LEARNER MODELING FOR INTEGRATION SKILLS IN PROGRAMMING

by

Yun Huang

B.E. in Intelligence Science and Technology, School of Computer Science, Beijing University of Posts and Telecommunications, China, 2011

M.S. in Intelligent Systems, Dietrich School of Arts and Sciences, University of Pittsburgh, 2015

Submitted to the Graduate Faculty of
the Dietrich School of Arts and Sciences, University of Pittsburgh
in partial fulfillment
of the requirements for the degree of

Doctor of Philosophy

University of Pittsburgh

2018

UNIVERSITY OF PITTSBURGH

DIETRICH SCHOOL OF ARTS AND SCIENCES

This dissertation was presented

by

Yun Huang

It was defended on

July 5, 2018

and approved by

Dr. Peter Brusilovsky, School of Computing and Information, University of Pittsburgh

Dr. Marek Druzdzel, School of Computing and Information, University of Pittsburgh

Dr. Christian Schunn, Learning Research and Development Center, University of Pittsburgh

Dr. Kenneth Koedinger, Human-Computer Interaction Institute, Carnegie Mellon University

Dissertation Director: Dr. Peter Brusilovsky, School of Computing and Information, University of Pittsburgh

LEARNER MODELING FOR INTEGRATION SKILLS IN PROGRAMMING

Yun Huang, PhD

University of Pittsburgh, 2018

Mastery development requires not only acquiring component skills, but also practicing their integration into more complex skills. When learning programming, an example is to first learn += and loops, then learn how to combine them into a loop that sums a sequence of numbers. The existence of integration skills has been supported by cognitive science research, yet it has rarely been considered in learner modeling, the key component for adaptive assistance in an intelligent tutoring system (ITS). Without this, early assertions of mastery in ITSs after only basic component skill practice or practice in limited contexts may be merely indicating shallow learning.

My dissertation introduces integration skills, widely acknowledged by cognitive science research, into learner modeling. To demonstrate this, I chose program comprehension with a complex integrative nature. To provide grounds for skill modeling, I applied a Difficulty Factors Assessment (DFA) approach (from cognitive science) and identified integration skills along with generalizable integration difficulty factors in common basic programming patterns. I used the DFA data to inform the construction of the learner model, CKM-HI, which incorporates integration skills in a hierarchical structure in a Bayesian network (BN). Compared with other machine learning approaches, BN naturally utilizes domain knowledge and maintains interpretable knowledge states for adaptation decisions. To address the limitation of prediction metrics to evaluate such multi-skill learner models, I proposed and applied a multifaceted evaluation framework. Data-driven evaluations on a real-world dataset show that CKM-HI is superior to two popular multi-skill learner models, CKM and WKT, regarding predictive performance, parameter plausibility, and expected instructional effectiveness.

To evaluate its real-world impact, I built a program comprehension ITS driven by learner modeling and a classroom study deploying this system suggests that CKM-HI could lead to better learning than the CKM model.

My dissertation work is the first to systematically demonstrate the value of integration skill modeling, and offers novel integration-level learner modeling and multifaceted evaluation approaches applicable to a broader context. Further, my work contributes recent ITS infrastructure and techniques to programming education, and also contributes an example of taking an interdisciplinary approach to ITS research.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

PREFACE

This PhD journey has been a very fulfilling experience for me thanks to all the amazing people I got to work with or share time with during these years. There are so many people I want to thank and I am sure I will end up missing a few.

Firstly, I want to thank my advisor, Peter, for his believing in me, his great support and advice on my research path and personal growth, for all these six years. I am very grateful to my other committee members, Chris, Ken and Marek, for their invaluable advice, and for their spending so much time to guide me, without which I couldn't have developed new perspectives to identify problems and form solutions. I want to thank my mentor, Jose, who guided me when I started the PhD, and taught me skills and rigorousness that are so important for doing research. Also, I am grateful to Rosta, Yuru, and Igor, for their help and inspiration when I got stuck, and UACH teachers for their patience to work with me.

Secondly, I want to thank all my friends and colleagues who have accompanied and helped me to grow on this journey, through collaborations, discussions, and their kind understanding, encouragement and support. I especially want to thank Rosta, my roommates Linda and Ziyi, Di, Roya, Khushboo, who gave me so much love and made me feel I was not alone, thank Yeye for running and sharing feelings and thoughts with me almost everyday during these last stressful months, thank Lingjia for helping me settling down Pittsburgh, and thank Jordan for helping me running initial studies regardless of his tight schedule.

Last but not least, I want to thank my my husband, Daniel, in whose arms I always feel so peaceful, from whose eyes and voice I feel so loved and strong. I also want to thank my parents-in-law for their support and love. I am eternally grateful to my parents, without whom I couldn't have reached here. Dad and mom, my every taste of happiness and peace, the quest for the truth and excellency, originated and have been molded by you. Thank you!

1.0 INTRODUCTION

1.1 MOTIVATION

Mastery development requires not only the acquisition of basic component skills, but also practicing the integration of these skills [Ambrose et al., 2010], during which students acquire common chunks, schemas or patterns, as evidenced by research into expertise acquisition in a wide range of domains, such as chess [Chase and Simon, 1973, De Groot, 1978], electronic circuitry [Egan and Schwartz, 1979], physics [Larkin et al., 1980] and geometry [Koedinger and Anderson, 1990]. Especially relevant to my study is Heffernan and Koedinger's research on algebra [Heffernan and Koedinger, 1997], which has provided empirical evidence demonstrating that additional knowledge is required in order to accomplish specific skill combinations. In this research, students were found to be significantly worse at translating two-step algebra story problems into expressions (e.g., 800 - 40x) than they were at translating two closely matched one-step problems (with answers 800 - y and 40x). The authors suggested that the students lacked a recursive grammar rule which would allow expressions (e.g., 40x) to be embedded within other expressions (e.g., 800 - 40x). Further, they called the situation where a two-operator problem is more difficult than two decomposed one-operator problems sequentially presented the *composition effect*. Similarly, research on computer science education and pedagogy has long argued that knowledge of a programming language can't be reduced to the sum of knowledge about different programming constructs, since there are many integrating patterns, schemas or plans that must be taught and practiced [Shneiderman, 1976, Adelson, 1981, Soloway and Ehrlich, 1984].

The above different constructs (e.g., chunks, schemas, plans, patterns, etc.), which have been identified across diverse domains, share the same essence: they refer to skills of integrating basic component skills that are essential to domain expertise, which I have referred to as *integration skills* in my dissertation. In the Knowledge-Learning-Instruction (KLI) framework [Koedinger et al., 2012a], a knowledge component (KC)[1] that "integrates or must be integrated (or connected) with other KCs to produce behavior" is called an *integrative knowledge component*, and "descriptions of integrative KCs make reference to internal mental states either in their condition (e.g., a deep feature produced by another KC) or in their response (e.g., a subgoal for another KC to achieve)." This is a formal and also more general definition of integration skills.

Despite being recognized in cognitive science research and teaching practices, skill integration has rarely been taken into account in learner modeling, the key component for adaptive assistance within an intelligent tutoring system (ITS). Traditional learner modeling approaches don't explicitly monitor the levels of knowledge that exist in the different types of skill integration [Corbett and Anderson, 1995, Mayo and Mitrovic, 2001, Conati et al., 2002]. For example, the most popular learner modeling approach, Bayesian knowledge tracing (BKT) [Corbett and Anderson, 1995], is based on decomposing domain knowledge into individual component skills, and monitors each individual component skill on decomposed steps. It doesn't address the possible skill integration that wholly underlies solving a multi-skill problem. Even some of the more advanced learner models [Mayo and Mitrovic, 2001, Conati et al., 2002, Millán and Pérez-De-La-Cruz, 2002, Xu and Mostow, 2012], which address the skill responsibility assignment in multi-skill problems in a more sophisticated way, still decompose domain knowledge into individual component skills, and don't explicitly monitor possible integration skills.

Specifically in the programming domain, where tasks intrinsically involve integrating multiple skills at the same time, only a few early attempts have been made to consider the integrative nature of problem-solving in programming [Brusilovsky, 1992, Weber, 1996a]. Moreover, learner modeling addressing integration skills by formal probabilistic approaches has barely been explored.

[1]In KLI framework [Koedinger et al., 2012a], a *knowledge component* is defined as an acquired unit of cognitive function or structure that can be inferred from students' performance on a set of related tasks. It generalizes multiple terms—production rule, schema, misconception, or facet, as well as everyday terms, such as concept, principle, fact, or skill—for describing pieces of cognition or knowledge.

Although ignoring explicit integration skill differentiation may make knowledge engineering and modeling easier, it may also result in a non-trivial negative impact on student learning. ITSs that ignore integration skill modeling risk giving early assertions of mastery after merely observing successes in decomposed component skill practices or practices in limited application contexts, which will lead to shallow, non-robust learning: students may still fail in problems applying the learned component skills in a new context, because they have learned the component skills in an overly specific way, encoded by surface features, or haven't encountered problems that reveal other aspects of the component skills; students may also make errors on learned component skills when they need to attend to other skills in an integration problem, because they haven't reached enough fluency in using the component skills, which would have been obtained by practicing the skills in varied contexts. Also, ITSs that ignore integration skill modeling risk giving ineffective remediation: when students who have mastered basic component skills fail in complex integration problems, such ITSs might not provide targeted remediation for learning (simpler) skill integration but instead keep supplying basic component skill practice, resulting in students "spinning their wheels."

My dissertation investigates the introduction of integration skills, long acknowledged by cognitive science research into learner modeling in ITS. I chose program comprehension with a complex integrative nature as a demonstration of integration skill modeling. Modeling integration skills is non-trivial and imposes several challenges, particularly in the context of programming. Firstly, although several empirical studies (mentioned above) have provided support for some integration skills in specific contexts, it is still not clear what factors decide the existence of an integration skill, and what the integration skills exactly are in a new context. In particular, there are still no unified, clear, fine-grained skill definitions (including those of integration skills) in programming, unlike the better-defined ones found in the math domain, although efforts have been made to identify CS standards on a coarse-grained, high level (e.g., CSTA K-12 Computer Science Standards[2], ISTE Standards[3], and the K-12 Computer Science framework[4]). Admittedly, less accurate, coarse-grained, or even "black box" skill models could be applied to build adaptation techniques, yet a clear, fine-

[2]https://dl.acm.org/citation.cfm?id=2593249
[3]https://www.iste.org/standards/for-computer-science-educators
[4]http://www.k12cs.org

grained understanding of domain skills and techniques built based on this foundation, would better enable the broader community, including researchers from different backgrounds and teachers, to interpret results, utilize the techniques in new contexts, and gain insights to better develop instructional policies and learning materials.

Secondly, machine learning methods for skill discovery/refinement [Barnes, 2005, Winters et al., 2005, Cen et al., 2006, Desmarais, 2012, Lan et al., 2014, González-Brenes, 2015] developed by the educational data mining (EDM) community have only shown effectiveness on datasets where each assessment unit (item) maps to one to three skills. Meanwhile, a program comprehension problem or an integration problem may involve an even larger number of skills mapping to one assessment unit. Recent work from [Van de Sande, 2016] has pointed out conditions[5] a multi-skill practice dataset has to meet in order to apply his method of plotting learning curves for items with multiple skills, which serves as an initial attempt in the community to understand the limitations brought by multi-skill practice datasets to distinguish between different skills with machine learning methods.

Thirdly, the improvement of an integration-level over a basic-level learner model might not be sufficiently measured by predictive performance, which is used as the only evaluation on learner models in most work in the EDM community, where new learner models are actively proposed. One possible reason (which is validated by my simulation study, see Section 7.3) is that integration practice involves multiple skills mapping to one assessment unit and a relatively large skill space, so models without integration skills could utilize the flexibility in the parameter space to reach a similar predictive performance. Several studies have pointed out the limitation of predictive performance evaluation. There may be a set of parameter estimates that make identical predictions where some are clearly incorrect about students' knowledge [Beck and Chang, 2007]. Also, learner models with a similar predictive performance might suggest that substantially different amounts of practice are necessary [Rollinson and Brunskill, 2015]. My prior work has also shown that a highly predictive

[5]Van de Sande in his work [Van de Sande, 2016] pointed out that if there is a skill that always appears along with another skill for several problems and all the students in the dataset solve nearly the same ordered sequence of problems, then there is no way to distinguish between the two skills for one or more practice opportunities, because this will result in a Hessian matrix that is not positive-definite and the matrix inversion will fail. He argued that this situation will rarely arise in practice, yet is common in many real-world programming datasets.

model can be useless for adaptive tutoring [González-Brenes and Huang, 2015], and may have implausible parameters [Huang et al., 2015]. Moreover, there is still little work in the EDM community that "closes the loop", i.e., completes the "4d cycle" of system design, deployment, data analysis and discovery, recycling back to design. To "close the loop", it would be necessary to examine whether a redesigned system based on a new learner model produces better student learning [Koedinger et al., 2013] compared with the original one.

1.2 MAIN DIRECTIONS OF WORK AND CONTRIBUTIONS

I proposed and applied the following approaches in my dissertation, to address the above challenges to modeling integration skills.

Since the nature of integration skills is still unclear (particularly in the programming domain) and automatic machine learning methods fall short in complex integration situations, I applied a Difficulty Factors Assessment (DFA) method [Koedinger and Nathan, 2004] borrowed from empirical cognitive task analysis methods [Clark et al., 2007] in cognitive science, to systematically generate and analyze data, and to study the existence and nature of integration skills and integration difficulties in common basic programming patterns.

The DFA methodology, combined with an automatic program text analysis, was used to build a skill model with integration skills. Based on the skill model, I constructed an integration-level learner model, CKM-HI, which incorporates integration skills in a hierarchical structure in a Bayesian network (BN). It learns parameters from student data, and gives dynamic, individualized knowledge estimates when deployed in a system. I chose BN among the many machine learning approaches in the area of artificial intelligence, because it naturally utilizes domain knowledge and maintains interpretable knowledge states for adaptation decisions, which provides an important practical value beyond predictive performance.

To address the limitations of prediction metrics in evaluating multi-skill learner models, I proposed a general, multifaceted evaluation framework for learner models that combines important analytic evaluations, such as predictive performance, parameter (model) plausibility and expected instructional effectiveness, with real-world intervention study evaluations.

I conducted extensive data-driven analytical evaluations based on a real-world dataset, comparing the proposed learner model, CKM-HI, with two popular multi-skill practice learner models that don't incorporate integration skills, CKM and WKT, over multiple aspects of the proposed evaluation framework. I also conducted a simulation study to better understand model behaviors observed in the real-world dataset.

To evaluate the real-world impact of the proposed learner model, I built an ITS, Trace Table Tutor (T3), that teaches students program comprehension skills by using trace tables. This tutor provides hints and adaptive problem selection, driven by underlying learner modeling. I conducted a classroom study comparing a version of the ITS driven by an integration-level learner model to another version driven by a basic-level learner model. The combination of analytic and empirical study evaluations gives a comprehensive examination of the proposed integration-level learner model.

To sum up, my dissertation makes five main contributions, listed as follows:

- Firstly, it is a pioneer work that systematically demonstrates the value of integration skill modeling, and offers a novel, general integration-level learner modeling approach which should be applicable to a broader context beyond program comprehension.
- Secondly, it contributes to both EDM and ITS research a unified, multifaceted evaluation framework for learner models, which includes both data-driven analytical evaluations and empirical user study evaluations, and demonstrates applying this framework to the evaluation of the proposed integration-level learner model.
- Thirdly, it is the first work to apply a DFA approach which originated in cognitive science research to computer science education research. The DFA approach provides a way to systematically generate and analyze data for investigating difficulty factors, and my work is the first to apply it to studying integration skills in program comprehension.
- Fourthly, it contributes to bringing recent ITS infrastructure and techniques into computer science education.
- Last but not least, it contributes an example to EDM and ITS research of utilizing cognitive science discoveries, and taking an interdisciplinary approach (combining empirical methods from cognitive science and analytic methods from computer science) to tackling a topic which is ultimately aimed at the practical value of improving student learning.

1.3 RESEARCH QUESTIONS

My dissertation addresses five main research questions. The motivations have been explained in detail in the previous Section 1.1 while in this section I briefly explain why I have asked each research question and how the research questions interconnect.

[**RQ 1**] **Does the composition effect exist in program comprehension, and if so, what is the nature of it?** This question sets the foundation for building a skill model for an integration-level learner model (RQ 2), and provides insights for building a program comprehension ITS for evaluating the integration-level learner model (RQ 5). The existence of a composition effect indicates the existence of an integration skill, and with deeper investigation into the errors made in integration, one should be able to reliably confirm the existence of integration skills, and abstract, generalizable integration difficulty factors for identifying integration skills in new contexts. Thus, this question is the first question investigated in my dissertation.

[**RQ 2**] **How to build a learner model for integration skills?** This question is the core question in my dissertation. As defined in my dissertation, learner modeling includes constructing a *skill model* (specifying the skill set, item-to-skill relationships and skill-to-skill relationships), and constructing a mechanism to infer knowledge from performance, based on a *modeling approach* (e.g., Bayesian networks, logistic regression). To construct a skill model, automation and generalization, based on the DFA discoveries from RQ 1, should be taken into consideration. To construct an inference mechanism based on the skill model, I have chosen Bayesian networks. Finding a proper way to represent different kinds of skills and relationships, based on discoveries from the RQ 1 studies, as well as how to enable effective adaptive tutoring in this context, should be taken into consideration. Thus, this question also needs to be carefully investigated.

[**RQ 3**] **How to evaluate a learner model?** This question aims to address the challenge of evaluating multi-skill practice learner models in general, and is necessary to reveal the value of my proposed integration-level learner modeling. The answer to this question is a unified, multifaceted evaluation framework for learner models that is applicable to both single-skill and multi-skill learner models, with a range of dimensions and metrics that is well-informed by prior work, including both analytical data-driven evaluations on collected datasets and empirical user study evaluations. This framework is necessary for later applications, when answering RQ 4 and RQ 5.

[RQ 4] **Is learner modeling for integration skills beneficial in terms of multi-faceted data-driven evaluations?** The data-driven evaluations under this question apply the evaluation framework proposed in RQ 3 and serve as one major part of the evaluations of the proposed learner model from RQ 2. The experimental setup—such as baseline models, density levels of assessment units and learner model initialization—needs to be considered.

[RQ 5] **Is learner modeling for integration skills in an ITS beneficial in terms of improving student learning?** This question relates to the ultimate goal of learner modeling, improving student learning, and completes the evaluation of the proposed learner model (RQ 2) under the proposed evaluation framework (RQ 3).

1.4 DISSERTATION ORGANIZATION

The chapters in my dissertation are organized as follows: Each chapter (except Chapter 2 and 9) focuses on one research question, and introduces the approach for answering the research question.

Chapter 2 gives a thorough literature review on the background and related work of my dissertation, including integration skills in expertise, learner modeling in ITS, Difficulty Factors Assessment (DFA) for skill modeling and learner model evaluation.

Chapter 3 reports my investigation into RQ 1. Classroom Study 1 was conducted. I applied the DFA approach with systematically designed problems and analyzed students' performance to examine the existence of the composition effect. I further conducted a drill-down integration errors analysis and a correlation analysis to identify integration skills and generalizable integration difficulty factors.

Chapter 4 explains how I constructed a learner model for integration skills for answering RQ 2. To construct the skill model, I proposed an automatic program text analysis algorithm based on a simple set of rules with contextual features, informed by the identified integration skills and generalizable integration difficulty factors from Chapter 3. To construct the mechanism for inferring knowledge from performance, I used the Bayesian network technique and built an integration-level learner model, CKM-HI, which has integration skills incorporated in a hierarchical structure and is based on the constructed skill model.

Chapter 5 explains my multifaceted evaluation framework to address RQ 3. Based on various important aspects identified in prior work, I proposed a multifaceted, general evaluation framework with a traditional analytical dimension (predictive performance) and new analytical dimensions (parameter plausibility and expected instructional effectiveness) instantiated by novel metrics, as well as real-world intervention study evaluations.

Chapter 6 explains the design and implementation of a novel program comprehension ITS, the Trace Table Tutor (T3), as the basis for my investigation into RQ 5. I designed trace table practice problems which naturally address the integration difficulties discovered from the composition effect study in Chapter 3. I designed an automatic process to generate hints and a problem selection algorithm, both of which are driven by underlying learner modeling. An existing ITS infrastructure was utilized to implement T3.

Chapter 7 reports the experiments of data-driven evaluations for the proposed learner model, answering RQ 4. An analytical study with cross-validation was conducted. I conducted experiments on a real-world dataset comparing the proposed learner model, CKM-HI, with two popular multi-skill learner models, CKM and WKT, that don't incorporate integration skills. I considered a range of aspects, which were proposed in my evaluation framework. Further, I conducted a simulation study to better understand the model comparison results obtained from the real-world dataset.

Chapter 8 reports the real-world intervention study evaluation of the proposed learner model, answering RQ 5. Classroom Study 2 was conducted, comparing two versions of T3: one version was driven by CKM-HI and the other version driven by CKM. I analyzed the results from several aspects, including posttest scores, time, scores under time constraints and composition effects. This analysis, combined with the data-driven analytical evaluations, gives a comprehensive evaluation of the proposed integration-level learner model, CKM-HI.

Chapter 9 summarizes conclusions, limitations and future work, as well as the contributions of my dissertation.

2.0 BACKGROUND AND RELATED WORK

In this chapter, I review the background and related work of my dissertation from following aspects: integration skills in expertise, learner modeling in ITSs in various domains including the programming domain, Difficulty Factors Assessment for skill modeling, and learner model evaluation.

2.1 INTEGRATION SKILLS IN EXPERTISE

It has been well-established in cognitive science research that experts organize and store knowledge by *chunks* or *schemas* while novices can't. A *chunk* is defined as a familiar collection of more elementary units that have been inter-associated and stored in memory repeatedly and act as a coherent, integrated group when retrieved [Tulving and Craik, 2000]; similarly, a *schema* is defined as mental constructs that allow patterns or configurations to be recognized as belonging to a previously learned category and which specify what moves are appropriate for that category [Sweller and Cooper, 1985]. In the context of my dissertation, I will use these two concepts interchangeably. Experts organize individual pieces of information into larger, meaningful, functional units as chunks and schemas, that allow them to retrieve and apply related knowledge with facility and autonomy. The early evidence of chunks comes from the chess domain [Chase and Simon, 1973, De Groot, 1978] where chess masters were found to recall more positions than novices did. Latter experiments across different domains including electronic circuitry [Egan and Schwartz, 1979], physics [Larkin et al., 1980], programming [Soloway and Ehrlich, 1984], geometry [Koedinger and Anderson, 1990] provided further evidence of chunks in defining expertise.

Particularly, researchers in the area of psychology of programming have long argued that *programming schemas*, *plans* or *patterns* form an important part of programming expertise [Shneiderman, 1976, Adelson, 1981, Soloway and Ehrlich, 1984]. For example, [Soloway and Ehrlich, 1984] compared the performance between experts and novices on program comprehension tasks involving various programming plans, and found out that experts have and use programming plans, which are generic program fragments that represent stereotypic action sequences in programming (e.g., *maximum search loop plan*, *running total variable plan*). Recent multi-institutional studies on program comprehension [Whalley et al., 2006, Lister et al., 2006] found out that experts or high level students presented more *relational responses* in theirs answers compared to low level students. They defined a *relational response* as a response where the student integrates the parts of the problem into a coherent structure and uses that structure to solve the task (e.g., inferring that a code counts the number of common elements in the two arrays and calculating answers without step-by-step tracing). In addition, a recent study on program construction [de Raadt, 2008] also discovered that experts applied algorithmic plans while novices didn't, which is consistent with the conclusions drawn from program comprehension studies. Further, programming patterns were used by researchers in the area of ITS to support intelligent analysis of student programs, such as PROUST [Johnson and Soloway, 1985], TALUS [Murray, 1985], ELM-PE [Weber, 1996b], INCOM [Le and Menzel, 2009] and Haskell [Gerdes et al., 2012].

What are the potential difficulties in the acquisition of schemas? I summarized research in cognitive science from two perspectives to explain the difficulties. One potential difficulty is that the acquisition of schemas requires a correct conceptual understanding of how elementary units are integrated together, which could be missed or implicit in the original learning process. For example, in an algebra symbolization study [Heffernan and Koedinger, 1997], students were found to be significantly worse at translating two-step algebra story problems into expressions (e.g., 800 - 40x) than they were at translating two closely matched one-step problems (with answers 800 - y and 40x). This phenomenon where a problem is harder than the parts made it up together was termed as the *composition effect*. Students might have viewed an expression as a recipe rather than a first-class object [Sfard and Linchevski, 1994], and thought that only single numerals or variables (e.g., 40 or x) could be combined with mathematical operators. The researchers further hypothesized that students lacked a hidden symbolic production skill (or recursive grammar rule) indicating expressions (e.g., 40x) can be embedded within other expressions (e.g., 800 - 40x). This work has an important implication that additional skills may arise supplementing or adjusting the conceptual understanding of previously learned skills when they are applied together in an unseen situation, and actually, such additional skills may only be manifested when its component skills are integrated in specific ways. In the programming domain, it is still unclear and thus it is one of my goals to investigate whether some difficulties in programming results from a lack of correct conceptual understanding in how basic constructs are integrated together.

Another potential difficulty in the acquisition of schemas is that novices are still very unfamiliar to the task and they have to use demanding process (e.g., means-ends analysis [Sweller, 1988]) with high *cognitive load*, which increases the chance of failure and also deprives them of cognitive resources to construct schemas [Sweller and Cooper, 1985]. Here, *cognitive load* refers to the effort being used in the working memory. According to [Miller, 1956], working-memory capacity has inherent limits, so the demands imposed by a complex task applying multiple skills simultaneously can easily exceed what a novice can manage [Kahneman, 1973]. However, experts could recognize and invoke schemas stored in the long-term memory to be processed in the working memory as a single unit, with a manageable cognitive load.

