A STUDENT MODEL FOR
AN INTELLIGENT TUTORING SYSTEM HELPING NOVICES
LEARN OBJECT-ORIENTED DESIGN

by

Fang Wei

Presented to the Graduate and Research Committee
of Lehigh University
in Candidacy for the Degree of
Doctor of Philosophy

In
Computer Science

Lehigh University
August 2007
Approved and recommended for acceptance as a dissertation in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

_______________________ Date

Professor Glenn D. Blank
Dissertation Advisor
Committee Chair
Computer Science and Engineering, Lehigh University

_______________________ Accepted Date

Committee Members:

_____________________
Professor Edwin Kay
Computer Science and Engineering, Lehigh University

_____________________
Professor Jeff Heflin
Computer Science and Engineering, Lehigh University

_____________________
Professor Helen L. Columba
College of Education, Lehigh University

_____________________
Professor William M. Pottenger
Center for Discrete Mathematics and Theoretical Computer Science, Rutgers University
Acknowledgement

I wish to express my sincere gratitude to my advisor, Dr. Glenn D. Blank, for giving me the opportunity to work on this research project and for the invaluable guidance he has given me throughout the research. Without his great patience and constant encouragement, I could not have finished this study. I would also like to thank Dr. Edwin Kay, Dr. Jeff Heflin, Dr. William M. Pottenger, and Dr. H. Lynn Columba for serving as my Ph.D. committee members and especially Dr. Edwin Kay and Dr. Jeff Heflin for their advice.

This research was supported by National Science Foundation (NSF) and the Pennsylvania Infrastructure Technology Alliance (PITA).

I am very grateful to many individuals at Lehigh University who has helped me over the past few years. I am especially grateful to Dr. Bennett Eisenberg who helped me about how to model students answering questions. I am also grateful to Shahida M. Parvez and Sally H. Moritz who provided me with invaluable information and help on the experiments and research and have been very patient and always open for discussion.

I am very grateful to Conni, Bob Schwoyer, and Barbara Sitler for their spiritual guidance and support and the wonderful folks at Beit Simcha. This dissertation would not be possible without the unconditioned love and support of my family. I dedicated this dissertation to my husband Dr. Daming Yu, who is a great help on my research.
# Table of Contents

ACKNOWLEDGEMENT .................................................................................................. III

LIST OF TABLES ........................................................................................................... VII

LIST OF FIGURES ......................................................................................................... VIII

ABSTRACT ..................................................................................................................... 1

1 INTRODUCTION ......................................................................................................... 3

1.1 THE GAP BETWEEN IDEAL AND EXISTING ITS .................................................. 3

1.2 RESEARCH QUESTIONS ....................................................................................... 5

1.3 CONTRIBUTIONS .................................................................................................... 6

1.4 ORGANIZATION .................................................................................................... 9

2 RESEARCH BACKGROUND ...................................................................................... 11

3 RELATED RESEARCH ............................................................................................. 17

3.1 STUDENT MODEL .................................................................................................. 17

3.2 STUDENTS’ KNOWLEDGE REPRESENTATION ................................................... 22

3.2.1 Domain knowledge .......................................................................................... 23

3.2.2 Reasoning knowledge .................................................................................... 23

3.2.3 Monitoring knowledge ................................................................................... 25

3.2.4 Reflective knowledge .................................................................................... 27

3.3 CONCRETE STUDENT MODEL EXAMPLES ......................................................... 29

3.3.1 Student models using non-advanced-numerical techniques ............................ 30

3.3.2 Student models using advanced-numerical techniques .................................. 40

3.4 COMMON PROBLEMS IN PREVIOUS STUDENT MODEL RESEARCH .............. 59

4 METHODOLOGY ....................................................................................................... 61

4.1 THREE-LAYERED STUDENT MODEL ARCHITECTURE ..................................... 61
5.3 E VALUATING STUDENT MODEL WITH REAL STUDENTS .................................................. 146
5.3.1 Parameters of evaluation ...................................................................................... 146
5.3.2 Experiment procedure ......................................................................................... 147
5.3.3 Evaluating student model for CIMEL multimedia with real students ................. 148
5.3.4 Evaluating student model for DesignFirst-ITS with real students .................... 151
5.3.5 Evaluating diagnoses integration in DesignFirst-ITS with real students ............. 154
5.3.6 Comparing with non-advanced-numerical student models ................................ 158

6 CONCLUSIONS AND FUTURE WORK ........................................................................ 164
6.1. SUMMARY OF DISSERTATION .............................................................................. 164
6.2. CONCLUSIONS ..................................................................................................... 169
6.3. CONTRIBUTION ................................................................................................... 169
6.4. FUTURE WORK .................................................................................................... 170

APPENDICES .................................................................................................................. 172

APPENDIX I: DEFINITIONS OF CONCEPTS IN CIN FOR STUDENT MODEL .................. 172
APPENDIX II: GLOSSARY FOR THE FIRST CHAPTER .................................................. 174

BIBLIOGRAPHY ............................................................................................................... 179

VITA .................................................................................................................................. 187
List of Tables

Table 3-1 Prior probability for root node Pollution ................................................. 46
Table 3-2 Prior probability for root node Smoking .................................................. 46
Table 3-3 Conditional Probability Table for non-root node Lung_cancer .................. 46
Table 4-1 Comparison between different student modeling approaches ................. 82
Table 4-2 Conditional probability table for concept Attribute_Method .................. 84
Table 4-3 Conditional probability table for node au .............................................. 84
Table 4-4 Conditional probability table for concept Attribute_Method^a ............... 107
Table 4-5 Conditional probability table for concept attribute^c or method^c .......... 107
Table 4-6 Conditional probability table for node au .............................................. 107
Table 4-7 Conditional probability table for concept Attribute_Method^c ............... 108
Table 5-1 ANOVA Tests of Between-Subjects Effects ............................................ 136
Table 5-2 ANOVA Tests of Between-Subjects Effects ............................................ 143
Table 5-3 ANOVA Tests of Between-Subjects Effects ............................................ 156
Table 5-4 ANOVA Tests of Between-Subjects Effects ............................................ 160
Table 5-5 ANOVA Tests of Between-Subjects Effects ............................................ 161
List of Figures

Figure 2-1 Architecture of DesignFirst-ITS ............................................................. 14
Figure 3-1 The overlay student model. (Adapted from Gürer (1993)) ...................... 19
Figure 3-2 The perturbation student model (Adapted from Gürer (1993)) ............. 20
Figure 3-3 Illustration of a possible derivation tree for the episodic student model .... 32
Figure 3-4 Illustration of a tree structure over constraints C, D, E, and F .......... 37
Figure 3-5 A Bayesian network Example ................................................................. 45
Figure 3-6 Illustration of the concept student model in ANDES ......................... 49
Figure 3-7 Illustration of the solution student model in ANDES ....................... 50
Figure 3-8 Static Bayesian student model example .................................................. 54
Figure 3-9 A student model example for aggregation relation ............................ 55
Figure 3-10 Dynamic Bayesian student model example ......................................... 56
Figure 4-1 Architecture of the Three-Layered student model in DesignFirst-ITS ...... 61
Figure 4-2 Curriculum information network (CIN) for the student model ............. 74
Figure 4-3 Atomic Bayesian Network (ABN) ......................................................... 76
Figure 4-4 A concrete example of an ABN ............................................................ 84
Figure 4-5 A simple example Bayesian network ..................................................... 85
Figure 4-6 Update the Bayesian network in two steps ......................................... 86
Figure 4-7 Updating an ABN in steps ............................................................... 88
Figure 4-8 An example of an ABN ............................................................... 88
Figure 4-9 An example of an ABN ............................................................... 90
Figure 4-10 An example of an ABN ............................................................... 93
Figure 5-9 Illustration of the evaluation procedure ................................................................. 150

Figure 5-10 Students do object-oriented design in LehighUML plug-in in Eclipse IDE
.................................................................................................................................................. 152

Figure 5-11 Illustration of evaluation procedure ................................................................. 153

Figure 5-12 Illustration of evaluation procedure ................................................................. 155

Figure 5-13 Average correct prediction rate vs. factor “method” ....................................... 157

Figure 5-14 Illustration of evaluation procedure ................................................................. 159

Figure 5-15 Averaged CPR of student models for close-ended problems ..................... 160

Figure 5-16 Average CPR for the student models for open-ended problems ............ 163
Abstract

Learning object-oriented design is a challenging task for many beginners. Intelligent Tutoring Systems can aid learners with complex problem-solving. Generally, an ITS contains three main components: an expert evaluator that observes a student’s work and identifies errors in his/her solution; a student model that diagnoses the gap in the student’s knowledge; and a pedagogical advisor that provides feedback to the student.

Existing student models have several common problems: 1) they only consider rule-based behaviors or concepts as students’ learning goals, while students oftentimes are confused about the relationship among concepts such as prerequisite, transition, similarity and distinction; 2) they do not represent layered student knowledge, taking into account domain, reasoning, monitoring and reflective knowledge; 3) they often use Bayesian networks requiring exponential time, and hence cannot provide consistent support for real time communicative tutoring; and 4) they rarely simulate students’ knowledge history.

This dissertation develops a student model applying Atomic Dynamic Bayesian Networks, which consists two connected Atomic Bayesian Networks. It advances the state of the art of student models by: 1) representing concepts and important relationships, such as prerequisites and distinction; 2) tracking a history of student learning; and 3) integrating a student model from both open- and close-ended work. This student model does all these things in real time, so that the ITS can be responsive to students as they work on an assigned problem.
We evaluated the ABN- and ADBN-based student models with 240 simulated students and 71 human subjects. The evaluation investigates the student models’ behavior for different types of students and different slip and guess values. Holding slip and guess to equal and small values, ADBNs are able to produce accurate diagnostic rates for student knowledge states over students’ learning history. The results also show that student models with ADBNs perform better than student models with ABNs only.

We evaluated the student model which integrates the diagnosed students’ learning state from closed- and open-ended exercises with the 71 human subjects. The results show that integrating diagnoses from closed- and open-ended exercises is an effective way to increase the accuracy of student models.

In addition, we compared the accuracy of non-advanced-numerical student models with the student models using ADBNs. The results show that student models using ADBNs perform much better than the non-advanced-numerical student models.
1 Introduction

Learning object-oriented design is a challenging task for many beginning students. An Intelligent Tutoring System (ITS) has been proposed to help beginners learn object-oriented design (Blank, Barnes, & Kay, 2005). Generally, an ITS contains three main components: the expert module, the student model and the Pedagogical Advisor. The expert module observes students’ work and identifies errors in their solutions; the student model diagnoses the gap in students’ knowledge structure and learning needs based on the history of the students’ performance; and the Pedagogical Advisor provides feedback to the students and tutoring when they need help.

In this research, a student model is developed based on Bayesian networks and rule-based approaches. It will advance the state of the art of student models in several directions: by representing concepts and important relationships, such as prerequisites and similarities that can cause confusion; and by tracking a history of student behavior, and inferring cognitive strategies germane to both the problem domain (object-oriented software development) and learning in general. This student model does all these things in real time, so that the ITS as a whole can be responsive to students as they work on an assigned problem and can issue understandable feedback, rebuild students’ knowledge structure, and tutor students’ cognitive strategies.

1.1 The gap between ideal and existing ITS

In an ideal ITS, a student receives understandable feedback that not only tells how to modify the current error but also helps to rebuild his/her knowledge structure.
After viewing the feedback, the student can quickly understand the missed knowledge based on his/her knowledge structure and hence will be less likely to repeat the same error in the future. Therefore, understanding feedback is crucial to shorten the time of students’ learning, and rebuilding the students’ knowledge structure can help students to achieve their learning goals. Further, the system can help the student cultivate positive cognitive strategies through monitoring the student’s learning actions and providing tutoring. For instance, if the student does not pay attention to the feedback and keeps gaming the system, i.e. repeatedly guessing the answer or repeatedly requiring hints until he/she gets the solution (Aleven, Roll, McLaren, Ryu & Koedinger, 2005; Roll, Baker, Aleven, McLaren, & Koedinger, 2005; Walker, Koedinger, McLaren, & Rummel, 2006), it is pedagogically appropriate to warn the student to stop the erroneous actions.

Existing ITSs can flag errors, provide hints for correct solution or possible next solution step according to students’ solution paths, and present definitions for all knowledge units relevant to the current problem. They issue feedback based on students’ errors instead of students’ knowledge structure. Ordinarily, the existing ITSs provide three levels of feedback, in which the first level identifies errors, the second level issues concept definitions relevant to the problem, and the third level outputs the correct solution. The existing ITSs intend to let students self-diagnose the deficiency in their knowledge structure and find their learning needs. Thus, students are required to read the information with all relevant knowledge units by themselves. Usually, students cannot understand the connection between the definitions of concepts and the correct solution due to their incomplete knowledge structure. Consequently, students
oftentimes are confused with the jumble of knowledge displayed, repeat the same errors, and finally quit prematurely.

Ohlsson and Mitrovic (2006) state that students’ knowledge acquisition requires the migration of structure from the experts to the students. Helping to rebuild the students’ knowledge structure is the essential step in students’ knowledge acquisition. However, none of the exiting ITSs considers whether the concept definitions can be understandable by the students and tries to help the student to rebuild their knowledge structure. In addition, there are few ITSs that can provide appropriate tutoring for students’ cognitive strategies. Hence, most of the existing ITSs lose the opportunities to motivate students, communicatively contact the students, and loosen up the connection between the tutors and the students.

The student model is an essential component in an ITS, which represents student’s knowledge state and updates it. To enable an ITS to provide concept definition that students can understand relevant to their problem solving, the student model needs to diagnose the representation of the students’ knowledge structure, which consists of granular knowledge items, in real time. To enable an ITS to provide appropriate tutoring of students’ cognitive strategies, the student model needs to diagnose students’ cognitive strategies in real time.

1.2 Research questions

To develop a student model that can diagnose students’ knowledge structure and cognitive strategies in real time, this study will examine and answer the following research questions:
1. Can this student model provide information about student knowledge state for pedagogical decisions?

2. How should this student model represent a student’s current knowledge state and the student’s knowledge structure?

3. How will the student model track student knowledge state over time? Under this research question there are two sub questions:
   a. Would tracking a history of student knowledge state be useful for pedagogical decisions?
   b. Can a history be maintained efficiently enough to be responsive in real time?

4. How can this student model synthesize information from two different sources, open-ended problem solving (object-oriented class diagram design) and close-ended exercises (multiple choice quizzes or drag-and-drop exercises)?

5. What cognitive strategies should the student model consider and how should it consider them?

1.3 Contributions

This research makes six contributions to student modeling. The first contribution is a novel way to represent the students’ knowledge structure, where both concepts and relationship among concepts are knowledge units that students need to learn. Previous research considers rule-based behaviors or individual concepts as part of the students’ learning goals. Nevertheless, students often are confused about the relationship among individual concepts such as prerequisite, transition, similarity, and
distinction. This contribution enables the student model to tell the pedagogical agent
the concepts a student needs to learn and helps to answer the research question 1.

Previous research introduced a variety of student modeling architectures, each
of which is confined to its own simulating techniques. Yet none of them reveals a
standard way to represent layered knowledge in a student model. This research will
describe a three-layered architecture that can be standardized for modeling various
layers of students’ knowledge. This architecture was presented at the tenth Annual
Consortium for Computing Sciences in Colleges Northeastern Conference (CCSCNE
2005) (awarded one of the best papers). This contribution helps to answer the research
question 2.

Student modeling embracing probabilistic techniques, such as Bayesian
networks, requires exponential computational time, since the number of parameters
and the updating time for Bayesian networks is of exponential order and hence cannot
provide consistent support for real-time communicative tutoring. This research will
present an Atomic Bayesian network that provides a refined representation of
prerequisite relationships, diagnoses student’s knowledge structure, and achieves real-
time responsiveness. The Atomic Bayesian network has been disseminated as a full
paper in the 8th International Intelligent Tutoring Systems Conference (ITS 2006).
This contribution helps to answer the research question 2.

There are few student models that simulate students’ knowledge within a
history. This is due to the complexity of the mechanism and volatility in the evolution
of students’ knowledge. Few researchers introduce dynamic Bayesian networks in
solving the problem, while none of them models the students’ hierarchical knowledge
structure. This research will present an innovative dynamic Bayesian network, Atomic Dynamic Bayesian network that represents refined prerequisite relationships and, by considering the learning history diagnoses in real-time students’ knowledge structure. The Atomic Dynamic Bayesian network has been disseminated as a poster in the 13th International Conference on Artificial Intelligence in Education (AIED 2007). This contribution helps to answer the research question 3.

Existing student models separate the inferred students’ knowledge from closed- and open-ended exercises. The corresponding ITSs determine the next step of learning according to the knowledge from close-ended questions and tutor problem solving errors based on the knowledge from open-ended tasks. Few of the existing student models integrate these two kinds of knowledge to represent the synthetic students’ learning state. This research will describe a unique student model that combines the knowledge from open-ended problem solving (object-oriented class diagram design) and close-ended exercises, which is suitable for a learning environment where students switch among various types of exercises frequently. This contribution helps to answer the research question 4.

The currently available student models simulate partial students’ knowledge including basic domain knowledge or problem-specific procedural problem solving skills. Few of them tackle the tutoring needs of students’ cognitive strategies. This research will develop a student model that simulates students’ knowledge from basic domain knowledge to positive cognitive strategies. Cognitive strategies include general and domain-specific strategies. One example of a general cognitive strategy is hacking. The domain-specific strategies are the strategies for going from problem
description to class diagram. This research will answer how the student model can
detect the general and domain-specific cognitive strategies. This contribution helps to
answer the research question 5.

The contributions of this research are as follows:

1. A novel way to represent students’ knowledge structure, where both concepts
   and relationship between concepts are knowledge units that students need to
   learn.

2. A novel three-layered architecture which can be standardized in modeling
   various stratum of students’ knowledge.

3. ABN – a novel Atomic Bayesian network that provides a refined
   representation of prerequisite relationships, diagnoses student’s knowledge
   structure, and guarantees real-time responsiveness.

4. ADBN – an innovative dynamic Bayesian network that represents refined
   representation of prerequisite relationships and diagnoses in real-time students’
   knowledge structure considering the learning history.

5. A unique student model that integrates knowledge from open-ended problem
   solving (object-oriented class diagram design) and close-ended exercises.

6. A unique student model that simulates students’ knowledge across a broad
   array from basic domain knowledge to positive cognitive strategies.

### 1.4 Organization

The rest of this paper is organized as follows: chapter 2 describes the research
background – how the student model fits into an embedded intelligent tutoring system,
DesignFirst-ITS, that tutors students objected oriented design; chapter 3 introduces the related theories in cognitive science and artificial intelligence, and reviews concrete examples that gives the state of art of student models; chapter 4 describes the methodology that is applied in building the student model in this dissertation; chapter 5 presents six empirical studies and their results, which validate the methodology; chapter 6 states the conclusion and future work.
2 Research background

Learning object-oriented programming, let alone object-oriented design, is a challenging task for many beginning students. Research by Ratcliffe (2002) has shown that lack of comprehension expressed by first year computer science students is a rising concern in academia. McCracken et al. (2001) performed a study that suggested that in UK and USA, approximately 30% of students do not understand the basics.

The first few lessons in object-orientation are rich and complex, so that many students get confused, and may withdraw from the course. Many students continue repeating similar errors after teachers tell them the right answers. They often struggle to solve problems after an instructor explains what they need to know. Meanwhile students having difficulties may not want to admit they are having problems or may have difficulties explaining their problems to an instructor. These situations happen for many reasons: 1) it is difficult to explain the problem from the student’s perspective—one must understand what the student knows and does not know; 2) it is hard to trace how many times a student commits similar errors and so observe repetition of problem solving patterns; 3) it is hard to remedy a student’s deficient problem solving patterns and encourage sound ones; and 4) an instructor may not know who is having difficulties until it is too late, may not be able to tell why the student is having these difficulties, and may simply not have enough time to look into every student’s needs in a large class.

Reiser, Anderson, and Farrell (1985) reported that students working with private tutors can learn given material four times faster than students who attend
traditional classroom lectures, study textbooks and work on homework alone. Bloom (1984) also reported that students have a better grasp of material working with a private tutor than attending traditional classroom lectures. When a qualified private human tutor is not available, the next best option is an intelligent tutoring system. An Intelligent Tutoring System (ITS) is a computer-based instructional system that has knowledge bases for instructional content and teaching strategies (Dağ & Erkan, 2003). It attempts to use a student’s level of mastery of topics to dynamically adapt instruction (Macisy & Castells 2001). Anderson and Skwarecki (1986) reported that an ITS is a cost-effective means of one-on-one tutoring that provides novices with the individualized attention needed to overcome learning difficulties. Intelligent tutoring systems are not only being used in academia to augment classroom teaching but have also penetrated various industries where companies are using ITSs to train employees to perform their job functions. As a result, ITSs have been built for various domains such as mathematics, medicine, engineering, public services, computer science, etc.

The application of ITS in computer science has been limited to tutoring database design and specific procedural aspects of programming languages such as Java, C++, and LISP, and have not kept up with the current technology focus of object-oriented design (Sykes & Franek 2004; Mitrovic, Mayo, Suraweera, & Martin 2001; Kumar, 2002; Reiser et al.,1985).

An ongoing project in Lehigh University, DesignFirst-ITS, is an intelligent tutoring system that provides one-on-one tutoring to help beginners with various learning styles in a CS1 course. Based on a design-first curriculum which subsumes an objects-first approach in lessons that also introduce object-oriented analysis and
design, using elements of UML before implementing any code (Moritz, Wei, Parvez, & Blank, 2005), DesignFirst-ITS helps students learn object-oriented design and eventually programming. DesignFirst-ITS is built on a prior project, the result of the NSF CRCD-sponsored CIMEL (Constructive and collaborative Inquiry-based Multimedia E-Learning) project, which developed multimedia courseware to introduce the breadth of computer science, including introductory concepts in Java and object-oriented programming, complementing a textbook, *The Universal Computer: Introducing Computer Science with Multimedia* (Blank, Barnes, et al., 2005). CIMEL multimedia uses many techniques including audio, video, text, animation, JUST THE FACTS summaries, and interactive, constructive, and inquiry-based learning exercises to reach students with a wide variety of learning styles (a web-based demonstration and documentation are available at www.cse.lehigh.edu/~cimel).

Hartley & Sleeman (1973) proposed an ITS that has three components, curriculum, student behaviors, and teaching methods. Curriculum is the subject matter that teachers and students cover in their studies. Student behaviors are how the students goes about solving the problem, why a student’s solution is incorrect, what the student currently knows about the problem and why a solution is incorrect. Teaching methods are how to provide the right answer to the question, how to tell if the student’s solution is right or not, and how to make decisions about tutoring tactics.
Dağ and Erkan (2003) added the User Interface as another component to the three-component ITS proposed by Hartley & Sleeman (1973). With these four components, Dağ and Erkan proposed an architecture of ITS. From this starting point, the architecture for DesignFirst-ITS is developed and shown in Figure 2-1.

DesignFirst-ITS consists of four components. The Curriculum Model, at the heart of the architecture, represents the knowledge of the first few lessons in a design-first CS1 course. It is organized as a Curriculum Information Network, or CIN, which links concepts that are intended to teach together to show relationships among them. The Expert Evaluator, the student model and the pedagogical advisor in the DesignFirst-ITS refer to the CIN that ties the student’s learning activities and state of knowledge to the curriculum.

The Expert Evaluator interfaces with the Eclipse IDE through a plug-in. The Eclipse IDE is an open-source development platform for Java. The Expert Evaluator observes the student’s work step by step in both the object diagram interface and the
code interface of Eclipse, and compares each step to its own solution(s) to the current problem. When a specific error is identified, it is linked to a concept within the CIN, and along with the recommended solution, is passed to the student model.

The student model, which is the focus of this research, maintains a model of the student’s current knowledge state based on information from both CIMEL and the Expert Evaluator. From CIMEL, the student model gets input on individual student performance based on exercise and quiz data from the object-oriented contents. From the Expert Evaluator, it receives information on both errors made by the student (as described above), and problems which the student completes successfully. The student model then performs a diagnosis based on the history of the student’s performance to determine the reasons for the student’s errors and where there are gaps in his/her knowledge.

The pedagogical advisor provides feedback to students and tutoring based on where the student model indicates that the students need help. The feedback consists of explanation of detail errors and relevant concepts.

The DesignFirst-ITS can interact with students either through CIMEL multimedia or the Eclipse IDE, each of which initiate different flows of control through the ITS architecture. The student can learn about object-oriented design from CIMEL multimedia. CIMEL records the student’s behaviors on the quizzes and exercises in a database. The student model uses these data to infer the student’s level of knowledge. Another flow of control begins with the Eclipse IDE. The Expert Evaluator observes the student as he/she enters a design solution, assesses what are right answers and errors in the student’s solution, and provides the results to the
student model and the pedagogical advisor. After receiving the input from the Expert Evaluator the student model performs a diagnosis based on the input and the history of student’s records and provides the diagnosis to the pedagogical advisor. The pedagogical advisor gives proper instructions to the student based on the results from the student model and the Expert Evaluator.
3 Related research

An ITS needs to understand what students know to provide sufficient help. This understanding can help to generate feedback that the students understand. To equip an ITS that understands students’ knowledge requires accurately modeling the knowledge, which includes the processes of representing and updating. Researchers of the student model represent and update not only the contents, but also the understanding level of the students’ knowledge. Cognitive researchers found that the student’s knowledge can be divided into various levels (Self, 1994). Existing student models adopt different approaches to model each level of the students’ knowledge. This chapter investigates the approaches, and has three major sections: the first section describes the basic approaches to build a student model; the second section introduces student’s knowledge representation; and the third section presents concrete student model examples.

3.1 Student model

A student model maintains a model of a student’s current knowledge state that allows more intelligent pedagogical decisions and actions to happen. These decisions and actions include curriculum sequencing, interactive problem solving support, intelligent analysis of student solutions, and understandable pedagogical tutoring customized to each individual student’s learning state (Brusilovsky, Schwarz, & Weber, 1996). Data for each student that are maintained in a student model are stored in student profiles. Each student profile belongs to a student and contains value-
knowledge pairs, where the value represents the understanding level for the knowledge unit in each pair. The student model first infers the student profiles from the student’s learning performance, and then infers the student’s learning needs from the profiles and performance. Hence, the student model provides fundamental information to understand the student’s learning state, i.e., why they make errors and what tutoring help they really need.

Student models have been studied since the beginning of ITS research. Many researchers argue that the main purpose of a student model is to guide pedagogical decision-making (Gürer, 1993; Zhou, 2000). Ohlsson (1986) called the student modeling problem “cognitive diagnosis.” He classified student models into four types: performance measures, overlays, bug library, and simulations. Later Ohlsson (1992) proposed a new approach of overlay modeling which is called constraint based modeling.

Performance measures (Ohlsson, 1986) access the student's knowledge state using simple computation. It gives a student an overall assessment using a single value such as grades. It does not represent details in the student’s knowledge state, i.e. which concepts he/she knows or does not know. Hence, there is no information available to present responsive feedback to the student for each error he/she made.

Overlay models (Ohlsson, 1986) simulate the student from a set of propositions in a knowledge domain. Propositions related to errors can be accounted for when these propositions have fine enough granularity. In an overlay model, the student knowledge is modeled as a subset of the expert knowledge. The basic function of this model is to find the missing chunks between the expert knowledge and the
subset of the expert knowledge – the student knowledge. The pedagogical advisor then tutors topics in the missing chunks. This model cannot model misconceptions since there is no representation for misconceptions in the expert knowledge; neither is in the subset of the expert knowledge. Therefore, overlay models do not simulate the actual student knowledge but the part that overlaps with the expert knowledge. The overlay student model is shown in Figure 3-1.

