation form dr/ds=f(s) as follows:

- On the LHS, the equation form and the Accepted Equation must have the same variable or derivative expression. For example, both should have a or both should have da/db
- On the RHS:
  ... If the equation form has, e.g., *f(b)*, the Accepted Equation must have an expression using the same variable – e.g., $3*b^5$ – on the RHS
  ... If the equation form has <number>, the Accepted Equation must have only a number on the RHS

Do

Select equation $dr/ds=10*s^2$

Sample Help Messages for Dialog Enter Equation

Prompt

Apply *differentiate* to $y=2*z^4$, then simplify the resulting equation

Hint

- Use the power rule to transform $y=2*z^4$ into a derivative equation
- Then simplify the resulting equation

Teach

differentiate transforms a regular equation into a derivative equation using the power rule

Example - *operand*: $a = 2 * b^3$ ==> $da/db = 3 * 2 * b^2$ ==> *result*: $da/db = 6 * b^2$

(1) Change the LHS into a derivative expression:
..... the numerator is d plus the variable on the operand's LHS
..... the denominator is d plus the variable on the operand's RHS
(2) Multiply the RHS by the exponent
(3) Subtract 1 from the exponent
(4) Simplify the resulting equation

Do

Enter equation $dy/dz=8*z^3$

BIBLIOGRAPHY

Aist, G., Kort, B., Reilly, R., Mostow, J., & Picard, R. (2002). Adding human-provided emotional scaffolding to an automated reading tutor that listens increases student persistence. In S. A. Cerri & G. Gouarderes & F. Paraguacu (Eds.), *Intelligent Tutoring Systems, 4th International Conference, ITS 2002*, 992.

Albrecht, D.W., Zukerman, I., & Nicholson, A.E. (1998). Bayesian models for keyhole plan recognition in an adventure game. *User Modeling and User-Adapted Interaction*, 8, 5-47.

Aleven, V., & Koedinger, K.R. (2000). Limitations of student control: Do students know when they need help? In G. Gauthier & C. Frasson & K. VanLehn (Eds.), *Intelligent Tutoring Systems, 5th International Conference, ITS 2000*, 292-303.

Aleven, V., McLaren, B.M., & Koedinger, K.R. (2004). Toward Tutoring Help Seeking: Applying Cognitive Modeling to Meta-Cognitive Skills. In S. Karabenick & R. Newman (Eds.), *Seventh International Conference on Intelligent Tutoring Systems, ITS 2004*, 227-239.

Anderson, J.R. (1983). *The Architecture of Cognition*. Cambridge, MA: Harvard University Press.

Anderson, J.R. (1993). *Rules of the Mind*. Hillsdale, NJ: Lawrence Erlbaum Associates.

Anderson, J.R., Corbett, A.T., Koedinger, K.R., & Pelletier, R. (1995). Cognitive Tutors: Lessons Learned. *The Journal of the Learning Sciences*, 4(2), 167-207.

Anderson, J.R., & Lebiere, C. (1998). *The atomic components of thought*: Mahwah, NJ: Erlbaum.

Arroyo, I., & Woolf, B.P. (2001). Improving student models by reasoning about cognitive ability, emotions and gender. In M. Bauer & P. J. Gmytrasiewicz & J. Vassileva (Eds.), *User Modeling 2001: Eighth International Conference*, 265-267.

Baker, R.S., Corbett, A.T., & Koedinger, K.R. (2004). Detecting Student Misuse of Intelligent Tutoring Systems. In S. Karabenick & R. Newman (Eds.), *Seventh International Conference on Intelligent Tutoring Systems, ITS 2004*, 531-540.

Beck, J.E., & Woolf, B.P. (2000). High-level student modeling with machine learning. In G. Gauthier & C. Frasson & K. VanLehn (Eds.), *Intelligent Tutoring Systems, 5th International Conference, ITS 2000*, 584-593.

Binder, J., Koller, D., Russell, S.J., & Kanazawa, K. (1997). Adaptive probabilistic networks with hidden variables. *Machine Learning Journal*, 29(2-3), 213-244.