Based on research in cognitive science and learning science, [Ambrose et al., 2010] has proposed a principle for developing mastery: students must acquire component skills, *practice integrating skills*, and know when to apply skills. Particularly, the second level they stated is the focus of my dissertation. In their Knowledge-Learning-Instruction framework, [Koedinger et al., 2012b] also pointed out the integrative aspect of some knowledge components, and defined a knowledge component that "integrates or must be integrated (or connected) with other KCs to produce behavior" as an *integrative knowledge component*, and "descriptions of integrative KCs make reference to internal mental states either in their condition (e.g., a deep feature produced by another KC) or in their response (e.g., a subgoal for another KC to achieve)." This is a formal and also more general definition of integration skills, because a knowledge component is a general term which could refer to a concept, a principle or a fact, but I primarily focus on skills in my dissertation. Borrowing the wording from these two work, I used the term *integration skills* to refer to any of the constructs mentioned in my literature review (chunks, schemas, plans, patterns, etc.) that describe skills of integration basic component skills which are essential to domain expertise, and allows for fluency in solving domain tasks.

Based on the numerous research on the existence and difficulty of acquiring integration skills, is there further evidence showing the effectiveness of teaching integration skills, so that it's worth the effort to modify instructions or build tutoring systems to address them? Prior work has provided some successful examples. To address the conceptual difficulty in integration, [Koedinger and McLaughlin, 2010] designed symbolic substitution problems in the ASSISTMENT system [Heffernan and Heffernan, 2014] for exercising two operator expressions (and thus should exercise the hidden integration skill, the recursive grammar rules) without the cover story, and students under this condition significantly improved their performance compared with the control group given simpler story problems. To address the cognitive load difficulty in integration, [Sweller and Cooper, 1985] demonstrated in a variety of quantitative fields from statistics to physics that *worked-examples* could free up cognitive resources that allowed students to notice the key features to construct schemas and enabled students to improve their performance on subsequent problem-solving.

In the programming domain, some preliminary studies have provided initial evidence of the effectiveness of teaching integration skills. When explicit instruction of programming patterns was incorporated into an actual introductory programming curriculum, [Proulx, 2000] qualitatively reported that students performed better on the midterm exam and seemed more confident than in the past; [Muller et al., 2007, de Raadt, 2008] used comparative research studies to show that novices who studied under the revised course exhibited better problem-solving competence than those who studied in a traditional manner. In the area of ITS, an example is the dialogue-based intelligent tutoring system ProPI [Lane and VanLehn, 2005] which elicited goal decompositions and program plans from students in natural language. Their small-scale evaluation showed that students who received tutoring from ProPl seemed to have developed an improved ability in problem composition and displayed behaviors that suggested they were able to think at greater levels of abstraction than the controlled group.

2.2 LEARNER MODELING IN ITSS

2.2.1 Popular Learner Models in ITSs

Intelligent tutoring systems (ITSs) strive to select educational activities and deliver individual feedback that is most relevant to the learner's level of knowledge, and a *learner model* which represents mostly the learner's knowledge of the subject domain is essential for ITSs to provide the adaptation effect, i.e., to behave differently for different learners [Brusilovsky and Millán, 2007].

The current most popular knowledge model mechanism in ITSs is to use an *overlay model* based on a *domain model* (*cognitive model*, *skill model*), which reflects the expert-level knowledge of the subject domain. For each fragment of domain knowledge (concept, skill, production rule, or in a general term knowledge component), an overlay model stores some estimation of the learner's knowledge level of this fragment [Brusilovsky and Millán, 2007]. Particularly, during the interaction with an ITS, learners' knowledge change dynamically, and the process of maintaining explicit knowledge estimations of domain skills over time, is also called *knowledge tracing* [Corbett and Anderson, 1995].

In the context of my dissertation, *learning modeling* includes both constructing a mechanism based on a *modeling approach* (e.g., Bayesian networks, logistic regression), and constructing a *skill model* (specifying the skills in the domain, skill-to-problem relationships and skill-to-skill relationships), and often these two aspects affect each other.

Bayesian networks (BN) [Pearl, 2014] has been one of the major probabilistic approaches to learning modeling, and it is the approach I applied in my dissertation. The main advantages of Bayesian networks are that they can encode domain expert knowledge in an interpretable, intuitive graphical representation (which are good for cognitive diagnosis), and offer a well-defined formalism for sound probability computations of unobservables from the evidence of observables [Desmarais and Baker, 2012].

In this section, I mainly review popular learner models in ITSs that apply Bayesian networks, and briefly mention others in the end. Also, for models used in the programming domain, I give more details in Section 2.2.2.

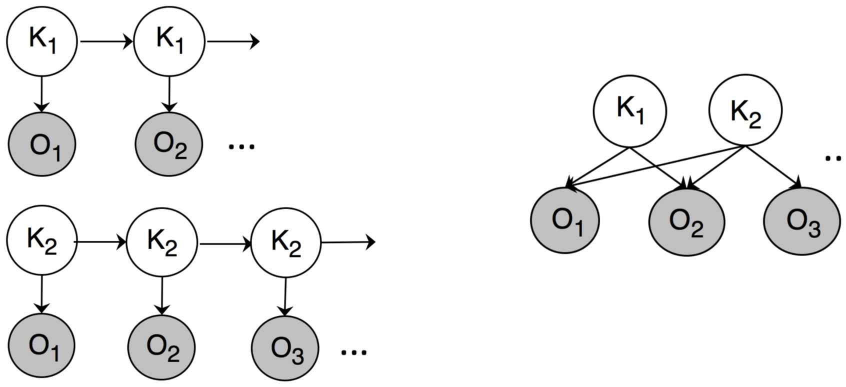
One of the most widely used learner model for single-skill practice situation is the classic *Bayesian knowledge tracing* (BKT) [Corbett and Anderson, 1995] based on the ACT-R theory [Anderson et al., 2004]. Here, each observed assessment unit (i.e., a practice opportunity, an observation, a step, a problem or in a general term an item) only requires a single skill. One major assumption made here is that skills are independent of each other. A hidden Markov model (HMM), one kind of the dynamic Bayesian networks, is used to model the learning process of each skill with four parameters: the probability of initially knowing the skill (*init*), the probability of transferring from an unlearned to a learned state (*learn*), the probability of accidentally failing a known item (*slip*), and the probability of correctly answering an item by chance (*guess*). BKT has been used in Cognitive Tutors for programming and maths learning and proved to be educationally successful [Ritter et al., 2007].

However, BKT has been designed for modeling basic component skills in decomposed practice opportunities but not for complex tasks where multiple skills or integration skills are required. In the latter situation, each observed assessment unit involving multiple skills poses substantial challenges in assigning responsibility (credit or blame) to each individual skill. Currently, there are two main streams of work that address this problem.

The first stream of work [Gong et al., 2010, Koedinger et al., 2011, Xu and Mostow, 2012, González-Brenes et al., 2014] converts the many-to-many skill-to-item mapping into a one-to-many (one-to-one) mapping during the model training process, where the classic BKT can be applied[1] (referred to as the one-to-many BN paradigm). Such models assume that skills are independent of each other and each skill is fully responsible for the performance (during training process). This is achieved by duplicating each observation for each required skill to form an individual skill practice sequence, and train a hidden Markov model for it. Variants within this paradigm differ in how they conduct prediction and updating during the predicting phrase. One variant I consider in my dissertation as a low baseline is called *weakest knowledge tracing* (WKT) (Figure 14a), which has been shown to have the best predictive performance on several datasets as compared with other variants [Gong et al., 2010, González-Brenes et al., 2014, Xu and Mostow, 2012]. It takes the minimum of the

[1]Note that recent work [Xu and Mostow, 2012, González-Brenes et al., 2014] still conducts single-skill practice knowledge tracing on coarse-grained skill levels and treats multiple fined-grained subskills as features.

predicted probability of success among involved skills as the final prediction; it only updates the knowledge of the weakest skill when observing an incorrect response, and updates all skills otherwise. Overall, models from this paradigm reduce modeling complexity, but simplifies the responsibility assignment issue. Existed applications have been focusing on modeling basic component skills and haven't considered modeling integration skills.



(a) WKT **(b) CKM**

Figure 1: Two main learner models for multi-skill practice situations. O nodes represent the observed binary student performance and K nodes represent binary skill knowledge levels.

The second stream of work [Conati et al., 2002, Mayo and Mitrovic, 2001, Millán and Pérez-De-La-Cruz, 2002] maintains the many-to-many skill-to-item mapping in both the training and predicting phrases (referred to as the many-to-many BN paradigm). In such models, skills are dependent conditioned on items, and could be dependent on other skills if relations among skills are considered in the structure. Each individual skill is assigned responsibility according to the conditional probability table and the Bayesian rule. Here, I focus on the models that assume a conjunctive relationship among skills (i.e., success in an item requires knowing all required skills) and that use noisy-AND gates for modeling the conjunctive relations. Noisy-AND gates were commonly used in many prior studies [Carmona et al., 2005, Conati et al., 2002, VanLehn et al., 1998], due to their linear rather than exponential complexity in inference. I call such a model that uses item-level noisy-AND gates with a flat structure among skills *conjunctive knowledge modeling* (CKM) and use it as a high baseline (Figure 14b). These models also closely relate to the popular psychometric model DINA [Junker and Sijtsma, 2001]. Each noisy-AND gate uses a *slip* parameter to capture the probability of accidentally failing a known item, and a *guess* parameter to capture the probability of correctly answering an item by chance. Further, in this avenue of work, some use a hierarchical structure among skills, focusing on either the prerequisite relations [Carmona et al., 2005, Conati et al., 2002, Käser et al., 2014] or granularity relationships (including competency-based networks) [Collins et al., 1996, Conati et al., 2002, Millán and Pérez-De-La-Cruz, 2002, Mislevy and Gitomer, 1995, Morales et al., 2006]. However, they are substantially different from the integration relationship that I model and the level of remediation that I target. In addition, most work doesn't model transition probabilities across time steps, due to the complexity imposed by the skill model in an arbitrary practice order, yet the online knowledge update could still be easily achieved by applying the Bayesian rule based on incoming evidence.

Besides the above Bayesian network based approach, logistic regression based approaches such as some models from the *Item Response Theory* [Embretson and Reise, 2013], Additive Factor Model [Cen et al., 2006], and Performance Factor Analysis [Pavlik et al., 2009] have also been used for assess learner ability (knowledge) and conduct adaptation [Chen et al., 2005] in ITSs. Under this paradigm, existed work except a recent one [Koedinger and McLaughlin, 2016] have also been focusing on modeling basic component skills and haven't considered integration skills. While it has been shown that some of these models could achieve equivalent or better predictive performance compared with the one-to-many BN paradigm in multi-skill practice situations [Gong et al., 2010], the logistic regression formulation intrinsically requires independence among skills to avoid multicollinearity in order to get reliable parameter estimates (which is not the case in the many-to-many BN paradigm). Moreover, efficient online estimation of student ability parameters (to be used as dynamic knowledge estimation) is still an active research [Weng et al., 2018, Ekanadham and Karklin, 2017], while the online update of knowledge could be easily done by applying the Bayesian rule in the Bayesian network approach. Giving the predictive advantages of logistic regression models, it will be an interesting future step to explore integration skill modeling under this approach.

2.2.2 Learner Models in the Programming Domain

Learner modeling in the programming domain is challenging since program comprehension or construction tasks usually require a wide range of knowledge and skills, and typically requires applying multiple skills simultaneously (multi-skill practice). It is still not yet clear what is the best representation for the skill model in the context of learning modeling for programming. In this section, I classify prior work by skill models, starting from approaches modeling basic component skills (e.g., using basic programming constructs), followed by approaches that explicitly model integration skills (e.g., applying programming patterns).

One of the early work is the classic BKT [Corbett and Anderson, 1995] introduced before (Section 2.2.1). It was deployed in ACT Programming Tutor for teaching students to write short Lisp, Prolog or Pascal programs. The procedural knowledge involved in the tasks here was represented by goal-specific production rules defined on fine-grained elementary skill level (e.g., if the goal is to move the last element to the front of a list, then code *reverse*, and set a goal to code the list), and it is the base for the underlying process called *model tracing*, where students' actions were compared with an *ideal student model*. BKT was used in the knowledge tracing process on such production rules to maintain the probabilities that the student has learned each of the rules. One important feature of the system is that the model tracing process made sure students maintained in a recognized solution path, so it was possible and suitable to use a single-skill practice knowledge tracing model, which is usually not the case in other programming systems.

Another early work is the *Constraint-Based Modeling* (CBM) [Ohlsson, 1994], with its major successful application in SQL-Tutor [Mitrovic, 2012] for teaching students to write SQL queries. CBM focuses on domain principles that correct solutions follow, and encodes domain knowledge into constraints describing fine-grained features of correct solutions (e.g., if the student's solution contains the JOIN keyword in the FROM clause, then the ON keyword must also appear in the same clause). This has been proved to be effective for many ill-defined domains. Originally, a simple heuristic overlay model (e.g., computing ratio of the correctness of each constraint) was used, and later a Bayesian network model was constructed and proved effective [Mayo and Mitrovic, 2000]. Although an attempt has been made in J-

LATTE [Holland et al., 2009] to apply CBM in Java (with 11 problems and 89 constraints), it is still unclear the knowledge engineering effort required to construct a sufficient set of constraints for the desired topic coverage for learning to program in imperative languages (e.g., Java, Python) compared with relatively simpler declarative languages (e.g., SQL).

A considerable amount of recent work in learner modeling in programming has used the simplified representation of domain knowledge by elementary programming constructs (concepts), such as *local variable*, *for statement*, or higher level constructs (concepts or topics), such as *variable*, *loop statement*, sometimes organized in a granularity hierarchy with different abstraction levels (which was called a domain *ontology*). Most of such work relied on expert manual annotation, while some recent work used automatic extraction by a parser [Hosseini and Brusilovsky, 2013] or an Abstract Syntax Tree library [Rivers et al., 2016]. Based on such a simplified skill model, different modeling approaches have been used: [Vesin et al., 2012] in their Protus 2.0 system for learning Java used a heuristic rule-based approach; some of my prior work [Huang et al., 2014, González-Brenes et al., 2014] and other people's work [Zapata-Rivera and Greer, 2001, Wang et al., 2017] applied Bayesian networks in Java or C programming, with [Zapata-Rivera and Greer, 2001] further modeled the prerequisite structure in the network; [Yudelson et al., 2014, Berges and Hubwieser, 2015] applied logistic regression based models (e.g., Rasch model, AFM) in Java programming.

On the other hand, a few researchers have explored learner modeling on the level of integration skills (patterns). One of the earliest work was by [Brusilovsky, 1992], where multiple programming concept pairs were connected by "usage" link in the domain model. A usage link indicated that one concept could be used in the context of another. For example, a specific kind of condition could be connected by one usage link with a loop and by another with a conditional operator. All problems in the system were indexed with usage links (i.e., concept pairs rather than individual concepts) to distinguish different applications of the same concept.

Another early work was by [Weber, 1996a] which demonstrated a complex approach of modeling patterns in the *episodic learner model (ELM)*: Firstly, a database was constructed by experts including task descriptions (with algorithmic plans), concept frames (with related rules for solving plans) and rule frames. Then, a cognitive diagnosis was conducted to generate a learner's code by applying rules resulting in a derivation tree, from which *episodic frames* was extracted, generalized and inserted into a student's learner model consisting of a hierarchy of episodic frames, their abstracted versions, and their associated concepts. Each episodic frame or its abstraction was actually an algorithmic pattern (e.g., checking whether the list contains a NIL value). When a new solution was presented, the stored model would be used to shorten the diagnosis process by giving higher priority to the patterns the learner used before. ELM with such pattern-level information has been proved effective for the adaptive recommendation of programming examples in ELM-PE [Weber, 1996b] and ELM-ART [Brusilovsky et al., 1996, Weber and Brusilovsky, 2001] systems.

However, the episodic learner model has never been expanded or ported to other systems or languages, due to its high demand for knowledge engineering. Instead, recent work has used simpler knowledge representations: [Kumar, 2006] enhanced a concept map with learning objectives and organized them in a hierarchical way, and the mastery of a domain concept requires mastering each pedagogic concepts (e.g., to understand *variable*, one needs to understand *variable declaration*, *variable assignment* and *variable referencing*); [Mathews, 2006] grouped elementary SQL problems that share a common SQL query pattern into problem templates, and conducted a small-scale preliminary study showing students using template-based ITS achieved high levels of learning within short periods of time; [Chrysafiadi and Virvou, 2013] constructed patterns (such as *count in a for loop*, *sum in a for loop*) as knowledge components and further constructed prerequisite relations among them to propagate performance. However, in terms of modeling approach, they have all employed heuristics (e.g., checking the number of correct attempts, giving heuristic thresholds in a fuzzy logic approach) rather than formal probabilistic approaches. It could be an interesting next step to compare my work based on probabilistic approaches with them. An exception is [Kasurinen and Nikula, 2009] which applied BKT to model students' knowledge levels in Python program construction. However, the patterns they modeled concerned the design aspect of programming (e.g., when to apply a for loop vs. a while loop) rather than the integration aspect (e.g., how to use a for loop with an addition assignment to sum over a sequence of numbers) that I focus in my current work.

Section 2.1 reviewed different representations for integration skills mainly from cognitive science research, and Section 2.2 reviewed learner models with their skill models in ITSs including the programming domain. Compared with prior work, my dissertation work offers a unique approach that possesses all of following characteristics: 1) a right complexity level of knowledge representation, which enables (semi-)automatic methods to construct the skill model, but doesn't reduce to oversimplified elementary skills since it also includes integration skills, 2) high cognitive fidelity, which is supported by the DFA and multifaceted evaluations, and 3) high robustness and generalizability, due to applying a modern probabilistic approach, Bayesian network which has a sound formalism to reason under uncertainty.

2.3 DIFFICULTY FACTORS ASSESSMENT FOR SKILL MODELING

Cognitive Task Analysis (CTA), a well-established methodology in cognitive science [Clark et al., 2007], has been applied as a way to uncover the explicit and implicit knowledge (skills) that experts use to perform complex tasks, and help to design or redesign instructions for novices [Velmahos et al., 2004, Feldon et al., 2010]. It uses qualitative research methods such as interviews, think-alouds, observations, which could be expensive and subjective.

The *Difficulty Factors Assessment* (DFA) approach has been proposed as a new way to conduct cognitive task analysis quantitatively for discovering hidden skills. It is a methodology of conducting studies with problems generated systematically by crossing factors expected to influence the degree of problem difficulty [Koedinger and Nathan, 2004]. Since such DFA data can be collected by paper form tests or tutoring systems, it can be more economical and has the potential to scale up which enables more objectivity, compared with traditional CTA. DFA doesn't focus on enumerating all of the errors any students make [VanLehn, 1990], but focuses on identifying factors that present the greatest difficulties for students. An example of using the DFA is the work from [Koedinger and Nathan, 2004] which found that students were more successful in solving simple algebra story problems than in solving mathematically equivalent equations, contrary to beliefs held by practitioners and researchers in mathematics education. In the same stream of work, [Heffernan and Koedinger, 1997] utilized DFA and discovered that the symbolic production skill (i.e., recursive grammar rule) is a critical element of student competence. Based on this discovery, symbolic substitution exercises were provided in a redesigned tutor which led to improved learning compared with the original one giving simpler story problems [Koedinger and McLaughlin, 2010]. [Baker et al., 2007] further expand the idea of DFA into a *difficulty factor approach* to design intelligent tutors. Albeit the effectiveness of DFA, to the best of my knowledge, there were still no attempts to use it in the programming domain for understanding the difficulty factors and nature in programming skills, which is one of the innovation in my dissertation.

In addition, it's worth mentioning that DFA could prove cognitive foundations and enhance interpretability for automated skill discovery or refinement methods actively explored in the ITS and EDM community. Learning Factor Analysis [Cen et al., 2006] is one example that takes the input of expert identified difficulty factors into a search algorithm and generates more predictive cognitive models.

2.4 LEARNER MODEL EVALUATION

In the field of *educational data mining* (EDM) particularly in the topic of learner modeling, data-driven assessment using predictive performance in a cross-validation setting has become the gold standard [Corbett and Anderson, 1995, Pavlik et al., 2009, Pardos and Heffernan, 2010a, Yudelson et al., 2013, González-Brenes et al., 2014, Khajah et al., 2014]. [Pelánek, 2015] gave a comprehensive comparison among different prediction metrics. However, prior work has expressed concerns about using prediction performance as the only evaluation approach. [Beck and Chang, 2007] pointed out that there could be a family of parameter estimates that make identical predictions where some are clearly incorrect about students' latent knowledge (i.e., the *identifiablity* problem). [Yudelson and Koedinger, 2013] suggested that even small differences in RMSE can have a significant impact on student over-practice and under-practice. [Rollinson and Brunskill, 2015] proposed a predictive similarity policy, and found out that learner models with similar predictive accuracies can suggest that substantially different amounts of practice are necessary. My prior work has also shown that a highly predictive model can be useless for adaptive tutoring [González-Brenes and Huang, 2015], and can have implausible parameters [Huang et al., 2015] reducing the reliability of the latent knowledge inference.

In particular, [Baker et al., 2008] has termed the problem where parameter values violate the learner model's conceptual meaning (such as a student being more likely to get a correct answer if he/she does not know a skill than if he/she does) as *Model Degeneracy*. Such parameters are not plausible and reduce the reliability and interpretability of the knowledge estimations. Several studies have taken parameter plausibility into consideration to evaluate learner models. Some used external measurements, such as pretest scores [Gong et al., 2010], exercise scores [Corbett and Anderson, 1995], or some domain-specific measurements [Beck and Chang, 2007]. However, such external resources are not always available. Some examined plausibility by internal validity. For example, [Baker et al., 2008] proposed 0.5 as the theoretical degeneration threshold for BKT *guess* and *slip* parameters, and considered two metrics for judging empirical degeneracy. However, their metrics relied on some heuristically selected values. One innovation of my dissertation work is that it provides internal validity metrics which don't rely on heuristic values for a family of learner models.

Another dimension that has been considered is a learner model's impact on adaptive tutoring or instructional decisions measured by the number of practices to reach inferred mastery [Gong et al., 2010, Yudelson and Koedinger, 2013, Lee and Brunskill, 2012]. However, it is hard to make grounded claims about the actual number of practices to reach mastery since real mastery can't be decided from the data. Simulation experiments provided ground truth for the moments of learning [Lee and Brunskill, 2012, Pardos and Yudelson, 2013, Pelánek and Řihák, 2017], yet they hardly reflect real-world learning. Evaluation independent of simulation, and with better validation of mastery assertion are needed. One of my co-authored work [González-Brenes and Huang, 2015], *learner effort-outcomes paradigm* (LEOPARD) was the first to address this. It introduced the expected outcome dimension when evaluating the expected effort (number of practices to reach inferred mastery). However, this framework is limited to single-skill practice situations and a single mastery threshold. My dissertation work extends this framework to a more general setting where multi-skill practice learner models under a range of mastery thresholds can be evaluated.

In addition, learning curves have been widely used to examine and refine skill models. They are plotted based on student performance data [Corbett and Anderson, 1995] and can be further inspected by parameters from fitted power law function [Martin et al., 2011], or fitted Additive Factors Models [Koedinger et al., 2012c]. Flat or jagged curves indicate potential problems in the skill model. However, this evaluation method is only readily applicable to single-skill practice situations, and only one recent work has made an initial attempt to generalize it to multi-skill practice situation [Van de Sande, 2016].

Meanwhile, in the field of ITS, conducting user studies deploying learner models into real systems has long been the standard [Anderson et al., 1995, Mitrovic et al., 2001, Vanlehn et al., 2005], since the ultimate goal of is to improve students' learning outcomes, and also data-driven evaluation is only reliable when the data is sufficient for running predictive models. However, the field of EDM which actively introduces new learner modeling (including skill modeling) approaches, mostly only conducted data-driven evaluations, and there are only very few *close-the-loop* studies (from the same research group) ([Cen et al., 2007, Koedinger and McLaughlin, 2010, Koedinger et al., 2013, Liu and Koedinger, 2017]) that actually redesigned the ITS according to data-driven discovered learner (skill) models, and examined whether the new instructions could lead to higher student outcomes, compared with the original system in a controlled experimental study.

A caution is that such user studies of on the system level typically jointly assess both learner modeling and adaptation decision making. The other extreme of only using such a user study evaluation doesn't suffice to evaluate the learner model either. This has been pointed out by the proponents of *layered evaluation* argued that holistic evaluation should be complemented by approaches that independently assess each layer [Brusilovsky et al., 2004a, Paramythis et al., 2010].

In my dissertation work, I developed a multifaceted evaluation framework for learner models that 1) combines and extends existed data-driven evaluation dimensions, and 2) includes close-the-loop user study evaluation. I demonstrated how I used this framework to conduct a comprehensive evaluation of my proposed learner model.

3.0 INVESTIGATING COMPOSITION EFFECTS AND INTEGRATION SKILLS IN PROGRAM COMPREHENSION

In this chapter, I describe a classroom study (Classroom Study 1) investigating composition effects and integration skills in program comprehension, for novice programmers. This chapter contributes to RQ 1, *Does the composition effect exist in program comprehension, and if so what is the nature of it?* Here, the *composition effect* is defined as the phenomenon where a problem requiring the integration of multiple component skills is more difficult than a problem made up of decomposed parts, each of which requires only one component skill.

As opposed to prior work, which has investigated the overall comprehension skills of novice programmers [Lister et al., 2004, Whalley et al., 2006], my work is the first to deploy a Difficulty Factors Assessment (DFA) approach to give an in-depth, systematic investigation of integration skills in program comprehension of common basic programming patterns. My studies aim to:

1. examine the existence of the composition effect in the common basic programming patterns for novice programmers;
2. reveal the nature of the composition effect and conceptualize the integration skills that explain the composition effect; and
3. investigate individual differences in integration skills and the generalizability of the composition effect.

The DFA approach applied here, combined with the automated program text analysis introduced in Section 4.1, provides the approach for building a skill model with integration skills. Also, analyses from this chapter shed light on the building of instructional materials (e.g., trace tables) in Chapter 6 for evaluating the proposed learner model.

3.1 METHOD

3.1.1 Overall Idea

The basic idea of the Difficulty Factors Assessment (DFA) approach [Koedinger and Nathan, 2004] is to systematically generate a pool of items based on the main factors which experts hypothesize would cause difficulties, and use the performance difference between items, with and without a difficulty factor, to confirm the existence of each hypothesized difficulty factor.