![Figure 3-1 The overlay student model. (Adapted from Gürer (1993))](image)

A Bug Library (Ohlsson, 1986) attempts to simulate students’ misconceptions. This model collects a great deal of information about likely errors and misconceptions for a given domain. It matches the student's errors to the gathered information to determine the misconception the student has. The pedagogical advisor can evoke remedial instructions when matches are found. Remedial instructions for misconceptions are more helpful than just providing missing expert knowledge to
students when they are confused by their errors. But building a bug library is very
time-consuming, and the bug library cannot identify new errors which have not been
collected in the bug library. Furthermore, even though matches are found, the bug
library cannot infer why the student made those errors, which limits the effectiveness
of the remedial instructions. The bug library model is shown in Figure 3-2, where the
novice knowledge covers area out of the expert knowledge, while the novice
knowledge does not in Figure 3-1.

![Diagram of student model](image)

**Figure 3-2 The perturbation student model (Adapted from Gürer (1993))**

The simulation model represents a student’s cognitive state and performs like
the student in a relevant knowledge domain. For a given task, the simulation model
can generate behaviors, which is a prediction of what the simulated person would do if
he or she solves the same task. This model simulates the student’s cognitive state at
each step in his/her answer to a problem and specifies the student's knowledge
including both declarative and procedural knowledge.
Model-tracing can be considered as a simple simulation model. Instead of inferring students’ knowledge from their final answers to several problems, model-tracing infers the students’ knowledge from each problem solving step (Anderson 1983, 1993). It keeps pace with the student by "tracing" the student’s actions (i.e., matching user and application events against the task model). The student is also able to consult the task model interactively. (VanLehn et al., 2005; Koedinger & Anderson, 1997; Koedinger, Aleven, & Heffernan, 2003; Amižić, Stankov, & Rosić, 2001)

The constraint based student model (Mitrovic et al., 2001) represents subject matter knowledge as sets of constraints (Ohlsson & Mitrovic, 2006). A constraint is an ordered pair \((C_t, C_s)\), where \(C_t\) is the relevance condition and \(C_s\) is the satisfaction condition. \(C_t\) is used to identify the class of problem states in which \(C_t\) is relevant. \(C_s\) identifies the class of relevant states in which \(C_s\) is satisfied. Each constraint specifies the property of the domain that is shared by all correct paths. If \(C_t\) is satisfied in a problem state, in order for that problem state to be a correct one, it must satisfy \(C_s\). The constraint violation on the part of the student’s solution indicates incomplete or incorrect knowledge and can therefore be used to guide the response of an ITS.

A constraint based student model costs less effort to build than a model-tracing student model, since it only provides a description of the student knowledge in terms of the constraints which he/she has violated and do not develop expert and buggy planning rules and buggy operator rules that the model-tracing student model needs. It also leaves open the question of what instruction is implied pertaining to individual student’s knowledge. In Ohlsson’s basic scheme, a remedial message is attached to each constraint and printed when that constraint is violated (Ohlsson, 1992).
A current trend of student modeling is to combine approaches of Overlay Model, Bug Library, and Model-Tracing Model (Brusilovsky et al., 1996; Weber & Brusilovsky, 2001; Weber, 1996; Shute, 1995). The researchers model students’ knowledge with not only expert knowledge but also students’ defective knowledge. Moreover, the researchers are not satisfied with modeling student knowledge after students finish several problems or problem solving steps. More and more researchers model students’ knowledge after each problem or each step in problem solving through model-tracing, the simple simulation model (VanLehn & Niu, 2001; Koedinger & Anderson 1997; Koedinger et al., 2003). There are also a few student models implemented that are constraint base (Mitrovic et al., 2001). To provide more details of the research background, the next two sections describe concrete examples of student modeling and analyze the student knowledge they modeled.

### 3.2 Students’ knowledge representation

All knowledge that students need to master before they successfully solve realistic problems can be written out (e.g. in a book). Knowledge representation for a student model represents the knowledge in a way that a computer program can easily retrieve and students can easily learn when the knowledge is shown in a window. Hence, the knowledge needs to be represented as reasonably granularized and logically connected. Self (1994) stated that the different levels of knowledge are domain knowledge, reasoning knowledge, monitoring knowledge, and reflective knowledge. The levels go from specific to general, from concrete to abstract knowledge.
3.2.1 Domain knowledge

Domain knowledge is a set of propositions which explain all concepts that belong to the *vocabulary* for discussing or solving problems. For example,

Domain Knowledge (astronomy)

= \{ \text{Planets are large celestial bodies that orbit the sun; Mars is a planet; Mars orbits the sun; X is a planet; X orbits the sun; …} \}

Domain knowledge is only declarative and tells nothing about how students can use the domain knowledge to solve a realistic problem.

3.2.2 Reasoning knowledge

Reasoning knowledge is a set of propositions which contain reasoning relationships among propositions in the domain knowledge. It covers part of the gap that the domain knowledge misses. For instance,

Reasoning Knowledge (astronomy)

= \{ \text{‗Planets are large celestial bodies that orbit the sun’ & ‘Mars is a planet’} \}

⇒ \text{‗Mars orbits the sun’;}

\text{Planets are large celestial bodies that orbit the sun’ & ‘X represents any planet’}

⇒ \text{‗X orbits the sun’; …} \}

In the example, because planets are large celestial bodies that orbit the sun and because Mars or X is a planet, Mars or X orbits the sun. Reasoning knowledge from Self (1994) enhances the understanding and remembering of domain knowledge. Self (1994) found that each unique proposition in the domain knowledge cannot stay by
itself in the students’ minds for long unless the students understand the reasoning knowledge that can connect the domain knowledge together. However, other than reasoning relationships, there are relationships among propositions that are essential for students to understand, remember, and apply the domain knowledge. The other relationships are the similarity, difference, and all possible relationships that can connect together the separate domain knowledge propositions. Suppose there are only two concepts, attribute, and parameter in the object-oriented design (Object-oriented design) domain:

\[
\text{Domain Knowledge (Object-oriented design)}
\]

\[
= \{ \text{Attribute} \quad \text{– An attribute is a characteristic of an object;}
\]

\[
\text{Parameter} \quad \text{– A parameter is a variable that passes data to a method} \}
\]

\[
\text{Reasoning Knowledge (Object-oriented design)}
\]

\[
= \{ \text{Attribute}_{\text{parameter}} \quad \text{– Though attributes and parameters are variables, an attribute is accessible anywhere within the scope of a class, while a parameter is accessible only within the scope of an object’s method} \}
\]

This example shows that the domain knowledge is inadequate for students to solve a realistic problem (e.g. design an attribute and a parameter in an Object-oriented design). This error has been observed from real students when they defined ‘movieTitle’ as a parameter after they has already defined ‘movieTitle’ as an attribute. One reason that the students made this error is that they did not understand the difference between a parameter and an attribute. Nevertheless, the domain knowledge and the reasoning knowledge together are inadequate to explain how students can use the domain knowledge to solve a realistic problem, since the connection between the
realistic problem and the domain knowledge has not been identified. Monitoring knowledge fills in the gap.

3.2.3 Monitoring knowledge

Monitoring knowledge is a set of propositions which specify how to solve a problem on the basis of the superficial characteristics of the problem. For example,

Reasoning Knowledge (astronomy)

= {‗Planets are large celestial bodies that orbit the sun’ & ‘Mars is a planet’

⇒ ‘Mars orbits the sun’;

‗Planets are large celestial bodies that orbit the sun’ & ‘X represents any planet’

⇒ ‘X orbits the sun’; …}

Monitoring Knowledge (astronomy)

= {If the subject from the problem is a planet and if the subject is Mars, then use the first rule in the reasoning knowledge;

if the subject from the problem is a planet and if the subject is not Mars, then use the second rule in the reasoning knowledge; …}

Suppose the problem is “Does Venus orbit the sun given Venus is a planet?”

According to the rules in the monitoring knowledge in the previous example, the second rule in the reasoning knowledge is applicable. Hence the answer to the problem is yes. This example shows how monitoring knowledge helps a student to solve a realistic problem using reasoning knowledge and domain knowledge. To
explain more about monitoring knowledge, another example on Object-oriented design domain is shown as follows:

Domain Knowledge (Object-oriented design)
= {An attribute is a characteristic of an object; a parameter is a variable that passes data to a method}

Reasoning Knowledge (Object-oriented design)
= {Though attributes and parameters are variables, attributes are accessible anywhere within the scope of a class, while parameters are accessible only within the scope of a method}

Monitoring Knowledge (Object-oriented design)
= {Parsing a sentence and simplifies it by using subjects, verbs, and nouns in objects only; a subject that has a simple value is a candidate attribute or parameter; an object is a candidate attribute or parameter; use the rule in the reasoning knowledge to distinguish attributes with parameters; …}

Assume the students need to find an attribute and a parameter for the problem description: “A movie ticket machine shows the movie title.” If the students apply the rules from the monitoring knowledge, they simplify the sentence to subject “movieTicketMachine”, verb “show”, object “movieTitle”. According to the domain knowledge (definition of attribute and parameter) and the second rule in the monitoring knowledge (a subject that has a simple value is a candidate attribute or parameter), the students will learn that the subject “movieTicketMachine” is not a candidate attribute or parameter since “movieTicketMachine” does not have a simple value, while the object “movieTitle” can be an attribute or a parameter since it can
have a simple value such as a string “Shrek”. According to the fourth rule in the monitoring knowledge (use the rules in the reasoning knowledge to distinguish attributes with parameters), the students can conclude that “movieTitle” is an attribute instead of a parameter, since “movieTitle” may need to be accessible in a larger scope than the scope of the method of “showMovieTitle”. This example shows that the monitoring knowledge is essential to help students to do object-oriented design from problem description.

Self (1994) stated that monitoring knowledge is hard to specify since different experts may have different interpretation about how to solve a problem given its description. Experts need to determine a set of efficient and robust monitoring knowledge, from which they generate the expert solutions. Domain knowledge, reasoning knowledge and monitoring knowledge together make students capable of solving a realistic problem. However, the three kinds of knowledge cannot help students to correctly apply all available sources, such as time, hints, and etc. The fourth level, reflective knowledge serves this purpose.

3.2.4 Reflective knowledge

Reflective knowledge is a set of propositions which specify appropriate strategies students should have in a learning environment. This kind of knowledge is about the correct learning actions students need to take and fallacious actions students need to avoid. Unlike domain knowledge, reasoning knowledge, and monitoring knowledge, reflective knowledge is not domain specific, since students can take same
learning actions in environments about different domain. The following is an example of the reflective knowledge:

Reflective Knowledge

= {Hints or feedback should be read carefully; problem description should be read repeatedly to reduce the chances of being misread; systematic guessing should be avoided; acquiring answers before taking serious thinking should be avoided; … }

If students commit errors such as ignoring hints or feedback, misreading the problem description, repeatedly guessing the answer for the same problem, or looking the answer before seriously thinking about the problem, some triggers will be invoked to tutor the students about the corresponding reflective knowledge. This example shows that the reflective knowledge helps students to become better learners through fostering learning habits which are beneficial for their future learning.

Different methods represent the four kinds of knowledge: 1) reflective knowledge can be represented by a set of rules since there is no relationship that needs to be modeled in this knowledge where each distinct piece of reflective knowledge corresponds to one rule; 2) domain knowledge, reasoning knowledge, and monitoring knowledge can be built on a concrete domain which is characterized by relationships among different knowledge items, and the three kinds of knowledge can be represented by a Curriculum Information Network (CIN).

A CIN describes the relationships between the concepts and skills in a curriculum (Wescourt et al., 1977), where a curriculum defines all knowledge that is intended to be used to teach students. A CIN is an acyclic directed graph, in which
each node represents a distinct knowledge item, and a set of directed arrows connects pairs of nodes representing prerequisite relations. A few existing Intelligent Tutoring Systems considered aggregation or prerequisite relationships in their CIN (Millán & Pérez-de-la-Cruz, 2002; Butz, Hua, & Maguire, 2004; Carmona, Millán, Pérez-de-la-Cruz, Trella, & Conejo, 2005). More details about how to model these relationships will be presented in the next section.

### 3.3 Concrete student model examples

A student model infers students’ knowledge by assessing students’ performances in a specific environment. In this environment, students’ performances are called evidence, and students’ knowledge are called inference outcomes. From a mathematical point of view, student modeling applies non-advanced-numerical and advanced-numerical techniques in its inference procedure. With advanced-numerical techniques, researchers look for nuances among understanding levels of knowledge units, while non-advanced-numerical techniques tend to treat each knowledge unit as either known or unknown without any states in between. Furthermore, advanced-numerical techniques can account for the relationship between knowledge concepts, which makes it possible to reach more accurate diagnosis with less evidence, while non-advanced-numerical techniques cannot.

Advanced-numerical techniques include fuzzy set theory, Bayesian networks, and various applications of Bayesian theory, whereas non-advanced-numerical techniques only involve simple algebraic calculations such as match, summation, and subtraction and are not related to any Bayesian networks or fuzzy set theory. The
following sections introduce examples of student models with non-advanced-numerical or advanced-numerical techniques. Although these student models have some deflects, they all belong to existing popular ITSs and demonstrate valuable designs that inform the later student modeling research.

3.3.1 Student models using non-advanced-numerical techniques

The popular student models that apply non-advanced-numerical techniques include the ones in the two ITSs, ELM-ART and SQL-Tutor. The two student models can represent the prototype of other non-advanced-numerical student models.

3.3.1.1 Student models in ELM-ART

Brusilovsky et al. (1996) and Weber and Brusilovsky (2001) present a web-based ITS, ELM-ART (Episodic Learner Model-Adaptive Remote Tutor, available online at http://apsymac33.uni-trier.de:8080/Lisp-Course). This ITS helps students to learn LISP programming through adaptively guided, on-line learning materials and leveled hints that help students code small-scale LISP programs. ELM-ART uses the on-line materials to teach students the domain knowledge. It can recommend adaptive learning paths according to students’ learning histories when students read on-line material. After students finish particular parts of the material, ELM-ART tests students with a list of multiple-choice, yes-or-no, or fill-in-blank questions to detect students’ levels of understanding of the domain knowledge. ELM-ART also presents students a few simple problems, each of which normally needs one or two LISP functions to be solved.
ELM-ART displays two kinds of communication with students in two different windows: feedback and knowledge pages for both non-programming and programming tasks. Feedback tells students whether they made a correct answer and how to modify their errors, while knowledge pages display detailed definitions of all relevant concepts and examples similar to the current problem. The feedback for non-programming exercises is in one level and specifies whether the student’s answer is correct and states the correct solution. The feedback for programming tasks has three levels: the first level specifies where the errors are, the second level specify how to modify the errors, and the third level gives the correct solution. The feedback level is increased when students re-submit wrong solutions.

ELM-ART has two student models, one is the collaborative student model, and the other is the episodic student model. Both of them use non-advanced-numerical techniques to update the students’ knowledge level. The collaborative student model diagnoses the students’ knowledge states from the results of tests. This student model uses counting method to estimate students’ understanding level of each piece of domain knowledge. In this counting method, the learning state of each knowledge unit is represented by a confidence value. If a student answers a question correctly, a static value is added to the confidence value. If the student gives a wrong answer, the static value is subtracted from the unit’s confidence value. A knowledge unit is considered as a master unit when its confidence value reaches or exceeds a critical value. ELM-ART then uses the results to recommend a learning path to each student, in which at each step, EML-ART assumes that the student has mastered all prerequisites for the
next step. Usually, a learning path recommended to a student consists of material about knowledge units that has not been mastered by this student.

![Diagram of a possible derivation tree for the episodic student model](image)

**Figure 3-3 Illustration of a possible derivation tree for the episodic student model**

The other student model, the episodic student model, uses a hybrid technique of rule-based and case-based reasoning. It collects and arranges all relevant knowledge for each test exercise or programming task. This student model works like a giant library. It stores students’ knowledge in derivation trees and episodes (cases). Each derivation tree contains all information about which concepts and rules are needed to solve a problem. Each rule or concept is associated with a list of episodes or cases, each of which contains an existing student’s solution which applied this rule or concept. The knowledge units in a derivation tree are connected in the order of applying those concepts and rules to generate the solution. Figure 3-3 shows a derivation tree. The arrows represent the order between solution steps and the undirected links represent associations.
The episodic student model (Weber, 1996) uses matching to diagnose detailed errors in students’ programming solution. After a student types in his/her solution, a LISP compiler compiles and runs the solution program. If the compiling and running results are wrong, the student model compares the errors with typical errors that were stored in the existing derivation tree. The student model stops when all matches are found. Otherwise the student model stores the errors into episodes that have the relevant concepts in the derivation tree. Once the student model finds the detailed errors, an explanation according to the errors is shown to the student. When the student model cannot match an error, the student will see a recommendation for a possible modification according to the derivation tree. On the other hand, if the compiling and running results are correct, and the student’s solution is different from the solution stored in the derivation tree, the student model generates episodes for concepts that are applied in the student’s solutions, and then adds the episodes into the existing derivation tree.

The collaborative student model represents domain knowledge since it describes each piece of domain knowledge with a confidence value, whereas the episodic student model simulates partial monitoring knowledge or procedural knowledge since it arranges a set of concepts in a derivation tree in the order of solution steps. The episodic student model does not represent general monitoring knowledge which fits a class of problems. Instead, it merely represents the problem-specific procedural knowledge. Both student models do not consider reasoning knowledge which represents relationships among knowledge units, and reflective knowledge which represents positive strategies when students learn. Without learning
the general monitoring knowledge, students in ELM-ART can hardly master effective strategies to cope with new problems; without learning the reasoning knowledge, students may find it difficult to distinguish LISP concepts such as null and nil, atom and symbol; without modeling reflective knowledge, ELM-ART loses the chance to motivate students, and loosens up the connection between the role of tutors and the role of students.

The collaborative student model applies a simple algorithm with straightforward representation and update of knowledge contents and understanding level. Although the algorithm consumes less computing time and effort, it has a less rigorous mathematical foundation than the advanced-numerical approaches. The decision is made instinctively and without precise mathematical analysis about what value should be added to (or subtracted from) the confidence value of a concept when the students give a correct (or wrong) answer. As a result, the collaborative student model chooses a small value to add to or subtract from the confidence value to avoid reaching abrupt conclusions, which requires more questions to determine the students’ knowledge state than the advanced-numerical approaches do.

The episodic student model can accurately point out where the detailed errors are due to the LISP compiler, the use of derivation trees, and the giant collection of students’ previous errors. The student model represents and updates the contents instead of the grasping level of students’ procedural knowledge. Further, it does not diagnose why the student makes the mistake, e.g. which particular concepts and prerequisites the student does not know. Instead, all relevant knowledge to the programming task is displayed in the knowledge page and all correct steps are listed in
the feedback window without considering whether the student can understand the contents or not. Students cannot find clear connection between the contents in the knowledge page and the feedback. Consequently, students oftentimes are confused with the jumble of knowledge displayed, repeat the same errors, and eventually quit prematurely.

3.3.1.2 The student models in SQL-Tutor

Martin (1999), Mitrovic and Ohlsson (1999), Martin and Mitrovic (2000), and Mitrovic et al. (2001) present an intelligent tutoring system that helps students compose queries in a database language, SQL (Structured Query Language). SQL-Tutor provides about 170 problems that students can practice writing queries. After students enter their solutions, they can choose from six levels of feedback: simple feedback, error flag, hints, partial solution, error list, and complete solution. The simple feedback tells students how many errors there are in their solutions, error flag points out where the first error is, hints coarsely explain how to correct the first error, partial solution shows the first part of the solution, and the error list displays all the errors made. The authors stated that the purpose of so many levels of feedback is to help students to self-diagnose deficiency in their SQL skills. After the students successfully finish a problem, SQL-Tutor can select the next problem according to the students’ knowledge of SQL skills.

SQL-Tutor includes a constraint-based student model which represents students’ knowledge and diagnoses students’ errors with around 500 constraints. SQL-Tutor researchers compose constraints based on ideal solutions of problems and
empirically collected students’ solutions. The constraints indicate the common requirements among different correct solutions and common characteristics among various faulty solutions which need to be avoided.

Each constraint contains two components, a relevance condition and a satisfaction condition. The relevance condition specifies whether the constraint is relevant to the students’ solution, whereas the satisfaction condition identifies whether the students’ solution is correct. After students enter their solution to a problem, the student model compares the solution with the relevant constraints according to the form: if <relevance condition> is true, then <satisfaction condition> should also be true, otherwise there is an error. (Martin & Mitrovic, 2000)

The authors illustrated the comparison with the following example. If a student enters his/her solution to a problem “List names of all movies” and one of the relevant constraint is:

relevance condition: there is a “SELECT” clause

satisfaction condition: the attribute asked in the problem description must be of the same name as the expression of the “SELECT” clause of the query

From this constraint, the student model first checks if there is a “SELECT” clause in the student’s solution. If the student model gets a positive result, then it abstracts “name” out of the problem description and checks if the student’s solution has “name” as the expression in the “SELECT” clause. Once the student model finds the match the “SELECT” clause is correct, otherwise the clause is erroneous.

The constraint-based student model keeps records of the number of times each constraint has been satisfied or violated by a student, and which constraints he/she
satisfies and contravenes for the current problem. Hence, the student model can generate the analogous correct solution for the students by replacing the defective components with the satisfaction states in the violated constraints. SQL-Tutor provides feedback according to students’ errors. It groups the constraints based on errors and attaches all levels of feedback to each of the groups. Different levels of feedback contain messages of error identification and correct solution similar to students’ fallacious input. Once students submit an erroneous solution, the feedback that is attached to all contravened constraints is solicited for students to review.

![Diagram of a tree structure over constraints C, D, E, and F](image)

**Figure 3-4 Illustration of a tree structure over constraints C, D, E, and F**

SQL-Tutor applies another student model (constraint-concept model) to introduce a structure over the constraints, so as to diagnose students’ knowledge state over time (Martin, 1999). The structure consists of several trees, each of which pertains to a basic topic in SQL language. In the trees, the root nodes represent the basic topics and the leaf nodes represent the constraints. A basic topic consists of several sub-topics, each sub-topic consists of several concepts, and each concept
associates with a constraint. Hence, nodes for sub-topics connect to nodes for concepts, and they link the root node and the leaf nodes together (see Figure 3-4).

Students’ knowledge state in SQL-Tutor includes students’ understanding level on constraints, sub-topics, and topics. The student model diagnoses the students’ knowledge state by checking the pattern in the last four students’ performance logs. If the last four logs end with two consecutive correct answers, then the student model considers that the student has learned the constraint. If there is only one log which is correct, the student model considers the constraint as tentatively learned. Other patterns imply that the constraint is not learned. The learned state can propagate to upper level in the tree structure: once a constraint is learned, the concept associated with the constraint is learned; a sub-topic and a topic are learned when all concepts of the sub-topic and all sub-topics of the topic are learned. Similar is the propagation of un-learned state. SQL-Tutor shows the diagnosis results to students in a separate window. Hence, the students can perceive how well they understand the domain knowledge.

The constraint-based student model and constraint-concept student model in SQL-Tutor represent the domain knowledge with a set of constraints. The latter also models concepts, sub-topics, and topics. However, both student models have no relationship modeled among the constraints, though some constraints are prerequisites of others. Neither student model represents the knowledge about how to apply rules or concepts to a given problem and effectual strategies when learning in the ITS environment. Therefore, neither student model represents reasoning knowledge, the monitoring knowledge or reflective knowledge, as described in (Self, 1994). Without
explicit reasoning knowledge, SQL-Tutor does not explicitly build knowledge structure in students’ minds (though students may learn principles of the domain from as they correct errors). Without explicit monitoring knowledge, SQL-Tutor does not teach students how to lead to find the solution, only what is wrong with their work.

Kodaganallur, Weitz, and Rosenthal (2005) claim that constraint-based tutors tend to be product-centric, making inferences from student solutions, whereas model tracing tutors are more process-centric, making inferences from each step of the student’s problem solving process. Hence a constraint-based student model may be more applicable for 1) exercises whose solutions have rich information; and 2) exercises whose solution processes have small depth and branching factors. Exercises with large depth and branching factors for the solution processes may have an enormous number of possible solutions. As a result, a constraint-based student model would have difficulty identifying which step in the students’ solutions is fallacious and recommend correct solutions in favor of students’ attempts.

Mitrovic and Ohlsson (2006) respond to this critique (there are many arguments and counter-arguments) by noting that a constraint-based model can model problem solving steps as path constraints, when the correctness of a solution crucially relies on the order of steps. Though most constraint-based tutors have relied exclusively on solution state constraints, path constraints have occasionally been used in constraint-based systems to model student learning in highly procedural domains. It is not clear, however, if path constraints can support inferences about intermediate problem solving steps at the same level of detail as rules in model-tracing tutors.
3.3.2 Student models using advanced-numerical techniques

Popular advanced-numerical techniques applied in existing student models include fuzzy sets and Bayesian networks. Compared to ad hoc (non-advanced-numerical) approaches, advanced-numerical techniques have more rigorous mathematical foundation which can generate more predictable and accurate diagnoses. However, student models using advanced-numerical techniques are more complicated to develop, which surely requires more design and implementation time.

3.3.2.1 Student models using fuzzy sets

Compared to ordinary sets, fuzzy sets have members who do not have sharp boundaries but have an area of overlap in between. Fuzzy sets can describe objects’ states that have continuous values. When a member in a fuzzy set represents a state of an object, it provides a vague description with a truth value between 0 and 1. For example, a fuzzy set with members of tall and short can describe how tall a person is vaguely. Say a person’s height is 5'5” and it is in the overlap area between “tall” and “short”, the person can be described as tall with a truth value of 0.6, and short with a truth value of 0.4.

Although fuzzy sets seem to be similar to the manner that human tutor evaluates a student (Gürer, 1993), since there are no accurate rules to initialize and update the truth values in student modeling, fuzzy set can introduce some degree of ambiguity and arbitrariness.

3.3.2.1.1 Student model SMART in Stat Lady
Shute (1995) presented a student model, SMART (student modeling Approach for Responsive Tutoring) for a web-based ITS called Stat Lady, which helps students to learn basic statistics, and is described on-line at http://www.galaxyscientific.com/areas/training/edt3.htm. The ITS intersperses exercises in the instructional materials, and provides three levels of feedback for each exercise, where the first level tells that the student’s answer is incorrect, the second level identifies the particular errors, and the third level gives the correct solution. The feedback level is increased by one every time the students enter erroneous input. Thus, students see the correct answer after entering wrong solution three times.

According to the diagnoses of SMART, the ITS determines when to advance to the next instructional section and when to go back into the curriculum for explanation of a knowledge item. The author states that the criteria for advancing is when a student masters all knowledge items in the current section, and the criteria for going back is when the student has been diagnosed as not understanding a piece of knowledge.

SMART represents students’ understanding level for each knowledge item with a fuzzy set, which only has one predicate, mastered. The author first identifies the truth value of mastered through discrete representation such as remedial, intermediate, and mastery, with low vs. high divisions within each, and then associates values with each of the six states. SMART increases or decreases the truth value of mastered based on the feedback level that the students have reached to successfully solve a problem, where feedback level 0 means that the students make a correct answer at the first trial.
The author determines the initial truth values based on a student’s average scores in the pretest, which has multiple exercises for a knowledge item.

The author provides several mapping rules for promotion or demotion of the truth values in which a higher level of feedback results in a lower truth value. These rules have been approved by two experienced instructors in the domain. With the mapping rules, SMART updates the truth value of mastered after each time students enter a correct answer. Further, the author transforms the mapping rules to four regression equations by plotting the discrete points. Each point represents a rule and each equation compiles the mapping rules for one feedback level.

The author states two conditions under which to place the students back into the curriculum from remedial instruction. The two conditions are: 1) when the truth value for a knowledge item is decreased to a value below 0.6 and this knowledge item has not been explained; and 2) when the truth value is decreased to a value below 0.5 after two consecutive trials and the knowledge item has been explained.