Bloom, B.S. (1984). The 2 sigma problem: The search for methods of group instruction as effective as one-to-one tutoring. *Educational Researcher*, 13(6), 4-16.

Boyen, X., & Koller, D. (1998). Tractable inference for complex stochastic processes, *Fourteenth Annual Conference on Uncertainty in Artificial Intelligence (UAI-98)*, 33-42.

Cheng, J., Bell, D., & Liu, W. (1998). Learning Bayesian networks from data: An efficient approach based on information theory. On the World Wide Web: www.cs.ualberta.ca/~jcheng/bnpc.htm

Cheng, J., & Druzdzel, M.J. (2000). AIS-BN: An adaptive importance sampling algorithm for evidential reasoning in large Bayesian networks. *Journal of Artificial Intelligence Research*, 13, 155-188.

Chi, M.T.H., Bassok, M., Lewis, M.W., Reimann, P., & Glaser, R. (1989). Self-explanations: How students study and use examples in learning to solve problems. *Cognitive Science*, 15, 145-182.

Chiu, B.C., & Webb, G.I. (1998). Using decision trees for agent modeling: Improving prediction performance. *User Modeling and User-Adapted Interaction*, 8, 131-152.

Clemen, R.T. (1996). *Making Hard Decisions: An Introduction to Decision Analysis*. New York: Duxbury Press, Wadsworth Publishing Company.

Collins, A., & Brown, J.S. (1988). The computer as a tool for learning through reflection. In H. Mandl & A. Lesgold (Eds.), *Learning Issues for Intelligent Tutoring Systems* (pp. 1-18). New York: Springer-Verlag.

Conati, C. (2002). Probabilistic Assessment of User's Emotions in Educational Games. *Journal of Applied Artificial Intelligence, Special issue on "Merging Cognition and Affect in HCI"*, 16(7-8).

Conati, C., Gertner, A., & VanLehn, K. (2002). Using Bayesian networks to manage uncertainty in student modeling. *User Modeling and User-Adapted Interaction*, 12(4), 371-417.

Conati, C., & VanLehn, K. (1996). POLA: A student modeling framework for probabilistic on-line assessment of problem solving performance. In D. N. Chin & M. Crosby & S. Carberry & I. Zukerman (Eds.), *Fifth International Conference on User Modeling (UM96)*, 75-82.

Cooper, G. (1990). The computational complexity of probabilistic inference using Bayesian belief networks. *Artificial Intelligence*, 42, 393-405.

Cooper, G., Horvitz, E., & Heckerman, D. (1989). A method for temporal probabilistic reasoning. Medical Computer Science KSL-88-30. Stanford, CA: Knowledge Systems Laboratory, Stanford University.

Corbett, A., McLaughlin, M., & Scarpinatto, K.C. (2000). Modeling student knowledge: Cognitive tutors in high school and college. *User Modeling and User-Adapted Interaction*, 10, 81-108.

Corbett, A.T., & Anderson, J.R. (1992). Student modeling and mastery learning in a computer-based programming tutor. In C. Frasson & G. Gauthier & G. I. McCalla (Eds.), *Intelligent Tutoring Systems, Proceedings of the Second International Conference, ITS '92*, 413-420.

Corston, R., & Colman, A. (2003). *A Crash Course in SPSS for Windows* (2nd ed.): Blackwell Publishing.

Dagum, P., & Luby, M. (1993). Approximating probabilistic inference in Bayesian belief networks is NP-hard. *Artificial Intelligence*, 60(1), 141-153.

de Vicente, A., & Pain, H. (2002). Informing the detection of the students' motivational state: an empirical study. In S. A. Cerri & G. Gouarderes & F. Paraguacu (Eds.), *Sixth International Conference on Intelligent Tutoring Systems, ITS 2002*, 933-943.

Dean, T., & Kanazawa, K. (1989). A model for reasoning about persistence and causation. *Computational Intelligence*, 5(3), 142-150.

Dean, T., & Wellman, M.P. (1991). *Planning and Control*. San Mateo, California: Morgan Kaufmann.

del Soldato, T., & du Boulay, B. (1995). Implementation of motivational tactics in tutoring systems. *Journal of Artificial Intelligence in Education*, 6(4), 337-378.