On a highly abstract level, the difficulty factor hypothesized here is *skill integration*. Following the basic idea of DFA, the key idea of the design is to examine whether problems requiring skill integration are (significantly) more difficult than tasks using the same component skills without skill integration. If significant or noticable performance differences are found, this will suggest that a *composition effect* exists [Heffernan and Koedinger, 1997, Koedinger and McLaughlin, 2016]. In the context of program comprehension, the key idea of the design is to examine whether novice programmers have significantly more difficulties in understanding code that requires the integration of basic programming skills (constructs) together, as compared to understanding code involving these same skills, but without their integration.

However, such a high, abstract level difficulty factor may not provide enough information to construct a learner model that gives sufficient integration practice to the student, nor to gain generalizable insights on skill integration. Therefore, I have hypothesized a range of integration skills which can also be interpreted as the finest-grained difficulty factors in the context of my dissertation, and used them to systematically design a set of problems with enough variety to reflect the wide range of common basic programming practices.

The data collected from students working on these problems is also called the *DFA data*. It was first used to examine the existence of the composition effect, and then to study the nature of the composition effect, to confirm whether each hypothesized integration skill exists or not, and to elicit integration difficulty factors with appropriate abstraction levels.

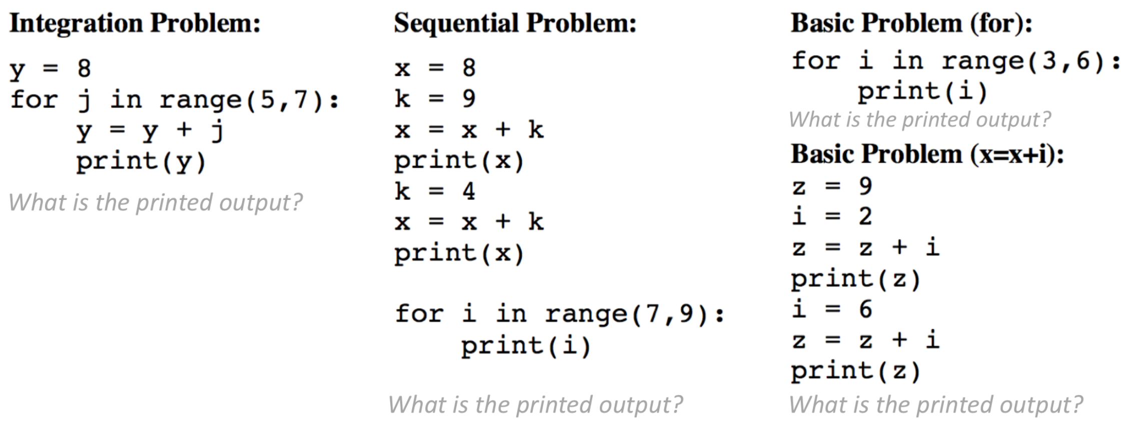
3.1.2 Design and Materials

Following the above overall idea, for each hypothesized integration skill, a problem set was constructed with three types of problems:

- *Basic* problem(s): Each basic problem requires applying a single basic component skill. For a problem set, some basic problems might have already appeared in a previous problem set, but only the remaining basic problem(s) is(are) included in the current problem set.
- *Sequential* problem: Each sequential problem requires applying basic component skills sequentially, as though directly solving one basic problem after another.
- *Integration* problem: Each integration problem requires integrating basic component skills together.

In each problem, a program is presented, and students are asked to write an execution table (which they have learned from lectures) recording the values of the variables and the printed outputs that will appear in the console. Figure 2 shows a problems set for a hypothesized integration skill *for&x=x+i*, i.e., being able to get cumulative sums for a sequence of consecutive numbers in a *for* loop. Figure 3 shows another problem set for a hypothesized integration skill *for&for*, i.e., being able to get the sequence of numbers resulting from a nested *for* loop.

The performance comparison between each matching sequential problem and integration problem examines the composition effect: if the integration version is significantly more difficult than the sequential version, then the composition effect exists. Similarly, the combined performance of the set of basic problems, compared with the matching integration problem, also examines the composition effect. The inclusion of both basic and sequential problems rather than only one of them enables two ways of analyzing the composition effect, as used in prior work [Heffernan and Koedinger, 1997, Koedinger and McLaughlin, 2016]. I used two guidelines when designing each problem set: 1) The basic and sequential problems include repetition so that in each pairwise comparison, the non-integrated version has the same number of steps (or more than) those in the integrated version. This is to avoid increasing the difficulty (if any) in the integrated version, merely as a result of having more steps (due



Integration Problem:

```
y = 8
for j in range(5,7):
    y = y + j
    print(y)
```

What is the printed output?

Sequential Problem:

```
x = 8
k = 9
x = x + k
print(x)
k = 4
x = x + k
print(x)

for i in range(7,9):
    print(i)
```

What is the printed output?

Basic Problem (for):

```
for i in range(3,6):
    print(i)
```

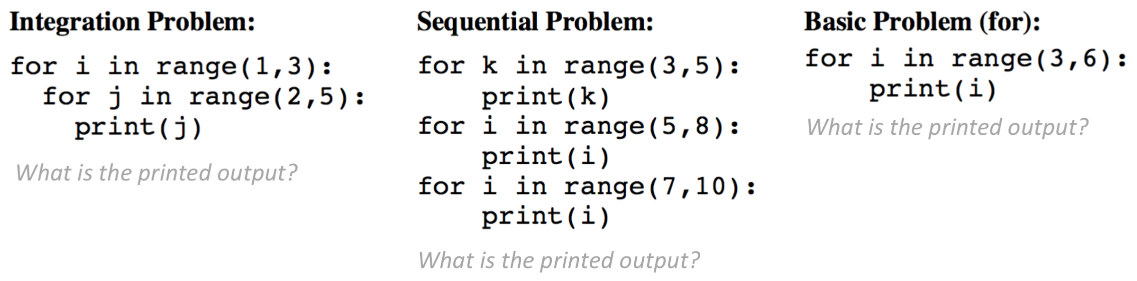
What is the printed output?

Basic Problem (x=x+i):

```
z = 9
i = 2
z = z + i
print(z)
i = 6
z = z + i
print(z)
```

What is the printed output?

Figure 2: The problem set for the hypothesized integration skill *for&x=x+i*.



Integration Problem:

```
for i in range(1,3):
  for j in range(2,5):
    print(j)
```

What is the printed output?

Sequential Problem:

```
for k in range(3,5):
    print(k)
for i in range(5,8):
    print(i)
for i in range(7,10):
    print(i)
```

What is the printed output?

Basic Problem (for):

```
for i in range(3,6):
    print(i)
```

What is the printed output?

Figure 3: The problem set for the hypothesized integration skill *for&for*.

to the loop), thus increasing the possibility of errors. 2) Problems vary in their variable names and literal values but keep the same number of variables and have similar values for the literals in each pairwise comparison.

The problem sets and hypothesized integration skills are listed in Table 1[1]. In the table, the symbol & means integration and the hypothesized integration skill is the integration of the basic skills connected by &. The problem set name is also used as the integration problem name. Each problem set (except the list-V one) is designed to investigate only one 2-component integration skill, which suffices to lay the foundation for investigating more complex integration skills in the future. But in the topic of *lists*, common patterns (e.g.,

[1]I removed the problem sets *for&x=x+167* and *while&x=x+167* from the analysis, because the hypothesized integration skills are of the same nature as *for&x=x+4* and *while&x=x+4*. Using larger numbers only increases the summation complexity, which is not a targeted factor in my thesis.

sorting) easily involve more than one integration skill, so I also include problem set V in the topic of *lists*. Skills with subscripts denote variants of the same programming construct corresponding to different common programming patterns. For example, for_{v2} denotes a for loop where the number of iterations depends on another variable rather than on a literal (i.e., the range function receives a variable rather than a literal as the argument). This kind of *for* loop could be used as an inner loop, while the outer loop iteration variable appears as the condition part of a nested loop pattern.

3.1.3 Latin Square and Within-Subject Design

In order to reduce the order effect when computing the composition effect for a problem set, a Latin square design rotating the order of basic, sequential and integration problems was deployed (Table 2). Since I don't (primarily) aim at comparing different hypothesized integration skills, a fixed order of integration skill problem sets was used. Each order corresponded to one quiz. Each student was randomly assigned to one of the three quizzes, each of which contained the same set of problems. This is a within-subject design.

3.1.4 Participants and Procedure

I conducted the classroom study (Classroom Study 1) in an introductory Python programming course at the Austral University of Chile (Universidad Austral de Chile) from April to June 2017 in three sessions covering three topics (*for* loops, *while* loops and *lists* consecutively, following the syllabus). The number of participants in each session are listed in Table 3. Each session was held as an in-class quiz session two weeks after the targeted topic was lectured on and students had already done some exercises on this topic. In each session, students were asked to complete a hard-copy quiz within one hour. In agreement with the instructor, the scores of the quiz were considered as part of the students' course grades.

Table 1: Problem sets and the corresponding hypothesized integration skills in the topic of *for* loops, *while* loops and *lists*. Roman numerals denote the order of problem sets in each quiz, while Arabic numbers denote the overall order.

Problem set			Hypothesized integration skill(s)	
		Name	Name	Definition (being able to get...)
I	1		for&for	A sequence of numbers resulting from a nested *for* loop.
II	2		for&x=x+i	The sum of a sequence of consecutive numbers in a *for* loop.
III	3		for&for$_{v2}$	A sequence of numbers resulting from a nested *for* where the outer loop iteration variable decides the number of inner loop iterations.
IV	4		for&x=x+4	The cumulative sum when a fixed number is added during each *for* loop iteration.
V	5		for&x=4+i	The sum of a fixed number and the value of the iteration variable in each *for* loop iteration.
I	6		while&while	A sequence of numbers resulting from a nested *while* loop.
II	7		while&x=x+i	The sum of a sequence of consecutive numbers in a *while* loop.
III	8		while&while$_{v2}$	A sequence of numbers resulting from a nested *while* where the outer loop iteration variable decides the number of inner loop iterations.
IV	9		while&x=x+4	The cumulative sum when a fixed number is added during each *while* loop iteration.
V	10		while&x=4+i	The sum of a fixed number and the value of the iteration variable in each *while* loop iteration.
VI	11		while&for	A sequence of numbers resulting from having a *for* loop nested inside a *while* loop.
I	12		for&a$_1$=a$_1$+a$_2$	List values after appending another list to it multiple times in a *for* loop.
II	13	search-max	for&if	The maximum value in a list with a *for* loop.
III	14	for&x=x+a[i]	for&x=x+i	The sum of a list, using a *for* loop.
IV	15		for&a[i]$_{v3}$	List values after creating the list as a Fibonacci sequence, using a *for* loop.
V	16	insert-sort	for&while$_{v3}$, a[i]$_{v4}$&a[i]$_{v5}$	List values after doing an insertion sort within the list, using a *for* loop.
VI	17		for&a[i]$_{v2}$	List values when creating a list of consecutive odd numbers, using a *for* loop.

Table 2: Latin square design for the composition effect DFA study. Each Roman numeral denotes one integration skill problem set.

Quiz	I			II			...
A	basic	integration	sequential	basic	integration	sequential	...
B	integration	sequential	basic	integration	sequential	basic	...
C	sequential	basic	integration	sequential	basic	integration	...

Table 3: Number of students for each study session (topic).

Topic(session)	*For*	*While*	*Lists*	Participated in all topics
#students	81	76	68	60

3.1.5 Grading

Each problem was graded as correct (1) or wrong (0), using the same rubrics. When a printed output was an exact match with the standard answer, it was graded as correct. In the cases of inexact match, if it was clear from the execution table (if written out) that a student knew the skill, but he/she committed a computation error (e.g., 4+9=12), missed some intermediate printed outputs, or wrote a list with correct elements but in a strange format (e.g., missing the brackets), then the answer was also graded as correct. All other cases were graded as wrong (including inertial thinking errors, as defined in Section 3.2.2).

3.2 RESULTS

In this section, I conduct an in-depth analysis investigating 1) the existence of the composition effects in common basic programming patterns, 2) the nature of the composition effects and integration skills, and 3) individual differences in integration skills.

3.2.1 Existence of Composition Effects

3.2.1.1 Comparing Sequential Problems with Integration Problems

First, I examined each problem set to see whether a composition effect exists. For each problem set, each student's correctness on the sequential problem was compared with his(her) correctness on the matching integration problem, which is similar to the analysis by [Heffernan and Koedinger, 1997]. A Wilcoxon signed-rank test was conducted on the differences, with the sample size equal to the number of students. If the difference between the two versions of problems was significant, then the composition effect exists for the problem set.

As shown in Table 4, among the 17 problem sets across three topics, 71% (12/17) shows a composition effect at a 0.05 significance level, and 82% (14/17) shows a composition effect at a 0.1 significance level. Only three problem sets don't show a composition effect, although their sequential versions are all easier than the integration versions.

Problem sets requiring $loop\&loop_{(v2)}$ consistently show significant composition effects across three topics with $p<0.1$. Specifically, considering only nested loops composed of the same loop construct (e.g., $for\&for_{(v2)}$, $while\&while_{(v2)}$), each composition effect is very strong with $p<0.003$. This provides some support for the existence of integration skills related to $loop\&loop_{(v2)}$. Problem sets requiring *loop&x=x+i* also consistently show significant composition effects across three topics with $p<0.05$. This provides some support for the existence of integration skills related to *loop&x=x+i*.

Meanwhile, problem sets requiring *loop&x=x+4* only show a significant composition effect in the *for* loops topic ($p< 0.001$) but not in the *while* loops topic ($p=0.157$); problem sets requiring *loop&x=4+i* show a significant composition effect in the *while* loops topic ($p<0.01$), but not in the *for* loops topic ($p=0.225$). The existence of the hypothesized integration skills related to *loop&x=x+4* and *loop&x=4+i* needs further investigation (see Section 3.2.2).

Other problem sets with hypothesized integration skills specific to the *lists* topic mostly show significant composition effects ($p<0.1$), with the exception of $for\&a[i]_{v3}$ ($p=0.346$). This provides some support for the existence of integration skills related to *a[i]&a[i]*, *for&if*, and $for\&a_1=a_1+a_2$, but *for&a[i]* needs further investigation (see Section 3.2.2).

Table 4: Examining the composition effect for each problem set and all problem sets. For each problem set, pK_x denotes the proportion correct among students on the corresponding basic problem; pM denotes the estimated proportion correct on the sequential problems, as computed by multiplying pK_xs (or picking the smallest in the cases of *for&for*$_{v2}$, *while&while*$_{v2}$, or *a[i]*$_{v4}$&*a[i]*$_{v5}$); pS (pI) denotes the proportion correct on the matching sequential (integration) problems. Wilcoxon's signed-rank test (WT) was used when examining each problem set or when normality was violated; otherwise a paired t-test (PT) was used. (Significance level ***:<.001, **:<.01, *:<.05, $^{\bullet}$:<.1; WT effect size (es) $^{+++}$:>.5, $^{++}$:>.3, $^{++}$:>.1; PT effect size $^{+++}$:>.8, $^{++}$:>.5, $^{+}$:>.2; Itgt.: Integration.)

ID	Hypothesized	K_1,(K_2	pK_1 (pK_2	pM pS pI	pM-pS	pM-pI	pS-pI		
	Itgt. skill(s)	K_3,K_4)	pK_3 pK_4)				val	p-val	es
3	for&for$_{v2}$	for,for$_{v2}$	.86 .80	.80 .77 .46	.04	.35	.31	<.001	.38$^{***}_{++}$
4	for&x=x+4	for,x=x+4	.86 .96	.83 .79 .62	.04	.21	.17	<.001	.26$^{***}_{+}$
1	for&for	for	.86	.86 .60 .38	.26	.48	.22	.002	.24$^{**}_{+}$
2	for&x=x+i	for,x=x+i	.86 .90	.78 .72 .62	.06	.16	.10	.033	.17$^{*}_{+}$
5	for&x=4+i	for,x=4+i	.86 .99	.85 .75 .69	.10	.16	.06	.225	.09
8	while&while$_{v2}$	while,while$_{v2}$	.89 .91	.89 .88 .57	.01	.33	.32	<.001	.40$^{***}_{++}$
6	while&while	while	.89	.89 .86 .61	.04	.29	.25	<.001	.34$^{***}_{++}$
10	while&x=4+i	while,x=4+i	.89 .97	.87 .86 .71	.02	.16	.14	.008	.22$^{**}_{+}$
7	while&x=x+i	while,x=x+i	.89 .95	.85 .89 .82	-.05	.03	.08	.034	.17$^{*}_{+}$
11	while&for	while,for	.89 .96	.86 .84 .76	.02	.10	.08	.058	.15$^{\bullet}_{+}$
9	while&x=x+4	while,x=x+4	.89 .97	.87 .89 .84	-.02	.03	.05	.157	.11$_{+}$
16	for&while$_{v3}$	for,while$_{v3}$	.90 .74	.44 .50 .22	-.06	.22	.28	<.001	.33$^{***}_{++}$
	a[i]$_{v4}$&a[i]$_{v5}$	a[i]$_{v4}$,a[i]$_{v5}$	.68 .66						
17	for&a[i]$_{v2}$	for,a[i]$_{v2}$	.90 .68	.61 .65 .44	-.04	.17	.21	<.001	.28$^{***}_{+}$
14	for&x=x+i	for,x=x+i,a[i]	.90 .97 .72	.63 .72 .57	-.09	.05	.15	.004	.25$^{**}_{+}$
13	for&if	for,if,a[i]	.90 .90 .72	.58 .68 .57	-.10	.01	.10	.035	.18$^{*}_{+}$
12	for&a_1=a_1+a_2	for,a_1=a_1+a_2	.90 .79	.71 .79 .72	-.08	-.01	.07	.096	.14$^{\bullet}_{+}$
15	for&a[i]$_{v3}$	for,a[i]$_{v3}$	.90 .63	.57 .60 .54	-.04	.02	.06	.346	.08
	Avg (p-value)	–	–	.76 .75 .60	.01(.8)	.16(<.001)$^{***}_{+++}$			

3.2.1.2 Comparing Basic Problems with Integration Problems

Secondly, I examined the composition effects using basic and integration problems. This was the method used in prior work [Koedinger and McLaughlin, 2016], where they compared the product of the proportion correct (treated as a probability) of each decomposed basic problem (pM) with the matching integration problem (pI). I use this as another way to examine the composition effects. Multiplying the single probabilities serves as an estimate of the proportion correct of the sequential problem (pS), with the added assumption that the performance on each basic problem is independent of performance on other basic problems. While the satisfaction of this assumption needs to be examined, a benefit of using basic problems over using sequential problems is that it could reduce the total number of problems in a study design, since many problem sets share the same basic skills. Meanwhile, this method of estimation belongs to the family of conjunctive models used in prior work [Conati et al., 2002, Mayo and Mitrovic, 2001, Millán and Pérez-De-La-Cruz, 2002] for multi-skill practice problems, with the simplification of assuming independence among skills, no integration skills, and no noise associated with items or skills.

To compute the joint probability in this new way, I used the following guidelines for two special cases: 1) If a basic skill is applied multiple times with the same complexity, the joint probability equates to the probability of succeeding in a problem applying the basic skill once, e.g, P(for)×P(for)=P(for); 2) if a basic skill is applied multiple times with different complexities (e.g., with or without the outer loop iteration variable in the condition part of the inner loop), the joint probability equates to the smaller one of the probabilities of succeeding in problems applying the basic skill in different ways, e.g., P(for)×P(for_{v2})=min(P(for), P(for_{v2})). Both 1) and 2) are based on the assumption that different instantiations of the same basic skill should be considered as highly dependent.

As shown in Table 4, all problem sets (except $for\&a_1\!=\!a_1\!+\!a_2$) have shown a positive difference (pM-pI), and the average difference over all problem sets is 0.16. A paired t-test (after confirming the normality) on the difference values (sample size=17) further reveals that the difference is statistically different from 0 (p<0.001) with a large effect size (1.07). This is consistent with the result of a Wilcoxon signed-rank test (due to violation of normality) on the difference values between the sequential and integration problems (pS-pI) with an average value of 0.16, p<0.001 and a large effect size (0.62). The results suggest that, in general, the composition effect exists in the pool of common basic programming patterns.

In addition, I also validated that pM and pS are not statistically different by a Wilcoxon signed-rank test (due to violation of normality) with p=0.83. This suggests that the independence assumption among basic skills is reasonable when using basic problems to estimate a sequential (joint) problem, for cases where sequential problems are not available. Yet, it's worth mentioning that sequential problems, compared with basic problems, offer a straightforward way to conduct statistical tests for examining each problem set.

3.2.1.3 Examining Composition Effects on Topic Level

I also examined the composition effects on groups of problem sets on each or all topics to see whether, in general, the composition effect exists in a selected pool of programming patterns.

Since the students and problem sets considered here are still only a subset of the entire student population and possible common basic patterns, I conducted a generalized linear mixed-effect modeling for further analysis. I constructed generalized linear mixed-effect models predicting the correctness of each problem for each student, given a binary indicator of whether the current problem was an integration problem or not (as a fixed effect), the problem set id (as a random effect) and the student id (as a random effect). Only the sequential problems and matching integration problems in a targeted group (*for*/*while*/*lists*/all) were used to construct a model. I conducted the analysis by using the R package *lme4*, using the formula: $correctness \sim is\_itgt\_prob + (1|prob\_set) + (1|stu)$.

Table 5 reports the results. As shown in the table, regardless of the unit of a group (*for*/*while*/*lists*/all), the fixed effect *is_itgt_prob* (the *Est.* column) is consistently significantly negative (p<0.001), i.e., the integration version is much more difficult than the matching sequential version. The results provide further support that, in general, the composition effect exists in the pool of common basic programming patterns. The results demonstrate the robustness of the composition effect across topics.

Table 5: Examining the composition effects for groups of problem sets. Generalized linear mixed effect models were constructed for predicting the correctness on each problem of each student based on the following formula: $correctness \sim is\_itgt\_prob + (1|prob\_set) + (1|stu)$. Residuals are (approximately) normally distributed. (ps: problem set; FE: fixed effect; RE: random effect; WT: Wald test; LRT: likelihood ratio test comparing this model with the model without the fixed effect; LL: log likelihood; significance level ***:<0.001.)

Topic	#ps.	#stu.	#obs.	FE *is_itgt_prob* p-value				RE SD		LL
				Est.	SE	WT	LRT	*stu*	*ps*	
for	5	81	810	-1.48	0.21	<.001***	<.001***	2.55	0.68	-378
while	6	76	912	-1.94	0.28	<.001***	<.001***	2.60	0.65	-314
lists	6	68	816	-1.23	0.21	<.001***	<.001***	2.52	1.01	-380
all	17	86	2538	-1.20	0.11	<.001***	<.001***	1.88	1.00	-1153

3.2.2 Nature of Composition Effects and Integration Skills

So far, I have gathered evidence proving the existence of composition effects and providing some preliminary support for the hypothesized integration skills. This section further investigates the nature of composition effects and integration skills by posing following questions:

- Why does the composition effect exist in some cases but not in others? Is the composition effect due to missing/wrong conceptual understanding, cognitive load or both?
- What are the integration skills and integration difficulty factors that could explain the composition effect?

These two sets of questions are interconnected with each other, and they provide the basis for generalizing the composition effect and integration skills over a broader range of topics and problems, and should form the basis for later learner model construction.

3.2.2.1 Integration Error Analysis

To answer the above questions, I analyzed the errors on integration problems for students who succeeded in a sequential problem but failed in the matching integration problem. I excluded sequential problem errors that resulted from the lack of basic skills, and only focused on the errors introduced by integration itself. I defined such errors as *integration errors* and classified them into the following types, according to the error analysis:

- *Conceptual errors*: In this type of error, students have a missing or wrong conceptual understanding of how integration works, such as in the following two cases:
  - *State update errors*: Students fail to update a variable correctly.
  - *Nested loop errors*: Students fail to execute the nested loop with the right procedure.
- *Nonconceptual errors*: In this type or error, students demonstrate enough conceptual understanding of how integration works, but commited errors similar to the following examples:
  - *Cognitive load errors*: Students make errors on basics skills, e.g., when summing over a list using a loop, in some iterations, the students used the value of the iteration variable *i* rather than the list element *a[i]*, while in other iterations they were able to retrieve a[i] correctly, as well as in the sequential problem. In this case, students lack fluency in the basic skills, so during a complex integration process they are deprived of cognitive resources and intermittently commit errors on basic skills.
  - *Inertial thinking errors*: Students use a similar operation with the same or increased complexity which has been used in previous problems of the quiz, or has been practiced more often in other materials, e.g., in *while&x=4+i* problem, students did x=4+i in the first iteration and also in the sequential problem, but did x=4+x in later iterations (note that *while&x=4+i* was positioned after *while&x=x+4*). This is different from the cognitive load errors, because students need the same or even more cognitive resources for the mistakenly chosen operation.
- *Uncategorized errors*: Such errors can't be categorized into the above categories, e.g., empty answers, errors with too few students (1 or 2).

My error analysis revealed that integration difficulty can be a result of multiple types of errors, as listed above. I classified an integration error into one of the following three types, depending on the dominant type of integration error among them state update conceptual errors, nested loop conceptual errors and nonconceptual errors.

- *Nested loop integration type*: The highest percentile is *nested loop conceptual errors*.
- *State update integration type*: The highest percentile is *state update conceptual errors*.
- *Nonconceptual integration type*: The highest percentile is *nonconceptual errors*.

Figure 4 summarizes the percentile of integration errors of each integration (problem set) and grouped integrations (problems sets) into integration types according to the major integration errors. The details are reported in Tables 31, 32 and 33 in the Appendix.