In addition to simulating the students’ knowledge, the author proposes to model the students’ abilities, such as working memory capacity, associative learning skill, inductive reasoning, and information processing speed. The author tried to modify the four equations considering the students’ abilities, though she never reveals any new results in her later publications.

The way that SMART updates students’ knowledge is problematic. The essential idea behind the mapping rules is that updating of the truth value is based on how often the students try to solve the problem. If a student only makes one attempt, SMART increases the belief a great amount that the student understands the
knowledge item. On the contrary, if the student makes four times as much effort, SMART then lowers the belief a great quantity. The mapping rules only benefits the particular manner that Stat Lady issues the feedback since they cannot be applied until the students provide a correct solution. Even if the students have entered their erroneous answers, SMART cannot make any diagnoses, which decreases the flexibility of student modeling. Further, it has little tutoring benefits to let the students copy the correct solution when they do not understand the knowledge item relevant to the exercise. Thus, entering a correct solution does not contribute to students’ learning, yet is required by the algorithm of the system. This contradicts one of the rules in ITS design – machine should benefit learning (Ohlsson, 1986).

SMART does provide two insights for later student modeling research. First, the author introduces the idea of a responsive tutor, which customizes tutoring actions according to dynamic update of students’ knowledge state. The responsive tutor responds not only to errors but also to current students’ knowledge level during students’ problem solving. It goes one step further to intelligently satisfy individual student’s learning needs comparing to the previous student modeling, which detaches the relationship between tutoring actions and the diagnostic results of student models. Second, the author broadens the horizons of student modeling researchers by trying to simulate other than conventional student knowledge. Even though the author is not successful in simulating students’ abilities, the endeavor encourages later researchers to explore more possibilities such as simulating student’s cognitive strategies.
3.3.2.2 Student models using Bayesian networks

3.3.2.2.1 Basic Probabilistic Knowledge

A Bayesian network is a data structure and has great power to represent the causal relationships and hence infer probabilistic outcomes in a domain. Since a student’s knowledge is full of uncertainty and characterized by causal relationships and hierarchical structures (Millán & Pérez-de-la-Cruz, 2002; VanLehn, Niu, Siler, & Gertner, 1998), Bayesian networks are increasingly popular in designing and implementing student models. The definitions about Bayesian networks and basic probabilistic concepts can be referred in Russell and Norvig (1995) and Nilsson (1998).

Since it is not possible to examine happening of specific events exhaustingly, when probability theory is used to model the real world, a probability is about the belief of an event based on the observed occurrences so far. Therefore, probabilities change after receiving more evidence. Before the acquisition of any evidence, the probability can be set to any value or be obtained from a small size of sample data. This probability is called a prior probability and needs to be refined with more evidence. After the acquisition of new evidence, the updated probability is called posterior probability.

3.3.2.2.2 Bayesian Networks

Random variables in a domain may have causal relationships with each other. A Bayesian network explains the relationship between independent variables and
dependent variables probabilistically. Technically, a Bayesian network is a directed acyclic graph (DAG) which consists of nodes and links. Each node represents a random variable in a domain, and each link is an arrow that represents causal influence and points from a node of cause to a node of effect. A full joint probability distribution of all the random variables can be obtained through the products of conditional probabilities of each random variable given all parents in the Bayesian network. Therefore, any probabilistic question about the random variables can be answered by the full joint probability distribution represented in the Bayesian network. The procedure of answering questions is called probabilistic inference.

One common probabilistic inference from a Bayesian network is to update prior probability of variables for causes. The inference procedure requires prior probabilities of root nodes and conditional probability table (CPT) for non-root nodes. For instance, a Bayesian network for a domain that has three random variables, Pollution, Smoking, and Lung_cancer is shown in Figure 3-5. In the network, variables Pollution and Smoking are causes for variable Lung_cancer. The parameters for the Bayesian network are shown in table Table 3-1, Table 3-2, and Table 3-3.

![Figure 3-5 A Bayesian network Example](image)
Table 3-1 Prior probability for root node Pollution

| Pollution=True | 0.1 |

Table 3-2 Prior probability for root node Smoking

| Smoking=True | 0.3 |

Table 3-3 Conditional Probability Table for non-root node Lung_cancer

| Pollution | Smoking | P(Lung_cancer | Pollution, Smoking) |
|-----------|---------|-------------------|
| True      | True    | 0.6               |
| True      | False   | 0.2               |
| False     | True    | 0.4               |
| False     | False   | 0.05              |

If new evidence that random variable Lung_cancer equals true is obtained, the prior probability of the random variable Pollution is updated to a posterior probability:

\[
P(Pollution=true | Lung_cancer=true) = \frac{P(Pollution=true, Lung_cancer=true)}{P(Lung_cancer=true)}
\]

\[
P(Lung_cancer=true) \text{ can be calculated through } P(Pollution=true | Lung_cancer=true) + P(Pollution=false | Lung_cancer=true)=1, \text{ and } P(Pollution=true, Lung_cancer=true) \text{ can be calculated from full joint probability distribution of the Bayesian network through the summation of other random variables:}
\]

\[
P(Pollution=true, Lung_cancer=true) = \sum_{Smoking} P(Pollution=true, Lung_cancer=true, Smoking)
\]
The value in the full joint probability distribution, \( P(\text{Pollution}=\text{true},\ \text{Lung}_\text{cancer}=\text{true},\ \text{Smoking}=\text{true}) \) can be obtained from the products of conditional probabilities of each node in the Bayesian network as:

\[
P(\text{Pollution}=\text{true},\ \text{Lung}_\text{cancer}=\text{true},\ \text{Smoking}=\text{true}) \\
= P(\text{Lung}_\text{cancer}=\text{true}|\text{Pollution}=\text{true},\ \text{Smoking}=\text{true}) \times P(\text{Pollution}=\text{true}) \times P(\text{Smoking}=\text{true})
\]

When a node is a root, it has no parent and its conditional probability becomes the prior probability. All the conditional probabilities and prior probabilities can be found from the tables associated with the Bayesian network. The other value in the full joint probability distribution \( P(\text{Pollution}=\text{true},\ \text{Lung}_\text{cancer}=\text{true},\ \text{Smoking}=\text{false}) \) can be calculated similarly, and the other posterior probability \( P(\ \text{Pollution}=\text{false} \mid \ \text{Lung}_\text{cancer}=\text{true}) \) can also be calculated with the similar method.

A Bayesian network can eliminate the relationship between a random variable and its non-ancestors and provides a precise way to calculate the values in the full joint probability distribution of the random variables, and hence can answer any probabilistic queries about the variables. However, a Bayesian network requires exponential computational time, which can be represented as \( O(2^n) \), where \( n \) is the number of random variables in the network. Therefore, the computational time may explode with the increase of the number of the random variables.

### 3.3.2.2.3 Student models in ANDES

VanLehn et al. (2005) presented the ITS ANDES that guides students to solve physics problems. ANDES can be downloaded from
ANDES provides a list of problems for students to solve and supplies help in the solving of each problem. In ANDES, students need to input how they solve a physics problem, including drawing axis, listing equations, and assigning values. ANDES can diagnose students’ multiple solutions and supply hints for the next possible step according to the student’s current solution path. When students make errors, ANDES provides three levels of feedback: the first level flags the error, the second level hints how to modify the error, and the third level tells the correct solution.

VanLehn et al. investigated two Bayesian network student models, the concept student model (1998) and the solution student model (1995 & 2001). The concept student model diagnoses close-ended questions and the solution student model diagnoses close-ended exercises. They tried to synthesize the diagnoses from the two exercises. However, through the research of the concept student model, the authors find that a Bayesian network does not need accurate priors, and assessment can be based on the difference between the prior and posterior probabilities, i.e., if the posterior probability is higher than the prior, the student masters the rule; otherwise there is insufficient evidence to conclude students’ mastery. The priors for the rules mean “the probability that a randomly drawn student from the population will have already mastered that rule before using ANDES.” (VanLehn et al., 1998)

The authors’ purpose for using the concept student model is to determine the prior probabilities of the solution student model. Before solving physics problems, students need to finish a list of problems that either have multiple choice or numerical answers. The students’ solutions are diagnosed by the concept student model to
ascertain individual student’s mastery level for each physics rule. In the concept student model, rules are causes for students’ answers and link nodes of relevant rules to the node of a problem. Figure 3-6 illustrates the structure of Bayesian network that the concept student model uses.

![Bayesian network diagram](image)

**Figure 3-6 Illustration of the concept student model in ANDES**

The concept student model applies a “noisy-and” relationship among the causes in the Bayesian network. Noisy-and is an uncertain relationship which is generalized from the logical AND (Pearl 1988). Whereas logical AND requires that a student understands all rules before he/she answers the problem correctly, noisy-and allows for uncertainty about the knowledge of rules. The authors determine the conditional probability table through slip and guess factor, where the slip factor represents the probability that students make an error when they understand the knowledge, and the guess factor represents the probability that the students make a correct answer by guessing when they do not master the knowledge. The authors apply the concept student model to 600 simulated students and find that the prior of 0.5, which represents the initial understanding level for each concept in the domain, yields the best accuracy of the diagnoses.
The solution student model simulates the application of multiple rules in a solution graph when students solve a physics problem. All possible students’ solutions are represented in the solution graph. The student model contains three kinds of nodes: 1) nodes for rules, 2) nodes for application of rules, and 3) nodes for facts that are given in the problem or derived by the students. Students’ actions are represented by the nodes for facts. The authors assume that the rules and the facts cause the application for the rules, and the application for rules cause the students’ answers which belong to deducted facts.

Figure 3-7 Illustration of the solution student model in ANDES

Figure 3-7 illustrates the structure of Bayesian network that the solution student model applies. This figure shows that fact3 is one of the students’ possible solutions, and the two arrows pointed to the node of facts represent two possible solution paths that reach the solution. All possible erroneous solutions are also incorporated in the solution student model along with their solution paths, which include all possible erroneous rule-application, rules, and facts. After a student enters a
recognizable solution step, the node associated with the step is invoked, and then the updates are propagated to its ancestors.

If the solution step is wrong, the student needs tutoring with the erroneous rules with high posterior probabilities and the correct rules with the low posterior probabilities (Gertner, Conati, & VanLehn, 1998). The next possible step along the student’s solution path is also predicted from the Bayesian network. Hence, hints for the next step can be provided to students. The authors state that predicting next step requires much less calculating time than the diagnoses of mastering rules. To alleviate the burden of long updating time of the Bayesian network, the authors propose to keep students in one solution path through the feedback. Therefore, the nodes not in the solution path do not need to be updated.

The solution student model applies noisy-and among the links pointing from the nodes of rules and facts to the nodes of rule-application, and noisy-or among the links from the nodes of rule-application to the nodes of students’ answers. Similar to noisy-and, noisy-or is an uncertain relationship which is generalized from the logical OR (Pearl 1988). Logical OR requires that at least one of the rule-application happens before a student answers the problem correctly, while noisy-or allows for uncertainty about the existence of rule-application.

Similar to the concept student model, the parameters of the Bayesian networks in the solution student model are determined by the slip and guess factor. For a node of rule-application, the conditional probability is 1 minus slip when all the values for its parents are true, and 0 for any value whose parent nodes are not true. This implies that students can make a correct rule-application only after they know all the relevant
facts and the rule. For a node of a student’s solution, the conditional probability is the value of *guess* when all the values for its parents are not true, and 1 for any value whose parents is true. This implies that students will enter a correct solution step by guessing only when they did not make any correct rule-application; otherwise they will input a correct step. For a node of a fact that has no parent, the prior is assigned to 0.95, which indicates that students have put the fact in their working memory. For a node of a rule, the prior is obtained from the results of the concept student model.

The solution student model predicts a student’s next solution step and diagnoses erroneously applied *rules*, instead of unknown *concepts*. The concept student model does not diagnose students’ understanding level of concepts either. Concepts represent knowledge at a finer granularity than rules. For example, one ANDES rule says that if the velocity of an object is constant, then its acceleration is zero, rather than representing the concepts velocity or acceleration.

Furthermore, both the concept student model and the solution student model do not model prerequisite relationships between rules. The successful real-time feedback in ANDES is to diagnose any error in the current step and to guide a student to the next correct solution step. It does not tell the underlying concepts that the student needs to learn to avoid errors. The breath and depth in the solution student model increase exponentially when the number of solution steps and possible solutions increases. Although ANDES has resorted to approximate algorithms in order to avoid exponential running time (VanLehn & Niu 2001), approximate algorithms are arbitrary with respect to how much of the network they consider. Furthermore, the
updating time for the Bayesian networks in the solution student model may still grow explosively because the augmenting size of the network is exponential.

The student modeling research in ANDES provides several useful conclusions: 1) Nodes of knowledge are causes of the nodes of students’ solutions in a Bayesian network that diagnoses students’ knowledge; 2) noisy-and is the relation among all causal links between the nodes of knowledge and the nodes of solutions; 3) assessment of students’ knowledge can be determined by the difference between the prior and the posterior probability of nodes of knowledge; and 4) priors of 0.5 yield the highest accuracy for diagnoses as compared to other values when diagnosing students’ mastery of knowledge.

3.3.2.2.4 A Student model for a CAT

Millán and Pérez-de-la-Cruz (2002) proposed a way to represent the knowledge of students and introduced a method of building a Bayesian student model from the results of a computer adaptive test (CAT). The authors propose to refine the Bayesian student model to approach the student’s real knowledge level through selected questions.

The authors represent the student’s knowledge by concepts, topics, and subjects. A subject consists of several related topics, a topic consists of several related concepts, and a concept is the basic unit of knowledge. The student model in this paper is composed of two parts. One part models the relationship between the student’s answers and his/her understanding level of each concept. The other part
models the aggregation relationships between different knowledge units such as subjects, topics, and concepts.

Figure 3-8 illustrates how to find a student’s knowledge about concepts based on his/her answers. For example, question2 (q2) is related to two concepts (c2 and c3). The probability of correctly answering q2 depends on the student’s knowledge about concepts c2 and c3, which is determined through a modified Item Characteristic Curve (ICC) model in Item Response Theory (IRT). Using the conditional probabilities \( P(a2|c2,T,c3=F) \) as parameters, the student’s knowledge about concepts c2 and c3 can be determined through Bayesian network algorithm. The modified ICC model considers the student’s knowledge level, and slip and guess factor. Slip factor simulates that students make an error when they have understood the relevant knowledge, while

| c1 | c3 | c4 | \( P(a1|c1,c3,c4) \) |
|----|----|----|----------------|
| T  | T  | T  | 0.99          |
| T  | T  | F  | 0.975         |
| T  | F  | T  | 0.939         |
| T  | F  | F  | 0.861         |
| F  | T  | T  | 0.717         |
| F  | T  | F  | 0.522         |
| F  | F  | T  | 0.345         |
| F  | F  | F  | 0.2           |

\( P(c1) \) – probability about how well a student understands concept1
\( P(a2|c2=T,c3=F) \) – probability of the student correctly answering question2 on conditional of knowing concept2 and not knowing concept3
guess factor models that students make a correct answer by guessing when they do not have the knowledge.

Once the degree of understanding of each concept has been found, the degree of understanding of topics and subject can be inferred. Figure 3-9 illustrates how to find a student’s knowledge about topics and subjects based on his/her knowledge about concepts. For example, topic2 is related to two concepts (c3 and c4). To measure the student’s knowledge about topic2, a variable \( t_2 \) is defined as

\[
t_2 = c_3 \cdot w_3 + c_4 \cdot w_4
\]

where \( w_3 \) and \( w_4 \) are weight of concept c3 and c4 respectively, and \( w_3 + w_4 = 1 \). \( c_i \) equals to 1 if the probability \( p(c_i) \) determined from the Bayesian student model is larger than or equal to 0.5, otherwise \( c_i \) equals to 0. Similarly, to measure the student’s knowledge about the subject, a variable \( s \) is defined as

\[
s = t_1 \cdot w_{t1} + t_2 \cdot w_{t2}
\]

where \( w_{t1} \) and \( w_{t2} \) are weight of topic t1 and t2 respectively, and \( w_{t1} + w_{t2} = 1 \).

**Figure 3-9 A student model example for aggregation relation**
*(adapted from Millán & Pérez-de-la-Cruz (2002))*

The authors propose to use the dynamic Bayesian student model to consider the influence of the student’s previous knowledge, as shown in Figure 3-10. Compared to the static Bayesian student model shown in Figure 3-8, the dynamic student model
uses two more parameters $P(c_1^1 | c_2^0)$ and $P(c_3^1 | c_3^0)$, which represent the influence of the estimated student’s knowledge at time 0 on the estimated knowledge at time 1. Since the authors assume that the student’s actual knowledge level did not change during the time when students are solving the problems in a CAT, the authors conclude that if the current estimate knowledge level equals the previous estimated knowledge level, the conditional probability is 1, otherwise it is 0.

\[
\begin{array}{c|c|c|c}
\text{c}_1 & \text{c}_3 & \text{c}_4 & \text{p(a1|c1,c3,c4)} \\
T & T & T & 0.99 \\
T & T & F & 0.975 \\
T & F & T & 0.939 \\
T & F & F & 0.861 \\
F & T & T & 0.717 \\
F & T & F & 0.522 \\
F & F & T & 0.345 \\
F & F & F & 0.2 \\
\end{array}
\]

\[
\begin{array}{c|c|c}
\text{c}_3 & \text{p(c}_3^1 | \text{c}_3^0) \\
\text{c}_3^0 = \text{c}_3^1 & 1 \\
\text{c}_3^0 \neq \text{c}_3^1 & 0 \\
\end{array}
\]

\[
\begin{array}{c|c|c|c|c|c}
\text{c}_2 & \text{c}_4 & \text{p(a2|c2,c3)} \\
T & T & 0.99 \\
T & F & 0.939 \\
F & T & 0.717 \\
F & F & 0.2 \\
\end{array}
\]

\[
\begin{array}{c|c|c|c|c|c|c}
\text{c}_1^0 & \text{p(c}_1^1 | \text{c}_1^0) \\
\text{c}_2^0 = \text{c}_2^1 & 1 \\
\text{c}_2^0 \neq \text{c}_2^1 & 0 \\
\end{array}
\]

c\_2^1 – updated knowledge state about concept c\_2 at time 1

**Figure 3-10 Dynamic Bayesian student model example**
(adapted from Millán & Pérez-de-la-Cruz (2002))

The authors propose to refine the student model to represent the student’s actual knowledge level through selected questions from the test pool. Before students take the CAT, the student model gives each student an initial knowledge level that can be 0.5 or the averaged level of previous students who did the CAT. Then the system
selects a question from the test pool for the student answer. The student model updates each student’s knowledge level based on his/her answers.

The authors evaluate the student model through 180 simulated students. At the beginning of a test, they set the prior probability for each concept for all students to 0.5. They stop the test when all concepts were evaluated. The concepts with the probability value higher than 0.7 are diagnosed as known, while lower than 0.3 as unknown. The validity of the student model is shown by comparing the derived knowledge level with the pre-described knowledge level of each simulate student, which is represented by the Accurate Diagnoses Rate (ADR). The student model reaches an ADR of 90.27% for randomly selected problems.

The authors represent and measure the student’s knowledge using concepts, topics, and subjects and model the aggregation relation among them. Nevertheless, the authors did not specify prerequisites relation among concepts. Instead, they consider the influences by putting all concepts in one layer in their dynamic Bayesian student model, which did not show the actual probabilistic causal mechanism in student learning. Thus, the student model cannot model the actual students’ knowledge efficiently.

In the proposed student model, the conditional probabilities of students’ answers giving concepts are not accurately determined. For example, it was randomly assumed that the probability of correctly answering the question knowing concept c3 would be higher than knowing concept c4, i.e., \( P(a_1|c_1=F,c_3=T,c_4=F) \) is larger than \( P(a_1|c_1=F,c_3=F,c_4=T) \), as shown in Figure 3-8. Further, the conditional probabilities of current knowledge giving previous knowledge are not accurate. The authors
propose that the conditional probability of current knowledge given previous knowledge was either 1 or 0 (as shown in Figure 3-9), which is equivalent to not considering the influence of previous knowledge level. The conditional parameters need to be determined accurately for highly dynamic problems that have much limitation on diagnosis time during a learning process full of feedback, and further investigation is necessary to mathematically quantify the influence among the knowledge levels at different time instances.

The authors also explain how to create simulated students and use them to evaluate the student model. Experimenting with human subjects is very expensive and many unexpected factors can make results deviate from intended. Large amounts of data are needed to prove that the student model really works. Using simulated students solves the problems of limited human subject resources. Furthermore, simulated students can be repeatedly used without any extra expenses when testing the student model. However, the authors assume the simulated students’ actual knowledge level do not change, which cannot satisfy the problems involving constantly-changing knowledge level during a learning process full of responsiveness.

In conclusion, Millán and Pérez-de-la-Cruz (2002) presented a good way to represent students’ knowledge and a good start of creating a dynamic Bayesian student model. They also introduced an economical way to evaluate and test the student model using simulated students. However, the authors did not consider hierarchical knowledge structure when diagnosing the understanding level of concepts on condition of the student’s answers. And more investigation is needed to mathematically quantify the causal effects of the students’ learning history on their
current knowledge level, especially for a highly dynamic student model in a responsive learning process.

### 3.4 Common problems in previous student model research

There are six common problems revealed from the review of previous student model research:

1. Previous student model research considers rule-based behaviors or individual concepts as part of the students’ learning goals. Nevertheless, students often are confused about the relationship among individual concepts such as prerequisite, transition, similarity and distinction.

2. Previous research introduces a variety of student model architectures. Yet none of them reveals a standard way to represent layered knowledge in a student model according to the students’ knowledge (Self, 1994).

3. Student modeling embracing probabilistic techniques such as Bayesian networks requires exponential computational time, since the number of parameters and the updating time for Bayesian networks is of exponential order, and hence cannot provide consistent support for real-time communicative tutoring.

4. There are few student models that simulate students’ knowledge history, because of the complex mechanism required and the volatility in the evolution of students’ knowledge. Fewer researchers introduced dynamic Bayesian networks in solving the problem, while none of them models the students’ hierarchical knowledge structure at the meantime.
5. Existing student models separate the inferred students’ knowledge from closed- and open-ended exercises. The corresponding ITSs determine the next step of learning materials according to the knowledge from close-ended questions, and tutor problem solving errors based on the knowledge from open-ended tasks. Few of the existing student models integrate these two kinds of knowledge to represent the synthetic students’ learning state.

6. The currently available student models simulate partial students’ knowledge, including basic domain knowledge and specific procedural problem solving skills. Few of them tackle the tutoring needs of students’ cognitive strategies, including general and domain-specific.

To tackle the six problems, this research presents a three-layered student model which 1) represents concepts and important relationships, such as prerequisites and similarities that can cause confusion; 2) tracks the history of student behavior; 3) integrates knowledge from open-ended problem solving (object-oriented class diagram design) and close-ended exercises; 4) infers cognitive strategies germane to both the problem domain (object-oriented software development) and learning in general; and 5) does all these things in real time. Therefore, the ITS as a whole can be responsive to students as they work on assigned problems.
4 Methodology

4.1 Three-layered student model architecture

This research presents a three-layered student model (Wei, Moritz, Parvez, & Blank, 2005) which provides adaptive tutoring for each student. The architecture of the student model is shown in Figure 4-1. The Problem-Domain Model infers how well the student understands relevant concepts, from student solutions annotated by the Expert Evaluator. The Historical Model infers the historical knowledge state of the student from a sequence of student solutions. Finally, the Cognitive Model infers general and domain-specific problem solving strategies from student work and errors. The three levels can then provide different kinds of information to the pedagogical advisor about where the student needs help.

![Figure 4-1 Architecture of the Three-Layered student model in DesignFirst-ITS](image-url)
4.1.1 The Problem-Domain Model (PDM)

The Problem-Domain Model (PDM) maintains a student’s knowledge state and solution history. The solution history consists of answers to exercises and quizzes in the CIMEL multimedia and design solution steps in the Eclipse IDE. The PDM specifies how the student’s knowledge state is tied to CIN (Curriculum Information Network) in terms of concepts he/she knows or does not know. Each concept is associated with a probabilistic value to show how well the student knows the concept in the context of the current problem the student is solving. The PDM uses Atomic Bayesian networks to provide a refined representation of prerequisite relationships, to diagnose student’s knowledge structure, and to achieve real-time responsiveness. The PDM also helps to maintain a history of when, what, and how well the student learns.

4.1.2 The Historical Model (HM)

The history maintained in the PDM enables the Historical Model (HM) to trace how many times the student commits similar errors. The HM simulates the student’s knowledge in a history and models the student’s hierarchical knowledge structure in the meantime. It generalizes an overall student’s knowledge state from what are observed in the PDM, by tying it to the CIN in terms of knowledge items he/she knows and does not know. The HM applies dynamic Bayesian networks to represent refined prerequisite relationships and diagnoses in real-time students’ knowledge structure considering the learning history. The HM helps to maintain the student’s learning history. This history enables explaining the problem from the student’s perspective.
4.1.3 The Cognitive Model (CM)

The Cognitive Model (CM) recognizes problem solving strategies that a student is using. During the past 40 years many student models attempted to select the appropriate level of advice and explanations, to determine readiness for advancement and dynamic planning of the student’s curriculum, to flag his/her current performance, and to hint at the next possible step in problem solving (Katz, Lesgold, Eggn, & Gordin, 1992). Few student models have incorporated a CM into their approach to find the student’s learning needs and to characterize the problem solving strategies that the student is using. Mitrovic et al. (2001) argued that feedback with only the correct answer would be sufficient to help the student when he/she makes an error. But oftentimes when the student sees the right answer he/she does not know what it means and why it is correct, especially in complex, open-ended domains such as object-oriented design. So the student’s problems are likely to recur. Tu, Hsu, and Wu (2002) used a mapping technique to map the tutoring with a specific error and maintained an information map which had all the hard coded mappings. This approach does not fit problems where the students’ learning needs are related to a wide variety of contexts in which the error is made.

To analyze a student’s problem solving performance, Gürer (1993) incorporated three kinds of cognitive strategies, i.e. “Knowledge type”, “Focus”, and “Approach”, into her student model for tutoring physics problems. The experimental results show that her CM is helpful in diagnosing the student’s solution. The “Knowledge type” strategy determines whether the student is using preconceived notions or actual physics knowledge. The “Focus” strategy determines whether the
student only focuses on the problem’s surface features or on physics principles. Finally, the “Approach” strategy determines whether the student uses a top-down approach or bottom-up approach. Physics problems are more procedurally oriented than object-oriented problem solving. Techniques for solving physics problems involve finding useful equations for given facts and deducing new facts that are underneath. This research alters the definition of cognitive strategies from Gürer (1993) and adds more kinds of strategies and hence adapts to the general and domain-specific problem solving.

The general and domain-specific problem solving strategies can be divided into patterns and anti-patterns. Previous literature shows that there are patterns of problem solving that produce effective, high-quality solutions to recurring problems while anti-patterns that produce ineffective, low-quality solutions or none at all. Patterns are often non-obvious to beginners; anti-patterns may often seem more obvious but actually are misleading blind alleys. If the problem solving patterns of a student are sound then the student can solve the object-oriented problems correctly and efficiently because of the problem solving patterns he/she is using. But students often use anti-patterns of problem solving which causes inefficient and even wrong solutions.

The general problem solving strategies that are incorporated into the CM include hacking, analogy, focus, and approach. Hacking is a common anti-pattern of novices. It includes two kinds: 1) hacking the solution, and 2) hacking the tutor. When students hack the solution, they enter variant answers by guessing and hope one of them happens to be the correct answer. When students hack the tutor, they take advantage of the way that the system provides feedback (Walker et al., 2006; Aleven
et al., 2005). They may require correct solutions too soon without serious thinking. Hacking is incorporated into the CM to encourage students to think thoroughly instead of entering reckless answers.