Fox, B.A. (1993). *The Human Tutorial Dialogue Project: Issues in the Design of Instructional Systems*. Hillsdale, NJ: Lawrence Erlbaum Associates.

Galdes, D. (1990). An empirical study of human tutors: The implications for intelligent tutoring systems. Unpublished doctoral dissertation, Ohio State University, Columbus, OH.

Ganeshan, R., Johnson, W.L., Shaw, E., & Wood, B.P. (2000). Tutoring diagnostic problem solving. In G. Gauthier & C. Frasson & K. VanLehn (Eds.), *Intelligent Tutoring Systems, 5th International Conference, ITS 2000*, 33-42.

Gertner, A., Conati, C., & VanLehn, K. (1998). Procedural help in Andes: Generating hints using a Bayesian network student model, *Fifteenth National Conference on Artificial Intelligence (AAAI-98)*, 106-111.

Gertner, A.S., & VanLehn, K. (2000). Andes: A coached problem solving environment for physics, *Intelligent Tutoring Systems, 5th International Conference, ITS 2000*, 133-142.

Goleman, D. (1995). *Emotional intelligence: Why it can matter more than IQ*: New York: Bantam.

Graesser, A.C., Person, N.K., & Magliano, J.P. (1995). Collaborative dialogue patterns in naturalistic one-to-one tutoring. *Applied Cognitive Psychology*, 9, 495-522.

Grossmann-Hutter, B., Jameson, A., & Witttig, F. (1999). Learning Bayesian networks with hidden variables for user modeling, *IJCAI-99 Workshop "Learning About Users"*, 29-34.

Grosz, B.J., & Sidner, C.L. (1986). Attention, intentions, and the structure of discourse. *Computational Linguistics*, 12(3), 175-204.

Heckerman, D. (1995). A tutorial on learning with Bayesian networks. MSR-TR-95-06: Microsoft Research.

Henrion, M. (1989). Some practical issues in constructing belief networks. In L. N. Kanal & T. S. Levitt & J. F. Lemmer (Eds.), *3rd Conference on Uncertainty in Artificial Intelligence*, 161-173.

Henrion, M., Pradhan, M., Del Favero, B., Huang, K., Provan, G., & O'Rorke, P. (1996). Why is diagnosis in belief networks insensitive to imprecision in probabilities? In E. Horvitz & F. V. Jensen (Eds.), *Twelfth Annual Conference on Uncertainty in Artificial Intelligence*, 307-314.

Horvitz, E., & Barry, M. (1995). Display of information for time-critical decision making, *Eleventh Conference on Uncertainty in Artificial Intelligence*, 296-305.

Horvitz, E., Breese, J., Heckerman, D., Hovel, D., & Rommelse, K. (1998). The Lumiere project: Bayesian user modeling for inferring the goals and needs of software users, *Fourteenth Conference on Uncertainty in Artificial Intelligence*, 256-265.

Horvitz, E., Jacobs, A., & Hovel, D. (1999). Attention-sensitive alerting, *Fifteenth Conference on Uncertainty in Artificial Intelligence*, 305-313.

Howard, R.A. (1966). Information value theory. *IEEE Transactions on Systems Science and Cybernetics*, SSC-2, 22-26.

Howard, R.A., & Matheson, J.E. (1984). Influence diagrams. In R. A. Howard & J. E. Matheson (Eds.), *Readings on the Principles and Applications of Decision Analysis* (pp. 721-762). Menlo Park: Strategic Decisions Group.

Huang, C., & Darwiche, A. (1996). Inference in belief networks: A procedural guide. *International Journal of Approximate Reasoning*, 15, 225-263.

Jameson, A. (1996). Numerical uncertainty management in user and student modeling: An overview of systems and issues. *User Modeling and User-Adapted Interaction*, 5(3-4), 193-251.

Jameson, A., Grossman-Hutter, B., March, L., Rummer, R., Bohnenberger, T., & Wittig, F. (2001). When actions have consequences: Empirically based decision making for intelligent user interfaces. *Knowledge-Based Systems*, 14, 75-92.