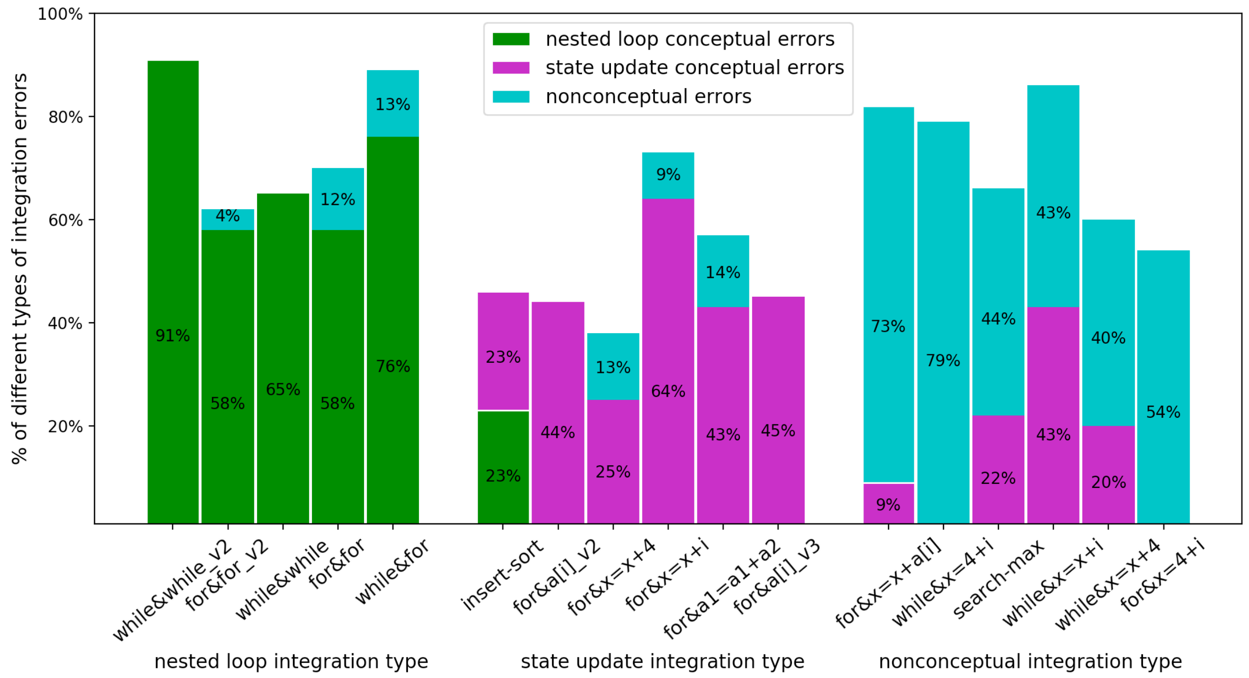


Figure 4: Percentiles of different integration errors for different problem sets. Problem sets are classified into three integration types, and are ordered by the effect size of the composition effect, from large to small (left to right), within each type.

Before examining each integration type, I would like to explain two situations that I need to specifically address when grouping integrations: 1) The problem set *insert-sort* with the hypothesized integration skills $for\&while_{v3}$, $a[i]_{v4}\&a[i]_{v5}$ has the same percentile of state update conceptual errors and nested loop conceptual errors (23%). I classified this as a state update integration type because its composition effect (pS-pI) has a much higher average correlation with those in the state update integration type than with those in the nested loop integration type (0.28 vs. -0.07). Similarly, the problem set *while&x=x+i* also has the same percentile of nonconceptual errors and state update conceptual errors (43%). I classified it into the nonconceptual integration type because its composition effect (pS-pI) has a much higher average correlation with those in the nonconceptual integration type than with those in the state update integration type (0.20 vs. 0.11).

Regarding the nested loop integration type (Figure 4, Table 31), the main integration errors are nested loop related conceptual errors ($\geq$58%). One major conceptual error which persisted across different types of nested loops was that students could only do one outer iteration with all its inner iterations. It could be that students only did the first outer iteration not knowing that they should return to the outer iteration again after finishing the inner iterations, or that the students only did the last outer iteration because they enumerated all values of the outer iteration variable first and then did the inner iterations. Other surprising nested loop conceptual errors are listed in the Table 31. Students should be able to figure out nested loops once they have learned the basic loop, since the inner loop as a whole could be treated as statements put in a standard loop body, but it turned out that many students had learned the basic loop on a surface level and failed to decompose a nested loop correctly. These results suggest that nested loops should be considered as integration skills, and students should benefit from sufficient, explicit practice on them.

Regarding the state update integration type (Figure 4, Table 32), the main integration errors are state update related conceptual errors ($\geq$23%). Student committed such conceptual errors mainly due to using the initial value of a variable (which points to a number or a list) in all iterations, instead of updating the variable or the list with each iteration and using the updated value in a new iteration. Interestingly, when the state update statement, the addition assignment statement[2], is explicitly written multiple times consecutively (see Figure 2 sequential or basic problems), many students were able to use the updated variable from a previous addition assignment statement in a repeated addition assignment statement, but failed to do so within a loop when the repetition is implicit and interrupted by the loop statement. This suggests that additional skills exist which require students to use the updated values of variables in loop iterations, and students should benefit from sufficient, explicit practice on these skills.

Unfortunately, the integration error analysis on the state update integration type didn't provide more insights into the understanding the non-existence of the composition effect in $\mathit{for\&a[i]}_{v3}$ in contrast to the existence of the composition effect in $\mathit{for\&a[i]}_{v2}$, since both problem sets share the similar composition of integration errors. The integration problem of $\mathit{for\&a[i]}_{v3}$ is expected to be more difficult than that of $\mathit{for\&a[i]}_{v2}$, since the former not only requires retrieving the previous list element (which is required in $\mathit{for\&a[i]}_{v2}$ for creating an odd number sequence), but also requires retrieving the element two positions before to create a Fibonacci sequence. As shown in Table 4, pI of the former is 0.54 and pI of the latter is 0.44, while their pSs are quite close (0.60 vs. 0.65). I posit several hypotheses requiring further testing in the future: $\mathit{for\&a[i]}_{v2}$ was positioned at the end of the quiz and students might lack time or attention to solve the integration problem correctly, but students might have decided and were able to pay more attention to the $\mathit{for\&a[i]}_{v3}$ integration problem which was positioned earlier after seeing its complexity; a Fibonacci sequence is inherently recursive, making it natural to express it as a[i]=a[i-1]+a[i-2] in the code, but an odd number sequence is not recursive, making it a bit unnatural to express it as a[i]=a[i-1]+2 in the code, so students might be more able to recognize the Fibonacci pattern rather than the odd number pattern and thus were more able to solve the $\mathit{for\&a[i]}_{v3}$ integration problem correctly.

The nonconceptual integration type (Figure 4, Table 33) contains the most unexpected cases for the (non)existence of the composition effect. The main integration errors are nonconceptual ($\geq$40%), and in some cases state update conceptual errors were also committed.

[2]In my dissertation, an assignment statement like x=x+i is also called an *addition assignment*, although strictly speaking, only an assignment with a += operator (e.g., x+=i) should be called this. I define it in this way because the courses for my dissertation studies had only taught the former one, at the time the studies were run, and both ways essentially share the same kind of state update structure.

According to the integration error analysis, the unexpected c omposition effect on *while&x=4+i* is mainly because many students (14×79%=11) did x=4+i in the first iteration, but did x=4+x or x=x+i afterwards due to inertial thinking. The reason this problem set has such severe inertial thinking slips is probably due to it being positioned at the end of quizzes after *while&x=x+4*, *while&x=x+i*. Meanwhile, *for&x=4+i*, as expected, doesn't show a composition effect; it has fewer students who committed inertial thinking integration errors and any kind of integration errors, compared with *while&x=4+i*. It might be that *while&x=4+i* has one more explicit addition assignment for the iteration variable within the loop body, which could increase the integration difficulty due to cognitive load.

Two more unexpected cases are the *while&x=x+i* and *while&x=x+4* problem sets. Compared with their counterparts, *for&x=x+i*, *for&x=x+4*, these integration problems also require updating the state, but they have shown a much higher ratio of nonconceptual errors, all belonging to the cognitive load error type. This suggests that students had gradually acquired the conceptual understanding of *loop&x=x+i* or *loop&x=x+4* by the time they reached *while* loops topic, but the *while* loops topic introduced a new level of difficulty, requiring a higher cognitive load, probably due to the explicit addition assignment statement of the iteration variable in the loop body.

Interestingly, although the integration problem *for&x=x+a[i]* also requires updating states as *loop&x=x+i*, the main errors are cognitive load errors (73%), such as using the iteration variable rather than the list element to do summation in some iterations, or starting the list element indexes with 1. This suggests that most students had already acquired the conceptual understanding of *loop&x=x+i* or *loop&x=x+4* when they reached the *lists* topic, but putting such an integration in the context of *lists* increased the cognitive load substantially, since students still hadn't reached fluency in basic skills in *lists*.

The integration error analysis also gives insight into what causes difficulty in the integration problem *search-max* with the hypothesized integration skill *for&if*: the main errors here are also cognitive load errors (44%), such as starting the list element indexes with 1, or doing one more or one less iterations. Despite being very different from *loop&x=x+i* in the surface, *for&if* actually shares with *loop&x=x+i* the deep feature of requiring a state update: here, it requires updating the variable storing the cumulative maximum value.

3.2.2.2 Identification of Integration Skills

Based on the above integration error analysis (Section 3.2.2.1) and previous composition effect existence analysis (Section 3.2.1), I examined which hypothesized integration skills can be accepted for explaining the composition effects. An *integration skill* exists and only exists when a composition effect exists and any (substantial number of) *conceptual* integration errors can be found in the DFA data. The existence of conceptual integration errors is necessary: without them, the composition effect could be an artifact of the study design (like the case of *while&4+i*, where a composition effect was found and no conceptual errors were associated with it), and also it was not clear how to define the skill. However, one should be cautious that this doesn't mean that integration skills don't involve a nonconceptual difficulty. As shown in the integration error analysis, a noticeable number of integration errors may be due to cognitive load. Table 6 gives the definitions of the conceptual parts of integration skills. The definitions are informed by the detailed conceptual and nonconceptual integration errors summarized from the DFA data shown in Table 31, 32 and 33 (in Appendix). Note that these definitions differ from the previous definitions in Table 1, which gives the overall definitions of integration skills in terms of their functionalities, including both their conceptual and nonconceptual parts.

Note that I have merged *for&x=x+i* and *for&x=x+4* into one integration skill *for&+=* (the same applies to *while*), because *for&x=x+i* and *for&x=x+4* contained the same kind of state update conceptual error, i.e., using the initial value of the sum variable in all iterations, and only differed in nonconceptual errors and uncategorized errors (the same applies to *while*), according to the detailed integration error analysis (Table 32, 33).

Also, I excluded the hypothesized integration skills *for&x=4+i* and *while&x=4+i* from the set of integration skills, because the integration error analysis (Table 33) revealed that most of their errors were due to inertial thinking, probably resulting from positioning this problem set at the end of the quiz, and there were no conceptual errors associated with them. Such a composition effect will probably disappear if the order of the problem sets is permuted.

Meanwhile, the integration skills *for&*a_1=a_1+a_2, *for&a[i]* and *for&if* may be generalizable to the *while* loop, but I leave it to future research to confirm such hypotheses.

Table 6: Defining the conceptual parts of integration (itgt.) skills, as identified by the integration error analysis of the DFA data. A student who demonstrates the described behavior is considered to know how the integration skill works, conceptually. *Loop* refers to either a *for* loop or a *while* loop.

Itgt. skill	Definition of the conceptual part (being able to ...)
while&while	Get the correct iteration variable values for at least two outer loop iterations with corresponding inner loop iterations inside a nested *while* loop.
for&for	Get the correct iteration variable values for at least two outer loop iterations with corresponding inner loop iterations inside a nested *for* loop.
while&for	Get the correct iteration variable values for at least two outer loop iterations with corresponding inner loop iterations, when a *for* loop is nested inside a *while* loop.
for&+=	Use the value of the updated variable x (which stores the cumulative sum of numbers) from the previous iteration in a new iteration in a *for* loop.
while&+=	Use the value of the updated variable x (which stores the cumulative sum of numbers) from the previous iteration in a new iteration in a *while* loop.
for&a_1=a_1+a_2	Use an updated list from a previous iteration to conduct list appending in a new iteration, in a *for* loop.
for&a[i]	Use an updated list element from a previous iteration to conduct numeric operations in a new iteration, in a *for* loop.
for&if	Update a variable x (which stores the current maximum value of some list elements) and use the value of the updated x from the previous iteration in a new iteration, in a *for* loop.
a[i]&a[i]	Update both list elements correctly in a value swap pattern.

Admittedly, the number of integration errors (i.e., the number of students who have succeeded in the sequential problem but failed in the integration problem) for conducting the integration error analysis is still not large enough, reducing the reliability of the conclusions drawn. I am considering conducting a large scale study to solidify the discoveries found here in the future.

3.2.2.3 Identification of Integration Difficulty Factors

The above integration skills can be further abstracted into three integration difficulty factors, *loop nesting*, *state update* and *cognitive load*[3], which are more coarse-grained than integration skills, but more fine-grained than the single difficulty factor, *skill integration*. This level of abstraction could enable generalization of composition effects and integration skills.

Table 7: Integration (itgt.) difficulty factors identified by the integration error analysis.

Itgt. difficulty factor	Type	Definition
Loop nesting (LN)	Conceptual	A loop nested inside another loop.
State update (SU)	Conceptual	Retrieving the previous value of a variable to conduct operations, then storing the result back in the same variable, when the statement is in a loop, or the variable points to a list (or a list element).
Cognitive load (CL)	Nonconceptual	The effort required to conduct an integration involving any of the above integration difficulty factors.

Table 7 gives the definitions of these three integration difficulty factors. They summarize the common integration difficulties, or common major patterns in integration errors, among integration skills across different topics (or programming constructs) in common basic programming patterns. *Loop nesting* summarizes the common integration difficulty involved in different kinds of nested loops (involving *for* and/or *while*). *State update* summarizes the common integration difficulty involved in updating and retrieving updated values of vari-

[3] *Inertial thinking* is excluded from being an integration difficulty factor since it is highly likely an artifact of the fixed ordering over problem sets, as explained before.

ables pointing to different kinds of objects (e.g., *integers*, *strings*, *lists*) in a loop or in list operations. *Cognitive load* refers to the effort required to conduct the above two types of integrations. In Section 3.2.3, I provide further support for the abstraction of these three integration difficulty factors by showing how they could explain individual student differences in integration skills. In Section 4.1.1, I show an example of using the two conceptual integration difficulty factors to identify potential integration skills in new contexts.

3.2.3 Individual Differences in Integration Skills Explained by Integration Difficulty Factors and Topics

So far, I have demonstrated and explained the existence of the composition effects (CEs) and integration skills in common basic programming patterns. A natural question to ask next, for informing later learner modeling, is whether there are individual differences in integration skills, e.g., whether some students have higher or lower integration skills than others (over a period of time). If individual differences in integration skills exist, then it is reasonable and beneficial for a learner model to monitor different students' levels of integration skills, and thus elicit personalized guidance to help individuals, based on their levels. An similar question, from another perspective, is to ask whether a CE or integration difficulty could generalize to new problems.

In order to answer these questions, I conducted the following analysis: I removed two problem sets, *for&x=4+i* and *while&x=4+i*, in which integration difficulties are due to the artifact of the fixed order of problem sets, and only kept those corresponding to the identified integration skills in Table 6. This resulted in 15 problem sets. I permuted 15 problem sets so that different types of integration skills, according to integration difficulty factors, were distributed evenly in the sequence. I called the first half of the problem sets Group 1 and the second half Group 2 (Table 8). Although Group 2 data has been used in previous sections to maximize the amount of data to be analyzed, in the analysis of this section, it serves as an *approximation* of a group of new problems. I used the strength of the CE (pS-pI) as the indicator for integration skill levels (the larger the CE, the lower the integration skill level). I only used the 60 students who participated in all three sessions for the analysis.

Table 8: A random splitting of all problem sets into two groups, where topics and integration difficulty factors are evenly distributed (Integration difficulty factors LN: loop nesting, SU: state update, CL: cognitive load).

		For	*While*	*Lists*
Group 1	LN	for&for	while&while$_{v2}$	–
(7)	SU	for&x=x+i	–	insert-sort(*for,while*), for&a[i]$_{v3}$
	CL	–	while&x=x+4	for&x=x+a[i]
Group 2	LN	for&for$_{v2}$	while&while, while&for	–
(8)	SU	for&x=x+4	–	for&a[i]$_{v2}$, for&a_1=a_1+a_2
	CL	–	while&x=x+i	search-max(*for*)

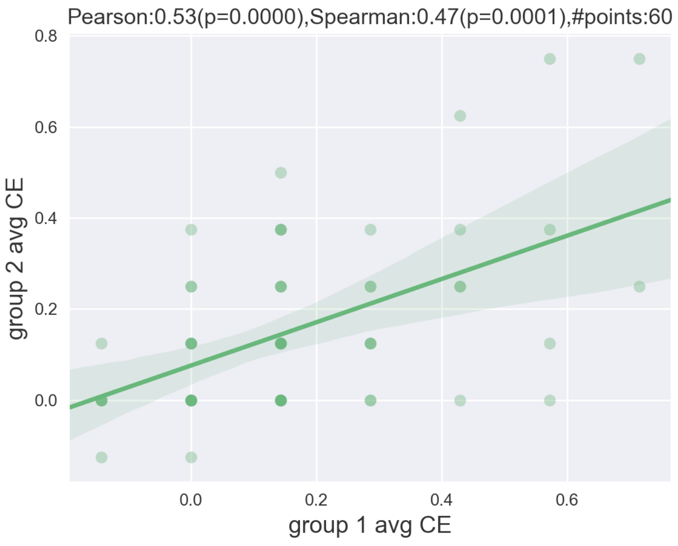


Figure 5: The correlation of the average CE strength (pS-pI), between the Group 1 and Group 2 problem sets. A linear regression line with a 95% confidence interval for the regression estimate has been added.

Table 9: The Spearman correlation (with p-value) of average CE strength among integration difficulty factors. The within-type correlation is between the same types in Group 1 and Group 2; while the cross-type correlation is between one type from both groups to another type from both groups. There were 60 students (data points) for computing each correlation. (Sig. level ***:<.001, **:<.01, *:<.05.)

	LN (*for,while*)	SU (*for,while,lists*)	CL (*for,while,lists*)
LN (*for,while*)	.39(.002)**	.08(.55)	.30(.02)*
SU (*for,[while,]lists*)	–	.43(<.001)***	.02(.87)
CL (*for,while,lists*)	–	–	.38(.003)**

First, I investigated whether individual differences in integration skills exist by considering two problem set groups, and investigating whether the CE could be generalized from Group 1 to Group 2. For each student, I computed the average value of the CE strength (pS-pI) over the problem sets of Group 1, and that of Group 2, and then computed the Spearman correlation (after confirming that bivariate normality is violated) over the 60 students. As shown in Figure 5, the correlation between these two groups is 0.47 and it is highly significant (p<0.001). This shows that students with have higher (lower) integration skills on a group of problems also have higher (lower) integration skills on a new group of problems, suggesting that individual differences in integration skills exist. This also supports that CEs in a group of problems could generalize to a new group of problems.

However, one might notice that both groups have the same coverage of integration difficulty factors and topics, so it is not clear whether and how integration difficulty factors and topics affect the interpretation of the correlation. Firstly, I investigated the effect of integration difficulty factors on individual difference and the generalizability of the CE. I compared the pairwise correlation of average CE strength among integration difficulty factors. Within each group obtained before, the problem sets were further split according to the their major integration difficulty factors (as displayed in Table 8).

To compute a within-type correlation, I compared the average CE strength of a type in Group 1 with that of the same type in Group 2; to compute a cross-type correlation, I gathered all problem sets from both groups of a type and computed the average CE strength, and did the same for another type. The result is shown in Table 9. As shown in the table, within each integration difficulty factor, there is a significant positive correlation ($p<0.01$). This shows that students with higher (lower) integration skills of an integration difficulty factor (type) on a group of problems also have higher (lower) integration skills of the same integration difficulty factor (type) on another group of problems, suggesting that individual differences in integration skills holds within the same integration difficulty factor. This also shows that CEs can be generalized within the same integration difficulty factor.

In terms of the cross-type correlations, there is no correlation between LN and SU, SU and CL, but there is a significantly positive correlation between LN and CL ($p<0.05$). The absence of a correlation between LN and SU, SU and CL suggests that LN and SU are essentially different integration difficulty factors, and SU and CL are also fundamentally different integration difficulty factors. We shouldn't merge LN and SU, merge SU and CL, or use a single integration difficulty factor, *skill integration*, to abstract the integration skills.

Meanwhile, the significant correlation between LN and CL ($p<0.05$) suggests that students' individual differences may still exist and the CEs may generalize when merging both types, and LN may be highly related to CL. This is actually consistent with our understanding that nested loop problems typically require a high cognitive load. However, since LN and CL are fundamentally different integration difficulty factors in that the former includes a conceptual part while the latter doesn't (see their definitions in Table 7), we should be cautious about merging them. I leave this for future investigation.

One might argue that the above correlations might have resulted from generalizability within the same topic (or main programming construct). Therefore, I conducted a similar analysis as above but focused on the correlation among topics. As shown in Table 10, significant positive correlations are found within each topic ($p<0.001$), but there is no correlation between *for* and *lists* topics, *while* and *lists* topics, although each pair shares one integration difficulty factor. This suggests that topic plays a role in explaining the existence of individual differences and generalized CEs. However, topic alone is not sufficient for the explanation:

Table 10: The Spearman correlation (with p-value) of average CE strength among topics. The within-type correlation is between the same topics in Group 1 and Group 2; while the cross-type correlation is between one topic from both groups to another topic from both groups. There were 60 students (data points) for computing each correlation. (Sig. level ***:<.001, **:<.01, *:<.05, • :<.1.)

	for (LN,SU)	*while* (LN,CL)	*lists* (SU,CL)
for (LN,SU)	.43(<.001)***	.24(.07)•	.19(.15)
while (LN,CL)	–	.50(<.001)***	-.04(.74)
lists (SU,CL)	–	–	.49(<.001)***

in Table 9, LN and SU share similar topics (*for*, *while*), and SU and CL also share similar topics (*for*, *while*, *lists*), but there is no correlation found in each pair; meanwhile, LN in Group 1 and LN in Group 2 each involves two topics, SU in Group 1 and SU in Group 2 each involves two to three topics, and CL in Group 1 and CL in Group 2 each involves all three topics, and significant correlations were found in each within-type pairwise relation. This contrast suggests that both integration difficulty factors and topics are needed for explaining the existence of individual differences and generalized CEs. Meanwhile, a marginally significant correlation was found between the *for* and the *while* topics (p<0.1), which is consistent with our understanding that both loops share some similarities.

A further analysis within the same integration difficulty factor removing the effect of sharing topics (*for* loop SU problem sets vs. *while* loop SU problem sets), or within the same topic removing the effect of sharing integration difficult factors (e.g., *for* loop SU problem sets vs. other *for* loop SU problem sets), is not supplied here, due to having insufficient problem sets to conduct such a correlation (e.g., having only one problem set to be analyzed with another problem set) and the fixed order over problem sets imposed in the study. Such a per-item level analysis probably contains a lot of noise introduced by the fixed order (e.g., a lower performance on a problem set could be due to its position), which can significantly reduce the reliability of the conclusion. I leave further investigation of this to future research.

Combining the results from both drill-down correlation analyses, I conclude that students' individual differences in integration skills and the generalizability of CEs are due to both the integration difficulty factors and topics. The results also provide additional evidence that the three factors *loop nesting*, *state update*, and *cognitive load* are playing the role of integration difficulty factors, while *loop nesting* and *state update* are playing the role of two different conceptual integration difficulty factors for identifying potential new integration skills.

One might also suggest that *loop nesting* and *state update* could be used to represent a student's higher level, more abstract integration skills, based on the results. However, one disadvantage of basing our model on such an abstracted representation is that students may lose explicit practice on different instantiations of the same integration type, using different constructs. As one can see from integration error analysis in Section 3.2.2.1, although *for&x=x+i* and *while&x=x+i* both involve *state update* and even have similar constructs (*loop* and *addition assignment*), students have exhibited substantially more cognitive load errors in the latter case. In addition, in this section, the insufficient number of problems doesn't allow for examining correlation within the same integration difficulty factor, which would tease apart the topics to provide supporting evidence. So, it is more grounded to represent integration skills on the current finer-grained level, and use the current integration difficulty factors more abstractly than the integration skills, as a bridge to help identify potential integration skills. I leave further investigation of this issue to future research.

3.3 DISCUSSION AND CONCLUSIONS

In this section, I describe the classroom study I conducted to investigate composition effects and integration skills in program comprehension for novice programmers, on three topics (*for* loops, *while* loops and *lists*). The results, on these three topics, consistently demonstrated the existence of composition effects and integration skills in common basic programming patterns. A drill-down integration error analysis revealed the nature of the composition effect and the integration skills: the difficulty can be conceptual or nonconceptual, resulting from one to multiple integration difficulty factors *loop nesting*, *state update*, and *cognitive* *load*. Further, I conducted correlation analysis between two separate pools of problems, and results suggest that students' individual differences in integration skills exist and composition effects could be generalized due to both integration difficulty factors and topics. Based on the above analysis, the two conceptual integration difficulty factors, *loop nesting* and *state update* could be used to identify potential new integration skills.

The results described in this section have several implications. First, they lay the foundation for building learner models addressing integration. For each validated integration skill, I will include it when conducting skill labeling for problems, and create a latent knowledge variable in the learner model. For unseen contexts, the identified integration difficulty factors could be used to identify potential integration skills, which could be used to reduce the user study design space, or an automated learner model's search space. Secondly, it provides insights into designing hints and trace tables for building a Python programming comprehension tutor (Chapter 6) teaching students program comprehension skills. For example, for problems requiring state update integration, a hint could be designed to emphasize retrieving the updated value from the previous iteration as input for the current iteration. Thirdly, this set of studies informs investigation of the composition effect in program construction in the future. Although comprehension and construction are very different tasks, they share conceptual knowledge, so the identified integration skills in program comprehension can serve as candidates for skill development, in the context of learning program construction.

An important next step to improve our current studies is to conduct a new study where problems sets are also permutated so that the inertial thinking errors caused by the order effect can be eliminated, and problem sets with varying, finer-grained potential difficulty factors can be compared. This will enable a cleaner investigation of the nature of composition effects and finer-grained integration difficulty factors. Moreover, increasing the scale of the current studies, in terms of participants, coverage of skills and topics and extending them to other programming languages domains would further solidify the conclusions drawn here.

4.0 BUILDING AN INTEGRATION-LEVEL LEARNER MODEL

This chapter explains how I built a learner model for integration skills. This chapter contributes to RQ 2, *How to build a learner model for integration skills?* A learner model consists of a skill model and a mechanism to infer knowledge from performance. Section 4.1 introduces how I constructed the skill model and Section 4.2 explains how I constructed the mechanism to infer knowledge, based on a Bayesian network for this skill model.