The analogy strategy has three cases: if_analogy, right_analogy, and analogy_adaptation. If_analogy determines whether the student used analogy by copying existing design in his/her solution. Right_analogy determines whether the student makes appropriate analogies. Only similar components (classes, attributes, methods, or parameters) can be candidates for analogy. Analogy_adaptation determines whether the student adapts the copied design into the proper class.

The focus strategy determines whether students focus on the requirement of problem or provide answers that are irrelevant to the problem. It has been observed that first-year college students oftentimes ignore the problem description during their design and bring irrelevant objects into the solution. The approach strategy implies a sound strategy where concrete and apparent components are the first to be designed and the abstract and ambiguous components are the second. Many novices of object-oriented design do not know this strategy and are stuck at composing the difficult components at the beginning when they try to solve the problem. Proper tutoring actions ought to be issued when the anti-patterns are detected.

The domain-specific cognitive strategies pertaining to object-oriented design include the strategies of designing correct class, attributes, methods, and parameters. The Design-first curriculum teaches students the procedures how to design class diagram from problem description:
1. Read the problem description at least twice and ask questions about any confusion.

2. Underline each noun phrases which has multiple nouns, e.g. “color of my car”.

3. Parse each sentence: put an ‘S’ below each subject, pub a ‘V’ below each meaningful verb (If a sentence mainly describes an action then only mark the main verb and ignore other verbs in the sentence), and put an ‘O’ below each noun in the objects,

   e.g. “To \underline{withdraw}, \underline{the\ customer\ enters\ the\ amount\ of\ money}.” where “enters” is ignored.

4. Analyze each subject with the following questions and steps:
   a. Does a subject represent a person performing an action? If yes, then it is an actor, add ‘A’ after ‘S’.
   b. Does a subject take a simple value (such as ‘color’ or ‘money’)? If yes, then it is probably an attribute, add ‘T’ after ‘S’.
   c. The remaining subjects are classes. Add ‘C’ after ‘S’ for each subject. Focus on one class at a time.

5. Analyze all nouns in all grammatical objects with the following questions and steps:
   a. Does noun represent a person? If yes, then it is an actor, ‘A’. Add ‘A’ after ‘O’.
b. Does it take a simple value? If yes, then it is probably an attribute, Add ‘T’ after ‘O’.

c. Figure out the datatype for each candidate attribute, e.g., ‘money’ has datatype of double, ‘name’ has datatype of String.

6. Examine all verb marked with ‘V’ in the sentences (if a verb is not marked with a ‘V’ then it is ignored):

   a. Each ‘V’ is a method. Add a ‘M’ after each ‘V’.

   b. Find all verb/object pairs. Each noun in the verb/object pair is a possible parameter for the corresponding methods.

   c. Add ‘P’ after ‘O’ for each possible parameter.

   d. Set the return type of the method to be the data type of the value the method returns. If the method returns nothing then the return type is void.

   e. If possible attributes and possible parameters have overlap, use following tips to distinguish them:

      i. An attribute is accessible anywhere within the scope of a class, e.g., inside one class, any methods can directly use any attributes. While a parameter is accessible only within the scope of an object’s method.

      ii. Once an item is set to an attribute, it should not be set to a parameter again.
Here is an example about how to generate an object-oriented design using the domain-specific cognitive strategies. The task is to design a movie ticket machine. The problem description is:

The movie ticket machine displays the movie title, displays the show time, and displays the price of a ticket. A customer enters money. The machine displays the customer balance. The customer enters the number of tickets. The machine prints number of tickets, movie title, and show time on a ticket when there are available seats, and returns the customer balance. The machine also tracks the number of available seats.

Following the rules in the domain-specific cognitive strategies, the problem description can be analyzed as:

The **movie ticket machine displays the movie title, displays the show time, and displays the price of a ticket.**

A **customer enters money.**

The **machine displays the customer balance.**

The **customer enters the number of tickets he needs.**

The **machine prints number of tickets, movie title, and show time when there are seats available.**

The **machine returns the customer balance.**
The machine also tracks the number of available seats.

The following shows a possible good object-oriented design from the problem description according to the domain-specific cognitive strategies:

Class: MovieTicketMachine

Attribute:

- String movieTitle
- String showTime
- double priceTicket
- int numberAvailableSeats
- double customerBalance
- int numberTickets

Method:

- void displayMovieTitle()
- void displayShowTime()
- void displayTicketPrice()
- void displayCustomerBalance()
- void enterMoney(double moneyEntered)
- void enterNumberTickets(int numberTicketEntered)
- void printTicket()
- double returnChange()
- void trackNumberSeats()
The Cognitive Model (CM) recognizes cognitive strategies that a student applies. It incorporates general cognitive strategies such as focus, approach, hacking, and analogy, and domain-specific cognitive strategies from problem description to class diagram. The strategies represent different problem solving patterns and anti-patterns in the domain of object-oriented design and learning in general. The CM enables the pedagogical advisor to remedy a student’s deficient problem solving anti-patterns and encourage sound ones.

**4.2 Flow of control in the student model**

The PDM receives student performance data from the multimedia and the Expert Evaluator (Blank, Parvez, Wei & Moritz, 2005). In the multimedia the student performance data consists of student’s performance on each quiz or exercise. In the Eclipse IDE the student performance data includes the problem presented to the student, the student’s solution, the expert’s solution, the CIN concepts used to create the expert solution, and the gaps between the expert and student solutions as identified by the Expert Evaluator. From the data, the PDM infers knowledge items the student have learned and how well he/she knows them. Each knowledge item in the inferred results has a time stamp and a probabilistic value and is associated with a question or a step in a design solution. All knowledge items are organized according to the CIN. The known/unknown concepts related to current problems are sent to the Historical Model (HM).

The HM calculates how well the student knows all the concepts in the CIN considering his/her learning history. Each concept in the result has a probabilistic
value and a time stamp. These concepts comprise the student’s knowledge state along his/her learning history. If a prerequisite concept with the newest time stamp for the missing concepts in the student solution has a very low probability, the possible reason for the student error can be he/she does not know the prerequisite. The unknown prerequisites are sent to the CM to let it infer the reasons for the student’s error.

The CM simulates general and domain-specific cognitive strategies and measures cognitive measurement flags for erroneous strategies or anti-patterns from the progress on students’ problem solving. Each cognitive measurement flag associates with a time stamp. The CM synthesizes and analyzes the unknown prerequisites and the cognitive measurement flags to infer the tutoring needs of the students.

4.3 Knowledge representation

According to cognitive science theory, a sound knowledge state should show a highly connected and well-defined structure (Chi & Koeske, 1983; Albacete & VanLehn 2000). Students need not only knowledge of individual concepts, but also the relationships among concepts, such as similarity, difference, usage and a-part-of, to build up a sound knowledge state. Knowledge representation for a student model needs to be represented with the granularity of concepts and logical relationships that students need to learn. Self (1994) describes different levels of knowledge: domain knowledge, reasoning knowledge, monitoring knowledge, and reflective knowledge. The levels cross from specific to general, from concrete to abstract knowledge. Our student model represents the two lower levels, domain and reasoning knowledge.
Domain knowledge is a set of propositions which explain all concepts that belong to the vocabulary for discussing or solving problems. Reasoning knowledge is a set of propositions which contain reasoning relationships between propositions in the domain knowledge. It covers some of the gap that the domain knowledge lacks. Reasoning knowledge enhances the understanding and remembrance of domain knowledge. Self found that each unique proposition in the domain knowledge cannot stay by itself in students’ minds for long unless the students understand the reasoning knowledge that can connect the domain knowledge together. However, other than reasoning relationships, there are still other relationships among propositions that are essential for students to understand, remember, and apply the domain knowledge. The other relationships are the similarity, difference, and all possible relationships that can connect together the separate domain knowledge propositions. Suppose there are only two concepts, attribute and parameter in the object-oriented (OO) design domain:

Domain Knowledge (Object-oriented design)

= {Attribute – An attribute is a characteristic of an object;
Parameter – A parameter is a variable that passes data into a method}

Reasoning Knowledge (Object-oriented design)

= {Attribute-parameter – Both attributes and parameters are variables. They have different scope. An attribute can be accessed in any methods in this class, while a parameter can only be accessed in its method}

This example shows that domain knowledge is not adequate for students to solve a realistic problem (e.g., design an attribute and a parameter in an Object-oriented design). Real students commit errors when they defined ‘movieTitle’ as a
parameter after they has already defined ‘movieTitle’ as an attribute. One reason why students make this error is that they do not understand the different between a parameter and an attribute. Therefore, in addition to unique concepts, such as attribute and parameter, students need to understand relationships that emerge between them, such as the differences between these concepts, and when to use which (Wei & Blank, 2006). Other examples of relationships in the domain of object-oriented design and programming include **datatype_variables, parameter_returntypes**, that a constructor shares the name of its class, etc. Though relationships tend to be harder to specify and understand the unique concepts, they are crucial to learning a complex knowledge structure.

The knowledge scheme in this research represents these relationships as explicit nodes in a network. A pair \((ku, au)\) is used to model the causal relationship between an immediate concept or knowledge unit and an action step that a student takes to solve a problem. A \(ku\) is a knowledge unit, which means the knowledge that students need to learn. There are two kinds of \(ku\): concepts and relationships between concepts. For example, the relationship Attribute_Parameter models the difference between concepts Attribute and Parameter (a common confusion for novices). It has been observed from preliminary results that students frequently struggle to understand relationships between concepts, such as the difference between Attribute and Parameter (and when to use which), or between integer and double, etc. An \(au\) is an action unit, which is a single step in a student’s solution, e.g. writing a name for an attribute. From the definition of the pair \((ku, au)\), \(ku\) directly causes \(au\).
As shown in Figure 4-2, the knowledge units that students need to understand to solve the object-oriented problems are modeled in a Curriculum Information Network (CIN) for the student model. All the knowledge units are connected by the prerequisite links. By convention, a prerequisite is a concept that a student needs to understand before understanding another concept. Different teachers may use different curricula which results to a different CIN. So more broadly, any concept one needs to teach before introducing a new concept is also a prerequisite. *Immediate* prerequisites are those concepts that are strongly related to a concept and play the most important role in understanding the concept. The concept Class in the CIN is not the aggregation of concepts Attribute and Method. Instead it represents a category of objects. A student understands the concept of Class if he/she can identify correct class names.
4.4 Atomic Bayesian Networks

ABNs implement the PDM layer in the student model. They apply Bayesian networks to diagnose students’ understanding level based on students’ solution. An Atomic Bayesian Network (ABN) focuses on just one concept, its immediate prerequisites, and its relationship to a solution step. An ABN models both the causal association between a student’s solution and the most relevant concepts, and models prerequisite relationships among these concepts. It indicates that mastery of those concepts causes whether the student makes the current solution step correctly or not. In other words, an ABN models two kinds of relationships: 1) the student needs to understand a concept at the center of an ABN before he/she can make the current solution step correctly, and 2) the student needs to understand all of the immediate prerequisites of the center concept before he/she is ready to understand the center concept.

4.4.1 Definition of an Atomic Bayesian Network (ABN)

As Figure 4-3 shows, an ABN is a directed graph composed of one edge \((k_u, a_u)\) and multiple edges \((\text{immediate-prerequisite}(k_u), k_u)\), in which \(k_u\) and \(a_u\) make a pair \((k_u, a_u)\). Immediate-prerequisite\((k_u)\) represents the knowledge units that must be taught right before teaching \(k_u\). A noisy-and relationship is enforced among all edges \((\text{immediate-prerequisite}(k_u), k_u)\). Noisy-and is an uncertain relationship which is generalized from the logical AND (Pearl 1988, Millán & Pérez-de-la-Cruz 2002). Whereas logical AND requires that a student understand all immediate prerequisites of \(k_u\) before he understands \(k_u\), noisy-and allows for uncertainty about the knowledge of
immediate prerequisites. It assumes that allowing each immediate prerequisite is independent of allowing the others. For example, the concept Numeric-datatype has two immediate prerequisites: Int and Double. Noisy-and assumes that the joint events (numeric-datatype, int) and (numeric-datatype, double) are mutually independent. Given this independence, parameters in the conditional probability table for a noisy-and relationship take the product of the conditional probability values of each parent.

![Diagram of Atomic Bayesian Network (ABN)]

Figure 4-3 Atomic Bayesian Network (ABN)

All of the variables (nodes) in an ABN have binary values, true or false. They are defined as follows:

- The variable $au$ (the leaf node) represents how a student makes a solution step in a constructive exercise, such as an Object-oriented design problem. The value of true means the student makes a correct step, while false means a wrong step.

- The variable $ku$ in the center represents if the student knows the most relevant concept for the current solution step. The value is true when the student understands the concept and false otherwise.
The variable $p_1(ku)$ (the root nodes) represents if a student knows the immediate prerequisites of the center concept. The value is true when the student understands the prerequisite and false otherwise.

A student’s knowledge about $ku$ is noted as $k|(\text{student}, ku)=\{0, 1\}$ where 1 means the student understands $ku$ while 0 not. When $k|(\text{student}, ku)=1$ (a student understands $ku$) he/she might still make a wrong step because of a slip or unintentional mistake. Or, when $k|(\text{student}, ku)=1$ (the student does not know $ku$) he/she might guess the solution correctly. Guessing here is not the same as blindly guessing since many students may have misconception about the knowledge and then make a wrong answer. Let’s also consider the possibility of errors deriving the center concept from its prerequisites. Even if the student knows all the immediate prerequisites, a student might not understand the center concept. Or a student might guess the correct meaning of the center concept even if he does not understand any immediate prerequisite. These characteristics in student learning can be applied to find the conditional probability tables in an ABN.

Four variables to determine the conditional probability tables in an ABN are formally defined as follows:

- $slip_e$ is the probability a student makes a wrong step when $k|(\text{student}, ku)=1$ (he/she knows $ku$) (Murray, 1998; VanLehn et al. 1998; VanLehn & Niu, 2001; Millán, Agosta, & Pérez-de-la-Cruz, 2001) where $e$ means an evidence or a solution step:

$$P (au = false \mid ku = true) = slip_e \quad \text{or} \quad P (au = true \mid ku = true) = 1 - slip_e$$
• $guess_e$ is the probability that a student makes a correct step when $k|(\text{student, } ku)=0$ (he/she does not understand $ku$) (Murray, 1998; VanLehn et al. 1998; VanLehn & Niu, 2001; Millán et al., 2001):

$$P (au = true \mid ku = false) = guess_e$$

• $slip_p$ is the probability a student fails to understand the center concept when he knows one immediate prerequisite, where $p$ means a prerequisite (i.e., a slip in the causal relationship from a prerequisite to center concept).

• $guess_p$ is the probability that a student understands the center concept when he does not know any immediate prerequisites.

A conditional probability table between the nodes of $au$ and $ku$ can be calculated from $slip_p$ and $guess_e$ from their above definitions. A conditional probability table between the nodes of immediate-prerequisite ($ku$) and $ku$ can be calculated by the definition of $slip_p$, $guess_p$ and noisy-and as

$$P(ku = true \mid p_i(ku), ... p_j(ku)) = \prod_{i \in K} (1 - slip_p) \prod_{j \in \overline{K}} guess_p$$

(4-1)

where $p_i(ku)_i = true$, $i \in K$, $p_i(ku)_j = false$, $j \in \overline{K}$, $K \cup \overline{K}$ is a set of all immediate prerequisites of the center concept $ku$, and $K = \{ p_i(ku)_i = true \mid i \in [1, n] \}$, a set of immediate prerequisites that the student knows, while $\overline{K}$ is a set of immediate prerequisites that the student does not know.

From the definition of conditional probability:

$$P(ku = true \mid p_i(ku)_i = true, ... p_j(ku)_j = false)$$

$$= \frac{P(ku = true, p_i(ku)_i = true, ... p_j(ku)_j = false)}{P(p_i(ku)_i = true, ... p_j(ku)_j = false)}$$
Because the events of knowing immediate prerequisites of a concept are mutually independent, and because noisy-and assumes that joint events of knowing an immediate prerequisite and knowing the concept are also mutually independent, then the conditional probability of a knowledge unit given its prerequisites becomes:

\[
P(ku = \text{true}, p_i(ku)_i = \text{true}) \quad \frac{\cdots \cdot \cdots \cdot \cdots P(ku = \text{true}, p_i(ku)_j = \text{false})}{P(p_i(ku)_i = \text{true}) \quad \frac{\cdots \cdot \cdots \cdot \cdots P(p_i(ku)_j = \text{false})}{}}
\]

Applying the definition of conditional probability again, the conditional probability of an ABN, taking slip and guess into account, becomes:

\[
P(ku = \text{true} \mid p_i(ku)_i = \text{true}) \ast \cdots \ast P(ku = \text{true} \mid p_i(ku)_j = \text{false})
\]

\[
\prod_{i \in K} (1 – \text{slip}_p) \prod_{j \in K} \text{guess}_p
\]

This proves Equation (4-1). Millán et al. (2001) used a similar formula to calculate the conditional probability table for the link between the knowledge nodes and a solution node. However, they did not consider prerequisite knowledge. We use the formula to calculate the conditional probability table between the immediate-prerequisite nodes and the center concept node. Thus we consider prerequisite knowledge in updating student knowledge level.

### 4.4.2 Advantages of an ABN

Every solution step correlates to an ABN which stores the updated value for the ABN of next solution step. Each knowledge unit has at most \(k\) immediate parents (smaller than six), and altogether there are \(n\) knowledge units in the domain. Therefore, an ABN will:
• Need \( O(1) \) running time instead of \( O(2^n) \) for each solution step because it has a small bounded number of immediate parents.

• Update \( O(1) \) nodes instead of \( O(n) \) nodes for each solution step.

• Determine 4 parameters instead of \( O(nk) \) parameters in a noisy-and relationship.

Using an ABN reduces the running time for each step from exponential for a complete Bayesian network to constant time because the ABN only considers its immediate parents, which is a small bounded number for any knowledge domain. The number of conditional parameters drops to 4—two pairs of guess and slip. The number of nodes that must be updated for each step drops to the number of parents. In any domain, the number of immediate parents is much smaller than the number of total variables.

Other tutoring systems using the Bayesian networks have resorted to approximate algorithms in order to avoid exponential running time (Millán & Pérez-de-la-Cruz, 2002; VanLehn & Niu, 2001). To improve the efficiency of the propagation process, Millán and Pérez-de-la-Cruz (2002) used a goal-oriented algorithm to compute a reduced sub-graph where the propagation took place. To balance the precision and computing time, VanLehn and Niu (2001) used Likelihood Sampling to update a network of 110 nodes and needed 30 seconds to reach the precision of 70%. Compared to their Bayesian student models, an ABN is more efficient. An ABN is also sufficient, because the relationship between a center concept and ancestor prerequisites is tenuous at best. For example, from the CIN in Figure 4-2, `class` is a prerequisite of `class_constructor` (the constructor has the same name as the
class), which in turn is a prerequisite of constructor. From the solution step focusing on **constructor** (ask a student to give a name for the constructor), an ABN only updates **class_constructor**, not **class**, because the prerequisite relation between **class** and **constructor** is intuitively tenuous. Understanding **constructor** implies that the student understands how to define the name of a constructor for a class instead of the understanding of **class**. It is sufficient just to update the immediate prerequisite. Simulation results preliminarily support the claim about the sufficiency of ABNs (experiments with real students to validate ABNs’ sufficiency will be performed as well in this dissertation).

The use of ABN accelerates the diagnosis of a student’s knowledge state because the ABN models the prerequisite concepts. Table 4-1 compares the student modeling approaches between considering and not considering prerequisites where C represents the related concept of S, a solution step, and A and B are the prerequisites of C. The student model that does consider prerequisites is based on Millán and Pérez-de-la-Cruz (2002). They consider that all the three concepts are the direct causes for the solution step. The ABN student model (Wei & Blank, 2006) considers the prerequisite relationship between concepts A, B, and C.
Using the two student models, the posterior probabilities corresponding to a correct solution and a wrong solution are calculated. A posterior probability from 0.7 to 1 means the concept is known, a posterior probability from 0.3 to 0.7 means undiagnosed, and a probability from 0 to 0.3 means unknown (Millán & Pérez-de-la-Cruz, 2002). Table 4-1 shows that when the prerequisites are considered, the student model is less possible to end up with undiagnosed states.

Millán and Pérez-de-la-Cruz determine the conditional probability for the concepts based on Item Response Theory and slip and guess values (Section 3.3.2.2.4). They rank all concepts before computing the conditional probability. In the example shown in Table 4-1, they consider C is more important than B and B is more important than A. Thus, conditional probabilities for concepts A, B, and C are different.

### 4.4.3 A concrete example of an ABN

To illustrate how to build an ABN, a concrete example is provided. Suppose a student is designing a class diagram for an ATM problem. A class diagram requires
the entering of a class name first, followed by either attributes or methods. Suppose that the student has added a correct class name and several correct attributes. The knowledge levels which are posterior probabilities for concepts related to these solution steps have been stored in a database. A current solution step would be that the student enters a name for his first method. Suppose the Expert Evaluator (EE) (Sally, 207; Blank, Parvez, et al., 2005; Wei et al., 2005) determines that the student entered a wrong name for the method, for example totalDeposit as a method (instead of an attribute). The Expert Evaluator produces a packet indicating that the student created a method where an expert would create an attribute. From this difference, the student model determines that the center concept of an ABN in question is attribute_method. It also finds all immediate prerequisites of this concept, attribute and method, from the CIN. Figure 4-4 shows the structure of the ABN.

The ABN needs prior probabilities for root nodes and conditional probability tables for nodes that are not root. Prior probabilities for concept attribute can be found from the database. Because it is the first time the student adds a method, there is no record for the concept method. Then its prior probability is 0.5, since the students’ knowledge state is accessed based on the difference between the prior and posterior probabilities (VanLehn et al., 1998; Millán et al., 2001). Suppose that slipe=guesse=0.1 and slipp=guessp=0.1, the conditional probability table for the concept attribute_method and the node au (action unit) according to Equation (4-1) is shown in Table 4-2 and Table 4-3.
Figure 4-4 A concrete example of an ABN

Table 4-2 Conditional probability table for concept Attribute_Method

| attribute | method | P(attribute_method|attribute, method) |
|-----------|--------|-----------------------------|
| True      | True   | 0.81                        |
| True      | False  | 0.09                        |
| False     | True   | 0.09                        |
| False     | False  | 0.01                        |

Table 4-3 Conditional probability table for node au

| attribute_method | P(au|attribute_method) |
|------------------|--------------------|
| True             | 0.9                |
| False            | 0.1                |

4.4.4 Updating an ABN

When students do exercises they will encounter several problems that relate to same concepts. There are two ways to calculate the students’ understanding level to each concept according to the problems that students answered. The first way is to calculate the understanding level of a concept based on students’ solution to all problems that relate to the concept after the students finish all the problems. It needs one time of calculation. The second way is to calculate the understanding level of a concept after the students finish each problem that relates to the concept. This requires
multiple times of calculation. The final results in the two ways are equivalent to each other. However, calculating the understanding level along with students doing the problems has more advantages, which allows the student model to monitor students’ learning at multiple spots instead of only one spot when students finish all problems. Let’s use a simple example to show the reasons why the final results from the two ways are same.

![Bayesian network diagram](image)

**Figure 4-5 A simple example Bayesian network**

Figure 4-5 shows a Bayesian network, where node $S_1$ represents a solution to a problem which relates to a concept $K$ at time spot 1 (represented by node $K_1$). Similarly, node $S_2$ represents a solution at time spot 2 which relates to node $K_2$, representing the student’s understanding level of concept $K$ at that time. We can have an updated understanding level for concept $K$ after the student did the two problems, which is $P(K|S_1, S_2)$. We have two ways to get the value $P(K|S_1, S_2)$. The first way is that we calculate $P(K|S_1, S_2)$ directly. The second way is that we calculate $P(K|S_1)$ first, and then use $P(K|S_1)$ as $P(K)$ for node $K_1$ to calculate $P(K|S_2)$ as in Figure 4-6.
Figure 4-6 Update the Bayesian network in two steps

First, we calculate $P(K|S_1, S_2)$ using the first way. In Figure 4-5, from the process of inference in Bayesian networks (Russell & Norvig 1995), the conditional probability of $P(K|S_1, S_2)$ is:

$$P(K | S_1, S_2) = \frac{P(K, S_1, S_2)}{P(S_1, S_2)}$$

$$= \frac{1}{P(S_1, S_2)} P(S_1 | K) P(S_2 | K) P(K)$$

From the chain rule (Nilsson, 1998) we get $P(S_1, S_2) = P(S_2 | S_1) * P(S_1)$. So we can rewrite the above result as,

$$P(K | S_1, S_2) = \frac{P(S_2 | K)}{P(S_2 | S_1) * P(S_1 | K) P(K)} P(S_1 | K) P(K)$$

Next, we calculate $P(K|S_1, S_2)$ using the second way. In Figure 4-6, since we sample and update in steps, we get every observance at the second step given the first step has happened. Thus we can use $P(K|S_1)$ as $P(K)$ for node K_1 and $P(S_2|S_1)$ as $P(S_2)$ for node S_2: $P(K)$ is set to $P(K | S_1)$, $P(S_2)$ is set to $P(S_2 | S_1)$. From the Bayes’ rule (Russell & Norvig 1995), we can calculate $P(K | S_1)$ as,
\[ P(K \mid S_1) = \frac{P(S_1 \mid K) \cdot P(K)}{P(S_1)} \]

Thus, \( P(K) \) is set to \( \frac{P(S_1 \mid K) \cdot P(K)}{P(S_1)} \).

From the process of inference in Bayesian networks (Russell & Norvig 1995), the conditional probability of \( P(K\mid S_2) \) is:

\[ P(K \mid S_2) = \frac{P(K, S_2)}{P(S_2)} = \frac{P(S_2 \mid K) \cdot P(K)}{P(S_2)} \]

Replace \( P(K) \) with \( \frac{P(S_1 \mid K) \cdot P(K)}{P(S_1)} \), we get:

\[ P(K \mid S_2) = \frac{P(S_2 \mid K)}{P(S_2)} \frac{P(S_1 \mid K) \cdot P(K)}{P(S_1)} \]

\[ = \frac{P(S_2 \mid K)}{P(S_2) \cdot P(S_1)} P(S_1 \mid K) P(K) \]

\[ = \frac{P(S_2 \mid K)}{P(S_2 \mid S_1) \cdot P(S_1)} P(S_1 \mid K) P(K) \]

Therefore, the two ways, calculating \( P(K\mid S_1, S_2) \) in one time or calculating \( P(K\mid S_1) \) first, and then using \( P(K\mid S_1) \) as \( P(K) \) for node \( K \) to calculate \( P(K\mid S_2) \) generate the same results. This dissertation applies the second way to update ABNs, e.g., we update an ABN each time a student answers a problem.
Updating an ABN includes updating the understanding level of the center concept node and the prerequisite concept nodes according to a student’s solutions. We can update the ABN using the overall ABN. We can also update an ABN in multiple steps. Each step involves a sub-network in an ABN as shown in Figure 4-7. Updating an ABN using sub-networks in the ABN is the same as updating the ABN using the overall network. However, updating in multiple steps makes it possible to generalize the updating procedure for ABNs with any number of parent nodes. The following example will show that the final results from the two ways of updating are equivalent.