Katz, S., & Lesgold, A. (1994). Implementing post-problem reflection within Coached Practice Environments. In P. Brusilovsky & S. Dikareva & J. Greer & V. Petrushin (Eds.), *Proceedings of the East-West International Conference on Computer Technologies in Education*, 125-130.

Keeney, R., & Raiffa, H. (1976). *Decisions with Multiple Objectives*. New York: Wiley.

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

Lepper, M.R., Aspinwall, L., Mumme, D., & Chabay, R.W. (1990). Self-perception and social perception processes in tutoring: Subtle control strategies of expert tutors. In J. M. Olson & M. P. Zanna (Eds.), *Self Inference Processes: The Sixth Ontario Symposium in Social Psychology* (Vol. 6, pp. 217-237): Hillsdale, NJ: Lawrence Erlbaum Associates.

Lepper, M.R., Woolverton, M., Mumme, D.L., & Gurtner, J.-L. (1993). Motivational techniques of expert human tutors: Lessons for the design of computer-based tutors. In S. P. Lajoie & S. J. Derry (Eds.), Computers as Cognitive Tools (pp. 75-105). Hillsdale, NJ: Lawrence Erlbaum Associates.

Lin, Y., & Druzdzel, M.J. (1999). Stochastic sampling and search in belief updating algorithms for very large Bayesian networks, *AAAI-1999 Spring Symposium on Search Techniques for Problem Solving Under Uncertainty and Incomplete Information*, 77-82.

Martin, J., & VanLehn, K. (1995). Discrete factor analysis: Learning hidden variables in Bayesian networks. Technical Report. Pittsburgh, PA: Department of Computer Science, University of Pittsburgh.

Mayo, M., & Mitrovic, A. (2001). Optimising ITS behaviour with Bayesian networks and decision theory. *International Journal of Artificial Intelligence in Education*, 12, 124-153.

McArthur, D., Stasz, C., & Zmuidzinas, M. (1990). Tutoring techniques in algebra. *Cognition and Instruction*, 7(3), 197-244.

Merrill, D.C., Reiser, B.J., Merrill, S.K., & Landes, S. (1995). Tutoring: Guided learning by doing. *Cognition and Instruction*, 13(3), 315-372.

Merrill, D.C., Reiser, B.J., Ranney, M., & Trafton, J.G. (1992). Effective tutoring techniques: A comparison of human tutors and intelligent tutoring systems. *The Journal of the Learning Sciences*, 2(3), 277-306.

Mostow, J., Huang, C., & Tobin, B. (2001). Pause the Video: Quick but quantitative expert evaluation of tutorial choices in a Reading Tutor that listens. In J. D. Moore & C. L. Redfield & W. L. Johnson (Eds.), *Artificial Intelligence in Education*, 343-353.

Murray, R.C., & VanLehn, K. (2000). DT Tutor: A dynamic, decision-theoretic approach for optimal selection of tutorial actions. In G. Gauthier & C. Frasson & K. VanLehn (Eds.), *Intelligent Tutoring Systems, 5th International Conference, ITS 2000*, 153-162.

Murray, R.C., & VanLehn, K. (2005). Effects of dissuading unnecessary help requests while providing proactive help. In C.-K. Looi & G. McCalla & B. Bredeweg & J. Breuker (Eds.), *Artificial Intelligence in Education*, 887-889.

Murray, R.C., VanLehn, K., & Mostow, J. (2001a). A decision-theoretic approach for selecting tutorial discourse actions, *NAACL 2001 Workshop on Adaptation in Dialogue Systems*, 41-48.

Murray, R.C., VanLehn, K., & Mostow, J. (2001b). A decision-theoretic architecture for selecting tutorial discourse actions, *AI-ED 2001 Workshop on Tutorial Dialogue Systems*, 35-46.

Murray, R.C., VanLehn, K., & Mostow, J. (2004). Looking ahead to select tutorial actions: A decision-theoretic approach. *International Journal of Artificial Intelligence in Education*, 14(3-4), 235-278.

Neapolitan, R.E. (2004). *Learning Bayesian Networks*. Upper Saddle River, NJ: Pearson Prentice Hall.