4.1 BUILDING THE SKILL MODEL

A *skill model* (defined in my dissertation) includes the item-to-skill structural relationship (mapping) and the skill-to-skill structural relationship (mapping), and is the foundation for a learner model. In this section, I describe my approach to construct these two mappings consecutively. I explain the general approach and demonstrate examples of applying the general approach to the targeted context of my later analytical and classroom studies, the topics of *for* loops and *lists*. The targeted topics serve as an example to illustrate the critical steps for building a skill model for integration-level learner models. These two topics have been chosen because they cover most common basic programming patterns.

4.1.1 Item-to-Skill Mapping

Integration skills are defined in a context, i.e., only when basic skills interact in specific ways can certain integration skills be associated with an item. Fortunately, programs give highly structured and abstract contexts for skill applications, and there are many program analysis techniques available. I hereby proposed a general *automated potential integration skill identification algorithm* (APIS) using automated program analysis based on contextual features, to construct item-to-skill mapping. APIS labels the required potential integration skills (if any) for each item (a problem or a step), and outputs a mapping from items to potential integration skills in the targeted study context. APIS requires two inputs:

- *conceptual integration difficulty factors* (CIDF) with *contextual features* describing them.
- *aspects* for describing potential integration skills which constitute *contextual features* describing integration skills.

Contextual features consist of the *necessary features* which must appear in an item, and *unwanted features* which should not appear in an item. These instantiated features, along with the *aspects* for each skill, constitute a rule. These two inputs can be obtained by directly consulting domain experts, or conducting Difficulty Factors Assessment (DFA) studies. A DFA study may be a more objective and reliable way to obtain the inputs, compared to directly consulting domain experts, since student performance data is collected to examine hypothesized difficulty factors, reducing expert bias and expert blind spots [Clark et al., 2007]. In my dissertation, I have conducted a DFA study, abstracted two conceptual difficulty factors and identified a set of aspects for describing (potential) integration skills.

To conduct automated program analysis, APIS performs string or regular expression matching, or checks variable states, using the debugger library of the programming language[1]. The parser or AST tree of a program can be further utilized for identifying potential complex integration skills. In my dissertation, I demonstrate how simple automated program analysis and a small set of rules are sufficient in my study context.

Before moving on, one aspect worth mentioning is the assessment unit that APIS operates at. Depending upon the assessment unit, the rules (or aspects) differ slightly. ITSs can be classified into two types, according to the assessment unit level. The first type places the assessment on the *problem level*, where only the final answer to a problem is evaluated by the system and used for learner modeling, although a fine-grained level of hints or decomposed scaffolding of problems might still be provided. Examples include the SQL-Tutor [Mitrovic

[1]I used the library from Online Python Tutor (https://github.com/pgbovine/OnlinePythonTutor/).

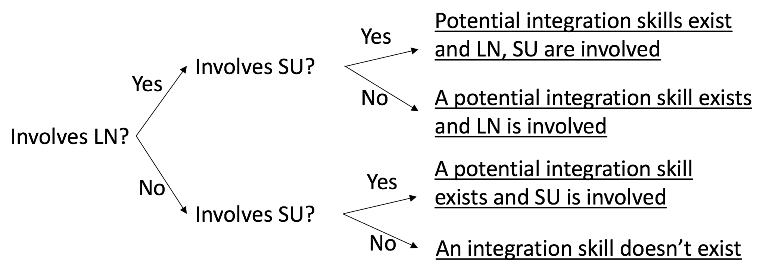

Involves LN?
Yes
No
Involves SU?
Involves SU?
Yes
No
Yes
No
Potential integration skills exist and LN, SU are involved
A potential integration skill exists and LN is involved
A potential integration skill exists and SU is involved
An integration skill doesn't exist

Figure 6: An example of a decision tree for judging the existence of potential integration skills and CIDFs in my study context (LN: loop nesting, SU: state update).

et al., 2001], ASSISTment [Heffernan and Heffernan, 2014] and QuizGuide [Brusilovsky et al., 2004b]. The second type places the assessment on the *step level*, where problems are decomposed into steps (according to the underlying model-tracing process). Step level answers are evaluated and step level assistance is provided by the system. Examples include Cognitive Tutor [Anderson et al., 1995] and Andes [Vanlehn et al., 2005]. I refer to an assessment unit (a problem or a step) as an item. APIS can operate at the problem level or the step level assessment unit.

APIS works as follows. For each item (to be labeled) in a targeted study context:

1. Judge the existence of potential integration skills by CIDFs and examine what CIDFs are involved for the problem. For example, Figure 6 shows an example of using *loop nesting* (LN) and *state update* (SU) discovered by my previous DFA study (Chapter 3) to judge whether potential integration skills exist or not and examine what CIDFs are involved in common basic programming patterns (among *for* loops, *while* loops, *lists* topics). Given definitions of CIDFs which encode features describing them (e.g., Table 7), an automated program analysis could be applied. If the item doesn't involve any of the factors, move to the next item; otherwise, continue to the next step.

2. For each matching CIDF, extract values of *aspects* from the problem (when the item is a problem) or from both the problem and the current step (when the item is a step) and look up the rule table to check whether any rules are matched. If a rule is matched, label the corresponding skill; otherwise, add a new rule to the table. For example, Table 11 lists the final rule table involving SU for problems in my study context.

Table 11: An example of the final rule table involving *state update* problems in my study context. *Operator* refers to the operator (which is not the assignment operator) used in the line where an assignment occurs or the retrieval of the previous value occurs.

Involves *for* loop?	What is being updated?	What is used to update?	Operator?	Potential integration skill
Yes	x (number)	x (number)	+, *	for&#=
Yes	x (number)	a[i] (list element)	$<, >, \leq, \geq$	for&if
Yes	a[i] (list element)	a[i] (list element)	+, -, *	for&a[i]
No	a (list)	a[i] (list element)	+, -, *	a[i]&=

APIS enables generalization, i.e., APIS can use CIDFs to identify new potential integration skills. For example, in Table 11, a new potential integration skill *for&#=* is identified because it involves the *state update* CIDF, and it is a natural extension to *for&+=*. It was identified in Chapter 3 by including the multiplication assignment[2] (*for&*=*) situation. In addition, another new potential integration skill *a[i]&=* is identified because it involves the *state update* CIDF for *list* and slightly differs from *for&a[i]* in that it requires writing the complete value of a list rather than one list element, and focuses on whether a list can be updated correctly, regardless of whether list elements in later positions depend on list elements in former positions or not.

Identifying this kind of potential integration skills can inform the design of new empirical user studies (e.g., my Classroom Study 2 in Chapter 8), or narrow down the search space for a machine learning model to discover integration skills, the results of which can also serve as further validation of potential integration skills.

[2]Similar to the *addition assignment* defined in my dissertation, an assignment like x=x*i is called a *multiplication assignment* although strictly speaking only an assignment with the *= operator (e.g., x*=i) should be called so. I defined it in this way because the courses had only been taught in the former way, by the time my studies were run, and both ways in essence share the same state update structure.

Table 12: (Potential) integration skills for my study context.

Name	Overall definition (being able to get ...)
for&for	A sequence of numbers resulting from a nested *for* where the outer loop iteration variable decides the number of inner loop iterations. In my study, since all instances of nested loops interact in this way, the subscript $v2$ is ignored.
for&#=	The sum or product of a sequence of consecutive numbers with a *for* loop and an addition or a multiplication assignment.
for&if	The maximum or minimum value of a list with a *for* loop. Although the name doesn't include *a[i]*, the context of list is required.
for&a[i]	List values after creating a list of numbers where the later numbers depend on the former numbers (e.g., a list of a Fibonacci sequence) with a *for* loop.
a[i]&=	List values when some list elements have been updated.

Table 13: Basic skills for my study context.

Name	Overall definition (being able to get ...)
for	The sequence of values of the iteration variable.
#=	The value of a variable after its addition or multiplication assignment(s).
if	The value of the condition expression, and enter/skip the body correctly.
a[i]	The value of a list element with the index involving a variable (e.g., i, i-1, i+2).
%	The value of an expression with the modulus operator.
//	The value of an expression resulting from a floor division.

By adding *aspects* for basic skills, APIS could be converted to *automated potential skills identification algorithm* (APS) for labeling both integration skills and basic skills. Definitions of skills could be obtained by converting the features into descriptions. A skill set could be obtained by gathering the pool of skills labeled for a set of items. Table 12 and 13 list the definitions of (potential) integration skills and basic skills in the context of my study.

4.1.2 Skill-to-Skill Integration Graph

Another component in the skill model is the skill-to-skill mapping. My dissertation focuses on the integration relationship. I introduce a new type of knowledge graph, the *integration graph*, which shows how component skills progressively integrate and form integration skills that are essential to describe domain expertise (Figure 7). Lower levels consist of more basic component skills and higher levels consist of integration skills integrating previous levels' skills. Although my dissertation focuses on 2-component integration skills, integration graphs could be built for more complex integrations. Nodes are created for each skill with a skill tag (e.g., *for&x=x+i*), and a description (e.g., *get the sum of numbers with a for loop and an addition assignment*). Edges denote component-integration prerequisite relationships.

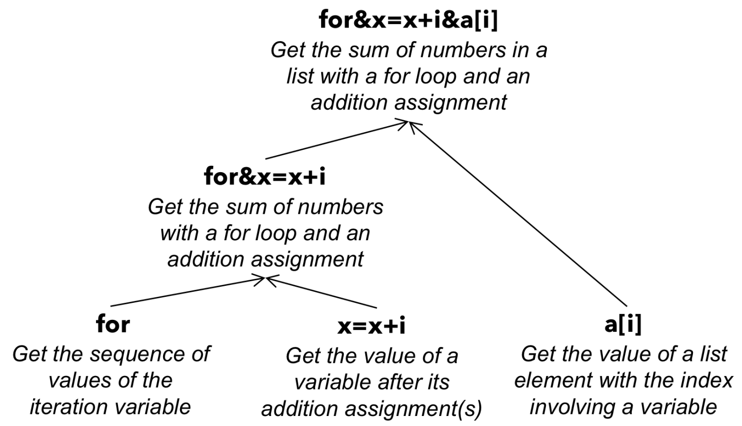

for&x=x+i&a[i]
Get the sum of numbers in a list with a for loop and an addition assignment
for&x=x+i
Get the sum of numbers with a for loop and an addition assignment
for
Get the sequence of values of the iteration variable
x=x+i
Get the value of a variable after its addition assignment(s)
a[i]
Get the value of a list element with the index involving a variable

Figure 7: An example of an integration graph showing how basic skills *x=x+i*, *for*, and *a[i]* integrate and form more complex integration skills.

One important principle for constructing an integration graph is that an integration skill is only included if a composition effect and integration conceptual difficulty exist. Otherwise, we end up with numerous ungrounded "integration skills" many of which students can naturally acquire once they have learned basic skills. For example, in Figure 7, the integration skill *for&+=* is included because a composition effect is found as shown in Chapter 3. However, *iterating through a list and printing its elements with a for loop* or *adding a list element to a variable with addition assignment* hasn't been proven to have a composition effect, so the hypothesized integration skills are not included.

Admittedly, to construct a reliable, complete integration graph, a sufficient number of DFA studies may be needed. Alternatively, when student performance data is available, one could construct a potential integration graph and then use fitness or predictive metrics computed from the data to tune or validate the learner model that is built on the foundation of that integration graph. However, this may require data with sufficient numbers of and diversity in problems.

As a general future direction for building a skill model with integration skills, one could collect and analyze data from MOOCs or large courses, which provide a large number and high variety of problems and students, as well as exploring machine learning approaches.

4.2 BUILDING THE BAYESIAN NETWORK GIVEN THE SKILL MODEL

In this section, I introduce how to construct a Bayesian network that is based on the skill model (which was constructed using the approaches introduced in the above Section 4.1) for integration skill modeling. Compared to other machine learning approaches, I chose the Bayesian network (BN) technique for learner modeling because it offers a natural mechanism to convert the given skill model, by encoding existent domain knowledge into a network structure. Also, it maintains interpretable knowledge states which can be directly used for diagnosis and adaptive decisions. The construction of the Bayesian network, including decisions about its structure, parameters and update mechanism are explained as follows.

4.2.1 Learner Model Structure and Parameters

Figure 8 shows the structure of an integration-level learner model based on a Bayesian network. The key features of such a BN are explained as follows:

- Basic component skills and integration skills are separately represented by different nodes. A basic component skill node (K_i^b) represents the basic understanding and application of a basic component skill (e.g., *conducting a simple addition assignment*). An integration skill node ($K_{i\&j}$) represents the understanding and application of an integra-

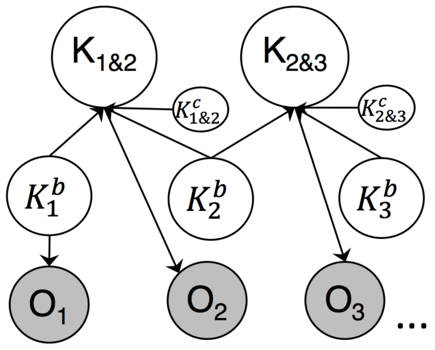


Figure 8: The BN structure of an integration-level learner model. A specific formulation, CKM-HI, which assumes conjunctive skills for item performance and uses a noisy-AND gate, shares the same structure as the one shown in the graph. O nodes represent the binary observed student performance, and K nodes represent binary latent skill knowledge levels.

tion skill (e.g., *getting the sum of numbers with a for loop and an addition assignment*). This separation of two kinds of skills and explicit representation of integration skills help to provide targeted practice, and to assure full practice coverage.

- Each integration skill node has its own parent node ($K^c_{i\&j}$ for $K_{i\&j}$), which denotes the conceptual part of the integration skill. The construction of $K^c_{i\&j}$ nodes was informed by the DFA study in Chapter 3, where results clearly showed that students have conceptual misunderstanding or are missing conceptual understandings of how basic skills integrate together, even if they already know how each separate, basic component skill works. Such nodes in BN indicate that the level of integration depends not only on the levels of basic component skills, but also on the conceptual understanding of each specific integration.
- Integration skills (if not serving as components for more complex integration skills) are directly connected to items, and edges from more basic component skills to items are removed if their integration skills are required. For example, in Figure 8, O_2 requires the integration skill $K_{1\&2}$, so the edges from K^b_1 to O_2, from K^b_2 to O_2 are removed. In this way, remediation can directly operate at integration skill levels. This is different from granularity-based networks [Collins et al., 1996, Conati et al., 2002, Millán and Pérez-De-La-Cruz, 2002] including competency-based networks [Mislevy and Gitomer,

1995, Morales et al., 2006] where higher level nodes represent aggregation or abstraction, not integration, of lower level skills and aren't directly connected to items. As a result, remediation can't directly operate at such higher levels.

- Latent skills are organized in a hierarchical way capturing the dependencies among skills. Lower levels consist of more basic components skills, and higher levels consist of integration skills that require the integration of lower-level skills. This hierarchical structure allows knowledge increase/decrease from one skill node to propagate to other skill nodes. This helps reduce over-practicing basic skills when students already know integration skills, and also helps reduce over-practicing very similar integration skills when students already know an integration skill along with its basic skills.
- Multinomial (here, binomial) distributions are used as conditional probability distributions for integration skills, where the basic component skills and conceptual parts of integration skills serve as parents, whose knowledge state combination determines the state of the integration skill. Multinomial distributions allow component skills to have different levels of importance to the integration, and have generated a better model than noisy-AND gates in my preliminary analytical cross-validation experiments, while maintaining a relatively low time and space complexity for 2-component integration skills. However, for highly complex integration skills requiring a larger number of components as parents, noisy-AND gates should be considered to keep the complexity linear rather than exponential.

In my thesis, I construct *Conjunctive Knowledge Modeling with Hierarchical Integration skills* (CKM-HI), which has the key features listed above, but further assumes a conjunctive effect of skills for item performance and uses a noisy-AND gate to model this conjunctive relationship. The assumption of the conjunctive relationship is suitable when each problem has only one solution, which requires that students know all of the underlying skills (rather than using alternative skills or only some of the skills), which is the case in my context of program comprehension as well as in many other contexts [Embretson, 1997, Cen, 2009, Carmona et al., 2005, Conati et al., 2002, VanLehn et al., 1998]. Furthermore, the use of a noisy-AND gate to model the conjunctive relationship in BN is beneficial when problems require a relatively large number of skills (e.g., more than 3) so that computational complexity can be reduced [Carmona et al., 2005, Conati et al., 2002, VanLehn et al., 1998]. However, the core of an integration-level learner model is not about the conjunctive or compensatory relationship chosen for depicting a multi-skill effect on item performance, but about the representation of skills, skill-to-skill relationships, and whether skills are directly linked with an item or not, i.e., the key features summarized previously. Whether other formulations (in other contexts) would be better suited than CKM-HI, is left for future investigation.

As mentioned before in Section 4.1.1, there are two granularity levels of assessment units that exist in the current tutoring systems, problem level and step level, which results in two granularity levels of observables and leads to two density levels of the item-to-skill relations. Using problem-level observables, CKM-HI has quite a complex (dense) structure between skills and items. It utilizes its specified conditional probability tables and Bayesian rules to manage the uncertainty underlying multi-skill practices. Then, using step-level observables, CKM-HI reduces the complexity (density) of the structure between skills and items. This likely reduces the level of uncertainty and increases the accuracy of the knowledge inference. Where the observables are on the problem level or the step level, CKM-HI maintains the key features of an integration-level learner model, as summarized above.

The next step for building a learner model is to specify/obtain values for the parameters. CKM-HI has three sets of parameters: 1) the probabilities of a student initially knowing a skill ($init/l_0$), 2) the conditional probability tables (CPTs) for integration skills, and 3) two noise parameters indicating the probability of correctly answering an item by chance ($guess/g$), and the probability of accidentally failing a known item ($slip/s$). Such parameters have been gathered from collected student interaction data. Since there are latent variables, the expectation maximization algorithm is used. Also, a standard junction-tree algorithm is used to conduct the exact inference. However, when there is no pre-existing student interaction data, the setting of these parameters depends on expert knowledge or the requirements of the study design. Then, with the incoming student interaction data in an online setting, each student's own BN will update the beliefs (estimations) of the skill levels, which will be explained in the following section.

4.2.2 Dynamic, Individualized Knowledge Update

Given the network's structure and parameters, one can use the network to predict item performance, and infer the knowledge level of each skill at each practice opportunity for a student. Different BNs are maintained for different students as their own learner models. All students' BNs are initialized with the same parameters, so for a student's first practice, the same prior probabilities (obtained by $init/l_0$ parameters) for latent skill nodes are used to predict performance; but after observing different students' practice sequences, the BNs start to differentiate among students by maintaining different up-to-date knowledge estimates. In order to achieve this, CKM-HI follows the same dynamic BN roll-up mechanism as in [Conati et al., 2002]: it uses posterior knowledge probabilities conditioned on historical observations as the priors for the next time steps. Currently, CKM-HI doesn't model the transition probabilities of latent skills between time steps (slices). There are three reasons why I consider such a static BN rather than a fully dynamic BN to be sufficient for my work:

- The change in knowledge estimates is mainly determined by new evidence (i.e. observed performance), since the learning gain from each practice will be ultimately translated into an observed performance serving as the evidence for updating the student's knowledge beliefs. This is suggested by prior work [Conati et al., 2002].
- Constructing a fully dynamic BN requires estimating posteriors for each latent variable at each time slice for each student, in an iterative EM setting. This significantly increases time and space complexity, and requires more data for reliable inference.
- According to the results in my prior work [Huang et al., 2017], such a static BN mechanism indeed achieves good performance, and it is better than a (simple) dynamic BN using HMMs (WKT).

I leave incorporating learning dynamics for future work, which is a non-trivial task for a network which has a hierarchical structure between the latent variables.

5.0 A MULTIFACETED EVALUATION FRAMEWORK FOR LEARNER MODELS

In this chapter, I introduce a general, multifaceted evaluation framework for learner models, in order to answer RQ 3, *How to evaluate a learner model?* There are two main motivations for developing this framework: 1) there has been a growing concern from various communities (EDM, UMAP, ITS, AIED) about evaluating learner models from a single aspect, 2) multi-skill learner models, such as my proposed integration-level learner model, posit challenges to traditional predictive performance metrics, most likely due to their flexibility in the parameter space. This multifaceted framework is applied later to evaluate my proposed integration-level learner model, in answer to RQ 4. This is one of the first unified frameworks to evaluate learner models and extends our two prior frameworks [Huang et al., 2015, González-Brenes and Huang, 2015]. Compared to our prior work [González-Brenes and Huang, 2015], it is applicable to a broader context, including both single- and multi-skill practice situations. This new framework contains two parts: 1) data-driven evaluations, and 2) real-world intervention study evaluations. Under the data-driven evaluations, there are multiple dimensions: predictive performance, parameter plausibility, and expected instructional effectiveness. Equal weights are given to these different dimensions. In the following sections, I will introduce these dimensions.

5.1 DATA-DRIVEN EVALUATIONS

This section introduces the three dimensions for data-driven analytical evaluations of learner models. The usage of various dimensions is to address the growing concern about using predictive performance as the only aspect to evaluate learner (including skill) models in the EDM community [Baker et al., 2008, Beck and Chang, 2007, Gong et al., 2010, González-Brenes and Huang, 2015, Huang et al., 2015], and to address the long-standing goal of finding a holistic evaluation method in the adaptive system community [Brusilovsky et al., 2004a, Paramythis et al., 2010]. A more thorough review, expressing the motivation behind using different dimensions, can be found in Chapter 2.

5.1.1 Predictive Performance

Predictive performance is the first dimension in this proposed evaluation framework. Predictive performance through cross-validation has long been a golden standard for evaluating learner models in the EDM community, and is the first dimension in my evaluation framework. For latent variable-based learner models, such as knowledge-tracing-based models (including CKM-HI, CKM, WKT), predictive performance indirectly assesses the accuracy of the latent knowledge inference, through testing the accuracy of the predictions.

While a wide range of prediction metrics could have been considered under this framework, in this dissertation, I demonstrate the use of the most relevant ones. I chose two popular prediction metrics used in evaluating learner (skill) models, the root mean squared error (RMSE) and area under the Receiver Operating Characteristic curve (AUC). The RMSE metric has demonstrated a high correlation to the 'moment of learning' (i.e., the step in which a student learned a skill) which could significantly affect student under-practice and over-practice for the BKT models [Pardos and Yudelson, 2013, Baker et al., 2011].

The AUC metric measures quite a different aspect of a learner model from that measured by RMSE. It assesses the models' abilities to discriminate each student's performance failures from successes, considering a range of decision thresholds for prediction, even if the data is imbalanced. The need to report both metrics is based on thorough investigations of predictive metrics in recent papers [Pardos and Yudelson, 2013, Pelánek, 2015], which raised concerns about using only AUC for evaluating learner (skill) models.

In addition, I also report the recall and precision metrics for incorrect outcomes. [Pardos and Yudelson, 2013] demonstrated that recall and F-measure for correct outcomes have a high correlation with the moment of learning in BKT simulation studies, yet a less explored aspect in the EDM field is the effectiveness of the corresponding metrics for incorrect outcomes. I argue that the recall for incorrect responses, which reflects the ratio of incorrect responses that can be identified by the learner model, is the most important of all the recall/precision metrics for incorrect/correct outcomes, since it is of primary concern that an ITS doesn't miss any knowledge/skill weakness that is reflected in each problem failure of a student.

5.1.2 Parameter Plausibility

Parameter plausibility is the second dimension of my proposed evaluation framework, and is inherited from my prior work [Huang et al., 2015]. The definition of parameter plausibility, under my framework, consists of two aspects. Firstly, the parameters of a learner model shouldn't violate common human understanding about learning (which typically is reflected in the learner model's underlying assumptions about learning). This has been pointed out by prior work [Baker et al., 2008]. For example, knowing required skills generally leads to successful performance, and not knowing any required skills generally leads to a failing performance [Baker et al., 2008]; practice shouldn't hurt learning [Newell and Rosenbloom, 1981]. Secondly, the parameters of a learner model should enable clear, explicit and meaningful interpretations of a student's cognitive states, which has also been pointed out by prior work [Gong et al., 2010, Beck and Chang, 2007]. The plausibility of the estimated parameters is of crucial importance, because it determines the plausibility of how latent knowledge is inferred, and ultimately affects the students' learning experience. For example, if a learner model has a high *slip* parameter, then even after observing a lot of incorrect responses, it will estimate that students already learned and fail because of slipping; at the same time, a model with a low *slip* parameter will estimate that students haven't learned because it is very unlikely they will fail if the student has already learned it. Under the first model, student practice will stop much earlier than under the second model, risking that many students will not have enough practice to learn the skill.

There are two sources that affect parameter plausibility. Firstly, students could exhibit noisy behaviors, affecting real-world performance data, such as guessing the answer correctly even if the student doesn't know the skills, or making careless errors even though the student already knows the skills. Such noisy behaviors could affect the fitting of noise parameters to learner models. Secondly, a skill model could mis-specify underlying skills which also affects the fitting of parameters. For example, if a skill model fails to identify several difficult skills of an item and students mostly reach a high knowledge level of the identified easier skills when facing this item, the learner model based on that skill model might fit high *slip* parameters to explain the high ratio of incorrect performance on this item that is observed in the data, instead of identifying the missing skill.

Under my proposed, flexible evaluation framework, parameter plausibility can be evaluated in various ways, depending on the learner model involved. My dissertation focuses on knowledge-tracing based learner models with *guess* and *slip* noise parameters. I propose two metrics, *guess* and *slip*, the value of which corresponds to the value of two parameters, to describe the parameter plausibility of such learner models. From the perspective of using student performance to predict/approximate knowledge, *guess* could be considered to be a *false positive*, and *slip* a *false negative*. For two learner models with statistically the same predictive performance, as evaluated by cross-validation on the same dataset, the model fitted with a smaller *guess* and *slip* is considered to be a model with a higher parameter plausibility, because student performance could be explained explicitly and clearly by knowledge levels of skills rather than with noise parameters, raising the explaining power of the learner model.