Figure 4-8 An example of an ABN
First, we update the ABN in Figure 4-8 using the ABN. From the process of inference in Bayesian networks (Russell & Norvig 1995), the conditional probability of \( P(D|S) \) is:

\[
P(D|S) = \frac{P(D,S)}{P(S)} = \frac{1}{P(S)} \sum_{A,B,C} P(S|D)P(D|A,B,C)P(A)P(B)P(C)
\]

Since \( P(S|D) \) can be moved to outside of the summation sign, we can get the conditional probability of \( P(D|S) \):

\[
P(D|S) = \frac{P(S|D)}{P(S)} \sum_{A,B,C} P(D|A,B,C)P(A)P(B)P(C) \quad (4-2)
\]

We can also get the conditional probability for parent nodes given the student’s solution,

\[
P(A|S) = \frac{P(A,S)}{P(S)} = \frac{1}{P(S)} \sum_{D,B,C} P(S|D)P(D|A,B,C)P(A)P(B)P(C) \quad (4-3)
\]

\[
P(B|S) = \frac{P(B,S)}{P(S)} = \frac{1}{P(S)} \sum_{D,A,C} P(S|D)P(D|A,B,C)P(A)P(B)P(C) \quad (4-4)
\]

\[
P(C|S) = \frac{P(C,S)}{P(S)} = \frac{1}{P(S)} \sum_{D,B,A} P(S|D)P(D|A,B,C)P(A)P(B)P(C) \quad (4-5)
\]
Second we show how to update the ABN in steps and the final result is the same as updating the ABN in one step. Figure 4-9 shows that we can update the ABN in three steps: 1) calculate the unconditional probability of the center node D based on its immediate parents; 2) update the center node according to a student’s solution; and 3) update the parent nodes according to the student’s solution.

From the process of marginalization in Bayesian networks (Russell & Norvig 1995), the unconditional probability, \( P(D) \) in the first sub-network in the ABN is:

\[
P(D) = \sum_{A,B,C} P(A,B,C,D) = \sum_{A,B,C} P(D | A,B,C) P(A) P(B) P(C)
\]  

(4-6)

From the process of inference in Bayesian networks, \( P(D | A) \) in the first sub-network is:

\[
P(D | A) = \frac{\sum_{B,C} P(A,B,C,D) P(D | A,B,C) P(A) P(B) P(C)}{P(A)} = \frac{\sum_{B,C} P(D | A,B,C) P(A) P(B) P(C)}{P(A)}
\]  

(4-7)
According to Bayes’ rule, the conditional probability given the solution, \( P(D|S) \) is:
\[
P(D \mid S) = \frac{P(S \mid D) \cdot P(D)}{P(S)}
\]  
(4-8)

Use Equation (4-6) to replace \( P(D) \) in Equation (4-8), we get the same result as in Equation (4-2). From the definition of conditional probability and marginalization in Bayesian networks, we get the conditional probability of parent nodes given the solution node:
\[
P(A \mid S) = \frac{P(A, S)}{P(S)}
\]

From marginalization or summing out in Russell & Norvig (1995), we can get the probability of \( P(A,S) \) by summing out variable \( D \). Thus, we get,
\[
P(A \mid S) = \frac{P(A, S)}{P(S)} = \frac{\sum \limits_{D} P(S, D, A)}{P(S)}
\]

From the chain rule (Nilsson, 1998), we can expend \( P(S, D, A) \) so that
\[
P(A \mid S) = \frac{\sum \limits_{D} P(S \mid D, A)P(D \mid A)P(A)}{P(S)}
\]
\[
= \frac{\sum \limits_{D} P(S \mid D)P(D \mid A)P(A)}{P(S)}
\]  
(4-9)

If we use Equation (4-7) to replace \( P(D|A) \) in Equation (4-9), we get the same result as in Equation(4-3):
\[
P(A | S) = \frac{\sum_{D} P(S | D) \sum_{B,C} P(D | A, B, C) P(B) P(C) P(A)}{P(S)}
\]

\[
= \frac{\sum_{D,B,C} P(S | D) P(D | A, B, C) P(A) P(B) P(C)}{P(S)}
\]

We can use a similar procedure to get the same results as in Equation (4-4) and Equation (4-5). Therefore, we can update an ABN in three steps using sub-networks and get the same results as using the overall ABN. This characteristic is due to the special structure of ABN which has one centered node. The characteristic guarantees that ABN can have \(O(1)\) updating time, since each knowledge unit has at most \(k\) immediate parents where \(k\) is a small bounded number of immediate parents. In this dissertation, \(k\) is less than six.

In the above calculation with multiple steps, two values, \(P(D)\) and \(P(D|A)\) from the second step, are referred to in the third step to calculate \(P(D|S)\) and \(P(A|S)\). The two values simplify the calculation of \(P(D|S)\) and \(P(A|S)\). Hence, we can generalize the following formulas.

The unconditional or parent-conditional probability of a node in an ABN only depends on all of its parents, e.g., in an ABN, define all parents of a node \(D\) as \(\text{parents}(D)\), then the unconditional probability of node \(D\) is:

\[
P(D) = \sum_{\text{parents}(D)} P(D | \text{parents}(D)) \prod_{\text{parents}(D)} P(\text{parent})
\]

The parent conditional probability of node \(D\) given \(X\) is:

\[
P(D | X) = \sum_{\text{parents}(D) - X} P(D | \text{parents}(D)) \prod_{\text{parents}(D) - X} P(\text{parent})
\]
Figure 4-10 An example of an ABN

We can infer a node’s unconditional probability and parental conditional probability using its immediate parent nodes. For example, we can use node D’s parent nodes A, B, …, C to calculate the probability of node D, $P(D)$, or parent-conditional probability of node A, $P(D|A)$, $P(D|B)$, …, $P(D|C)$. Then, we can update nodes in an ABN using the two formulas. To be more specific, the detail formulaa using the example in Figure 4-10 are listed below:

$$P(D) = \sum_{A,B,\ldots,C} P(D|A,B,\ldots,C) \cdot P(A) \cdot P(B) \cdot \ldots \cdot P(C) \quad (4-10)$$

$$P(D|A) = \sum_{B,\ldots,C} P(D|A,B,\ldots,C) \cdot P(B) \cdot \ldots \cdot P(C) \quad (4-11)$$

$$P(D|B) = \sum_{A,\ldots,C} P(D|A,B,\ldots,C) \cdot P(A) \cdot \ldots \cdot P(C) \quad (4-12)$$

$$\ldots$$

$$P(D|C) = \sum_{A,B,\ldots} P(D|A,B,\ldots,C) \cdot P(A) \cdot \ldots \cdot P(B) \quad (4-13)$$

The formulas can help to calculate the conditional probabilities of nodes A, B, C, …, D given the student’s answer in an ABN as shown in Figure 4-10. The following will describe how to update an ABN in different situations.
Figure 4-11 An ABN with no parent

Figure 4-11 shows an ABN with no parent. Node A represents a concept and node S represents a student’s solution. From the process of marginalization (Russell & Norvig 1995), we get the unconditional probability of node S:

$$P(S) = \sum_A P(S, A) = P(S, A) + P(S, \neg A)$$

If we use Bayes’ rule to extend $P(S, \neg A)$ and $P(S, A)$, the formula changes to

$$P(S) = P(S \mid A)P(A) + P(S \mid \neg A)P(\neg A)$$

If we use the definition of $slip_e$ and $guess_e$ (section 4.4.1), the formula changes to

$$P(S) = slip_e \ast P(A) + guess_e \ast P(\neg A) \quad (4-14)$$

From the definition of conditional probability (Russell & Norvig 1995), we get the updated conditional probability for node A:

$$P(A \mid S) = \frac{P(S, A)}{P(S)}$$

From the process of marginalization (Russell & Norvig 1995), we get the conditional probability for node A given solution node S,
If we use Bayes’ rule to extend $P(S, \neg A)$ and $P(S, A)$, the formula changes to,

$$P(A \mid S) = \frac{1}{1 + \frac{P(S \mid \neg A)P(\neg A)}{P(S \mid A)P(A)}}$$

From the definition of $slip_e$ and $guess_e$ (Section 4.4.1),

$slip_e = P(\neg S \mid A) = 1 - P(S \mid A)$, $guess_e = P(S \mid \neg A)$ we can get the formula for updating an ABN with no parent,

$$P(A \mid S) = \frac{1}{1 + \frac{guess_e * P(\neg A)}{(1 - slip_e) * P(A)}} \quad (4-15)$$

Figure 4-12 An ABN with two parents

Figure 4-12 shows an ABN with one parent. The updating for the ABN can be divided to three steps: 1) calculate the unconditional probability of node D based on its parent A; 2) update the node D according to the solution node S; and 3) update the
parent nodes A according to the solution node S. The three steps are show in Figure 4-13.

![Figure 4-13 Three steps to update an ABN](image)

From Equation (4-10) we can get the unconditional probability of the center node D,

\[
P(D) = \sum_A P(D \mid A)P(A) = P(D \mid A)P(A) + P(D \mid \neg A)P(\neg A)
\]

From the definition of \(slip_p\) and \(guess_p\) (Section 4.4.1), \(slip_p = P(\neg D \mid A) = 1 - P(D \mid A)\), \(guess_p = P(D \mid \neg A)\) we can get the expression for P(D) with \(slip_p\) and \(guess_p\) and the parent node A,

\[
P(D) = (1 - slip_p) * P(A) + guess_p * P(\neg A)
\]  

(4-16)

This equation is denoted as \(P(D) = ((1 - slip_p) + guess_p)A\), which means that Equation (4-16) consists of \((1 - slip_p)\) and \(guess_p\) and involves the probability of node A.

From Equation (4-15) we can get the conditional probability, P(D|S),
\[ P(D \mid S) = \frac{1}{1 + \frac{\text{guess}_e \cdot P(\neg D)}{(1 - \text{slip}_e) \cdot P(D)}} \]

where \( P(D) \) can be calculated by Equation (4-16).

According to the definition of conditional probability, marginalization, and chain rule in the Bayesian network, we can calculate the conditional probability of the parent node given the solution node,

\[
P(A \mid S) = \frac{P(S, A)}{P(S)} = \frac{\sum_D P(S, D, A)}{\sum_D P(S, D)} = \frac{\sum_D P(S \mid D) P(D \mid A) P(A)}{\sum_D P(S \mid D) P(D)}
\]

\[
= \frac{P(A) \cdot (P(S \mid D) P(D \mid A) + P(S \mid \neg D) P(\neg D \mid A))}{P(S \mid D) P(D) + P(S \mid \neg D) P(\neg D)}
\]

\[
= \frac{P(A) \cdot (P(S \mid D) P(D \mid A) \cdot \frac{P(S \mid \neg D) \cdot P(\neg D \mid A)}{P(S \mid \neg D) \cdot P(D) \cdot P(\neg D)}}{1 + \frac{P(S \mid \neg D) \cdot P(D)}{P(S \mid \neg D) \cdot P(D)}} \cdot \frac{1 + \frac{\text{guess}_e \cdot P(\neg D \mid A)}{1 - \text{slip}_e \cdot P(D)}}{1 + \frac{\text{guess}_e \cdot P(D)}{1 - \text{slip}_e \cdot P(\neg D)}}
\]

From the definition of \( \text{slip}_p, \text{guess}_p, \text{slip}_e, \) and \( \text{guess}_e \), we can get the formula for \( P(A \mid S) \),

\[
P(A \mid S) = \frac{P(A) \cdot P(D \mid A)}{P(D)} \cdot \frac{1 + \frac{\text{guess}_e \cdot P(\neg D \mid A)}{1 - \text{slip}_e \cdot P(D)}}{1 + \frac{\text{guess}_e \cdot P(D)}{1 - \text{slip}_e \cdot P(\neg D)}}
\]

Equation (4-17) shows that updating of the parent node needs \( \text{slip}_e, \text{guess}_e, \) \( P(A), P(D), \) and \( P(D \mid A) \), where \( P(D) \) is from Equation (4-16), \( P(D \mid A) \) is \( 1 - \text{slip}_p \), and \( P(A) \) is from previous ABNs.
Figure 4-14 An ABN with two parents

Figure 4-14 shows an ABN with two parents. The updating for the ABN can be divided to three steps. The three steps are show in Figure 4-15.

Figure 4-15 Three steps to update an ABN

Similar to the procedure in updating an ABN with no parent,

\[
P(D) = \sum_{A,B} P(D \mid A, B)P(A)P(B)
\]

\[
= P(D \mid A, B)P(A)P(B) + P(D \mid \neg A, B)P(\neg A)P(B)
+ P(D \mid A, \neg B)P(A)P(\neg B) + P(D \mid \neg A, \neg B)P(\neg A)P(\neg B)
\]

From Equation (4-1) we express P(D) with \(\text{slip}_p\) and \(\text{guess}_p\) and the parent nodes A and B,

\[
P(D) = (1 - \text{slip}_p)^2 P(A)P(B) + (1 - \text{slip}_p)\text{guess}_p P(A)P(\neg B)
+ \text{guess}_p (1 - \text{slip}_p)P(\neg A)P(B) + \text{guess}_p^2 P(\neg A)P(\neg B)
\]

(4-18)
This equation is denoted as \( P(D) = ((1 - \text{slip}_p) + \text{guess}_p)_{A,B} \), which means that Equation (4-16) consists of teams \( (1 - \text{slip}_p)^2 \), \( (1 - \text{slip}_p)\text{guess}_p \), and \( \text{guess}_p^2 \), and involves the probabilities of nodes A and B.

From Equation (4-11) and Equation (4-12) we can get the conditional probability, \( P(D|A) \) and \( P(D|B) \),

\[
P(D|A) = \sum_B P(D|A,B)P(B) = P(D|A,B)P(B) + P(D|A,\neg B)P(\neg B)
\]

\[
P(D|B) = \sum_A P(D|A,B)P(A) = P(D|A,B)P(A) + P(D|A,\neg B)P(\neg A)
\]

From Equation (4-1) we express \( P(D|A) \) with \( \text{slip}_p \) and \( \text{guess}_p \) and the parent nodes A and B,

\[
P(D|A) = (1 - \text{slip}_p)^2 P(B) + (1 - \text{slip}_p)\text{guess}_p P(\neg B)
\]

\[
= (1 - \text{slip}_p)((1 - \text{slip}_p)P(B) + \text{guess}_p P(\neg B)) \tag{4-19}
\]

\[
P(D|B) = (1 - \text{slip}_p)^2 P(A) + (1 - \text{slip}_p)\text{guess}_p P(\neg A)
\]

\[
= (1 - \text{slip}_p)((1 - \text{slip}_p)P(A) + \text{guess}_p P(\neg A)) \tag{4-20}
\]

Equation (4-19) is denoted as \( P(D|A) = (1 - \text{slip}_p)((1 - \text{slip}_p) + \text{guess}_p)_{B} \), which means that Equation (4-19) consists of teams \( (1 - \text{slip}_p)(1 - \text{slip}_p) \) and \( (1 - \text{slip}_p)\text{guess}_p \), and involves the probability of node B. Similarity, Equation (4-20) is denoted as \( P(D|B) = (1 - \text{slip}_p)((1 - \text{slip}_p) + \text{guess}_p)_{A} \).

From Equation (4-17) we can calculate the conditional probability for the parent nodes given the student’s solution,
where \(P(D), P(D|A),\) and \(P(D|B)\) can be found in Equation (4-17), (4-18), and (4-19).

In conclusion, an ABN that has \(n\) parent nodes and a center node \(D\) as shown in Figure 4-10 can be updated in three steps: 1) calculate the unconditional probability for the center node \(D\), and calculate the conditional probability for the center node given each immediate parent node; 2) update the center node according to the solution node; and 3) update each immediate parent node according to the solution node.

If we define all parents of a node \(D\) as \(\text{parents}(D)\) and \(X\) as an immediate parent of node \(D\), the formulas are listed as following: (Equation (4-21) consists of teams of \( (1 - slip_p)^n \), \( (1 - slip_p)^{n-1} \text{guess}_p \), \( (1 - slip_p)^{n-2} \text{guess}_p^2 \), ..., \( (1 - slip_p)^2 \text{guess}_p^{n-2} \), \( (1 - slip_p) \text{guess}_p^{n-1} \), and \( \text{guess}_p^n \), and involves the probabilities of each of the nodes in \(\text{parents}(D)\))

\[
P(D) = ((1 - slip_p) + \text{guess}_p)^{\text{parents}(D)}
\]

(4-21)

\[
P(D | S) = \frac{1}{1 + \frac{\text{guess}_e \cdot P(-D)}{(1 - slip_e) \cdot P(-D)}}
\]

(4-22)

\[
P(D | X) = (1 - slip_p)((1 - slip_p) + \text{guess}_p)^{\text{parents}(D) - X}
\]

(4-23)
\[ P(X \mid S) = \frac{P(X) \cdot P(D \mid X)}{P(D)} \cdot \left( 1 + \frac{\text{guess}_e \cdot P(-D \mid X)}{1 - \text{slip}_e \cdot P(D \mid X)} \right) \frac{1}{1 + \frac{\text{guess}_e \cdot P(D)}{1 - \text{slip}_e \cdot P(-D)}} \]  

(4-24)

Since updates of node D depends on its immediate parents, to update node D, we can look for the unconditional probability of each immediate parent. The conditional probability of node D given its immediate parents can be calculated from values of slips and guesses. Any previous updates for the immediate parents have been stored in their unconditional probability when the immediate parents are center nodes in other ABNs, where their conditional probability are calculated given the related solution steps. After this time of updating, the conditional probability of node D is stored as unconditional probability of node D for next time of calculation of another ABN.

### 4.4.5 Pseudocode for updating an ABN

Below is the pseudocode for the functions that update an ABN.

```plaintext
function ABNUpdate (student, targetConcept, CIN, allProfile) returns allProfile
    FindAllPrerequisites(CIN, targetConcept)
    find <student, theProfile> for the student from allProfile
    find studentProfile for current student
    FindProbabilities(targetConcept, prerequisites[num], theProfile)
    if targetConcept has no prerequisites then
        update targetConcept according to equation (4-22)
        insert < targetConcept, probability> in theProfile
```
else if current targetConcept has prerequisites then

calculate value according to equation (4-21)

update targetConcept according to equation (4-22)

put probability of targetConcept in theProfile

UpdateParentNodes(prerequisites[num], theProfile)

insert <student, theProfile> into allStudentProfile

return allStudentProfile

function FindAllPrerequisites(CIN, targetConcept) returns prerequisites[num]

find <targetConcept, prerequisites> from CIN

num←number of prerequisites

prerequisites[num] ←all prerequisites of the targetConcept

return prerequisites[num]

function FindProbabilities(targetConcept, prerequisites[num], theProfile) returns probabilities[num+1]

if theProfile is not null then

if find <targetConcept, probability> in theProfile

then set value to probability of the targetConcept

else

then set 0.5 to probability of targetConcept

for each of the num prerequisites for targetConcept do

if find <prerequisite, probability> in theProfile

then set the value to probability of the prerequisite

else
then set 0.5 to probability of the prerequisite

return probabilities[num+1]

function UpdateParentNodes (prerequisites[num], theProfile) returns theProfile

update parent nodes according to equation (4-23) and (4-24)

insert <prerequisite, probability> for each parent node in theProfile

return theProfile

4.5 Atomic Dynamic Bayesian Networks

ADBNs (Wei & Blank, 2007) implement the HM layer in the student model. They apply dynamic Bayesian networks to take into account students’ learning history. An Atomic Dynamic Bayesian Network (ADBN) consists of multiple Atomic Bayesian Networks (ABN). Each ABN belongs to a single time slice when a student does a solution step. An ABN (Wei & Blank, 2006) focuses on just one knowledge unit, its immediate prerequisite, and its relationship to a solution step. Based on the first-order Markov process (Russell & Norvig, 1995), an ABN for the current solution step depends on the ABN of the last solution step for the same knowledge unit and not on any earlier solution steps. This simplification assumes that students’ learning states at future time slices are independent of the ones at the past, given the students’ current learning state.

As Figure 4-16 shows, an ADBN is a directed graph composed of two smaller directed graphs, in which ku′ and au′ make a pair (t=0, 1). Immediate-prerequisite(ku) represents the knowledge units that must be taught before teaching ku. Causal links connect nodes for knowledge units at time slice 0 and 1. A noisy-and relationship
(Millán et al., 2001; Peral, 1988; Wei & Blank, 2006) is enforced among all edges (immediate-prerequisite$(ku)$, $ku$) and edge $(ku^0, ku^1)$. Noisy-and allows for uncertainty about a student’ knowledge of $ku$ at previous time slice and the knowledge of immediate prerequisites of $ku$. Applying of noisy-and relationship models that a student will be very likely to understand $ku$ after he/she understands all immediate prerequisites and understands $ku$ at previous time slice, and if he/she does not know $ku$ at previous time slice, the probability he/she knows $ku$ will drop a lot.

Similar to an ABN, all variables (nodes) in an ADBN have binary values, true or false. When a student understands $ku$ at a previous time slice, he/she might slip in understanding $ku$ at the current time slice. When a student does not know $ku$ at previous time slice, he/she might guess the correct meaning of $ku$ at the current time slice. These characteristics in student learning can be applied to find out the conditional tables for links between different time slices in an ADBN.

Figure 4-16 Atomic Dynamic Bayesian Network (ADBN) consists of ABNs at time 0 and 1
Two variables to determine the conditional probability tables in an ADBN are defined as follows:

- \( slip_t \) is the probability that a student forgets \( ku \) when he/she understands it at previous time slice where \( t \) means time: \( P(ku^1=\text{false}|ku^0=\text{true})=slip_t \) or \( P(ku^1=\text{true}|ku^0=\text{true})=1-slip_t \)

- \( guess_t \) is the probability a student guesses \( ku \) correctly when he/she does not know it at previous time slice: \( P(ku^1=\text{true}|ku^0=\text{false})=guess_t \)

Along with the four variables \( slip_e, guess_e, slip_p, \) and \( guess_p \) in an ABN (Wei & Blank, 2006), there are six variables in three pairs that determine all conditional probability tables in an ADBN. A conditional probability table between the nodes of immediate-prerequisite \((ku), ku^0 \) and \( ku^1 \) can be calculated by the definition of \( slip_e, guess_e, slip_p, \) \( guess_p, \) \( slip_t, guess_t, \) and noisy-and as

\[
P(ku^1=\text{true} \mid p_i(ku)_i^1=\text{true}, \ldots, p_i(ku)_j^1=\text{false}, ku^0=\text{true}) = (1-slip_t) \prod_{i \in K} (1-slip_p) \prod_{j \in K} guess_p
\]

\[
P(ku^1=\text{true} \mid p_i(ku)_i^1=\text{true}, \ldots, p_i(ku)_j^1=\text{false}, ku^0=\text{false}) = guess_t \prod_{i \in K} (1-slip_p) \prod_{j \in K} guess_p
\]

where if \( i \in K, j \in \overline{K} \), then \( p_i(ku)_i^1=\text{true}, p_i(ku)_j^1=\text{false} \), \( K \cup \overline{K} \) is a set of all immediate prerequisites of the center concept, and \( K = \{P_i(ku)_i^1=\text{true} \mid i \in [1,n]\} \), a set of immediate prerequisites that the student knows at time slice 1, while \( \overline{K} \) is a set of immediate prerequisites that the student does not know at that time.
4.5.1 A concrete example of an ADBN

To illustrate how to build an ADBN, we provide a concrete example. Suppose a student is designing a class diagram for an ATM problem. A class diagram requires entering a class name first, then no preference between attributes and methods next. Suppose the student added a correct class name and several correct attributes. The knowledge levels which are posterior probabilities for concepts related to these solution steps have been stored in a database. A current solution step would be that the student enters a name for his first method. Suppose the Expert Evaluator (EE) (Moritz, 2007) determines that the student entered a wrong name for the method, for example totalDeposit as a method (instead of an attribute). The Expert Evaluator produces a packet indicating that the student created a method where an expert would create an attribute. From this difference, the student model determines that the center concept of an ABN in question is attribute\_method. It finds all immediate prerequisites of this concept, attribute and method, from the CIN. It also finds the probability for these concepts from the database which records the diagnosed probability values for each solution step. Figure 4-17 shows the structure of the ADBN.

Figure 4-17 A concrete example of an ADBN
The ADBN needs prior probabilities for root nodes and conditional probability tables for nodes that are not root. Prior probabilities for concept Attribute can be found from the database. Because it is the first time the student adds a method, there is no record for the concept Method. Then its prior probability is 0.5. Suppose \( \text{slip}_e = \text{guess}_e = 0.1 \), \( \text{slip}_p = \text{guess}_p = 0.1 \), and \( \text{slip}_t = \text{guess}_t = 0.1 \), the conditional probability table for the concept Attribute_Method and the node \( au \) (action unit) is shown in Table 4-4, Table 4-5, Table 4-6, and Table 4-7.

**Table 4-4 Conditional probability table for concept Attribute_Method**

| attribute\(^p\) | method\(^c\) | \( P(\text{attribute}\_\text{method}\_p | \text{attribute}\_c, \text{method}\_c) \) |
|------------------|-------------|--------------------------------------------------|
| True             | True        | 0.81                                             |
| True             | False       | 0.09                                             |
| False            | True        | 0.09                                             |
| False            | False       | 0.01                                             |

**Table 4-5 Conditional probability table for concept attribute\(^c\) or method\(^c\)**

| attribute/method\(^c\) | \( P(\text{attribute}\_c | \text{attribute}\_p) / P(\text{method}\_c | \text{method}\_p) \) |
|-------------------------|------------------------------------------------------------------------|
| True                    | 0.9                                                                     |
| False                   | 0.1                                                                     |

**Table 4-6 Conditional probability table for node \( au \)**

| attribute_method\(^p\_p\) | \( P(au\_p | \text{attribute}\_\text{method}\_p) \) |
|---------------------------|--------------------------------------------|
| True                      | 0.9                                        |
| False                     | 0.1                                        |
Table 4-7  Conditional probability table for concept Attribute_Method

| attribute | method | attribute_method | P(attribute_method|attribute, method, attribute_method) |
|-----------|--------|------------------|-------------------------------------|
| True      | True   | True             | 0.729                               |
| True      | True   | False            | 0.081                               |
| True      | False  | True             | 0.081                               |
| True      | False  | False            | 0.009                               |
| False     | True   | True             | 0.081                               |
| False     | True   | False            | 0.009                               |
| False     | False  | True             | 0.009                               |
| False     | False  | False            | 0.001                               |

4.5.2 Updating an ADBN

Figure 4-18 A simple example Bayesian network considering time effect

Similar to section 4.4.4, when we consider the time effects, there are two ways to calculate the students’ understanding level to each concept according to the problems that students answered. The first way is to calculate the understanding level of a concept based on students’ solution to all problems that relate to the concept. It needs a calculation at one point in time. The second way is to calculate the understanding level of a concept after the students finish each problem that relates to the concept. This requires calculation at multiple points in time. The final results in the
two ways are equivalent to each other. Let’s use a simple example to show the reasons why the final results from the two ways are same.

Figure 4-18 shows a Bayesian network, where node $S_1$ represents a solution to a problem which relates to a concept $K$ at time spot 1 (represented by node $K_1$). Similarly, node $S_2$ represents a solution at time spot 2 which relates to node $K_2$, representing the student’s understanding level of concept $K$ at that time. We can have an updated understanding level for concept $K$ after the student did the two problems, which is $P(K_2|S_1, S_2)$. We have two ways to get the value $P(K_2|S_1, S_2)$. The first way is that we calculate $P(K_2|S_1, S_2)$ directly. The second way is that we calculate $P(K_1|S_1)$ first, and then use $P(K_1|S_1)$ as $P(K_1)$ for node $K_1$ to calculate $P(K_2|S_2)$ as in Figure 4-6.