Newell, A., & Simon, H.A. (1972). *Human Problem Solving*. Englewood Cliffs, NJ: Prentice-Hall, Inc.

Ngo, L., Haddawy, P., Krieger, R.A., & Helwig, J. (1997). Efficient temporal probabilistic reasoning via context-sensitive model construction. *Computers in Biology and Medicine*, 27(5), 453-476.

Paek, T., & Horvitz, E. (2000). Conversation as action under uncertainty. In C. Boutilier & M. Goldszmidt (Eds.), *16th Conference on Uncertainty in Artificial Intelligence (UAI-00)*, 455-464.

Pearl, J. (1988). *Probabilistic reasoning in intelligent systems: Networks of plausible inference*. San Mateo, CA: Morgan-Kaufmann.

Pek, P.-K. (2003). Decision-Theoretic Intelligent Tutoring System. PhD dissertation, National University of Singapore. ftp://ftp.medcomp.comp.nus.edu.sg/pub/pohkl/pekpk-thesis-2003.pdf.

Putnam, R.T. (1987). Structuring and adjusting content for students: A study of live and simulated tutoring of addition. *American Educational Research Journal*, 24(1), 13-48.

Reye, J. (1995). A goal-centred architecture for intelligent tutoring systems. In J. Greer (Ed.), *Proceedings of AI-ED 95 - World Conference on Artificial Intelligence in Education*, 307-314.

Reye, J. (1996). A belief net backbone for student modeling. In C. Frasson & G. Gauthier & A. Lesgold (Eds.), *Intelligent Tutoring Systems, Third International Conference*, 596-604.

Reye, J. (1998). Two-phase updating of student models based on dynamic belief networks. In B. P. Goettl & H. M. Halff & C. L. Redfield & V. J. Shute (Eds.), *Intelligent Tutoring Systems, Fourth International Conference*, 274-283.

Reye, J. (2004). Student modelling based on belief networks. *International Journal of Artificial Intelligence in Education*, 14, 63-96.

Russell, S., & Norvig, P. (1995). *Artificial Intelligence: A Modern Approach*. Englewood Cliffs, New Jersey: Prentice Hall.

Schäfer, R., & Weyrath, T. (1997). Assessing temporally variable user properties with dynamic Bayesian networks. In A. Jameson & C. Paris & C. Tasso (Eds.), *User Modeling: Proceedings of the Sixth International Conference, UM97*, 377-388.

Shachter, R., & Peot, M. (1992). Decision making using probabilistic inference methods, *Eighth Annual Conference on Uncertainty in Artificial Intelligence*, 276-283.

Shute, V.J. (1995). SMART evaluation: cognitive diagnosis, mastery learning & remediation. In J. Greer (Ed.), *Proceedings of AI-ED 95 - World Conference on Artificial Intelligence in Education*, 123-130.

Singley, M.K. (1986). Developing Models of Skill Acquisition in the Context of Intelligent Tutoring Systems. PhD Thesis, Carnegie-Mellon University, Pittsburgh, PA.

Singley, M.K. (1990). The reification of goal structures in a calculus tutor: Effects on problem solving performance. *Interactive Learning Environments*, 1, 102-123.

Tversky, A., & Kahneman, D. (1974). Judgment under uncertainty: Heuristics and biases. *Science*, 185, 1124-1131.

VanLehn, K., Ball, W., & Kowalski, B. (1989). Non-LIFO execution of cognitive procedures. *Cognitive Science*, 13, 415-465.

VanLehn, K., Siler, S., Murray, C., Yamauchi, T., & Baggett, W.B. (2003). Why do only some events cause learning during human tutoring? *Cognition and Instruction*, 21(3), 209-249.

von Winterfeldt, D., & Edwards, W. (1986). *Decision Analysis and Behavioral Research*: Cambridge: Cambridge University Press.

Vygotsky, L. (1978). *Mind in society*: Cambridge, MA: Harvard University Press.

Walker, M.A. (1996). Limited attention and discourse structure. *Computational Linguistics*, 22(2), 255-264.

Wellman, M.P. (1990). Fundamental concepts of qualitative probabilistic networks. *Artificial Intelligence*, 44(3), 257-303.