Admittedly, one might argue that certain levels of *guess* and *slip* are needed to reflect/express a student's true noisy behaviors. Also, it might be possible to give theoretical upper bounds for *guess* (using assumptions), such as giving an upper bound of 0.25 of *guess* for multiple-choice questions with four choices. Yet it is non-trivial to solidly specify the upper and lower bounds of *guess* and *slip* in a general setting, regarding dataset and item characteristics. I leave the investigation of this issue as future work.

5.1.3 Expected Instructional Effectiveness

Expected instructional effectiveness (EIE) is the third dimension of my proposed evaluation framework. The ultimate goal of a learner model is to improve *instructional effectiveness*, i.e., whether students can acquire the targeted skills without wasting any effort. Usually, one deploys the learner model in a real system to examine this, yet considering the cost and constraints of deployment, is there an analytical way to forecast the expected instructional effectiveness, in order to further improve a learner model or eliminate some learner models before final deployment? When there is no intention to deploy a learner model further, is there a way to estimate the instructional effectiveness of learner models? Unfortunately, the above-mentioned predictive performance and parameter plausibility metrics still wouldn't be able to give direct answers to these questions. Our prior framework, the *learner effort-outcomes paradigm* (LEOPARD) [González-Brenes and Huang, 2015] offers a way to quantify the amount of effort required to achieve a learning outcome. It is designed for single-skill learner models.

In my dissertation, I extend this framework from a single-skill to a multi-skill learner model evaluation, from a single hypothesized mastery threshold to a range of mastery thresholds, allowing for a more systematic joint examination of effort and scores. I call this new, general evaluation metric, *expected instructional effectiveness* (EIE). The key idea of EIE is to examine student effort and performance by simulating different instructional decisions (i.e., when to stop instruction), based on real-world data, according to the learner model's knowledge inference. In my thesis, I use the test data to compute the metric, after fitting a learner model from the training data. The EIE is based on the calculation of two metrics, *effort* and *score* and is reflected in a *score-effort* curve across a range of mastery thresholds. The details are explained as follows.

5.1.3.1 Score

This metric empirically quantifies the expected performance of students when they reach mastery of skills inferred by a learner model. For a given mastery threshold and a given student, the metric is calculated by the mean performance (accuracy) of the student when the learner model asserts that the student reaches the given mastery threshold for all of the required skills for a current item, by comparing the estimated probabilities of the student knowing the skills with the threshold, based on collected real-world data. The original metric suggested in our prior work [González-Brenes and Huang, 2015] only applies when each item maps to a single skill by simply examining the performance sequence of a skill. It is not applicable to multi-skill practice situations, since the responsibility of each skill for the final performance is not clear, i.e., if we convert the item level performance to single skill level performance, it is not clear what correctness should be assigned. To address this, my thesis introduces an extension of the original metric by jointly examining the multiple skill knowledge states.

The following formula explains the computation of the score averaging over a set of students U for mastering the set of skills Q given a mastery threshold m:

$$\begin{aligned} Score_{m,u} &= \frac{\sum_{1\le t\le|\mathbf{O}_u|}\prod_{q\in Q_{u,t}}\mathbf{I}(K_{u,q,t}\ge m)\cdot\mathbf{I}(O_{u,t}=1)}{\sum_{1\le t\le|\mathbf{O}_u|}\prod_{q\in Q_{u,t}}\mathbf{I}(K_{u,q,t}\ge m)} \\ Score_m &= \frac{1}{|U|}\sum_{u\in U} Score_{m,u} \end{aligned} \tag{5.1}$$

$\mathbf{O}_u$ denotes a student u's observed practice sequence; $Q_{u,t}$ denotes the set of direct parent skills of the item corresponding to $O_{u,t}$; $K_{u,q,t}$ denotes the estimated probability of student u knowing a skill q right before t^{th} observation (i.e., before being updated by this observation); U denotes the set of students on the dataset; $\mathbf{I}$ denotes an indicator function taking value 1 if the condition in the bracket is satisfied and 0 otherwise.

5.1.3.2 Effort

This metric empirically quantifies the expected number of practice opportunities that are needed to reach mastery for all skills under investigation inferred by a learner model. For a given mastery threshold and a given student, it is calculated by counting the number of practice opportunities the student needs in order to reach the given mastery threshold for a set of targeted skills, by comparing the estimated probabilities of the student knowing the skills with the threshold, based on collected real-world data. This metric was originally suggested in our prior work [González-Brenes and Huang, 2015], but this dissertation has adjusted it for multi-skill practice cases: integration items are considered in computing the effort for integration skills, but not considered in computing the effort for corresponding basic component skills or conceptual parts of integration skills, so that effort already considered in integration skills won't be repeatedly counted. Meanwhile, integration items are still considered for computing the effort for non-corresponding basic component skills, and items solely for practicing basic component skills are still considered for computing the effort for basic component skills.

The following formulas explain the computation of the effort averaging over students for mastering the set of skills Q, given a mastery threshold m:

$$\begin{aligned} \mathit{Effort}_{m,q} &= \frac{1}{|U_q|} \sum_{u \in U_q} \sum_{1 \leq t \leq |\mathbf{K}_{u,q}|+1} \prod_{1 \leq t' \leq t} \mathbf{I}(K_{u,q,t'} < m) \\ \mathit{Effort}_{m} &= \sum_{q \in Q} \mathit{Effort}_{m,q} \end{aligned} \tag{5.2}$$

$\mathbf{K}_{u,q}$ denotes the sequence of estimated probabilities of a student u knowing a skill q right before each practice that directly requires this skill, which has the same length of the student u's direct practice sequence for the skill q; $K_{u,q,t'}$ denotes the estimated probability of a student u knowing a skill q right before t'^{th} observation (i.e. before updated by this observation); U_q denotes the set of students on the dataset that have ever practiced on the skill q; $\mathbf{I}$ denotes an indicator function taking value 1 if the condition in the bracket is satisfied and 0 otherwise. Note that when computing $\mathit{Effort}_{m,q}$, index t is up to $|K_{u,q}|+1$ because the updated knowledge states that after the last observation is considered, i.e., if a student still hasn't reached mastery of a skill after the last practice, then he/she still needs at least one more practice.

As shown in the formula, to compute $\mathit{Effort}_{m,u,q}$ for a student u, we incrementally examine the estimated knowledge levels of skill q for each position t, and if $K_{u,q,t'}$ for the current and all previous observations of t hasn't reached the mastery threshold, i.e., $\prod_{1 \leq t' \leq t} \mathbf{I}(K_{u,q,t'} < m) = 1$, the effort value adds 1; otherwise no effort is added.

5.1.3.3 Mastery Thresholds

A mastery threshold determines when an ITS supported by a learner model stops (focused) instruction on a skill, and moves on to the next skill(s). Conventionally, 0.95 is chosen as the mastery threshold probability, yet to the best of my knowledge, only one work has provided justification for this threshold and it based this on simulated data [Fancsali et al., 2013]. It is still unclear whether this 0.95 threshold in a real-world system could lead to better learning than alternative thresholds. As demonstrated by [Fancsali et al., 2013], a mastery threshold can be treated as a parameter that can be tuned according to course preference or constraints, in the sense of trading off false positives (in which a student without knowledge is judged to have mastered a skill) for false negatives (in which a student is presented with additional practice opportunities after acquiring knowledge). Thus, a *good* mastery threshold might also depend on the context in which the learner model is deployed. This work and the scarcity of work on this topic suggest that more attention needs to paid to understanding the effect of a mastery threshold, how to set it in a given context, or even to consider methods that do not require a deterministic mastery threshold when eliciting instructions from the learner model, such as the predictive similarity policy [Rollinson and Brunskill, 2015] and the method proposed in [VanLehn et al., 1998].

Here, as a strategy with the least assumptions made, I consider a full range of mastery thresholds in the range from 0.5 to 0.99, at intervals of 0.01 which improves over our original framework [González-Brenes and Huang, 2015] of using a heuristic 0.6 as the threshold. Although primarily aiming at examining the effort and scores under a range of situations, this treatment also helps to explain the effect of mastery thresholds.

5.1.3.4 Imputation

As can been seen in the above score metric, at each mastery threshold m, only observations in which the required skills reached the mastery threshold will be examined. On high thresholds and datasets without sufficient practice, there may be insufficient data for computing the score. This could render the metric susceptible to outliers and reduces the reliability of the estimation. Following the original LEOPARD framework [González-Brenes and Huang, 2015], I conducted imputation as follows: for students who have never reached the mastery threshold for any skill, the student's average performance on all his/her observations will be used as the score. Regarding effort, for students who have never reached the mastery threshold for a skill, the effort value will be the number of observed student practice opportunities on this skill, and for students who have never had practice on a skill, they are excluded when computing the average effort for that skill. Such treatments on computing the effort happen because it is not straightforward to impose imputation. In the future, I will consider using the fitted learner model to forecast/project future effort not observed from the data, similar to the method used in [Lee and Brunskill, 2012, Yudelson and Koedinger, 2013].

As can be seen, both score and effort metrics were computed after the students reached a mastery threshold. So EIE is more suitable for datasets where students haven't been stopped from practicing by using pre-defined mastery thresholds, because in these datasets a full range of simulated situations can be examined with sufficient data for the chosen learner model. This is actually the case for the dataset used in my dissertation.

5.1.3.5 Summative Score-Effort Curve

Only the effort or only the score metric isn't enough to indicate the expected instructional effectiveness. Consider a learner model that tends to overestimate students' knowledge levels but on which students still frequently fail after the model has asserted mastery. Although the expected effort to reach mastery will be low, the expected score will also be low. Such a learner model will not be preferable in an adaptive tutor system, since it risks leading students toward under-practice. However, prior work that examines instructional effectiveness on real-world datasets has only computed the expected effort (number of practice opportunities) needed to reach mastery, assuming mastery is achieved when the inferred knowledge level from a chosen learner model reaches a chosen threshold (e.g., 0.95) [Gong et al., 2010, Koedinger et al., 2011, Lee and Brunskill, 2012, Yudelson and Koedinger, 2013].

In this framework, scores and efforts are jointly examined, and further, they are examined by a summative score-effort curve, which extends our original framework [González-Brenes and Huang, 2015]. Previously, the score and effort were only examined at one threshold. To plot the curve, the score and effort under the same mastery threshold are paired as a dot with x=effort and y=score, and such dots are ordered from low to high mastery thresholds;

then a curve is plotted to connect these dots. Using the common assumption in the ITS field that practice doesn't hurt learning [Newell and Rosenbloom, 1981], increasing effort should result in a non-decreasing score, but due to noise in the data and the imputation to address insufficient data, a score-effort curve might fluctuate. To construct a more reliable score-effort curve, a smoothing rule is imposed: the score on a new mastery threshold is set as the largest value of all the scores from the current and previous smaller mastery thresholds. This results in an adjusted, non-decreasing score-effort curve. The original curve has been shown in the graph with a light grey color. Figure 9 shows an example.

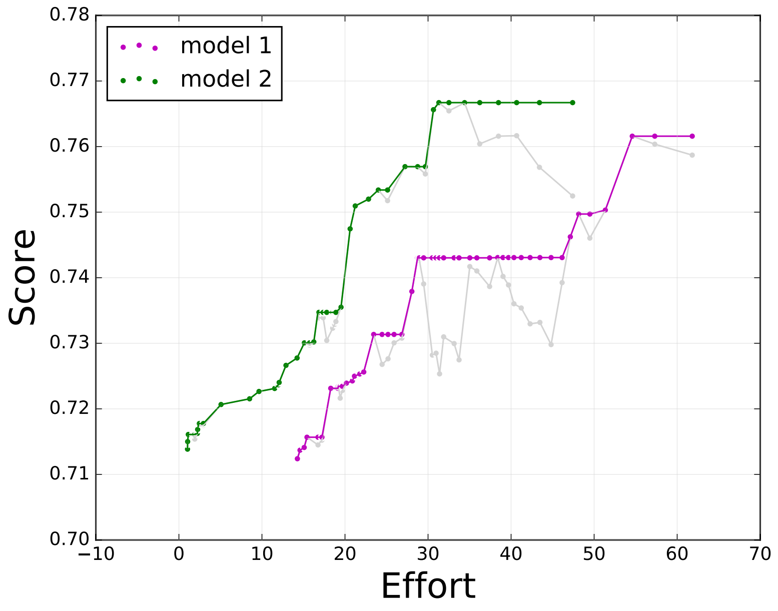


Figure 9: Score-effort curves for two learner models. Grey curves are non-adjusted curves.

To use the curve to compare learner models to find the one with the best learning effectiveness, choose the curve with the larger area under the curve or the steeper slope, which indicates that the learner model has a higher expected instructional effectiveness (EIE): using the same effort, students under this learner model are more likely to have a higher score (performance); or when reaching the same score, students under this learner model are more likely to use less effort (the number of practice opportunities). For example, in Figure 9 , model 2 is evaluated as having a better EIE than model 2. Currently, one still needs to visually inspect the curve to judge the expected instructional effectiveness, so a future step would be to convert this curve into a single metric, similar to the AUC metric for the receiver operating characteristic (ROC) curve, or the *instructional effectiveness measure* [Paas and Van Merriënboer, 1993], but it needs a more thorough investigation beyond the scope of my dissertation.

5.2 REAL-WORLD INTERVENTION STUDY EVALUATIONS

The ultimate goal of learner modeling is to improve student learning, and the ultimate way to examine whether a (new) learner model leads to better learning is to deploy it in an ITS, and compare the impact on student learning between this redesigned ITS and the original ITS, in an experimental study. Such a study is called a *real-world intervention study*, or a *close-the-loop* study (as it completes the "4d cycle" of system design, deployment, data analysis, and discovery–leading back to design) [Koedinger et al., 2013] in my dissertation. Unfortunately, in the EDM community where new learner modeling approaches are actively proposed (and accepted) once an increase in predictive performance is achieved, corresponding close-the-loop studies examining their real-world effectiveness are rarely performed.

To examine the real-world effectiveness of an intervention (e.g., a new learner model), the most widely-used measurement is student learning gain, as measured by the difference between posttest and pretest scores. If pretest scores from the control and experimental groups are statistically the same, posttest performance can also be directly compared. Higher learning gain or posttest performance (controlled for pretest performance) is a clear evidence of the effectiveness of an intervention, as compared with its original design. Note that learning gain is usually compared when time is controlled to be the same for both conditions.

Another measurement that should be paid more attention to is student learning time, or posttest time. If students under a redesigned ITS use less learning time or solve the posttest faster with the same posttest performance, compared with the original ITS, then the new intervention (e.g., the learner model) should be considered as more effective. An example is the accelerated learning effect achieved by the SHERLOCK system, which reduced four-year's troubleshooting experience to only seven days of practice [Lesgold et al., 1988].

One concern of evaluating a learner model deployed in an ITS is that other components such as the problem selection approach could affect the overall effectiveness, too. So, this evaluation should also be combined with the above data-driven evaluations which focus on only the learner model component. This issue has been pointed out by the proponents of *layered evaluation*, who argue that holistic evaluation should be complemented by approaches that independently assess each layer [Brusilovsky et al., 2004a, Paramythis et al., 2010].

5.3 DISCUSSION AND CONCLUSIONS

The main concept of my multifaceted evaluation framework is that a learner model should be examined from multiple perspectives beyond the traditional comparison of predictive performance, and should include both data-driven evaluations and real-world intervention study evaluations. Each dimension included in this framework has been supported by prior work with clear groundings [Pelánek, 2015, Baker et al., 2008, Beck and Chang, 2007, Van de Sande, 2013, Lee and Brunskill, 2012, Huang et al., 2015, González-Brenes and Huang, 2015].

However, the metrics used in my framework can be further improved. For example, regarding the EIE metric, it would be helpful to summarize the score-effort graph into a single value, and use the fitted learner model to conduct a simulation to forecast future effort not observed from the data, similar to the method used in [Lee and Brunskill, 2012]; regarding the parameter plausibility metric, it could take other forms (rather than *guess* or *slip* parameters) in other types of learner models.

Also, more dimensions or metrics could be considered to enhance this framework, such as the Brier score decomposed into additive components for better understanding learner models' prediction behaviors[Käser et al., 2017], parameter consistency examining whether a learner model converge to consistent parameters with different training settings [Huang et al., 2015], predictive similarity policy [Rollinson and Brunskill, 2015] which reveals the instructional effect of a learner model without relying on specific mastery thresholds.

In the remaining part of my thesis, I demonstrate how this framework is used and provide a comprehensive picture of the effectiveness of the proposed learner model.

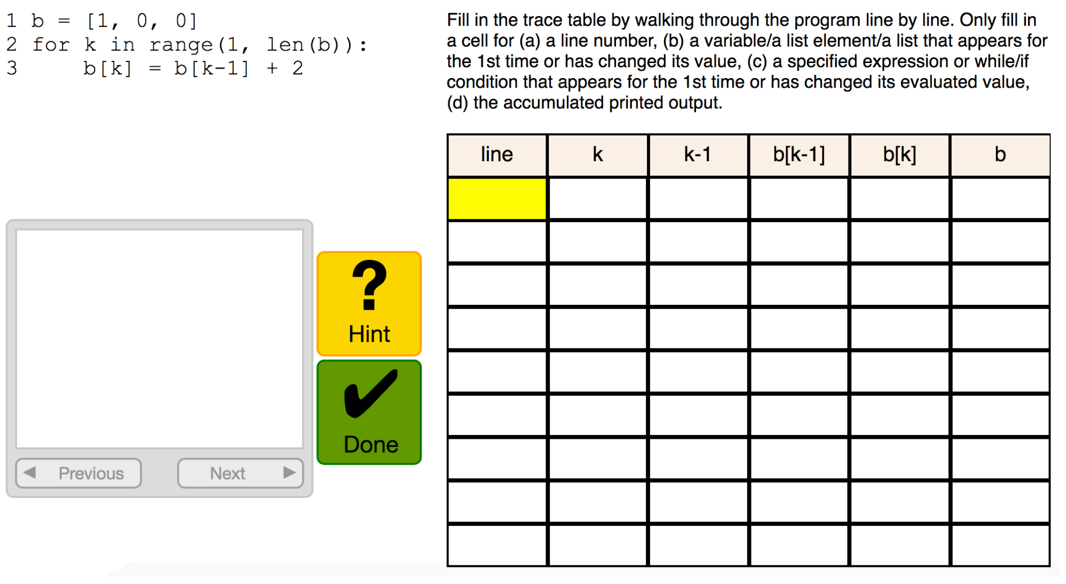
6.0 BUILDING A PROGRAM COMPREHENSION ITS DRIVEN BY LEARNER MODELING

This chapter introduces a Python program comprehension intelligent tutoring system, *Trace Table Tutor (T3)*, which was designed and implemented for the evaluation of the proposed learner model in a real-world intervention study to answer RQ 5 (results reported in Chapter 8). At the same time, the tutor also serves as an example of adopting ITS techniques and infrastructure to the computer science education domain. The key features of T3 are as follows: 1) it provides trace table practice problems which teach both basic and integration skills in program comprehension, and 2) it provides hints and personalized practice problems driven by underlying learner modeling. In the following part of this chapter, I first introduce the trace table practice interface and its implementation, then I describe hint generation and skill labeling, followed by an introduction of the learner modeling service and the problem selection service. Finally, I briefly explain system deployment and logging.

6.1 TRACE TABLE PRACTICE INTERFACE

In this section, I introduce the main practice interfaces and basic functionalities of *T3*, including the practice interface with trace tables, and the practice interface with partially-filled trace tables for learning the conceptual parts of integration skills.

The main component of a practice problem in *T3* is a *trace table* which requires students to fill the values of the variables which are initialized or updated after each line execution of the program in the cells. There are two reasons why I have chosen trace tables for the tutor. Firstly, trace tables are among the first tools for teaching programming to novices, dating



```
b = [1, 0, 0]
for k in range(1, len(b)):
    b[k] = b[k-1] + 2
```

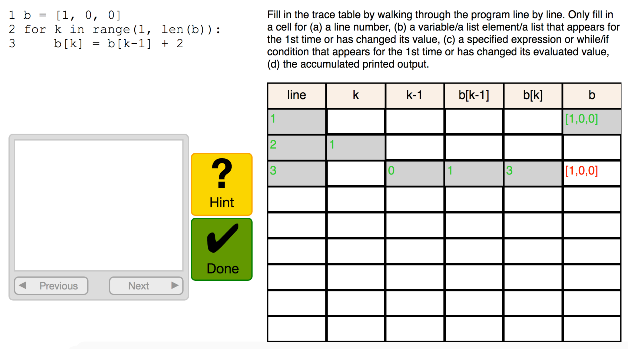
Fill in the trace table by walking through the program line by line. Only fill in a cell for (a) a line number, (b) a variable/a list element/a list that appears for the 1st time or has changed its value, (c) a specified expression or while/if condition that appears for the 1st time or has changed its evaluated value, (d) the accumulated printed output.

line	k	k-1	b[k-1]	b[k]	b

Figure 10: The initial interface of a practice problem with a trace table.

as far back as the 1970s when they were used for teaching Pascal and Fortran programming languages [Taylor, 1977, Du Boulay, 1986], and specifically, the introductory Python courses where I conducted all classroom studies for my dissertation have been teaching trace tables. Although there is no formal assessment of the effectiveness of teaching trace tables (to the best of my knowledge), teachers at Austral University of Chile have commented that trace tables seem to be useful at least for students' initial learning and for explaining some complex programs. Secondly, trace tables naturally offer a way to address the integration difficulty factors discovered in the composition effect studies (Chapter 3) for teaching integration skills in program comprehension. It decomposes the program into small individual steps, which could reduce the cognitive load, one of the integration difficulty factors, allowing students to focus on one step at one time. Meanwhile, some cells in trace tables directly match the conceptual parts of the integration skills, and hint messages could be attached to these cells (steps) to address the conceptual integration difficulties.

Figure 10 shows the main interface of the system: the initial interface of a practice problem with a trace table. Each practice problem provides a program (top left), some instructional text (top right) and requires students to fill in a trace table (bottom right) with rows corresponding to the line-by-line execution of the program. The cell to be filled next is highlighted. A student can enter a value in a cell multiple times. When a student enters a value wrongly and wants to move to the next cell, the answer will be marked as red, which serves as immediate feedback (Figure 11a). Then, a student can request a hint by clicking the hint button and hint messages will be displayed in the hint window (bottom left) (Figure 11b). There are first-level hints providing conceptual knowledge and procedural guidance for the cells, and bottom-level hints which tells the correct answer directly. When a student enters a value correctly, the cell will turn grey and the next cell to be filled will be highlighted. A student can only move to the next cell when the current cell is correct; a student can only move to the next problem when all required cells in the current problem are correct. The next problem is selected according to the underlying learning modeling and problem selection algorithm, and it is displayed in the same window in the same format.

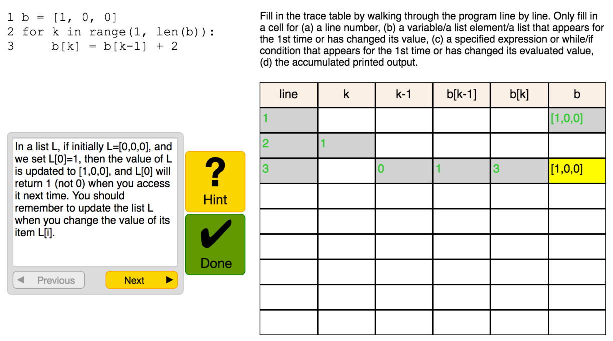


```
1 b = [1, 0, 0]
2 for k in range(1, len(b)):
3     b[k] = b[k-1] + 2
```

Fill in the trace table by walking through the program line by line. Only fill in a cell for (a) a line number, (b) a variable/a list element/a list that appears for the 1st time or has changed its value, (c) a specified expression or while/if condition that appears for the 1st time or has changed its evaluated value, (d) the accumulated printed output.

line	k	k-1	b[k-1]	b[k]	b
1					[1,0,0]
2	1				
3		0	1	3	[1,0,0]

(a) feedback interface



```
1 b = [1, 0, 0]
2 for k in range(1, len(b)):
3     b[k] = b[k-1] + 2
```

Fill in the trace table by walking through the program line by line. Only fill in a cell for (a) a line number, (b) a variable/a list element/a list that appears for the 1st time or has changed its value, (c) a specified expression or while/if condition that appears for the 1st time or has changed its evaluated value, (d) the accumulated printed output.

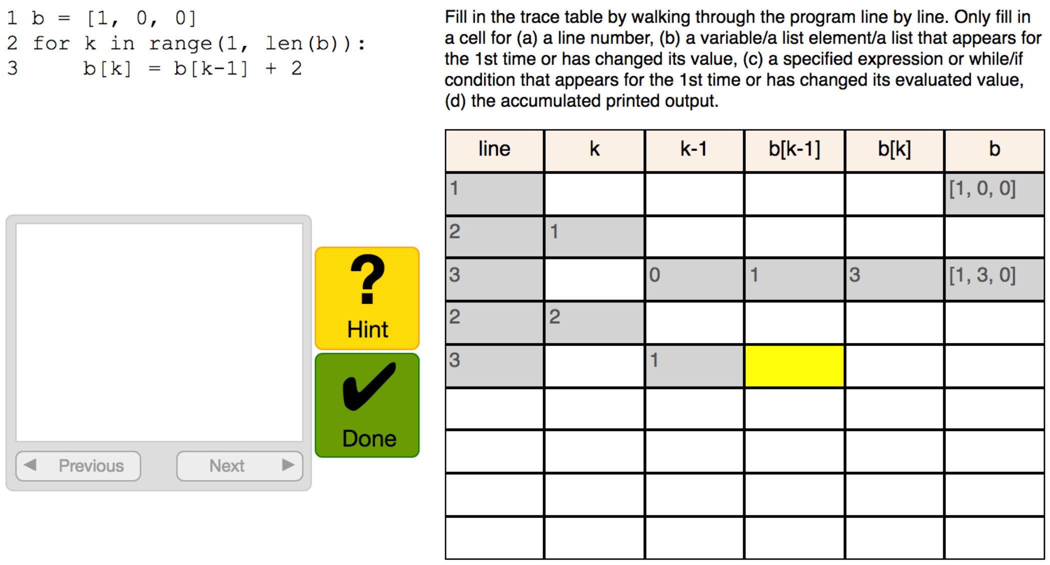
In a list L, if initially L=[0,0,0], and we set L[0]=1, then the value of L is updated to [1,0,0], and L[0] will return 1 (not 0) when you access it next time. You should remember to update the list L when you change the value of its item L[i].

line	k	k-1	b[k-1]	b[k]	b
1					[1,0,0]
2	1				
3		0	1	3	[1,0,0]

(b) hint interface

Figure 11: The interfaces of a practice problem for the situations where a student enters a wrong answer and wants to move to the next cell (a), and the situation where a student asks for a hint (b).