Figure 4-19 Update the Bayesian network in two steps considering time effect

First, we calculate $P(K_2|S_1, S_2)$ using the first way. In Figure 4-19, from the process of inference in Bayesian networks (Russell & Norvig 1995), the conditional probability of $P(K_2|S_1, S_2)$ is:

$$P(K_2 \mid S_1, S_2) = \frac{P(K_2, S_1, S_2)}{P(S_1, S_2)} = \frac{\sum_{K_1} P(K_1, K_2, S_1, S_2)}{P(S_1, S_2)}$$

$$= \frac{1}{P(S_1, S_2)} \sum_{K_1} P(S_1 \mid K_1)P(K_2 \mid K_1)P(S_2 \mid K_2)P(K_1)$$
\[ P(S_2 \mid K_2) = \frac{P(S_2 \mid S_1, S_2) \sum_{K_1} P(S_1 \mid K_1) P(K_2 \mid K_1) P(K_1)}{P(S_1, S_2)} \]

From the chain rule (Nilsson, 1998) we get \( P(S_1, S_2) = P(S_2 \mid S_1) \cdot P(S_1) \). So we can rewrite the above result as,

\[ \frac{P(S_2 \mid K_2)}{P(S_2 \mid S_1) \cdot P(S_1)} = \sum_{K_1} P(S_1 \mid K_1) P(K_2 \mid K_1) P(K_1) \]

Next, we calculate \( P(K_2 \mid S_1, S_2) \) using the second way. In Figure 4-6, since we sample and update in steps, we get every observance at the second step given the first step has happened. Thus we can use \( P(K_1 \mid S_1) \) as \( P(K_1) \) for node \( K_1 \) and \( P(S_2 \mid S_1) \) as \( P(S_2) \) for node \( S_2 \): \( P(K_1) \) is set to \( P(K_1 \mid S_1) \), \( P(S_2) \) is set to \( P(S_2 \mid S_1) \). From the Bayes’ rule (Russell & Norvig 1995), we can calculate \( P(K_1 \mid S_1) \) as,

\[ P(K_1 \mid S_1) = \frac{P(S_1 \mid K_1) \cdot P(K_1)}{P(S_1)} \]

Thus, \( P(K_1) \) is set to \( \frac{P(S_1 \mid K_1) \cdot P(K_1)}{P(S_1)} \).

From the process of inference in Bayesian networks (Russell & Norvig 1995), the conditional probability of \( P(K_2 \mid S_2) \) is:

\[ P(K_2 \mid S_2) = \frac{P(K_2, S_2)}{P(S_2)} = \frac{\sum_{K_1} P(K_1, K_2, S_2)}{P(S_2)} = \frac{1}{P(S_2)} \sum_{K_1} P(K_2 \mid K_1) P(S_2 \mid K_2) P(K_1) \]

\[ = \frac{P(S_2 \mid K_2)}{P(S_2)} \sum_{K_1} P(K_2 \mid K_1) P(K_1) \]
Replace \( P(K_1) \) with \( \frac{P(S_1 \mid K_1) \cdot P(K_1)}{P(S_1)} \), we get:

\[
P(K_2 \mid S_2) = \frac{P(S_2 \mid K_2)}{P(S_2)} \sum_{K_1} P(K_2 \mid K_1) P(S_1 \mid K_1) \frac{P(S_1 \mid K_1) \cdot P(K_1)}{P(S_1)}
\]

\[
= \frac{P(S_2 \mid K_2)}{P(S_2) \cdot P(S_1 \mid K_1)} \sum_{K_1} P(K_2 \mid K_1) P(S_1 \mid K_1) P(K_1)
\]

Therefore, the two ways, calculating \( P(K_2 \mid S_1, S_2) \) in one time or calculating \( P(K_1 \mid S_1) \) first, and then using \( P(K_1 \mid S_1) \) as \( P(K_1) \) for node \( K_1 \) to calculate \( P(K_2 \mid S_2) \) generate the same results. This dissertation applies the second way to update ABNs, e.g., we update an ABN each time a student answers a problem.

Figure 4-20 shows how to update the student model with ADBNs at time slice 0 and 1 which includes three steps: 1) update the ABN at time slice 0, i.e., to update the center concept and parent nodes according to evidence at time slice 0 based on equations (4-21), (4-22), (4-23), and (4-24); 2) calculate the probability for a parent node \( X \) at time slice 1 before considering any new evidence based on equation (4-25); and 3) update the ABN at time slice 1, i.e., to update the center concept and parent nodes according to evidence at time slice 1 based on equations (4-26), (4-27), (4-28), and (4-29). Step 2 simulates that a student possibly understands an immediate prerequisite concept when he/she understands the concept at previous time slice. Step 3 models that a student is very likely to understand a center concept when he/she understands all of the immediate prerequisites of the center concept and also knows the center concept at previous time slice.
\[
P(X^i) = (1 - \text{slip}_i) \cdot P(X^0) + \text{guess}_i \cdot P(\neg X^0) \quad (4-25)
\]

\[
P(D^i) = ((1 - \text{slip}) + \text{guess}) \prod_{\text{parents}(D^i) \neq D^0}
\]

\[
P(D^i \mid S^i) = \frac{1 + \text{guess}_e \cdot P(-D^i)}{1 + (1 - \text{slip}_e) \cdot P(D^i)} \quad (4-27)
\]

\[
P(D^i \mid X^i) = (1 - \text{slip})(1 - \text{slip}) + \text{guess} \prod_{\text{parents}(D^i) \neq X^i \neq D^0} \quad (4-28)
\]

\[
P(X^i \mid S^i) = \frac{P(X^i) \cdot P(D^i \mid X^i)}{P(D^i)} \cdot \frac{1 + \text{guess}_e \cdot P(-D^i \mid X^i)}{1 + (1 - \text{slip}_e) \cdot P(D^i)} \quad (4-29)
\]

Since an ADBN consists of two ABNs, its algorithm is an extension of the ABN updating algorithm (Section 4.4.4). The first ABN at time slice 0 is an ABN with \( n \) immediate parent nodes, and the second ABN at time slice 1 is an ABN with \( n+1 \) immediate parent nodes. Both of them can use the formulas of updating an ABN. Updating an ADBN still uses \( O(1) \) running time because each concept has at most \( k \) immediate parents, and there are \( n \) concepts in the domain. Therefore, an ADBN needs \( O(1) \) running time, updates \( O(1) \) nodes, and determines three parameters \( \text{slip}_e/\text{guess}_e, \text{slip}_p/\text{guess}_p, \text{slip}_t/\text{guess}_t \) for each solution step.

Using an ADBN reduces the running time for each step from exponential for a complete dynamic Bayesian network to constant because the ADBN stores the updated value for the next solution step, and according to first-order Markov process the current state only depends on the immediate previous state.
4.5.3 Pseudocode for updating an ADBN

Below is the pseudocode for the functions that update an ADBN.

function ADBNUpdate (student, targetConcept, CIN, allProfile) returns allProfile

FindAllPrerequisites(CIN, targetConcept)

find <student, theProfile> for the student from allProfile

FindProbabilities(targetConcept, prerequisites[num], theProfile)

if targetConcept has no previous record then

    if targetConcept has no prerequisites then

        update targetConcept according to equation (4-22)
insert <student, concept, probability> for targetConcept in theProfile

else if current targetConcept has prerequisites then

calculate value according to equation (4-21)

update targetConcept according to equation (4-22)

put probability of targetConcept in theProfile

UpdateParentNodes (prerequisites[num], theProfile)

else if targetConcept has previous record then

if targetConcept has no prerequisites then

update targetConcept according to equation (4-27)

put probability of targetConcept in theProfile

else if targetConcept has prerequisites then

calculate values according to equation (4-25) and (4-26)

update center node according to equation (4-27)

insert < targetConcept, probability> in theProfile

UpdateParentNodes (prerequisites[num], theProfile)

insert <student, theProfile> into allStudentProfile

return allStudentProfile

function FindAllPrerequisites (CIN, targetConcept) returns prerequisites[num]

find < targetConcept, prerequisites> of targetConcept from CIN

num←number of prerequisites

prerequisites[num] ←all prerequisites of the targetConcept

return prerequisites[num]
function FindProbabilities (targetConcept, prerequisites[num], theProfile) returns probabilities[num+1]

if theProfile is not null then

if find < targetConcept, probability> in theProfile

then set value to old probability of targetConcept

else

then set 0.5 to probability of targetConcept

for each of the num prerequisites for targetConcept do

if find <prerequisite, probability> in theProfile

then set the value to old probability of the prerequisite

else

then set 0.5 to current probability of the prerequisite

return probabilities[num+1]

function UpdateParentNodes (prerequisites[num], theProfile) returns theProfile

update parent nodes according to equation (4-28) and (4-29)

studentProfile←probability of parent nodes

insert <prerequisite, probability> for each parent nodes in theProfile

return theProfile

4.6 Integrating diagnoses of close-ended and open-ended exercises

Intelligent tutoring systems often use Bayesian networks to model how well students understand the rules or concepts in a domain of study. Such student models
have predicted student understanding from close-ended problems (such as multiple choice questions) or from open-ended exercises (such as designing a class diagram), but not integrated them. This dissertation investigates how to integrate a student model from both open- and close-ended work to diagnose how well students have learned object-oriented concepts. Close-ended exercises include multiple-choice questions and drag-and-drop exercises. Open-ended exercises involve designing a novice object-oriented system, for which students select a response from a limited range. The design of a UML class diagram is an open-ended exercise, for which no two students are expected to produce the same solution.

Since both the closed- and open-ended exercises cover the same domain, object-oriented concepts, these two kinds of exercises diagnose knowledge of the same set of concepts. Hence, the student model diagnosing student responses to closed- or opened-ended exercises can be stored in the same database for the pedagogical advisor and the student model’s historical layer. Students can switch back and forth between the multimedia and the Eclipse IDE. While they switch, the same student has one student profile which records his/her understanding level to the object-oriented concepts.

There are several advantages of integrating a student’s profile from both closed- and open-ended exercises: 1) The student model has more opportunity to gather information from students’ performance and hence generate more accurate students’ learning state; 2) It enhances the student model’s accuracy for open-ended exercises since it provides initial data about students’ understanding level if students
do close-ended exercise first; and 3) it enhances the student model’s accuracy for close-ended exercises if students do open-ended exercise first.

4.6.1 Diagnosing open-ended exercises

Figure 4-21 Environment where students do open-ended object-oriented design

Students do object-oriented design with a plug-in called LehighUML in Eclipse IDE. Figure 4-21 shows that a student is entering the design solution for a movie ticket machine. The LehighUML plug-in records all solution steps students enter and stores them in a database. The Expert Evaluator (Moritz, 2007) analyzes the records and decides whether the student’s solution step is correct. The Expert Evaluator generates a record named packet for each student solution step and stores it
into the database. The student model analyzes each packet and diagnoses the student’s understanding level using ADBNs.

<table>
<thead>
<tr>
<th>packetID</th>
<th>546</th>
</tr>
</thead>
<tbody>
<tr>
<td>packetNumber</td>
<td>1</td>
</tr>
<tr>
<td>studentID</td>
<td>xxxx</td>
</tr>
<tr>
<td>problemID</td>
<td>1</td>
</tr>
<tr>
<td>packetType</td>
<td>error</td>
</tr>
<tr>
<td>correctAction</td>
<td>add_attribute</td>
</tr>
<tr>
<td>correctProblemAction</td>
<td>price</td>
</tr>
<tr>
<td>correctDatatype</td>
<td>double</td>
</tr>
<tr>
<td>correctConcept</td>
<td>attribute</td>
</tr>
<tr>
<td>errorCode</td>
<td>UND_PARAMETER</td>
</tr>
<tr>
<td>studentAction</td>
<td>add_parameter</td>
</tr>
<tr>
<td>studentProblemAction</td>
<td>costOfTicket</td>
</tr>
<tr>
<td>studentDatatype</td>
<td>double</td>
</tr>
<tr>
<td>studentConcept</td>
<td>parameter</td>
</tr>
</tbody>
</table>

**Figure 4-22 A packet generated by the Expert Evaluator**

The student model diagnoses students’ learning state according to the information provided by the Expert Evaluator. Figure 4-22 shows a packet that was generated by the Expert Evaluator when a student made an error in designing parameters. The student entered “costOfTicket” as a parameter of the method “displayPriceOfTicket.” The Expert Evaluator identified the student entered a wrong parameter since it believes that “costOfTicket” should be an attribute of the class “MovieTicketMachine.” Therefore, the errorCode issued was “UND_PARAMETER,” which meant that this parameter should be some other component, such as an attribute. The packet also contains information about the data type that the student entered for this component. In this example, the student made an error in applying concept attribute-parameter, but correctly applied concepts datatype and double. Based on this information, the student model generates three ADBNs for three concepts,
double, datatype, and attribute parameter. These three ADBNs diagnose the student’s understanding level to these three concepts and their immediate prerequisite concepts. According to the information generated by the Expert Evaluator (Sally, 2007), the student model needs to determine how to build ADBNs. Sometimes even though the Expert Evaluator issued an error packet, the student actually designed a correct component. For example, the Expert Evaluator may diagnose an error called “NO_JAVADOC” which means a student did not add any comment though the student’s solution step may otherwise be correct. Therefore, the student model needs to consider that the student applies the related concepts correctly.

In addition, there can be multiple ADBNs set up based on one packet when a packet actually involves multiple concepts. For example, if a student enters a correct method and correct return type “void” for the method, there will be a single packet generated for the two actions. However, the packet involves three concepts which are void, returntype, and method. If a student corrects his/her previous error by updating an existing component, it implies that the student gains understanding not only on the concepts related to the current solution step but also on the concepts related to the previous error. Therefore, the student model needs to set up ADBNs for each relevant concept.

4.6.2 Pseudocode for algorithm of setting up ADBNs

Below is the pseudocode for the functions that setting ADBNs from input of the Expert Evaluator (Moritz, 2007) about open-ended exercises.

function CheckPacket(record, targetConcept, CIN, allProfile) returns allProfile
if errorCode in the record contains NO then packetType ← info

if packetType in the record contains is error then

    if errorCode in the record contains contains UNK
        then return unkPacket(student, targetConcept, CIN, allProfile)
    if errorCode in the record contains contains UND
        then return undPacket(student, targetConcept, CIN, allProfile)
    if errorCode contains INC
        then return incPacket(student, targetConcept, CIN, allProfile)

if packetType is info then

    then return infoPacket((targetConcept, CIN, allProfile))

function unkPacket(student, targetConcept, CIN, allProfile) returns allProfile

    targetConcept ← CorrectConcept

    return ADBNUpdate (student, targetConcept, CIN, allProfile)

function undPacket(student, targetConcept, CIN, allProfile) returns allProfile

    targetConcept ← CorrectConcept+StudentConcept

    return ADBNUpdate (student, targetConcept, CIN, allProfile)

if StudentDatatype is wrong

    then targetConcept ← StudentDatatype

    return ADBNUpdate (targetConcept, CIN, allProfile)

function incPacket(student, targetConcept, CIN, allProfile) returns allProfile

    if errorCode contains CLASS then

        targetConcept←StudentConcept+class

        return ADBNUpdate (student, targetConcept, CIN, allProfile)
if StudentDatatype for an attribute is wrong
   then return incDatatype(student, targetConcept, CIN, allProfile)
else if StudentDatatype for a method is wrong
   then return incReturntype(student, targetConcept, CIN, allProfile)
else if errorCode contains method then
   targetConcept←method
   return ADBNUpdate (student, targetConcept, CIN, allProfile)
if StudentDatatype for a method is wrong
   then return incReturntype(student, targetConcept, CIN, allProfile)
else if errorCode contains attribute then
   targetConcept←attribute
   return ADBNUpdate (student, targetConcept, CIN, allProfile)
if StudentDatatype for an attribute is wrong
   then return incDatatype(student, targetConcept, CIN, allProfile)

function incDatatype (student, targetConcept, CIN, allProfile) returns 0
   targetConcept←StudentDatatype+CorrectDatatype
   return ADBNUpdate (student, targetConcept, CIN, allProfile)

function incReturntype (student, targetConcept, CIN, allProfile) returns allProfile
   targetConcept←StudentReturntype+CorrectReturntype
   return ADBNUpdate (student, targetConcept, CIN, allProfile)

function infoPacket (student, targetConcept, CIN, allProfile) returns allProfile
   targetConcept←correctConcept
   return ADBNUpdate (student, targetConcept, CIN, allProfile)
if correctAction contains attribute or parameter then

targetConcept←correctDatatype

return ADBNUpdate (student, targetConcept, CIN, allProfile)

if correctAction contains method then

targetConcept←correctReturntype

return ADBNUpdate (student, targetConcept, CIN, allProfile)

4.6.3 Diagnosing close-ended exercises

Students do close-ended exercises such as multiple choice exercises and drag and drop problems in the multimedia. Figure 4-23 shows an example of a multiple choice question. The multimedia analyzes students’ performance and the records the results in a database through a PERL program on the multimedia server. Since the exercises are close-ended it is straightforward for the multimedia to know whether a student makes an error. The student model analyzes the records in the database and diagnoses the student’s understanding level using ADBNs.
Figure 4-23 Environment where students do close-ended exercises

Figure 4-24 shows a record generated by the multimedia. It specifies that: 1) which problem the student just finished, which is represented by the unique combination of three fields, “currChaplabel”, “currScreenlabel”, and “questionNumber”; 2) what answer he/she chose, which is represented by field “qusetionLevel”; and 3) whether it is a correct answer or not, which is represented by field “correct”. The target concepts that relate to the student’s solution are stored in a pre-made table and can be referred to by the student model. The student model sets up an ADBN every time the student finishes a solution.
4.7 Examples of understandable feedback

The student model helps the ITS to provide understandable feedback to individual student since it diagnoses his/her knowledge state and hence learning needs from the student’s learning history. This section illustrates the role of the student model with several examples of possible interactions with students.

Suppose student A defined `movieTitle` as a parameter for a method `displayMovieTitle` after she has already defined `movieTitle` as an attribute to class `TicketMachine`. The Expert Evaluator (Sally, 2007) diagnoses that `movieTitle` should not be a parameter. Then the student model determines that 1) student A has good understanding of the concepts `class`, `attribute`, `methods`, and `parameter` but not so good understanding of the relationship concept `attribute_parameter` (Section 3.2.2 & Section 4.3); and 2) the tutoring needs for student A are i) explanation of error details, and ii) explanation of concept `attribute_parameter`.

With this input from the student model, the pedagogical advisor could provide something like the following feedback: “Since you have added `movieTitle` as an attribute to the class `TicketMachine`, you cannot also make it a parameter to method `displayMovieTitle`. To decide whether `movieTitle` should be an attribute or a
parameter, remember: an attribute can be accessed in any methods in this class, while a parameter can only be accessed in its method.”

The first clause in this feedback: “Since you have added movieTitle as an attribute to the class TicketMachine” explains the error details by repeating what the student did, and is from the diagnostic results of the PDM layer which compares the student’s current solution with his/her solution history; “you cannot make it a parameter to the method displayMovieTitle” tells the student what the error is, and is from the Expert Evaluator’s assessment; and “since an attribute can be accessed in any methods in this class, while a parameter can only be accessed in its method” explains the relationship concept attribute_parameter, and is from the diagnoses of the PDM and HM layers.

The student model found that the reason that student A committed the error is because she does not know the relationship concept attribute_parameter. Without understanding this concept, the similar error may recur in this student’s work.

Suppose student B did the same error as student A did, but he has different knowledge state. Based on its observations of work done in LehighUML and the multimedia, the Student Model diagnoses that Student B does not understand concepts attribute and parameter (while student A has more understanding of these concepts). If student B were to get the same feedback as student A does, which explains attribute_parameter, student B would not understand it because he does not understand attribute_parameter’s prerequisites, which are attribute and parameter. Hence, feedback to student B should include the explanation of these prerequisites.
The student model diagnosed that the reason that student B committed the error is because he does not know the relationship concept **attribute-parameter** and its prerequisites **attribute** and **parameter**. Without understanding these concepts, student B may repeat the similar error. Therefore, the understandable feedback is based on the accurately diagnosed knowledge state of a student.

Technically, Java permits attributes (or instance variables) and parameters to have the same name in the same scope. However, DesignFirst-ITS is not actually helping students with Java programming but with creating simple class diagrams from problem descriptions. During class diagram design, we have found that the distinction between attribute and parameter is a common source of confusion for novices. When designing a class, one needs to understand whether an item in the problem description is an attribute or just a parameter. An attribute typically stores state information in an object, distinguishing it from other objects of that class. Parameters pass more transient information into methods. Novices need to understand this distinction to design attributes and methods correctly.

The pedagogical advisor (Parvez, 2007) can use various contents in the feedback, such as images or examples to tutor the concepts **attribute-parameter**, **attribute**, and **parameter**, and hence to rebuild student B’s knowledge structure. It can present the feedback in various formats to suit the student’s favor as well. However, the questions about how to present the feedback (Parvez, 2007) are beyond the scope of this dissertation.
5 Evaluation of the student model

Experimenting with human subjects is expensive and unexpected factors may make results deviate from what is hypothesized if the student model has not been tested thoroughly. Large amounts of data are needed to prove that the student model really works. Using simulated students avoids problems with limited human subject resources.

Simulated students can be generated in large numbers to evaluate the model thoroughly, with the only cost being computational running time; simulated students are an effective way to test the effectiveness of the algorithm before incurring the cost of human subjects; and simulated students eliminate other variables, such as the accuracy of other components of a tutoring system (user interface, expert module to the pedagogical advice), providing results that focus exclusively on the performance of the student model. However, the simulated students are just instantiations of the student model. How well the actual student behaviors are simulated by the student model needs to be evaluated with human subjects.

Some researchers use simulated students to evaluate their student models (Carmona et al., 2005; Collins, Greer, & Huang, 1996; Millán & Pérez-de-la-Cruz, 2002; VanLehn et al., 1998). Some researchers use human subjects instead (Koedinger & Anderson, 1997; Corbett, McLaughlin, & Scarpinatto, 2000). This dissertation presents six empirical studies which evaluate the student model thoroughly with simulation students, and with 71 human subjects (49 college freshmen and 22 high school students), all novices with object-oriented design.
In the six empirical studies, we first evaluate the student model with ABNs with simulated students to justify that ABNs provide reliable diagnoses. Then we evaluate the student model with ADBNs with simulated students to investigate whether the student model with ADBNs performs better than the student model with ABNs.

Next, we evaluated the student model with human subjects. We evaluated the student model with close-ended exercises in CIMEL multimedia (Blank, Barnes, & Kay, 2005) and open-ended problems in DesignFirst-ITS with the 71 human subjects. Then we evaluated the student model for DesignFirst-ITS which integrates the diagnoses results from the student model for the multimedia.

In addition, we compared the accuracy of non-advanced-numerical student models with the student models using ADBNs.

5.1 Evaluating ABNs with simulated students

One problem with simulated students is how confident we can be that they represent real students. Millán and Pérez-de-la-Cruz (2002) found that adding more pre-knowledge to simulated students—i.e., assuming that some simulated students already know some prerequisite knowledge before running a simulation—improves evaluation results. They divided the concepts into categories with different difficulty levels, and divided simulated students into novice, intermediate, good and expert groups, each of which knows a corresponding category of concepts. They arrived at their categories by analyzing real students.
The probability that a real student knows a concept will be very small when the student does not have any prerequisite knowledge. Following this intuitive rule from observing real students (Carmona et al., 2005) a simulated student will not know a concept without having any prerequisite knowledge.

An experiment evaluating the problem-domain layer of our student model began by generating 240 students in six categories, each consists of 40 students. Students in the six categories understand 5, 10, 15, 20, 25 and 30 concepts out of 38 concepts, respectively, in the Curriculum Information Network (Figure 4-2). We pick the concepts that students know randomly and follow the rule that a concept is chosen only when all of its prerequisites are chosen. Therefore, we choose the roots in the Curriculum Information Network first. If a simulated student understands a concept, k|(student, concept)=1 (his/her knowledge level for this concept is 1), otherwise k|(student, concept)=0.

Below is the pseudocode for the functions that generate known concepts in the knowledge profiles for simulated students.

function GenerateSimStudnets(CIN) returns allPreDefined

for each of the six categories that has a certain number of known concepts do

for each student in 40 students do

possibleStack ← GetRoot(CIN)

While (number of concepts in realStack<knownConcepts) do

choice ← random index of possible next concept in possibleStack

if the chosen concept is not in realStack then
add the chosen concept into realStack

possibleStack ← findAllPossibleNext (possibleStack)

add all concepts in realStack into allPreDefined for
the student

clean possibleStack

clean realStack

return allPreDefined

function GetRoot(CIN) returns possibleStack

roots ← concepts that have no parent in CIN

put all roots in possibleStack

return possibleStack

function findAllPossibleNext() returns possibleStack

for each concept in possibleStack do

find all children of the concept

addChildrenToPossibleStack(concept)

cleanPossibleStack()

return possibleStack

function addChildrenToPossibleStack (currconcept) returns possibleStack

for each child of all children of currconcept do

find all parents of the child

if any of the parents in realStack then

add the child into possibleStack
return possibleStack

function cleanPossibleStack() returns possibleStack

for each concept in possibleStack do
    if all children of the concept has been added into possibleStack then
        delete the concept from the possibleStack

return possibleStack

5.1.1 Evaluation

In the simulation study, we let each of the 240 students do 38 open-ended problems, each of which focuses on one concept in the Curriculum Information Network. Once a student finishes a problem, we use the evidence to update the student learning state. After the 240 students finish the 38 problems, we get an updated student learning state for each student. Then we compare the updated student learning state with the predefined student profile.

Figure 5-1 illustrates the procedure of the simulation. For clarity, the experiment distinguishes between pre-determined knowledge levels of the simulated students, on the left, and the updated student knowledge levels, on the right, developed as a result of each solution step. Following the arrows, the simulation starts from a randomly picked problem and the pre-defined knowledge state of a simulated student, which represents that a simulated student who has certain knowledge answers a randomly picked problem. Using the pre-defined knowledge state as prior probabilities, P(problem)—the probability of answering the problem correctly can be calculate by Equation (5-1).
\[ P(\text{problem}) = \text{slip}_d \ast P(\text{concept}) + \text{guess}_d \ast P(\neg\text{concept}) \]  \hspace{1cm} (5-1)

where \( \text{slip}_d \) and \( \text{guess}_d \) have the same meaning of \( \text{slip}_e \) and \( \text{guess}_e \) (Section 4.4.1).

If \( P(\text{problem}) \) is greater than or equal to a random number \( n \) from 0 to 1 (generated by Java’s pseudo-random-number generator with a uniform distribution), the model determines that the simulated student made a correct step, and if \( P(\text{problem}) \) is less than the random number \( n \), the model infers an error. Hence the experiment gathers evidence for updating the simulated student’s estimated knowledge level. The initial value of an estimated knowledge level for any concept is 0.5 (VanLehn et al., 1998; Millán & Pérez-de-la-Cruz, 2002; Carmona et al., 2005). The posterior probability values for concepts in the ABN are stored in a database and are used as prior probabilities for updating another ABN for the next solution step. The experiment repeats this procedure for 38 randomly picked solution steps, for each simulated student.

\[ n \leq P(\text{problem}) : \text{correct} \]
\[ n > P(\text{problem}) : \text{wrong} \]

\( n \): a random number between 0 and 1

**Figure 5-1 Illustration of evaluation procedure in the simulation study**
There are three possible diagnoses for a knowledge unit (\(ku\)) in the evaluation, known, unknown, and undiagnosed. A posterior probability from 0.7 to 1 means \(k|(\text{student,} \ ku)=1\) (\(ku\) is known by the student), a posterior probability from 0.3 to 0.7 means undiagnosed, and a probability from 0 to 0.3 means \(k|(\text{student,} \ ku)=0\) (\(ku\) is unknown by the student) (following Millán & Pérez-de-la-Cruz 2002). The student model diagnoses a \(ku\) correctly when the updated \(k|(\text{student,} \ ku)=1\) and the corresponding simulated \(k|(\text{student,} \ ku)=1\), or when the \(ku\) is diagnosed as \(k|(\text{student,} \ ku)=0\) and the corresponding simulated \(k|(\text{student,} \ ku)=0\), which is pre-defined in the simulated student’s knowledge profile.

To reduce the number of parameters in this student model, we choose

\[
\begin{align*}
slip_{do} &= \text{guess}_{do}, \quad slip_{e} = \text{guess}_{e}, \quad \text{and} \quad slip_{p} = \text{guess}_{p},
\end{align*}
\]

where \(slip_{do}\) and \(guess_{do}\) are used in calculating the probability of answering a problem correctly, \(slip_{e}\) and \(guess_{e}\) are used in updating the understanding level of the target concept, and \(slip_{p}\) and \(guess_{p}\) are used in considering the prerequisites (Section 4.4.1). Each parameter can assume one of the following four values: 0.001, 0.01, 0.1, and 0.2. The experiment evaluated 64 combinations of the above parameters (4x4x4) to investigate their influence on the accuracy of the student model. The accuracy of the student model is represented by the Correct Diagnostic Rate (CDR), the percentage of concepts diagnosed correctly for each simulated student, which is defined as

\[
CDR = \frac{\text{correctConceptsCorrectlyDiagnosed} + \text{wrongConceptsCorrectlyDiagnosed}}{\text{totalDomainConcepts}}
\]

(5-2)

Below is the pseudocode for the functions that do the simulation experiment using ABNs.

133
function simulation(CIN, allPreDefined\(^1\)) returns CDR[240x64]

for each simulate student in 240 simulate students do

for each concept in 38 concepts do

    evidence ← generateEvidence(concept)

for each of 64 triples of slip_e, slip_p, and slip_do do

    simProfile ← ABNUpdate\(^2\) (evidence, CIN, simProfile)

CDR[simulate student] ← compare simProfile with allPreDefined

return CDR[240x64]

function generateEvidence(concept) returns evidence

if the student knows the current concept then

    probability for knowing the concept ← 1.0

else

    probability for knowing the concept ← 0.0

probability of answering the problem ← abnexpendoproblem()

temp ← a random number

if temp <= probability of answering the problem then

    evidence ← 1

else evidence ← 0

return evidence

---

\(^1\) allPreDefined is generated in the function GenerateSimStudnets in Section 5.1

\(^2\) Please see the function ABNUpdate in Section 4.4.5.
function abnexpendoprob() returns probability

calculate the probability of answering problem according to Equation (5-1)

return probability

5.1.2 Results

In the ANOVA analysis, the correct diagnostic rate of each simulated student depends on four fixed-effect independent factors that are 4 (slip_do) x 4 (slip_e) x 4 (slip_p) x 6 (concept) and one random factor that is 240 (student) nested (40 each) in the “concept”. Slip_do (slip_{do}) is the slip factor in predicting students’ performance from predefined knowledge. Slip_e (slip_{e}) and Slip_p (slip_{p}) are the slip factors in diagnosing students’ knowledge state from students’ performance. Slip_e belongs to the link between evidence and target concept. Slip_p belongs to the link between prerequisite concepts and the target concept. Concept is the number of concepts that each simulated student knows.

Table 5-1 reports the results of the ANOVA on correct diagnostic rates that depend on the factors. The interaction of slip_e x slip_p (F(9, 6318)=118.5, P<0.001) is significant. A visual inspection of Figure 5-2 indicates that the correct diagnostic rates are higher when slip_e and slip_p take the same value. Figure 5-2 also shows that when slip_e and slip_p take the same value, the average correct diagnostic rate is at least 92%. The high correct diagnostic rates indicate that the student model can correctly diagnose student’s knowledge states in the simulate study. The correct diagnostic rate increases when both slip_e and slip_p decreases. This explains that higher diagnostic rate happens when slip_e and slip_p take equal and small values.
### Table 5-1 ANOVA Tests of Between-Subjects Effects

Dependent Variable: correct diagnostic rate

<table>
<thead>
<tr>
<th>Source</th>
<th>SS</th>
<th>DoF</th>
<th>MS</th>
<th>F Ratio</th>
<th>P Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>slip_do (do)</td>
<td>64.453</td>
<td>3</td>
<td>21.484</td>
<td>10672.467</td>
<td>.000</td>
</tr>
<tr>
<td>slip_e (e)</td>
<td>1.386</td>
<td>3</td>
<td>.462</td>
<td>229.541</td>
<td>.000</td>
</tr>
<tr>
<td>slip_p (p)</td>
<td>2.660</td>
<td>3</td>
<td>.887</td>
<td>440.520</td>
<td>.000</td>
</tr>
<tr>
<td>concept</td>
<td>3.326</td>
<td>5</td>
<td>.665</td>
<td>15.333</td>
<td>.000</td>
</tr>
<tr>
<td>sid (concept)</td>
<td>10.153</td>
<td>234</td>
<td>.043</td>
<td>21.554</td>
<td>.000</td>
</tr>
<tr>
<td>do * e</td>
<td>1.402</td>
<td>9</td>
<td>.156</td>
<td>81.093</td>
<td>.000</td>
</tr>
<tr>
<td>do * p</td>
<td>1.540</td>
<td>9</td>
<td>.171</td>
<td>89.451</td>
<td>.000</td>
</tr>
<tr>
<td>do * concept</td>
<td>.701</td>
<td>15</td>
<td>.047</td>
<td>23.726</td>
<td>.000</td>
</tr>
<tr>
<td>do * sid</td>
<td>12.446</td>
<td>702</td>
<td>.018</td>
<td>13.901</td>
<td>.000</td>
</tr>
<tr>
<td>e * p</td>
<td>2.006</td>
<td>9</td>
<td>.223</td>
<td>118.467</td>
<td>.000</td>
</tr>
<tr>
<td>e * concept</td>
<td>1.106</td>
<td>15</td>
<td>.074</td>
<td>37.984</td>
<td>.000</td>
</tr>
<tr>
<td>e * sid</td>
<td>2.728</td>
<td>702</td>
<td>.004</td>
<td>1.978</td>
<td>.000</td>
</tr>
<tr>
<td>p * concept</td>
<td>1.426</td>
<td>15</td>
<td>.095</td>
<td>49.511</td>
<td>.000</td>
</tr>
<tr>
<td>p * sid</td>
<td>3.263</td>
<td>702</td>
<td>.005</td>
<td>2.412</td>
<td>.000</td>
</tr>
<tr>
<td>do * e * p</td>
<td>.591</td>
<td>27</td>
<td>.022</td>
<td>15.439</td>
<td>.000</td>
</tr>
<tr>
<td>do * e * concept</td>
<td>.072</td>
<td>45</td>
<td>.002</td>
<td>1.128</td>
<td>.257</td>
</tr>
<tr>
<td>do * e * sid</td>
<td>17.932</td>
<td>2106</td>
<td>.003</td>
<td>4.602</td>
<td>.000</td>
</tr>
<tr>
<td>do * p * concept</td>
<td>.039</td>
<td>45</td>
<td>.001</td>
<td>.616</td>
<td>.980</td>
</tr>
<tr>
<td>do * p * sid</td>
<td>18.666</td>
<td>2106</td>
<td>.009</td>
<td>5.089</td>
<td>.000</td>
</tr>
<tr>
<td>e * p * concept</td>
<td>.491</td>
<td>45</td>
<td>.011</td>
<td>7.695</td>
<td>.000</td>
</tr>
<tr>
<td>e * p * sid</td>
<td>10.145</td>
<td>2106</td>
<td>.005</td>
<td>1.603</td>
<td>.000</td>
</tr>
<tr>
<td>do * e * p * concept</td>
<td>.128</td>
<td>135</td>
<td>.001</td>
<td>.667</td>
<td>.999</td>
</tr>
<tr>
<td>Error</td>
<td>12.636</td>
<td>6318</td>
<td>.002</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Figure 5.2 Average correct diagnostic rate vs. parameters slip_e and slip_p

The interaction of slip_do x slip_e (F(9, 6318)= 81.1, p<0.001) is significant and a visual inspection of Figure 5-3 and Figure 5-4 shows that correct diagnostic rate values increase when slip_do decreases. The troubling trend of slip_do indicates that it is unlikely that ship_do would be a high value (0.1 or greater). Figure 5-4 also suggests that slip_do’s influence on the correct diagnostic rate is much larger than that of slip_e’s. Even when slip_do takes the value of 0.2, the average CDR is still around 0.81 regardless of slip_e and slip_p. A diagnostic rate of at least 80% is good for a student model; moreover, as Figure 5.3 shows, the diagnostic rate improves significantly, to over 97% accuracy, when slip_do is set at smaller values, 0.01 or 0.001. Slip_do is hard to determine for the real students. The ANOVA results confirm our intuition and experience with actual students: students rarely slip or guess when
adding elements to a UML diagram, since they are not presented with a list of multiple choice alternatives from which they could guess or slip. Therefore, slip_e and slip_do should be a small value.

![Estimated Marginal Means of cdr](image)

**Figure 5-3** Average correct diagnostic rate vs. parameter slip_do
Figure 5-4 Average correct diagnostic rate vs. parameters slip_do and slip_e

5.2 Evaluating ADBNs with simulated students

Following the procedure described in Figure 5-1, we used 60 simulated students to evaluate the student model with ADBNs. We had 10 simulated students instead of having 40 students in each of the six groups. Thus, the 60 simulated students are a subset of the 240 simulated students that were generated in the study of ABN. In the simulation experiment for ADBNs, we let each student do the 38 open-ended problems. To reduce the number of parameters in the student model, we used the same value for slip_do and guess_do, slip_p and guess_p, slip_e and guess_e, and slip_t and guess_t, where the parameters slip_t and guess_t are used in considering the understanding
level of a concept in previous time spot (Section 4.5). Each parameter can assume one of the following four values: 0.001, 0.01, 0.1, and 0.2. The experiment evaluated 256 combinations of the above parameters (4x4x4x4). To compare the two types of student models, ABN and ADBN in this experiment, we included a sub-set of data from the ABN experiment to be the data when \textit{slip}, is 0. The sub-set of data belongs to the same simulated students that are chosen in this experiment.

### 5.2.1 Pseudocode

Below is the pseudocode for the functions that do the simulation experiment using ADBNs.

```plaintext
function simulation(CIN, allPreDefined\textsuperscript{3}) returns CDR[240x256]

  for each simulate student in 60 simulate students do

    for each concept in 38 concepts do

      evidence ← generateEvidence(concept)

      for 256 quadruples of slip_e, slip_p, slip_t, and slip_do do

        simProfile ← ADBNUpdate\textsuperscript{4} (evidence, CIN, simProfile)

        CDR[simulate student] ← compare simProfile with allPreDefined

    return CDR[240x64]

function generateEvidence(concept) returns evidence

  if the student knows the current concept then
```

\textsuperscript{3} allPreDefined is generated in the function GenerateSimStudents in Section 5.1.

\textsuperscript{4} Please see the function ABNUpdate in Section 4.5.3.
probability for knowing the concept ← 1.0
else
    probability for knowing the concept ← 0.0
probability of answering the problem ← abnexpendoproblem()
temp ← a random number
if temp<= probability of answering the problem then
    evidence ← 1
else
    evidence ← 0
return evidence

function abnexpendoproblem() returns probability
calculate the probability of answering problem according to Equation (5-1)
return probability

5.2.2 Result
In the ANOVA analysis, the correct diagnostic rate of each simulated student depends on four fixed-effect independent factors that are 4 (slip_do) x 4 (slip_e) x 4 (slip_p) x 5 (slip_t) x 6 (concept) and one random factor that is 60 (student) nested (10 each) in the “concept.” Slip_do (slip_{do}) is the slip factor in predicting students’ performance from predefined knowledge. Slip_e and Slip_p are the slip factors in diagnosing students’ knowledge state from students’ performance. Slip_e (slip_{e}) belongs to the link between evidence and target concept. Slip_p (slip_{p}) belongs to the link between prerequisite concepts and the target concept. Slip_t (slip_{t}) is the slip
factor to consider the effect of time on the students’ knowledge state. Slip_t belongs to the link between the concept at the previous time spot and the concept at the current time spot. Concept is the number of concepts that each simulated student knows.

Table 5-2 reports the results of the ANOVA on correct diagnostic rates that depend on the factors: slip_do, slip_e, slip_p, slip_t, and concept. The interaction of slip_e x slip_p (F(9, 5832)= 55.3, P<0.001) is significant and a visual inspection of Figure 5-5 indicates that correct diagnostic rate values are higher when slip_e and slip_p take the same value and slip_e and slip_p are less than or equal to 0.1. Figure 5-5 also shows a simulated student model that presets slip_e and slip_p to the same (relatively small e.g. <=0.2) values estimates knowledge levels with an accuracy of at least 92.5%. The high correct diagnostic rates indicate that the student model in the simulation study can accurately diagnose students’ knowledge states.

The effect of slip_do (F(3, 5832)= 19458.5, p<0.001) is significant and a visual inspection of Figure 5-6 suggests that correct diagnostic rate values increase when slip_do decreases. This result is similar to the one for ABNs (see Figure 2-1 and the section 5.1.2). Even after we consider the time effect, slip_do still should be a small value. As noted above, real students rarely slip or guess creating design elements.

The effect of slip_t (F(3, 5832)= 72.3, p<0.001) is significant and a visual inspection of Figure 5-7 suggests that the correct diagnostic rate values increase with slip_t when slip_t is less than or equal to 0.1. When slip_t is 0.1 the correct diagnostic rate reaches the peak value. Figure 5-7 also shows that the correct diagnostic rates when slip_t is not zero are higher than the correct diagnostic rates when slip_t is zero. This indicates that the ADBN student model provides significantly higher accuracy than the
ABN student model does. In the ABN student model, the probability of the center concept is always calculated from the probabilities of its parent nodes first and then updated from the student solution. The ABN student model does not take into account the student’s previous understanding level for the center concept. By taking the student’s learning history in account, the ADBN student model performs better than the ABN student model.

Table 5-2 ANOVA Tests of Between-Subjects Effects
Dependent Variable: correct diagnostic rate

<table>
<thead>
<tr>
<th>Source</th>
<th>SS</th>
<th>DoF</th>
<th>MS</th>
<th>F Ratio</th>
<th>P Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>slip_do (do)</td>
<td>95.589</td>
<td>3</td>
<td>31.863</td>
<td>19458.506</td>
<td>.000</td>
</tr>
<tr>
<td>slip_e (e)</td>
<td>.335</td>
<td>3</td>
<td>.112</td>
<td>68.288</td>
<td>.000</td>
</tr>
<tr>
<td>slip_p (p)</td>
<td>.471</td>
<td>3</td>
<td>.157</td>
<td>95.882</td>
<td>.000</td>
</tr>
<tr>
<td>slip_t (t)</td>
<td>.474</td>
<td>4</td>
<td>.118</td>
<td>72.347</td>
<td>.000</td>
</tr>
<tr>
<td>concept</td>
<td>3.552</td>
<td>5</td>
<td>.710</td>
<td>4.146</td>
<td>.003</td>
</tr>
<tr>
<td>sid (concept)</td>
<td>9.252</td>
<td>54</td>
<td>.171</td>
<td>104.631</td>
<td>.000</td>
</tr>
<tr>
<td>do * e</td>
<td>1.221</td>
<td>9</td>
<td>.136</td>
<td>117.602</td>
<td>.000</td>
</tr>
<tr>
<td>do * p</td>
<td>.539</td>
<td>9</td>
<td>.060</td>
<td>51.930</td>
<td>.000</td>
</tr>
<tr>
<td>do * t</td>
<td>.366</td>
<td>12</td>
<td>.031</td>
<td>26.452</td>
<td>.000</td>
</tr>
<tr>
<td>do * concept</td>
<td>2.654</td>
<td>15</td>
<td>.177</td>
<td>153.359</td>
<td>.000</td>
</tr>
<tr>
<td>do * sid</td>
<td>15.903</td>
<td>162</td>
<td>.098</td>
<td>120.267</td>
<td>.000</td>
</tr>
<tr>
<td>e * p</td>
<td>.574</td>
<td>9</td>
<td>.064</td>
<td>55.287</td>
<td>.000</td>
</tr>
<tr>
<td>e * t</td>
<td>.471</td>
<td>12</td>
<td>.039</td>
<td>34.035</td>
<td>.000</td>
</tr>
<tr>
<td>e * concept</td>
<td>.855</td>
<td>15</td>
<td>.057</td>
<td>49.396</td>
<td>.000</td>
</tr>
<tr>
<td>e * sid</td>
<td>1.631</td>
<td>162</td>
<td>.010</td>
<td>6.580</td>
<td>.000</td>
</tr>
<tr>
<td>p * t</td>
<td>.592</td>
<td>12</td>
<td>.049</td>
<td>42.753</td>
<td>.000</td>
</tr>
<tr>
<td>p * concept</td>
<td>.361</td>
<td>15</td>
<td>.024</td>
<td>20.838</td>
<td>.000</td>
</tr>
<tr>
<td>p * sid</td>
<td>.837</td>
<td>162</td>
<td>.005</td>
<td>2.940</td>
<td>.000</td>
</tr>
<tr>
<td>t * concept</td>
<td>.317</td>
<td>20</td>
<td>.016</td>
<td>13.466</td>
<td>.000</td>
</tr>
<tr>
<td>t * sid</td>
<td>.842</td>
<td>216</td>
<td>.004</td>
<td>2.211</td>
<td>.000</td>
</tr>
<tr>
<td>do * e * p</td>
<td>.180</td>
<td>27</td>
<td>.007</td>
<td>5.771</td>
<td>.000</td>
</tr>
<tr>
<td>do * e * t</td>
<td>.090</td>
<td>36</td>
<td>.002</td>
<td>2.162</td>
<td>.000</td>
</tr>
<tr>
<td>do * e * concept</td>
<td>.264</td>
<td>45</td>
<td>.006</td>
<td>5.091</td>
<td>.000</td>
</tr>
<tr>
<td>do * e * sid</td>
<td>19.902</td>
<td>486</td>
<td>.041</td>
<td>35.554</td>
<td>.000</td>
</tr>
<tr>
<td>do * p * t</td>
<td>.219</td>
<td>36</td>
<td>.006</td>
<td>5.273</td>
<td>.000</td>
</tr>
<tr>
<td>do * p * concept</td>
<td>.078</td>
<td>45</td>
<td>.002</td>
<td>1.496</td>
<td>.017</td>
</tr>
<tr>
<td>do * p * sid</td>
<td>17.847</td>
<td>486</td>
<td>.037</td>
<td>27.019</td>
<td>.000</td>
</tr>
<tr>
<td>do * t * concept</td>
<td>.080</td>
<td>60</td>
<td>.001</td>
<td>1.151</td>
<td>.199</td>
</tr>
</tbody>
</table>
As Table 5-2 shows, there are significant interactions for all the combinations of variables except in two cases, do * t * concept (F(60, 5832)= 1.2, P>0.05, with significance 0.199), and do * e * p * t * concept (F(540, 5832)=0.4, P>0.05, with significance 1.000). The latter interaction is of more interest to us, because it involves slip_e and slip_p, and suggests that the significant interaction of these variables may not hold in the context of all other factors taken together. This is an interesting outcome which deserves further investigation. Nevertheless, results from smaller combinations of factors involving slip_e and slip_p do suggest significant effects and taken together support the conclusions drawn in this section.
Figure 5-5 Average correct diagnostic rate vs. parameter slip_e and slip_p

Figure 5-6 Average correct diagnostic rate vs. parameter slip_do
5.3 Evaluating student model with real students

The purpose of the evaluations with simulated students is to thoroughly test the student models to make sure that they perform as expected. However, how well the actual student behaviors are simulated by the student model needs to be evaluated with human subjects.

5.3.1 Parameters of evaluation

In previous research, human students’ performance in post-tests had been compared with the predicted performance to determine how accurate a student model is (Shute, 1995; Corbett & Bhatnagar, 1997; Corbett et al., 2000). The comparison
results are represented in terms of the Pearson product-moment correlation coefficient, \( r \) (Howell, 2002).

However, when the student model needs to provide accurate information about students’ understanding level of each concept, the Pearson coefficient, \( r \) and mean cannot serve the purpose of evaluating the accuracy of the student model. Instead, we should use the value of the Correct Prediction Rate (CPR) which is defined as

\[
CPR = \frac{\text{correct ProblemsCorrectly Predicted} + \text{wrong ProblemsCorrectly Predicted}}{\text{total Problems}}
\]  

(5-3)

Since the correct prediction rate only counts when a problem is predicted correctly, it guarantees that the student model with the highest correct prediction rate has the highest accuracy in diagnosing students’ learning state towards each concept in the knowledge domain.

### 5.3.2 Experiment procedure

We did the experiment with 71 human subjects, all novices with regard to object-oriented design. The 71 human subjects include 49 first-year undergraduates in Lehigh University, 12 high school students in Memphis, TN, and 10 high school students participating in the Launch-IT program in Bethlehem, PA (an NSF project). We had two sections in each experiment, a multimedia section and an object-oriented design section. In the multimedia section, we first introduced the object-oriented concepts and then let the students read a chapter about object-oriented concepts in the multimedia. Students did quizzes and exercises along with reading the multimedia. All
students had their own accounts and all of their performance was recorded in a database on the CIMEL server. The student model for the multimedia diagnosed the students’ knowledge state based on their performance. After students finished the multimedia, they were given a post-test in which each problem focuses on one concept. We calculated the accuracy of the student model by comparing the post-test results with the predicted post-test results by the student model for the multimedia.

In the object-oriented design section, we first introduced the method of doing object-oriented design from problem description and then let students do a design about a ticket machine. The students did the design in LehighUML plug-in in the Eclipse IDE. All students had their own accounts and their performances were recorded in a database on the ITS server. The student model for the ITS diagnosed the students’ knowledge level according to their performance. After the students finished the design, all the students were given a post-test in which each problem focused on one concept. We calculated the accuracy of the student model by comparing the post-test results with the predicted post-test results by the student model for the ITS for the LehighUML plug-in in the Eclipse IDE.

5.3.3 Evaluating student model for CIMEL multimedia with real students

We evaluated the student model for the CIMEL multimedia with the 71 human subjects. The materials for the experiment are the multimedia that involves graphics, audio, close-ended interactive exercises, and introduction about objects and classes, a student model for the multimedia, and a post-test. The post-test has 26 multiple-choice
questions. Each question in the tests focuses on a concept that belongs to the Concept Information Network. Since each student’s performance will be compared to the predicted one to determine the accuracy of the student model, no control group is needed.

The experiments were proceeded as follows: 1) students interact with the multimedia by doing all exercises and quizzes in the multimedia (as shown in Figure 5-8); and 2) students take the post-test. Human tutors did not answer any questions about problem solving during the experiment in order to eliminate human tutoring as a variable in the post-test.

Figure 5-9 illustrates the procedure of the evaluation experiments. In the experiment the student model diagnoses students’ learning state, and predicts real students’ performance in the post-test. Following the arrows, the experiments start from the human subject’s performance with the multimedia. Using the pre-defined knowledge state, the student model predicts whether the student can answer each problem correctly in the post-test. Since each problem in the post-test focuses on one concept, if P(concept of the problem) is greater than 0.5, the model predicts that the student made a correct step, and if P(concept of the problem) is less than 0.5, the model infers an error. The student model predicts the post-test results of each of the students.
We analyzed the experimental results as follows: 1) calculate the correct prediction rate by Equation (5-3), which determines the accuracy of the student model across all students; and 2) find the running time which measures the performance of the student model. The running time is defined as how long it takes the student model to issue the diagnostic results to the database after receiving data from the multimedia.
The average response time of the student model after a student enters the solution step is 0.09 seconds with a standard deviation of 0.07 seconds. Thus, the student model is judged to have performed in real time. The student model for close-ended questions predicts real students’ performance with an averaged accuracy of 80.1% and a standard deviation of 11.4%.

In this experiment, the students provide limited amount of evidence when the student model monitors their learning. Students were required to do only 26 exercises in the multimedia and provided evidence for about 13 concepts. The ADBN student model takes into account the prerequisites. For each student, it diagnosed 21 (26x80.1%) concepts with a standard deviation of 3 (26x11.4%) concepts from the evidence for 13 concepts. The result indicates that the ADBN student model for close-ended problems performs well. When the student model can obtain more evidence from the real students it will produce higher accuracy.

5.3.4 Evaluating student model for DesignFirst-ITS with real students

In the previous study, we evaluated the student model that diagnoses student’s performance with close-ended exercises. In this study, we evaluate our student model for DesignFirst-ITS with the same 71 human subjects.

The materials used in the experiment are the student model for Design-first ITS, Eclipse IDE with Lehigh UML plug-in, and a post-test. The post-test has 29 multiple choice questions. Each of the questions focuses on a concept that belongs to
the CIN. Since student’s performance will be compared to the predicted performance to evaluate the accuracy of the student model, no control group is needed.

![Image of object-oriented design in LehighUML plug-in in Eclipse IDE](image)

**Figure 5-10 Students do object-oriented design in LehighUML plug-in in Eclipse IDE**

The experiment was performed in the following steps: 1) students do object-oriented design of a novice problem in the LehighUML plug-in in the Eclipse IDE as shown in Figure 5-10, getting tutorial help from DesignFirst-ITS; and 2) students take the post-test. Human tutors did not answer any questions about problem solving during the experiment to eliminate human tutoring as a variable in the post-test.

During the experiment, the students developed an object-oriented design for a movie ticket machine using Eclipse IDE with our LehighUML plug-in. DesignFirst-ITS provides students with help while they are working on the Object-oriented design.
The student model is one of the components in DesignFirst-ITS (Blank, Parvez, et al., 2005; Wei et al., 2005). After a student finished a design solution step, the Expert Evaluator identifies whether the solution step is correct (Moritz, 2007), then the student model diagnoses the student’s knowledge state.

Figure 5-11 illustrates the procedure of the experiments. The experiments distinguish between the student model that diagnoses students’ learning state and the student model that predicts the real student’s performance in the post-test. Following the arrows, the experiments start from the human subject’s performance in the LehighUML plug-in in the Eclipse IDE. Using the pre-defined knowledge state as prior probabilities, the student model predicts whether a student answers each problem correctly in the post-test. If \( P(\text{concept of a problem}) \) is greater than 0.5, the model predicts that the student made a correct step, and if \( P(\text{concept of a problem}) \) is less than 0.5, the model infers an error. The student model predicts the post-test results for each of the students.

![Figure 5-11 Illustration of evaluation procedure](image)

The experimental results were examined as the following: 1) the accuracy of the student model was evaluated by the correct prediction rate defined in Equation (5-3) across all students; and 2) the responsiveness performance of the student model
was evaluated by the running time used to issue the diagnostic results to database after receiving data from the Expert Evaluator.

The averaged response time of the student model after a students enters the solution step is 0.34 seconds with a standard deviation of 0.25 seconds. Thus, the student model is judged to be able to perform in real time with open-ended questions. The student model for open-ended questions predicts real students’ performance with average accuracy of 81.8%, and the standard deviation of 11.9%.

In this experiment, the amount of evidence of the student performance is limited when the student model monitors the student’s learning. Students provided evidence for about 15 concepts. The ADBN student model takes into account the prerequisites. For each student, it diagnosed 24 (29x81.8%) concepts with a standard deviation of 3 (29x11.9%) concepts from the evidence for 15 concepts. The result indicates that the ADBN student model for open-ended problems performs well. When the student model can obtain more evidence from the real students it will produce higher accuracy.

5.3.5 Evaluating diagnoses integration in DesignFirst-ITS with real students

This section investigates whether the accuracy of the student model can be increased by integrating the diagnosed knowledge from the closed- and open-ended problems.

We evaluated the student model for the DesignFirst-ITS when considering the prior knowledge from the CIMEL multimedia with the 71 human subjects. The
materials for the experiment are the diagnosed students’ learning state from the previous experiments with the multimedia, the students’ performance data from the DesignFirst-ITS, the post-tests, and the student model for the DesignFirst-ITS. Since each student’s performance will be compared to the predicted performance to justify the accuracy of the student model, no control group is needed.