I also created partially-filled trace tables for problems focusing on the conceptual parts of integration skills. Here, only the cells with the conceptual parts of integration skills are missing and other cells are automatically filled. Figure 12 shows the initial interface of an isolated integration problem for practicing *for&a[i]*^{c} in contrast to the initial interface of the full integration problem for practicing *for&a[i]* in Figure 10.



```
b = [1, 0, 0]
for k in range(1, len(b)):
    b[k] = b[k-1] + 2
```

Fill in the trace table by walking through the program line by line. Only fill in a cell for (a) a line number, (b) a variable/a list element/a list that appears for the 1st time or has changed its value, (c) a specified expression or while/if condition that appears for the 1st time or has changed its evaluated value, (d) the accumulated printed output.

line	k	k-1	b[k-1]	b[k]	b
1					[1, 0, 0]
2	1				
3		0	1	3	[1, 3, 0]
2	2				
3		1			

Figure 12: The initial interface of an isolated integration problem for the conceptual part of an integration skill, *for&a[i]*^{c}, where cells not addressing it are automatically filled.

I conducted a pilot study to check whether the above practice problem design with trace tables could be easily understood by students. The pilot study involved 5 undergraduates at the University of Pittsburgh and 4 undergraduates at the Austral University of Chile, taking introductory Python programming courses. They were required to try a demo (5 minutes), answer a pretest (5 minutes), practice in the system (45 minutes), answer a posttest (10 minutes), and answer interview questions (15 minutes). Interview results showed the trace tables were in general easy to understand for the students, and they gave some suggestions to make the instructional text clearer.

6.2 INTERFACE IMPLEMENTATION

To implement the above interfaces and basic functionalities, I utilized the Cognitive Tutor Authoring Tools (CTAT)[1]. More details can be found elsewhere [Aleven et al., 2006,

[1]https://github.com/CMUCTAT/CTAT/wiki

Koedinger et al., 2003]. CTAT provides a set of tools with user-friendly GUIs, and it has a detailed wiki and a responsive support team. The adoption of the Cognitive Tutor infrastructure in building *T3* has only required a small amount of effort.

One important feature of CTAT is that it supports the model tracing process, which is a key component for Cognitive Tutors. A solution path for each problem needs to be constructed (in BRD files) so that each step (cell) can be mapped to a state in this path, and the answer could be compared with the pre-specified correct answer, based on which feedback and hints can be given. CTAT automatically generates such a solution path given a demonstration (through a GUI) of how to solve a problem (as a student). To further speed up and scale up the content generation process, I implemented a process to automatically generate HTML and BRD files for any given programs utilizing the library provided by the Online Python Tutor[2].

6.3 HINT GENERATION AND SKILL LABELING

One intelligent functionality of *T3* is that it provides targeted hint messages for remediating weak skills. The hint generation is enabled by underlying learning modeling. In the practice problems with trace tables, step-level hint messages are attached to cells based on an underlying learner model. A learner model (defined in my dissertation) contains a skill model which includes the specification of skills required by items (on the problem or the step level). For each skill (across different cells), I created a corresponding hint message explaining the conceptual knowledge and procedural guidance to apply the skill.

This kind of hint-to-skill relationship helps to differentiate skill (learner) models for better evaluation: for the version of ITS driven by an integration-level learner model (e.g., CKM-HI), students can receive hints for understanding the conceptual parts of integration skills; for the version of ITS driven by a basic-level learner model (e.g., CKM, WKT), students only receive hints for basic skills. Also, this kind of hint-to-skill one-to-one mapping, compared with constructing different hints for the same skill in different problems, allows

[2]https://github.com/pgbovine/OnlinePythonTutor

for automation and scaling up. Admittedly, the latter method could offer more fine-grained assistance, and I leave it for future work. The hint generation process is addressed by a skill labeling process. I tailored the general skill labeling algorithm described in Chapter 4 to addressing the situation here where each step maps to a cell in a trace table. Details of this tailored skill labeling algorithm are explained as follows:

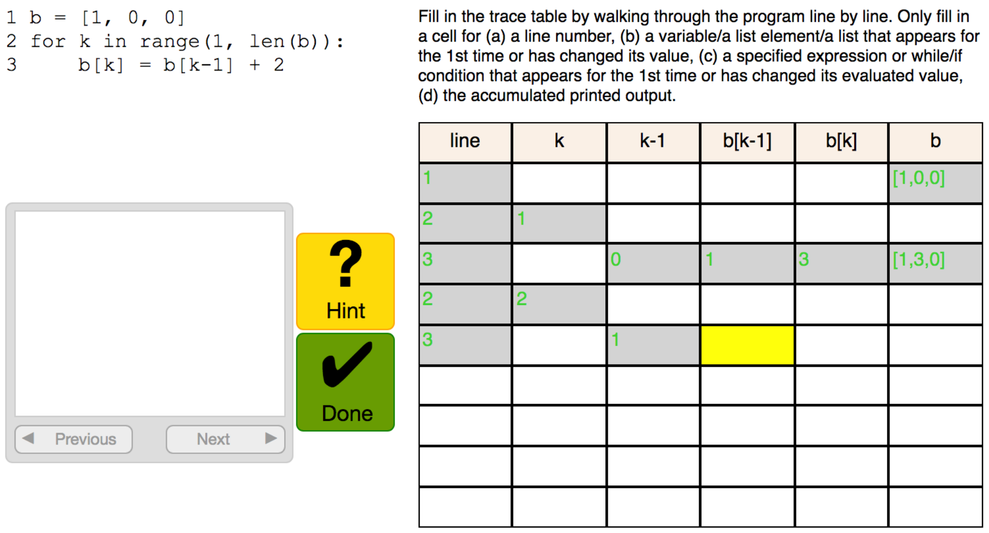
1. Generate a JSON-formatted *execution trace* after the line-by-line execution of the program, based on the library provided by the Online Python Tutor[3]. An execution trace is an ordered list of execution points. An execution point is an object representing the state of the computer's (abstract) memory at a certain point in execution (including the values of the variables). Details can be found elsewhere [Guo, 2013].
2. Get the names of the variables (or headers of the trace table) whose values need to be filled in the trace table (e.g., iteration variables, cumulative sum variables).
3. Iterate over the execution trace object by object, which is equivalent to iterating over the program line by line, and at each line:
   a. Iterate over each variable listed in the trace table headers. If the variable is initialized or its value has been changed, conduct the skill labeling for this step (cell) according to a set of rules based on its contextual features. The judgement of the existence of each feature is based on a straightforward program text analysis (such as string matching, indentation checking). Table 14 shows the features of a rule for labeling the conceptual part of an integration skill, *for&a[i]*^{c}, for the highlighted cell (step) shown in Figure 13.
   b. Update the variables.
4. Return an ordered list of steps (cells) with the associated skill(s). Label each cell in the solution path file with its required skill.

After the above skill labeling process, hint messages for each problem can be generated by matching the hints with skills, and adding the hint messages into the solution path files. Table 15 shows an example of hints for the highlighted cell in Figure 13, based on an integration-level learner model or a basic-level learner model.

[3]https://github.com/pgbovine/OnlinePythonTutor

Table 14: A rule with necessary features for labeling the conceptual part of an integration skill, *for&a[i]*^{c}, for a cell (step). Note that this rule doesn't have *unwanted features*.

ID	Features
1	Current corresponding variable is a list element and its index uses the iteration variable, e.g., a[i], a[i-1].
2	Current corresponding variable is on the right side of the assignment operator.
3	Current corresponding line is inside a *for* loop.
4	Current corresponding line has an assignment operator.
5	The variable on the left side of the assignment operator is a list element with the iteration variable as its index, e.g., a[i].
6	This is the second visit of the current corresponding line.



```
1 b = [1, 0, 0]
2 for k in range(1, len(b)):
3     b[k] = b[k-1] + 2
```

Fill in the trace table by walking through the program line by line. Only fill in a cell for (a) a line number, (b) a variable/a list element/a list that appears for the 1st time or has changed its value, (c) a specified expression or while/if condition that appears for the 1st time or has changed its evaluated value, (d) the accumulated printed output.

line	k	k-1	b[k-1]	b[k]	b
1					[1,0,0]
2	1				
3		0	1	3	[1,3,0]
2	2				
3		1			

Figure 13: The highlighted cell (step) is labeled with the conceptual part of an integration skill, *for&a[i]*^{c}, based on an integration-level learner model.

Table 15: Hints for the highlighted cell in Figure 13, based on an integration-level learner model or a basic-level learner model.

Learner model	Skill label	Hint message
basic-level	a[i]	In a list L, L[0] returns the item at index 0, which is the first item, and L[i] returns the the item at index i, which is the (i+1)th item. If initially L=[0,0,0], and we set L[0]=1, then L[0] will return 1 when you access it next time.
integration-level	for&a[i][c]	The list assignment statement (e.g. L[i]=L[i-1]+1) is in a *for* loop. You should use the updated value (not the initial value) of L[i-1] from the previous iteration. The new value of L[i] will be used in the next iteration when you access L[i-1].

6.4 LEARNER MODELING SERVICE

In the backend of the system, I have implemented a learner modeling service that maintains and updates each student's learner model based on the incoming student's performance sequence. Additionally, it also logs the problem-level student interaction data in real time.

The learner modeling service works as follows in an online system: After each problem submission, a student's performance on the cell level is extracted from the JSON formatted data in the HTTP requests from the TutorShop (the LMS that hosts the content), and it is used to update the corresponding skills in the student's learner model. The BN structure and parameters are pre-stored on the server. Students share the same BN structure (in the same condition group) and obtain the same initial estimations of skill levels in their learner models. The learner model is implemented based on the SMILE library [Druzdzel, 1999], which is a reasoning engine for graphical models such as Bayesian networks. It is a highly efficient engine with fast parameter learning and latent variable inference, and it has enabled responsive adaptation functionality of T3. In addition, the learner models for students are stored in a server database (and also through the TutorShop service).

6.5 PROBLEM SELECTION SERVICE

Another major intelligent and adaptive functionality of T3 is the problem selection, and it is also driven by the underlying learner modeling. Based on the learner modeling service, I developed a problem selection service[4] that takes as input a student's latest learner model (a vector of knowledge estimations) and his/her recent performance, and returns a problem for the student to work on next. The service is implemented in a general way so that any specific problem selection algorithms could be embedded. Since problem selection is not the focus of my dissertation, I only describe the conceptual idea and the generic algorithm for problem selection here. Note that when deploying T3 in Classroom Study 2 (Chapter 8), some modifications (preserving the general idea) were introduced according to the need of the study design.

The key idea of the problem selection algorithm is to give higher chances to stronger unmastered skills to be remediated first, and after picking a skill, pick the problem with the most focus on this skill. The reason of preferring stronger unmastered skills rather than weaker unmastered skills is that in many situations students' stronger skills are prerequisites for weaker skills, and if a student still hasn't mastered his/her stronger skills, then probably he/she is not ready to learn weaker skills.

The generic algorithm is explained as follows:

1. Update a student's learner model based on the just submitted problem.
2. Pick an unmastered skill according to following steps:
   a. Retrieve the student's latest learner model, i.e., the probabilities of knowing each skill k, $\mathrm{P(known)}_k$.
   b. Find unmastered skills, which are the skills with $\mathrm{P(known)}_k \leq m$ (the pre-specified mastery threshold); if there are no unmastered skills, stop practicing the current topic and move to the next topic[5].
   c. Compute the probability of an unmastered skill i being selected, $\mathrm{P(select)}_i$, as follows, so that stronger unmastered skills are more likely to be remediated first:

[4]The learning modeling and problem selection are implemented in a single service.

[5]In my Classroom Study 2 reported in Chapter 8, a modification was made in this step regarding the stop-practice criterion in order to adjust to the study design.

$$P(select)_i = \frac{P(known)_i}{\sum_u P(known)_u} \tag{6.1}$$

where u ∈ unmastered skills in the skill set of the current topic.

d. Pick a skill according to the multinomial distribution (of the skill index) specified by the set of P(select)$_i$, so that skills with higher P(select)$_i$ values have higher probabilities to be picked. Then, move to pick a problem.

3. Pick a problem focusing on remediating the chosen skill w:

   a. Compute the *focus score* F_{wj} for each problem j requiring the chosen skill w as follows, considering the relative strength of the skill w compared with other skills and the total amount of unlearned knowledge in the problem j:

$$F_{wj} = \frac{P(unknown)_w}{\sum_s P(unknown)_s} \tag{6.2}$$

   where s ∈ skills of the current problem j.

   b. Pick the problem with the highest F_{wj} as the final selected problem, so that a problem with the most focus on skill w will be chosen.

6.6 SYSTEM DEPLOYMENT AND DATA LOGGING

To deploy *T3*, I used the TutorShop[6] platform, a customized learning management system (LMS) for CTAT tutors. The learner modeling and problem selection service is called through an HTTP URL from the TutorShop. The TutorShop platform provides a user-friendly GUI for deploying and testing learning materials developed by CTAT, with a responsive support team. The deployment of the *T3* into TutorShop has been straightforward.

Besides showing the content interface and the links of different assignments (stages), TutorShop also has some functionalities to control the access: 1) a student can only access the next assignment (stage) once he/she finishes all the problems in the current assignment

[6]https://tutorshop.web.cmu.edu/

(stage), 2) a teacher can activate/deactivate any assignments at any time (even if students haven't finished them), so that the the teacher can have better controls of the maximum amount of time on an assignment.

Regarding logging the student interaction data, the fine-grained cell (step) level data is logged by DataShop[7], and the more coarse-grained problem level data is also logged by my learner modeling service.

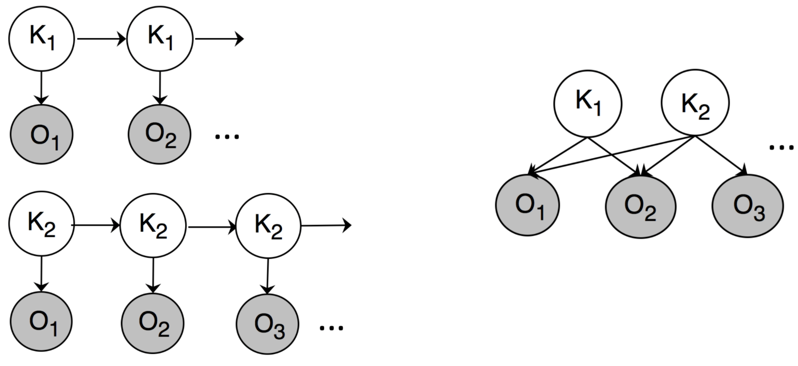
[7]https://pslcdatashop.web.cmu.edu/

7.0 DATA-DRIVEN EVALUATIONS OF THE PROPOSED INTEGRATION-LEVEL LEARNER MODEL

In this chapter I experiment on real-world and simulated datasets to evaluate the proposed learner model CKM-HI (introduced in Chapter 4), based on the data-driven evaluation dimensions and metrics under the proposed multifaceted evaluation framework (introduced in Chapter 5). This chapter contributes to my RQ 4, *Is learner modeling for integration skills beneficial in terms of multifaceted data-driven evaluations?* I first describe the baseline models chosen for comparison, then I introduce the main experiments on the real-world dataset collected from *T3* to demonstrate the evaluations using both problem and step level data, and finally I describe a focused experiment using simulated datasets to further investigate issues discovered in the real-world dataset experiments.

7.1 BASELINE MODELS

To evaluate the proposed integration-level learner model CKM-HI, two popular multi-skill practice learner models that don't incorporate integration skills were considered. The first baseline is the *weakest knowledge tracing* (WKT) model. As introduced in the Chapter 2, WKT is a representative model from one family of multi-skill practice learner models which assumes (marginal and conditional) independence between skills, and ignores modeling possible integration between the skills. Prior work [Gong et al., 2010, González-Brenes et al., 2014, Xu and Mostow, 2012] has shown that WKT has the best predictive performance, based on several datasets that compared it with other variants in the same family. WKT predicts the probability of success for each of the the involved skills, then picks the minimum



(a) WKT **(b) CKM**

Figure 14: BN structures of two popular multi-skill practice learner models. O nodes represent the observed binary student performance and the K nodes represent the latent binary knowledge levels.

probability as the final prediction; it only updates the knowledge of the weakest skill when observing an incorrect response, and updates all skills otherwise. The network structure of WKT is shown in Figure 14(a). WKT differs from CKM-HI in the multi-skill responsibility assignment mechanism, the explicit differentiation between component and integration skills, and the skill-to-skill integration relations. WKT serves as a low baseline.

The second baseline is the *conjunctive knowledge modeling* (CKM) model. As introduced in Chapter 2, CKM is a representative model of another family of multi-skill practice learner models where skills are marginally independent but conditionally dependent on items, and uses noisy-AND gates on items with Bayesian rule to assign responsibility for performance to skills and to update skills. It has been used in several prior studies [Carmona et al., 2005, Conati et al., 2002, VanLehn et al., 1998]. However, it also ignores the modeling of possible integration between the skills. The network structure of CKM is shown in Figure 14(b). CKM has the same multi-skill responsibility assignment mechanism as CKM-HI, but still differs from CKM-HI in the explicit differentiation between component and integration skills and the skill-to-skill integration relationships. The comparison between CKM-HI and CKM evaluates the benefit of incorporating integration skills. CKM serves as a high baseline model for evaluating CKM-HI. Meanwhile, CKM has the same skill specifications and skill-to-skill relations as WKT, and only differs from WKT in the multi-skill responsibility assignment mechanism. The comparison between CKM and WKT helps to tease apart the effect of the multi-skill responsibility assignment mechanism which exists in both CKM and CKM-HI models, so that we can cleanly examine the effect of incorporating integration skills in the comparison between CKM-HI and WKT.

7.2 REAL-WORLD DATASET EXPERIMENTS

The section provides multifaceted data-driven evaluation of the proposed integration-level learner model CKM-HI, as compared to CKM and WKT, on a real-world dataset. I start by explaining the experimental set up and then report the analysis results for both the problem- and step-level data.

7.2.1 Experimental Setup

7.2.1.1 Dataset

I used the dataset collected from classroom and lab studies deploying *T3* (Trace Table Tutor), an ITS I designed and implemented to teach skills in Python program comprehension, using trace tables (details in Chapter 6). The classroom and lab studies participants were students taking introductory level Python programming courses from Fall 2017 to Spring 2018, at the University of Pittsburgh and Austral University of Chile (Universidad Austral de Chile). The majority of the students in the dataset were from the Classroom Study 2 introduced in Chapter 8. Note that although the dataset was obtained after running studies deploying *T3*, CKM-HI had already been tested on a dataset and a user study of a Java program comprehension tutor, QuizJET [Hsiao et al., 2010], and had shown advantages over CKM and WKT over a wide range of aspects. The details were reported in my prior paper [Huang et al., 2017]. In this prior work, the skill model was constructed by expert-labeling, which is very different from the skill modeling approach used here.

This dataset contains problems on the *for* loops topic and the *lists* topic. The complete skill set can be found in Table 12 and Table 13 in Chapter 4. The characteristics of this tutor enabled me to build Bayesian networks for learner models on two density levels, the problem level and the step level. The motivation and construction of these two levels of learner models can be found in Chapter 4. The original finest-grained data logged students' attempts on the step level. To convert the data to the problem level, I examined all steps in a problem and labeled it as correct if the first attempts of all steps were correct, and labeled it as incorrect, otherwise.

I used students' first attempts on the practice problems and the submitted attempts (which were their last attempts) on the pretest and posttest problems to train and test the learner models. Using only their first attempts in step level tutoring data is a common way to create learner models in the educational data mining (EDM) field, since this method may be able to reflect student knowledge levels more accurately. Later attempts at solving practice problems may carry too much noise for knowledge inference, since students might have asked for hints and even bottom-level hints showing correct answers, in later attempts. Keep in mind that the tutor only allows students to move to the next step or next problem if the student has made a correct attempt in the current step or current problem during the practice state. Therefore, the last attempts to solve the practice problems are all correct. However, on the pretest and posttest problems, students don't get any feedback after each attempt, and may enter answers multiple times before finally submitting a problem. Therefore, the submitted attempts on the pre/posttests are the ones that are preserved for the dataset. The descriptive statistics for the final dataset can be found in Table 16, 17.

Table 16: Descriptive statistics for the data on problem and the step levels.

	#stu.	#obs.	Proportion	#items			
			correct	Pretest	Posttest	Practice	Total
Problem level	129	4,531	0.55	10	13	66	89
Step level	129	18,484	0.81	10	13	540	563

Table 17: Descriptive statistics for skill models for different learner models on problem and step levels. For computing the number of basic/integration/all skills per item, the mean with minimum and maximum are reported in the format of *mean[min, max]*. An integration item refers to an item labeled as having integration skills.

	WKT/CKM		CKM-HI	
	Problem	Step	Problem	Step
#labeled basic skills	6		6	
#labeled (conceptual parts of/full) integration skills	0		10	
#skills labeled per pre/posttest problem	1.7[1,3]		3.0[1,11]	
%pre/posttest problems labeled with integration skills	0		48%	
#skills labeled per practice item	1.8[1,5]	1.0[1,2]	3.3[1,11]	1.4[1,4]
%practice items labeled with integration skills	0	0	64%	33%

7.2.1.2 Learner Models' Initial Parameters

The parameters of all my learner models were learned from data. However, since the learner models involved latent variables, an expectation maximization (EM) algorithm was used. The EM algorithm is known to be prone to local optimum and sensitive to initial parameters. Prior work [Pardos and Heffernan, 2010b] has shown that when setting the initial parameters to plausible ranges, the BKT model [Corbett and Anderson, 1995] is less prone to getting stuck in a local optimum. Being aware of this issue and trying to maintain fairness, for all learner models, I initialized the *init* parameters (the probability of initially knowing the skill) of all root skill nodes (including the basic skills $K_i^{(b)}$, and the conceptual parts of integration skills $K_{i\&j}^c$) by the average proportion correct over the number of problems that require this skill. I initialized all learner models with the same (plausible) parameters explained as follows. I initialized the *learn* parameter for WKT as 0.15, and all models' *guess* and *slip* parameters as 0.1. For CKM-HI, I initialized the conditional probability table of each integration skill node $K_{i\&j}$, given the values of its parents K_i^b, K_j^b and $K_{i\&j}^c$ by setting {P(T|TTT)=0.999, P(T|TTF)=P(T|TFT)= P(T|TFF)=P(T|FTT)=P(T|FTF)=P(T|FFT)=P(T|FFF)=0.001}, indicating that a student has a low probability (0.001) of knowing an integration skill if he/she doesn't know any of the prerequisite basic skills or conceptual parts of integration skills, and a high probability (0.999) otherwise. I used the SMILE library [Druzdzel, 1999], which is a reasoning engine for graphical models, such as Bayesian networks, to construct all learner models. It is a highly efficient engine with fast latent variable inference and parameter learning.

7.2.1.3 Cross-Validation

I conducted a modified 10-fold student stratified cross-validation explained as follows. Students were randomly split into ten folds. For each fold, data from the other nine folds (90% of students) was used as the training data, and the data from the current fold (10% of students) was further split into two parts with these students' pretest data being placed into the training set, and the practice data being held back as the testing set. This way is slightly different from the standard method of student stratified cross-validation in that the pretest data of test students is also used for training. This is because in a real-world setting, the pretest would be constructed to initialize the learner models or for computing learning gain, and the prediction of scores on it doesn't affect adaptation decisions. The most important part is the performance prediction and knowledge inference of the learner model in the practice stage, because this affects adaptation decisions.

7.2.2 Results for the Problem Level Assessment Data

In this section, I compare CKM-HI with CKM and WKT using the problem level assessment data from multifaceted perspectives: predictive performance, parameter plausibility and expected instructional effectiveness. The statistical tests and values of predictive performance and parameter plausibility metrics are reported in Table 18 and Table 19. The score-effort curve for expected instructional effectiveness is displayed in Figure 15. I report and interpret results of each pairwise comparison between the three learner models, considering the three dimensions as follows.

Table 18: Comparison of predictive performance and parameter plausibility between the three learner models, based on the problem level data. The tables report the mean difference with a 95% confidence interval (CI) over ten folds, the paired t-test (approximate normality met) p-value (df=9) and Cohen's d effect size. (Sig. level: ***:<.001/3=.00033, **:<.01/3=.0033, *:<.05/3=.017, $^{\bullet}$:<.1/3=.033; effect size $^{+++}$:>.8, $^{++}$:>.5, $^{+}$:>.2.)

(a) Predictive Performance

	CKM-HI vs. WKT				CKM-HI vs. CKM				CKM vs. WKT			
	M	p	d		M	p	d		M	p	d	
RMSE	-.034±.014	.00032	3	*** +++	.0003±.002	.77	.02		-.034±.013	<.0003	3	*** +++
AUC	.103±.038	<.0003	3	*** +++	-.003±.006	.28	.07		.106±.036	<.0003	3	*** +++
Recall(0)	.157±.044	<.0003	4	*** +++	.015±.011	.01	.36	* +	.142±.043	<.0003	3	*** +++
Precs.(0)	.042±.039	.038	.6	++	-.012±.012	.06	.19		.054±.041	.016	.78	* ++

(b) Parameter Plausibility

	CKM-HI vs. WKT				CKM-HI vs. CKM				CKM vs. WKT			
	M	p	d		M	p	d		M	p	d	
guess+slip	-.141±.012	<.0003	10	*** +++	-.065±.012	<.0003	4	*** +++	-.076±.007	<.0003	7	*** +++
guess	-.149±.007	<.0003	12	*** +++	-.050±.006	<.0003	3	*** +++	-.099±.007	<.0003	8	*** +++
slip	.008±.011	.126	/		-.015±.009	.005	1	* +++	.024±.004	<.0003	5	*** +++

Comparing CKM-HI with WKT, CKM-HI clearly outperforms WKT in all three aspects. Regarding predictive performance, CKM-HI has performed significantly better on the RMSE, AUC and recall (for incorrect responses) metrics (p<0.00033) with large effect sizes, and is close to marginally significantly better than WKT in the precision for incorrect responses metric (p=0.038). Regarding parameter plausibility, CKM-HI also has significantly smaller *guess+slip* and *guess* (p<0.0003) with large effect sizes, yet has statistically the same *slip* as WKT. Regarding expected instructional effectiveness, CKM-HI shows an advantage over

Table 19: Predictive performance and parameter plausibility metrics of the learner models, based on the problem level data. The table reports the mean with a 95% CI over ten folds.