The experiment is performed in the following steps: 1) the student model for DesignFirst-ITS diagnoses the student’s performance taking the diagnosed students’ learning state from the multimedia as prior probabilities; 2) we compare the predicted students’ performances with the real students’ performances to calculate the correct prediction rate; and 3) we compare the correct prediction rate with the one from the same student model for the DesignFirst-ITS without considering the student’s prior knowledge state.

Figure 5-12 Illustration of evaluation procedure

Figure 5-12 illustrates the procedure of the evaluation experiment. Following the arrows, the experiments start from the human subject’s performance in the LehighUML plug-in in the Eclipse IDE. Using the diagnosed student’s knowledge state from the multimedia as prior probabilities, the student model calculates the students’ knowledge state based on students’ performance in the LehighUML plug-in. Then, the student model predicts whether a student answers each problem in the post-test correctly. If P(concept of a problem) is greater than 0.5, the model predicts that
the student made a correct step, and if $P(\text{concept of a problem})$ is less than 0.5, the model infers an error. Hence the student model predicts the post-test results of each of the students.

In the ANOVA analysis, the correct prediction rate of each simulated student depends on one fixed-effect independent factor that is 2 (method) and one random factor that is 71 (student). The factor “method” represents that whether the student model uses the diagnosed student’s learning state from the CIMEL multimedia as prior probabilities.

Table 5-3 reports the results of the ANOVA on the correct prediction rate that depends on the factor “method” (the effect of whether taking into account the evidence in the multimedia). The ANOVA shows that the effect “method” is significant ($F(1,70)=172.9$, $p<0.001$). A visual inspection of Figure 5-13 indicates that the correct prediction rate is higher when considering the knowledge state from the multimedia. This result implies that integrating diagnoses from the close-ended and the open-ended exercises is an effective way to increase the accuracy of student models.

**Table 5-3 ANOVA Tests of Between-Subjects Effects**

<table>
<thead>
<tr>
<th>Source</th>
<th>SS</th>
<th>DoF</th>
<th>MS</th>
<th>F Ratio</th>
<th>P Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>method</td>
<td>.312</td>
<td>1</td>
<td>.312</td>
<td>172.946</td>
<td>.000</td>
</tr>
<tr>
<td>sid</td>
<td>1.654</td>
<td>70</td>
<td>.024</td>
<td>13.103</td>
<td>1.000</td>
</tr>
<tr>
<td>Error</td>
<td>.126</td>
<td>70</td>
<td>.002</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

This result indicates that initializing the student model with data from the multimedia works better than starting with the arbitrary value 0.5. It also works better
than relying on evidence from UML problem solving alone. The new evidence includes students’ performance of applying some new concepts that are never covered in the evidence in the DesignFirst-ITS, and some old concepts that have already been covered. The evidence about the new concepts enable the student model for the DesignFirst-ITS to add more information in the diagnosed student’s learning state, and all the new evidence enables the student model to generate more accurate diagnoses about students’ understanding level to each concept. The combination of integrating evidence from the multimedia plus a history of prior student work significantly improves the accuracy of ADBNs.

![Estimated Marginal Means of CPR](image)

*Figure 5-13 Average correct prediction rate vs. factor “method”*

2 – without considering the prior knowledge from the multimedia
3 – consider the prior knowledge from the multimedia
5.3.6 Comparing with non-advanced-numerical student models

There are some student models that use non-advanced-numerical techniques such as match, summation and subtraction (Weber & Brusilovsky, 2001; Mitrovic et al., 2001). In this section, we evaluated the non-advanced-numerical student model for the CIMEL multimedia and the DesignFirst-ITS with the 71 human subjects and compared its performance with the student model with the ADBN student model.

The materials for the experiments are the non-advanced-numerical student models for the multimedia and the DesignFirst-ITS, the student’s performance data in the multimedia and in the LehighUML plug-in in the Eclipse IDE, and the post-test. Since the student’s real performance will be compared to the predicted performance to determine the accuracy of the student model, no control group is needed.

The experiment for the multimedia proceeded as follows: 1) the non-advanced-numerical student model diagnoses the students’ real performances in the multimedia and generate the learning state for each student; 2) the non-advanced-numerical student models students predict whether a student will answer the questions in the post-test correctly; 3) we compare the predicted students’ performances with the real students’ performances to generate the correct prediction rate; and 4) we compare the correct prediction rate with the one from the ADBN student model for the multimedia. The experiment for the DesignFirst-ITS is performed in the same sequence but with everything that relates to the DesignFirst-ITS including the students’ performance data and the student models.

Figure 5-14 illustrates the procedure of the experiments. The experiments distinguish between the student model that diagnoses students’ learning state and the
student model that predicts real students’ performance in the post-test. Following the arrows, the experiments start from the human subject’s performance. Using the diagnosed knowledge state, the non-advanced-numerical student model generates the prediction of students’ performance in the post-tests.

![Diagram](image_url)

**Figure 5-14 Illustration of evaluation procedure**

In the algorithm for the non-advanced-numerical student model, each student has an initial confidence value of 0 for each concept. When a student answers a question correctly, the student model adds 1 to the confidence value for the concept that is related to the question. When the student answers a question erroneously, the student model subtracts 1 from the confidence value for the concept that is related to the question. After the student finishes all problems, if the confidence value for a concept is greater than or equal to a fixed value then student model predicts that the student knows the concept, otherwise, the student does not know the concept. The fixed value is two since each student has at least two pieces of evidence for each problem for the non-advanced-numerical student model.

In the ANOVA analysis for the student models for the multimedia, the correct prediction rate of each simulated student depends on one fixed-effect independent factor that is 2 (method) and one random factor that is 71 (student). The factor “method” represents that whether the student model uses the ADBN student model or the non-advanced-numerical.
Table 5-4 reports the results of the ANOVA on the correct prediction rate that comes from the two levels of the factor “method” (i.e., the effect of the multimedia). The ANOVA shows that the effect of “method” on the correct prediction rate is statistically significant (F(1,70)= 102.0, p<0.001). A visual inspection of Figure 5-15 indicates that the correct prediction rate value is significantly higher when using the ADBN student model for the multimedia.

### Table 5-4 ANOVA Tests of Between-Subjects Effects

Dependent Variable: CPR for the student models for the multimedia

<table>
<thead>
<tr>
<th>Source</th>
<th>SS</th>
<th>DoF</th>
<th>MS</th>
<th>F Ratio</th>
<th>P Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>method</td>
<td>2.630</td>
<td>1</td>
<td>2.630</td>
<td>101.957</td>
<td>.000</td>
</tr>
<tr>
<td>sid</td>
<td>.486</td>
<td>70</td>
<td>.007</td>
<td>.269</td>
<td>1.000</td>
</tr>
<tr>
<td>Error</td>
<td>1.806</td>
<td>70</td>
<td>.026</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Estimated Marginal Means of CPR for the student models for the multimedia

5 – use the non-advanced-numerical student model
6 – use the ADBN student model

**Figure 5-15 Averaged CPR of student models for close-ended problems**

160
In the ANOVA analysis for the student models in the Eclipse environment, the correct prediction rate of each simulated student depends on one fixed-effect independent factor that is 4 (method) and one random factor that is 71 (student). The factor “method” represents: 1) using the non-advanced-numerical student model which does not consider the prior knowledge from the multimedia; 2) using the non-advanced-numerical student model which considers the prior knowledge from the multimedia; 3) using the ADBN student model which does not consider the prior knowledge from the multimedia; and 4) using the ADBN student model which considers the prior knowledge from the multimedia.

Table 5-5 reports the results of the ANOVA on the correct prediction rate that comes from the four levels of the factor “method.” The ANOVA shows that the effect of “method” on the correct prediction rate is statistically significant (F(3, 210)= 263.3, p<0.001).

Table 5-5 ANOVA Tests of Between-Subjects Effects
Dependent Variable: CPR for the student models in Eclipses environment

<table>
<thead>
<tr>
<th>Source</th>
<th>SS</th>
<th>DoF</th>
<th>MS</th>
<th>F Ratio</th>
<th>P Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>method</td>
<td>9.773</td>
<td>3</td>
<td>3.258</td>
<td>263.289</td>
<td>.000</td>
</tr>
<tr>
<td></td>
<td>.568</td>
<td>70</td>
<td>.008</td>
<td>.656</td>
<td>1.000</td>
</tr>
<tr>
<td>Error</td>
<td>2.599</td>
<td>210</td>
<td>.012</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

A visual inspection of Figure 5-16 indicates that from the same set of evidence, the ADBN student model performs better than the non-advanced-numerical student model. The reasons are: 1) the ADBN student model considers students’ knowledge structure where related concepts and their prerequisites are diagnosed from each piece
of evidence, while the non-advanced-numerical student models only diagnose the related concepts; 2) the ADBN student model considers students’ learning history and a student’s understanding level to a concept depends on the student’s learning history, while the non-advanced-numerical student models only consider the very last step of the student’s performance; 3) the ADBN student model captures a student’s learning behaviors by simulating that the student’s slip or guess when he/she answers problems, and hence generate more accurate students’ learning state, while the non-advanced-numerical student models do not simulate any students’ learning behaviors; and 4) the ADBN student model applies Bayesian theory which has been proved to be successful in solving problems with statistical technology, while the non-advanced-numerical student models only apply non-statistical scheme which cannot model real students.

Figure 5-16 also shows that the correct prediction rate values are higher when considering the knowledge state from the multimedia for both the non-advanced-numerical student model and the ADBN student model. This implies that integrating diagnoses from close-ended and open-ended exercises is an effective way to increase the accuracy of student models.
0 – the non-advanced-numerical student model does not consider the prior knowledge
1 – the non-advanced-numerical student model considers the prior knowledge
2 – the ADBN student model does not consider the prior knowledge
3 – the ADBN student model considers the prior knowledge

Figure 5-16 Average CPR for the student models for open-ended problems
6 Conclusions and future work

6.1. Summary of dissertation

Learning object-oriented design and programming is challenging for novices. Many students have learning difficulties which cannot be entirely overcome by teachers. To help student learn, we have developed an Intelligent Tutoring System (ITS) which integrates observations of student actions from both the CIMEL multimedia and the Eclipse IDE. As one of the major components in the ITS, a Student Model determines which concepts a student understands so that he or she can apply them in design problems and which concepts he/she needs more help.

Our ADBN-based student model provides a more accurate knowledge profile of a student so that the ITS can support adaptive tutoring. It infers the problem-domain knowledge state from a student's work and the historical knowledge state of the student. Ultimately, our three-layered architecture is based on the different levels of knowledge, domain knowledge, reasoning knowledge, monitoring knowledge, and reflective knowledge.

The first layer of the student model diagnoses the domain knowledge and the reasoning knowledge. It is implemented by Atomic Bayesian Networks (ABNs) which are a new approach to student modeling. Each ABN considers a small number of immediate prerequisites, a center concept and an action step. A student model implemented with ABNs uses prerequisites to estimate the current students' knowledge level. We set \( \text{slip}_e = \text{guess}_e \) and \( \text{slip}_p = \text{guess}_p \) where \( e \) represents evidence
and $p$ represents prerequisite to limit the number of parameters that an ABN needs to combine.

We developed a three-step algorithm to update an ABN: 1) calculate the unconditional probability of the center node D based on its immediate parents; 2) update the center node according to a student’s solution; and 3) update the parent nodes according to the student’s solution. We defined a mock power value $n$, which is like a power value in an exponential expression since the equations used to update an ABN are similar to an exponential expression with terms $(1 - \text{slip}_p)$ and $\text{guess}_p$ to the power of $n$. We derived the formulas for the three-step algorithm using the simplified notation with mock power value.

The second layer of the student model diagnoses the domain knowledge and the reasoning knowledge in a history. It is implemented by Atomic Dynamic Bayesian Networks (ADBNs) which are a new approach to student modeling as well. Each ADBN consists of two ABNs from two successive time slices. Each ABN considers a small number of immediate prerequisites, a center concept and an action step. A student model implemented with ADBNs estimates the current students’ knowledge level from prerequisites in the current time slice and the previous time slice. The constraints of ABN and ADBN structure and updating method lead to the student model’s responsiveness in real time.

The third layer of the student model, the Cognitive Model, diagnoses the reflective knowledge and the monitoring knowledge. This model incorporates general cognitive strategies that the student uses such as focus, approach, hacking, and analogy, and domain-specific cognitive strategies from problem description to class
diagram. The strategies represent different problem solving patterns and anti-patterns in the domain of object-oriented design and learning in general.

We developed the student model for CIMEL multimedia which diagnoses close-ended exercises such as multiple choice exercises and drag and drop problems. For this student model we designed and implemented the procedure of abstracting information from students’ performance records. We also developed the student model for DesignFirst-ITS which diagnoses open-ended exercises, creating an object-oriented class diagram. For this student model, we designed and implemented the procedure of abstracting information from the Expert Evaluator’s diagnoses about students’ performance. Both student models use ADBN to diagnose students’ learning state. We integrated the diagnosed students’ learning state from close-ended exercises and open-ended exercises to represent the synthetic students’ learning state.

We developed a non-advanced-numerical student model which uses a straightforward algorithm. We compared the accuracy of the non-advanced-numerical student models and the student models with ADBNs in the CIMEL multimedia and in DesignFirst-ITS to investigate the merits of ADBNs.

We evaluated the sufficiency of ABNs for student models with 240 simulated students, and also investigated the influence of different slip and guess values based on the results of the simulated students. The results show that correct diagnostic rates are higher when \(slip_p\), \(guess_p\) and \(slip_e\), \(guess_e\) take the same value. There is no significant difference when slip and guess values are different and relatively small. The high correct diagnostic rates indicate that the student model can correctly diagnose students’ knowledge states.
We evaluated the sufficiency of ADBNs for student models with 240 simulated students. There is no significant difference when three pairs of slip and guess values are different and relatively small. The high correct diagnostic rates indicate that the student model with ADBNs can diagnose students’ knowledge states accurately. We also investigated the difference between student models using ADBNs and ABNs. The results show that correct diagnostic rates with of ADBN student model are significantly higher than ABN student model.

We evaluated the student model for CIMEL multimedia with 71 human subjects. The results show that by presetting slip and guess with relatively small values (<=0.1) the student model for close-ended exercises can generate reliable diagnoses about students’ learning state, and pre-setting slip and guess to various values will affect the accuracy of the student model as much as 8.4%. The average response time of the student model after a students enters the solution step is 0.24 seconds, which shows that the student model performs in real time, and supports responses as students are working on exercises.

We evaluated the student model for DesignFirst-ITS with 71 human subjects. The results show that the student model for open-ended questions that have presetting slip and guess with relatively small values (<=0.1) can predict real students’ performance with an accuracy of at least 81.8%, confirming that pre-setting slip, guess will lead to a reliable student model. The average response time of the student model after a students enters the solution step is 0.63 seconds, confirming that the student model supports responses as students are working on problem-solving steps.
We found that there is a difference between the highest correct diagnoses rate in the studies with simulated students, and the highest correct prediction rate in the experiments with real students. The reason is that the simulated students do not “read” any feedback; while real students are given feedback every time they make an error and they take the post tests after they review the feedback. Therefore, some real students’ learning state may change after they have reviewed the feedback. The student model does not catch this change which leads to a lower correct prediction rate comparing to the correct diagnoses rate in the simulate studies.

We evaluated the student model which integrates the diagnosed students’ learning state from closed- and open-ended exercises with 71 human subjects. This student model diagnoses students’ learning state in DesignFirst-ITS environment and considers the prior knowledge from the CIMEL multimedia. The results show that the correct prediction rate is increased by 7.7% if considering the prior students’ knowledge diagnosed from the close-ended exercises in CIMEL multimedia. It implies that integrating diagnoses from closed- and open-ended exercises is an effective way to increase the accuracy of student models.

We evaluated the non-advanced-numerical student models with 71 human subjects’ data from CIMEL multimedia and DesignFirst-ITS, and compared the accuracy of the non-advanced-numerical student models with the student model using ADBNs in both environments. The results show that from the same set of evidence, student models using ADBNs perform more than two times better than the non-advanced-numerical student models. The accuracy of the non-advanced-numerical student model is too low to meet the requirement of a reliable student model. The
reason is that ADBNs apply a far more sophisticated structure which considers students’ knowledge structure and learning history with a novel development of Bayesian networks theory.

6.2. Conclusions

This dissertation has the following major conclusions:

1. The correct diagnostic rates of ADBN student model are higher than ABN student model.
2. Integrating diagnoses from closed- and open-ended exercises is an effective way to increase the accuracy of student models.
3. Student models using ADBNs perform much better than the non-advanced-numerical student models.
4. Student models with ADBNs can diagnose students’ knowledge states accurately in real time.

6.3. Contribution

This research makes seven contributions to student modeling. The contributions of this research can be summarized as follows:

1. A novel way to represent students’ knowledge structure, where both concepts and relationship between concepts are knowledge units that students need to learn.
2. A novel three-layered architecture which can be standardized in modeling various strata of students’ knowledge.
3. ABN – a novel Atomic Bayesian Network scheme that provides a refined representation of prerequisite relationships, diagnoses student’s knowledge structure, and guarantees real-time responsiveness.

4. ADBN – an innovative Dynamic Bayesian Network scheme that represents refined representation of prerequisite relationships and diagnoses students’ knowledge structure considering the learning history in real time.

5. A unique student model that integrates knowledge from open-ended problem solving (object-oriented class diagram design) and close-ended exercises.

6. A unique student model that simulates students’ knowledge across a broad array from basic domain knowledge to positive cognitive strategies.

7. This study presents a general approach for student models that help students learn complex problem solving in real time.

6.4. Future work

This research did not implement the details for the cognitive model in the student model. The future work can extend our student model to simulate monitoring knowledge and reflective knowledge. The monitoring knowledge is about domain-specific cognitive strategies from problem description to class diagram, while the reflective knowledge is about general cognitive strategies that the student is using such as focus, approach, hacking, and analogy. Future work can investigate the details about how to represent the two kinds of knowledge and evaluate the cognitive model with simulated and human subjects. Synthesizing the diagnostic results from the different layers of the student model, i.e. a student’s knowledge state and cognitive
strategies to identify his/her tutoring needs after the student commits a specific error is also interesting future work.

The scope of this research is novice object-oriented design. Future work can investigate the possibilities of using ADBNs to diagnose students’ learning state when learning in other domains, such as object-oriented programming. This new domain would require determining how to represent the structural knowledge of programming syntax, and the domain-specific cognitive strategies and the general cognitive strategies for programming. Other than these domain-specific knowledge representation issues, the new theory and technology for student models should transfer to other domains straightforwardly.

This research focused on the effect of small slip and guess values. Unlike $slip_e$ and $guess_e$, which intuitively should be small because students are creating design elements, it may be reasonable to consider the possibility of larger $guess_p$ values, since, intuitively, students may possibly try design moves without fully understanding the prerequisite knowledge. Small $slip_p$ and $guess_p$ values do seem to work well for ADBNs, but the effect of different $slip_p$ and $guess_p$ values is an open question for future investigation.
## Appendices

### Appendix I: Definitions of concepts in CIN for student model

<table>
<thead>
<tr>
<th>Concept Name</th>
<th>Relationship Specified by the Concept</th>
<th>Concept Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>method_parameter</td>
<td>Uses</td>
<td>A parameter can only be accessed inside its method. Changes to the value of a formal parameter within the method do not affect the corresponding actual parameter passed into the method.</td>
</tr>
<tr>
<td>parameter_variable</td>
<td>Extended Prerequisite</td>
<td>A parameter is similar to a variable.</td>
</tr>
<tr>
<td>attribute_object</td>
<td>Extended Prerequisite</td>
<td>Attributes contain values that accessible within an object.</td>
</tr>
<tr>
<td>method_object</td>
<td>Extended Prerequisite</td>
<td>Methods are operations that an object can perform.</td>
</tr>
<tr>
<td>class_method</td>
<td>Extended Prerequisite</td>
<td>A method is a behavior of a class and can be invoked by any methods inside the class.</td>
</tr>
<tr>
<td>class_object</td>
<td>Extended Prerequisite</td>
<td>A class is an abstraction of a category of objects which have similar characteristics and can perform similar operations.</td>
</tr>
<tr>
<td>constructor_object</td>
<td>Uses</td>
<td>A constructor helps to create an object of a class.</td>
</tr>
<tr>
<td>attribute_class</td>
<td>Extended Prerequisite</td>
<td>Attributes contain values that accessible within a class.</td>
</tr>
<tr>
<td>class_constructor</td>
<td>Extended Prerequisite</td>
<td>The constructor has the same name of the class.</td>
</tr>
<tr>
<td>attribute_constructor</td>
<td>Manipulate</td>
<td>A constructor assigns initial values to attributes.</td>
</tr>
<tr>
<td>attribute_parameter</td>
<td>Contrast</td>
<td>Though attributes and parameters are variables, attributes are accessible anywhere within the scope of a class, while parameters are accessible only within the scope of a method.</td>
</tr>
<tr>
<td>attribute_method</td>
<td>Manipulate</td>
<td>A method modifies, accesses, and displays values of attributes.</td>
</tr>
<tr>
<td>method_returntype</td>
<td>Uses</td>
<td>A method uses a returntype to define the type of value that is returned.</td>
</tr>
<tr>
<td>constructor_method</td>
<td>Contrast</td>
<td>A constructor has no returntype while a method has one.</td>
</tr>
<tr>
<td>datatype_variable</td>
<td>Extended Prerequisite</td>
<td>A variable has a datatype.</td>
</tr>
<tr>
<td>attribute_variable</td>
<td>Extended Prerequisite</td>
<td>An attribute is a variable.</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>returntype_variable</td>
<td>Extended Prerequisite</td>
<td>Returntype is the datatype of a variable that is returned from a method.</td>
</tr>
<tr>
<td>datatype_returntype</td>
<td>Extended Prerequisite</td>
<td>Returntype is a datatype for a return value.</td>
</tr>
<tr>
<td>class_parameter</td>
<td>Scope contrast</td>
<td>Parameters are accessible only within the scope of a method, not the wider scope of the class.</td>
</tr>
<tr>
<td>double_int</td>
<td>Contrast</td>
<td>A double holds a decimal number while an integer holds a whole number</td>
</tr>
<tr>
<td>double_String</td>
<td>Contrast</td>
<td>A double holds a decimal number while a string holds a sequence of letters</td>
</tr>
<tr>
<td>int_String</td>
<td>Contrast</td>
<td>An integer holds a whole number while a string holds a sequence of letters</td>
</tr>
<tr>
<td>boolean_int</td>
<td>Contrast</td>
<td>A Boolean holds a true or false value while an integer holds a whole number</td>
</tr>
<tr>
<td>char_String</td>
<td>Contrast</td>
<td>A string holds a sequence of letters while a char holds one letter</td>
</tr>
</tbody>
</table>
Appendix II: Glossary for the first chapter

**Architectures:**

The conceptual design and fundamental operational structure of a computer system. It is a blueprint and functional description of requirements (especially speeds and interconnections) and design implementations for the various parts of a computer.

**Artificial intelligence:**

Artificial intelligence (AI) refers the science and engineering of making intelligent machines. It can also refer to intelligence as exhibited by an artificial (man-made, non-natural, manufactured) entity. The terms strong and weak AI can be used to narrow the definition for classifying such systems. AI is studied in overlapping fields of computer science, psychology, philosophy, neuroscience and engineering, dealing with intelligent behavior, learning and adaptation and usually developed using customized machines or computers.

**Bayesian network:**

A probabilistic graphical model that represents a set of variables and their probabilistic dependencies.

**Class:**

An abstraction of a category of objects.

**Class diagram:**

A type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, methods, and the relationships between the classes.
**Close-ended problem:**

An exercise with only one correct answer which a student selects from a small set of possibilities, such as a multiple-choice, true-false, or a drag and drop activity.

**Drag-and-drop problem:**

A kind of multimedia exercises where students see several movable choices and non-movable matching slots and must drag each choice to the correct matching slot.

**Dynamic Bayesian network:**

A Bayesian network that represents a temporal probability model. It consists of variables at different time slices and considers the dependencies between these connected variables.

**DesignFirst-ITS:**

An intelligent tutoring system that provides one-on-one tutoring to help beginners learn object-oriented design in an introductory course in computer science. DesignFirst-ITS is based on a design-first curriculum which subsumes an objects-first approach into lessons that also introduce object-oriented analysis and design, using elements of UML before implementing any code.

**Expert evaluator:**

A module of an ITS that compares a students work with a solution that a domain expert or teacher might generate for the same problem and diagnoses whether the student’s work is correct or incorrect.

**Exponential computational time:**

In complexity theory, exponential time is the computation time of a problem where the time to complete the computation, m(n), is bounded by an exponential function of the
problem size, \( n \) (i.e., as the size of the problem increases linearly, the time to solve the problem increases exponentially).

Computer scientists sometimes think of polynomial time as "fast," and anything running in greater than polynomial time as "slow." Exponential time would therefore be considered dangerously slow. This notion provides a useful intuition, but is imprecise. In practice, the actual running time of any algorithm depends on the value of \( n \) and the constants (see Big O notation for details). For a given value of \( n \), a specific polynomial time algorithm may have greater running time than a specific exponential-time algorithm. However, for sufficiently-large values of \( n \), the running time of the exponential algorithm will dominate.

**Hacking:**

A kind of consistent learning behavior of making correct answers without understanding the related concepts. It has two kinds: 1) hacking the solution, and 2) hacking the tutor. When students hack the solution, they enter variant answers by guessing and hope one of them happens to be the correct answer. When students hack the tutor, they take advantage of the way that the system provides feedback.

**Intelligent Tutoring System (ITS):**

An artificially intelligent computer system that provides direct - i.e. without the intervention of human beings - customized instruction or feedback to students.

**Object:**

An entity distinguishable from others by its attributes and behaviors.

**Object (Software):**

A unit of run-time data storing attribute values which are manipulable via methods.
**Object-oriented design:**
A method for defining the objects and their interactions to solve a problem. It encourages software developers to think in terms of objects with typically small methods rather than large blocks of code or procedural decomposition.

**Object-oriented class diagram design:**
Object-oriented class diagram design creates class diagrams to model a problem domain.

**Open-ended problem:**
An exercise with many potentially correct answers, and several ways to the correct answer(s). In responding to open-ended items, tutors not only care about students’ work, but also care about how they got their answers or why they chose the method they did.

**Pedagogical advisor:**
A module of an ITS that provides feedback to the student.

**Real time:**
Real-time computing (RTC) is the study of hardware and software systems which are subject to a "real-time constraint"—i.e., operational deadlines from event to system response. By contrast, a non-real-time system is one for which there is no deadline, even if fast response or high performance is desired or even preferred.

**Rule-based system:**
Represents knowledge in terms of if-then productions: if a condition is true, then update the knowledge state or perform some action.
**Student model:**

A module of an ITS that represents a student’s current knowledge state and diagnoses gaps in the student’s knowledge.

**Unified Modeling Language:**

a standardized specification language for object modeling. UML is a general-purpose modeling language that includes a graphical notation used to create an abstract model of a system.
Bibliography


Vita

Fang Wei was born to Mr. Gujian Wei and Mrs. Shitao Tan in Beijing China on March 13, 1972. She is married to Dr. Daming Yu. She graduated with a B.S. degree in Civil Engineering from Shijiazhuang Railway Institute in Shijiazhuang, Hebei province of China in 1994. She then began her graduate study under Dr. Guiping Yan and Prof. He Xia at Northern Jiaotong University (now Beijing Jiaotong University) and received her M.S. degree in Civil Engineering in 1998. The title of her thesis was, “Analyzing Dynamic Responses of High Speed Railroad Beam Bridges under Seismic Loads.” Beginning in 2000, she spent two years in Kutztown University to take undergraduate and graduate level courses in computer science. In 2002, she began to pursue her Ph.D. in Computer Science at Lehigh University, Bethlehem, where she was a research assistant under the guidance of Dr. Glenn D. Blank. She will receive her Ph.D. degree in Computer Science for her work on student modeling in Intelligent Tutoring System in September 2007.