Model	Predictive Performance				Parameter Plausibility		
	RMSE	AUC	Recall(0)	Precs.(0)	*guess+slip*	*guess*	*slip*
CKM-HI	.445±.012	.761±.033	.778±.030	.711±.043	.526±.014	.205±.011	.321±.013
CKM	.444±.011	.764±.030	.763±.031	.723±.045	.591±.009	.255±.011	.336±.004
WKT	.478±.006	.657±.020	.620±.035	.670±.054	.667±.006	.444±.011	.312±.003

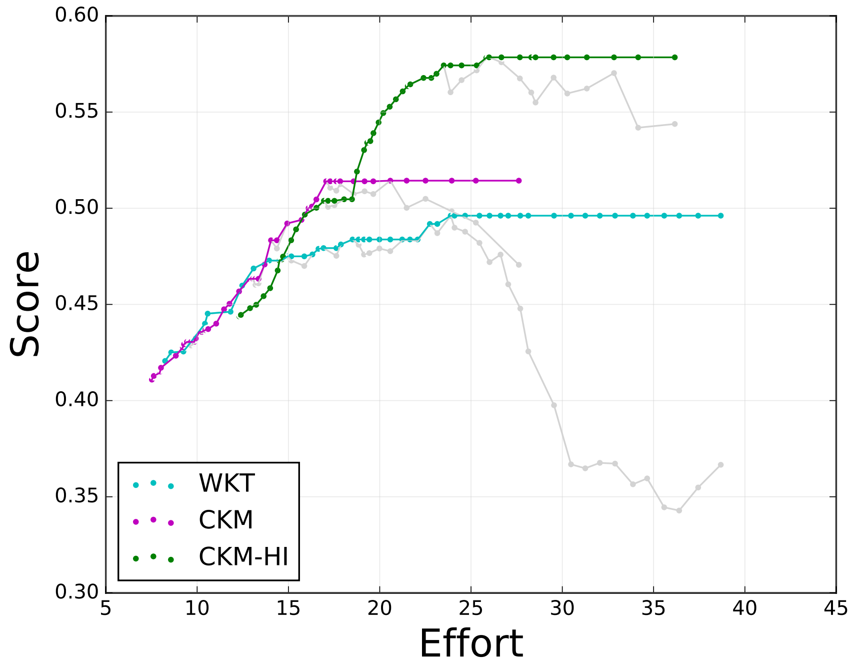


Figure 15: Comparison of expected instructional effectiveness between the three learner models, based on the problem level data. For each model, the sequence of values (dots) corresponds to mastery thresholds ranging from 0.50 to 0.99, in increments of 0.01. The grey curve is the original one, without smoothing.

WKT on most mastery thresholds (≥0.62): using the same effort, CKM-HI appears to enable students to reach higher scores, or when reaching the same score, CKM-HI appears to enable students to take less effort.

Comparing CKM-HI to CKM, CKM-HI is superior to CKM considering all three aspects. Although the RMSE, AUC and precision for incorrect responses metrics are statistically the same, CKM-HI has a significantly higher recall for incorrect responses ($p=0.01$, $d=0.36$). This recall metric is important in a tutoring system, since it describes the ability of a learner model to retrieve all failures of students when weak skills exist, and is arguably more important than the precision for incorrect responses metric. When it comes to parameter plausibility, the advantage of CKM-HI over CKM is dramatic: it has significantly smaller *guess+slip*, *guess* and *slip* values ($p<0.017$) with large effect sizes. In addition, in terms of expected instructional effectiveness, CKM-HI clearly outperforms CKM on high mastery thresholds (≥0.92), in that it can enable students to reach higher scores given the same effort, and it can enable students to reach the same score with less effort.

The comparison between CKM and WKT is a bit mixed. CKM significantly defeats WKT in all predictive performance metrics ($p<0.017$, with medium to large effect sizes) and has significantly smaller *guess+slip*, *guess* ($p<0.0003$, with large effect sizes), but CKM has a significantly larger *slip* ($p<0.0003$, with a large effect size). Regarding the expected instructional effectiveness, CKM seems to be better than WKT on high thresholds (≥0.80), yet the significance and persistence of the advantage is not clear.

7.2.3 Results for the Step Level Assessment Data

In this section, I compare CKM-HI with CKM and WKT on multifaceted perspectives: predictive performance, parameter plausibility and expected instructional effectiveness, based on the step level assessment data. The statistical tests and values of predictive performance and parameter plausibility metrics are reported in Table 20 and Table 21. The score-effort curve for expected instructional effectiveness is displayed in Figure 16. I report and interpret results of each pairwise comparison between the three learner models, considering the three dimensions as follows.

Table 20: Comparison of predictive performance and parameter plausibility between the three learner models, based on the step level data. The table reports the mean difference with a 95% confidence interval (CI) over ten folds, the paired t-test (PT) (approximate normality met) p-value (df=9) and Cohen's d effect size. In the comparison between CKM and WKT on RMSE, Wilcoxon signed-rank (WT) test was used since normality is strongly violated. (Sig. level: ***:<.001/3=.00033, **:<.01/3=.0033, *:<.05/3=.017, $^{\bullet}$:<.1/3=.033; PT effect size $^{+++}$:>.8, $^{++}$:>.5, $^{+}$:>.2; WT effect size $^{+++}$:>.5.)

(a) Predictive Performance

	CKM-HI vs. WKT			CKM-HI vs. CKM			CKM vs. WKT		
	M	p	d	M	p	d	M	p	d
RMSE	-.015±.005	<.0003	.4 $^{***}_{+}$	-.0034±.0026	.015	.1 *	-.012±.005	.007	.6 $^{*}_{+++}$
AUC	.083±.022	<.0003	3 $^{***}_{+++}$	.0069±.0068	.048	.3 $_{+}$	.076±.019	<.0003	2 $^{***}_{+++}$
Recall(0)	.252±.053	<.0003	6 $^{***}_{+++}$	.018±.023	.10	/	.233±.066	<.0003	4 $^{***}_{+++}$
Precs.(0)	.100±.064	.006	1 $^{*}_{+++}$	.021±.022	.06	/	.079±.071	.0328	1 $^{\bullet}_{+++}$

(b) Parameter Plausibility

	CKM-HI vs. WKT			CKM-HI vs. CKM			CKM vs. WKT		
	M	p	d	M	p	d	M	p	d
guess+slip	-.231±.007	<.0003	31 $^{***}_{+++}$	-.048±.011	<.0003	4 $^{***}_{+++}$	-.183±.012	<.0003	16 $^{***}_{+++}$
guess	-.216±.007	<.0003	33 $^{***}_{+++}$	-.040±.013	<.0003	3 $^{***}_{+++}$	-.176±.014	<.0003	14 $^{***}_{+++}$
slip	-.015±.001	<.0003	7 $^{***}_{+++}$	-.008±.002	<.0003	2 $^{***}_{+++}$	-.0075±.002	<.0003	2 $^{***}_{+++}$

As in the problem level results, the step level results show that CKM-HI clearly outperforms WKT in all three aspects. CKM-HI is significantly more predictive in all predictive performance metrics (p<0.017), with small to large effect sizes; CKM-HI has significantly smaller *guess+slip*, *guess* and *slip* values (p<0.0003), with large effect sizes; and CKM-HI has higher expected instructional effectiveness on all mastery thresholds. However, we still need to compare CKM with WKT to investigate where the advantages come from.

Table 21: Predictive performance and parameter plausibility of the learner models, based on the step level data. The table reports the mean with a 95% CI over ten folds.

Model	Predictive Performance				Parameter Plausibility		
	RMSE	AUC	Recall(0)	Precs.(0)	*guess+slip*	*guess*	*slip*
CKM-HI	.351±.025	.790±.015	.319±.047	.575±.033	.314±.005	.192±.004	.122±.002
CKM	.354±.026	.783±.020	.301±.063	.555±.037	.362±.011	.232±.013	.130±.003
WKT	.366±.025	.707±.030	.067±.017	.475±.074	.545±.006	.408±.005	.137±.002

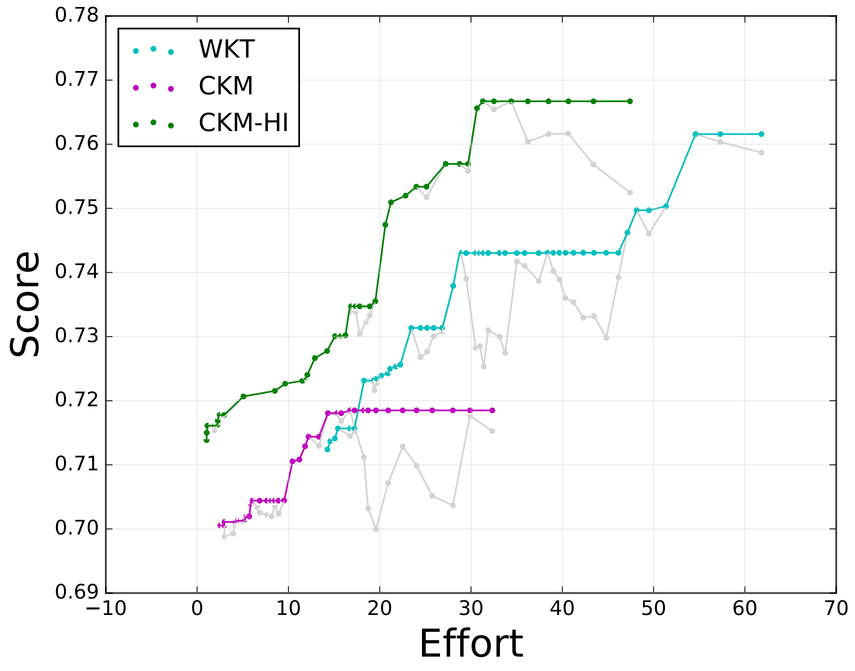


Figure 16: Comparison of expected instructional effectiveness between the three learner models, based on the step level data. For each model, the sequence of values (dots) corresponds to mastery thresholds from 0.50 to 0.99, in increments of 0.01. The grey curve is the original one without smoothing.

The step level comparison between CKM-HI and CKM reaches the same conclusions as the problem level did: CKM-HI is superior to CKM, considering all aspects. CKM-HI has a significantly lower RMSE ($p<0.017$) and a close to marginally significantly higher AUC ($p=0.048$), with recall and precision for incorrect responses statistically the same; CKM-HI still has significantly smaller *guess+slip*, *guess* and *slip* values ($p\leq0.0003$), with large effect sizes; CKM-HI clearly outperforms CKM on expected instructional effectiveness on all mastery thresholds. Together with the problem level results, the comparison between CKM-HI and CKM suggests that it's beneficial to add integration skills with a hierarchical structure to a learner model for multi-skill practice involving integration. Mostly, it helps correctly assign credit/blame to the right skills rather than increase the *guess* and *slip* noise parameters, and it helps improve the expected instructional effectiveness. This will increase the learner model's ability to diagnose weak skills, and will significantly affect students' learning efficiency.

Comparing CKM and WKT on the step level, CKM still significantly defeats WKT in all predictive performance metrics ($p<0.033$, with large effect sizes) with only the precision on incorrect responses metric having a decreased significance level, from 0.05 to 0.1. Different from the problem level results, the step level results show that CKM has not only significantly smaller *guess+slip*, *guess* ($p<0.0003$, with large effect sizes), but also a smaller *slip* ($p<0.0003$, with a large effect size); different from the problem level results, CKM seems to have a lower expected instructional effectiveness on high thresholds (≥0.90), while the significance and persistence of the disadvantage is not clear. Together with the problem level results, the comparison between CKM and WKT suggests that the item-to-skill credit/blame assignment mechanism that both CKM and CKM-HI possess brings a predictive improvement over WKT, but it doesn't necessarily bring improvement on parameter plausibility or expected instructional effectiveness over WKT.

7.2.4 Discussion and Conclusions

Summarizing the results on both the problem and step level assessment data collected from the real-world Python tutor, the proposed model CKM-HI offers significant improvements over two popular multi-skill practice learner models, WKT and CKM. I considered a range of aspects in my multifaceted evaluation framework: performance prediction accuracy, parameter plausibility, and expected instructional effectiveness.

Modeling integration skills (in CKM-HI) dramatically increases the parameter plausibility and expected instructional effectiveness, while it increases or maintains similar predictive performance, as evidenced by the comparisons between CKM-HI and CKM. Meanwhile, modeling conditional dependencies between skills with a conjunctive noisy-AND gate brings predictive advantages, but not necessarily improves parameter plausibility or expected instructional effectiveness, as evidenced by the comparison between CKM and WKT. These results clearly demonstrate the feasibility and value of learner modeling for integration skills, and the validity of the skill model and modeling approach I have proposed.

As a showcase for my proposed multifaceted data-driven evaluation framework, the above analysis also provides evidence about the limitations of performance prediction evaluation: by examining the expected instructional effectiveness, a model with a limited performance prediction gain can still have a significant improvement in adaptive tutoring (CKM-HI vs. CKM); and surprisingly, a model with a significant prediction gain can have a slight improvement or no improvements in adaptive tutoring (CKM vs. WKT).

One limitation of the current analysis is that it doesn't allow us to conclude whether the addition of a hierarchical structure to CKM brings an advantage to CKM-HI. To answer this question, another model using the flat structure of CKM, incorporating integration skills should be compared with CKM-HI. Yet, the conclusions obtained from the current analysis are still valuable: at a minimum, even with one model formulation investigated here, the results clearly demonstrate the advantage of using an integration-level model, which inspires further investigation.

One phenomenon is worthy to be further investigated: CKM-HI only achieves a limited predictive improvement over CKM, but the advantage in parameter plausibility is non-trivial, which should significantly increase the accuracy of latent knowledge inference. I hypothesize that this may be due to the fact that CKM uses less plausible parameters to fit the data, and that a larger prediction gain should be revealed if we impose parameter constraints. In the next section, I describe experiments on simulated datasets for testing this hypothesis.

In future work, I plan to explore skill integration in a broader range of contexts, while continuing to contribute to best practices in evaluating adaptive educational systems. Specifically, I plan to explore automated methods for extracting integration skills, advancing my preliminary approach [Huang et al., 2016].

7.3 SIMULATED DATASET EXPERIMENTS

The goal of this section is to understand the limited predictive improvement of CKM-HI over CKM, as shown in the real-world dataset, and answer the following question: will the magnitude of predictive improvement of CKM-HI over CKM increase when we impose parameter constraints on both models?

As reported above, CKM-HI has only achieved a limited predictive improvement over CKM, but it has significantly reduced the *guess* and *slip* noise parameters on the real-world dataset. This might be caused by CKM utilizing higher noise parameters to fit the data, which leads me to the hypothesis that a larger prediction gain would have been revealed if we had imposed parameter constraints on both models. However, in real-world datasets, it's not clear what the ground truth values of *guess* and *slip* noise parameters are, and thus it is not clear how strict the parameter constraints should be. So, I created simulated datasets where the ground truth values of *guess* and *slip* noise parameters were specified, and examined both models' predictive performance with different levels of parameter constraints (including the one exactly matching the ground truth level). Note that using simulated datasets to examine learner models has been used in several previous research [Pardos and Heffernan, 2010b, Lee and Brunskill, 2012, Klingler et al., 2015].

7.3.1 Method

Figure 17 shows the Bayesian network used to generate simulated datasets, which is called the *true* network in the experiments. In terms of the structure, it consists of three basic skills, three conceptual parts of the integration skills, three full integration skills, and seven

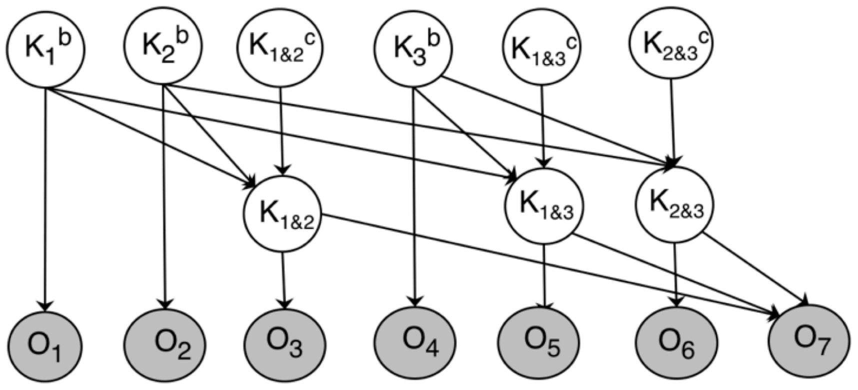


Figure 17: The Bayesian network for generating simulated datasets. O nodes represent the observed binary performance and K nodes represent latent binary knowledge levels.

items which progressively assess these skills. No learning is assumed. Each sample consists of observations on all items. A total number of 500 samples (corresponding to 500 students) was generated. 80% (400) of the samples (students) were used as the training set, and the remaining 20% (100) samples (students) were used as the test set.

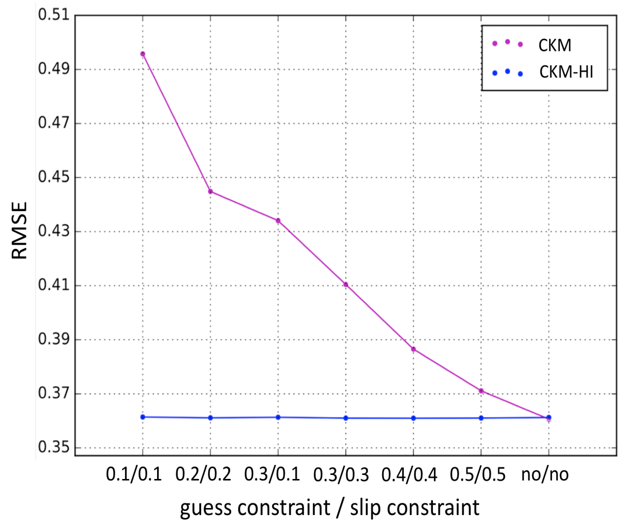
The parameters of the true network were set as follows. For each basic skill K_i^b, the probability of a student initially knowing the skill, P(K_i^b=T) was set to be 0.95; for each conceptual part of the integration skill $K_{i\&j}^c$, the probability of a student initially knowing the skill, P($K_{i\&j}^c$=T) was set to be 0.05; the conditional probabilities of each integration skill $K_{i\&j}$ given the values of its parents K_i^b, K_j^b and $K_{i\&j}^c$ was set as follows: {P(T|TTT)=0.99, P(T|TTF)=P(T|TFT)=P(T|FTT)=0.1, P(T|TFF)=P(T|FTF)=P(T|FFT)=0.05, P(T|FFF)=0.01}, where students have a low probability of knowing an integration skill if any of its basic and conceptual parts is unknown and has a high probability of knowing the integration skill otherwise. For all items, *guess* and *slip* values found in prior literature [VanLehn et al., 1998, Pavlik et al., 2009, Liu and Koedinger, 2017] were used as the parameter values in the simulation: *guess*/*slip*=0.1/0.1, 0.2/0.2, 0.3/0.1, 0.3/0.3. As can be seen, the true network for generating the simulated data is ideal for the CKM-HI model where CKM-HI should demonstrate a higher predictive performance than CKM, even without imposing parameter constraints; if not, it will suggest that the noise parameters of CKM might have played an important role in fitting the data.

The CKM-HI model used to learn from the simulated datasets has the same structure as the true network, but it has different initial parameters for the network: P(K_i^b=T)=0.85, P($K_{i\&j}^c$=T)=0.15, the conditional probabilities of each integration skill node $K_{i\&j}$ given the values of its parents K_i^b, K_j^b and $K_{i\&j}^c$ are set as follows: {P(T|TTT)=0.99, P(T|TTF)= P(T|TFT)=P(T|FTT)=0.25, P(T|TFF)=P(T|FTF)=P(T|FFT)=0.1, P(T|FFF)=0.01}, *guess*=*slip*=0.3. The CKM model used to learn from the simulated datasets has the same structure as the one obtained by removing all integration skill related nodes and edges from the true network, and the initial values of its overlapping parameters with CKM-HI are set to the same: P(K_i^b=T)=0.85, P($K_{i\&j}^c$=T)=0.15, *guess*=*slip*=0.3.

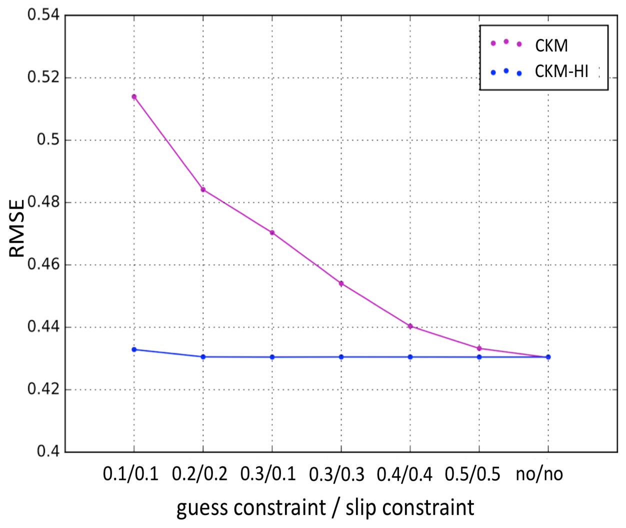
To impose parameter constraints on the training process, I used the technique from Bayesian statistical modeling of adding strong Bayesian priors to the *guess* and *slip* parameters. I achieved this by using a large effective sample size (ESS=10,000) in the prior distribution for *guess* and *slip* parameters. I use the usual conjugate prior, beta distribution, for the binomially distributed *guess* and *slip* parameters (random variables) here. Such strong priors should only be imposed on large *guess* and *slip* parameters, and not on *guess* and *slip* parameters that are already small; otherwise, it will fundamentally distort the entire fitting procedure. In order to address this, I firstly fit the parameters without strong priors (constraints), and then added strong priors to a *guess*/*slip* parameter if it exceeds 0.3 and conducted a new round of fitting. In each *guess* and *slip* ground-truth setting, I tried a range of priors ranging from more strict to more relaxed constraints on the *guess/slip* parameters.

7.3.2 Results

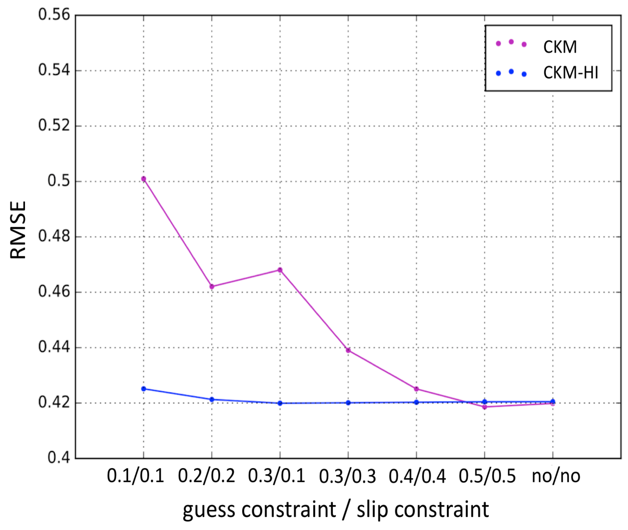
Figure 18 shows the comparison of RMSE values on test sets with training and test sets generated from different ground truth *guess* and *slip* parameters. In each setting (subfigure), each learner model is fitted with different parameter constraints, ranging from the most strict (*guess*$\leq$0.1, *slip*$\leq$0.1) to the least strict (*guess*$\leq$0.5, *slip*$\leq$0.5), to no constraint. Surprisingly, without any parameter constraints, CKM-HI and CKM don't have any difference in predictive performance, as measured by RMSE, even though the simulated datasets are generated by an ideal CKM-HI network.



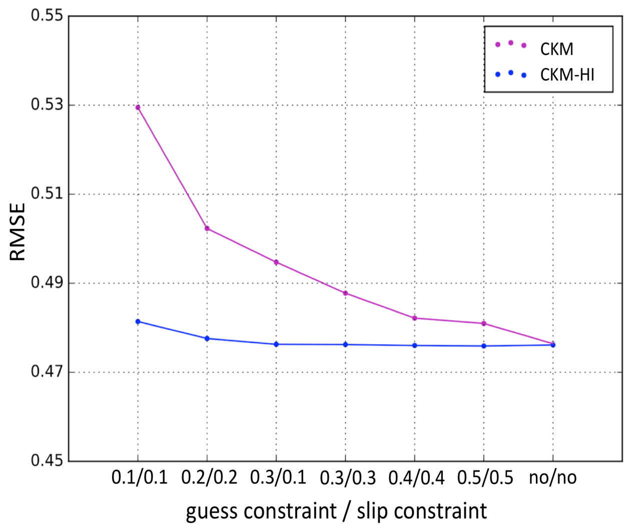
(a) True guess/slip=0.1/0.1



(b) True guess/slip=0.2/0.2

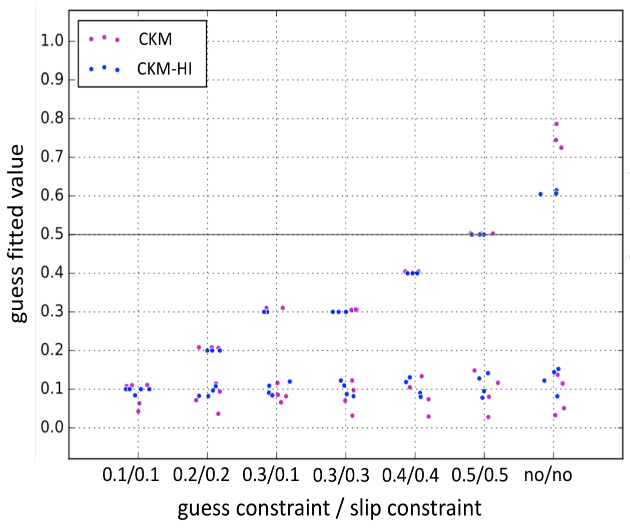


(c) True guess/slip=0.3/0.1

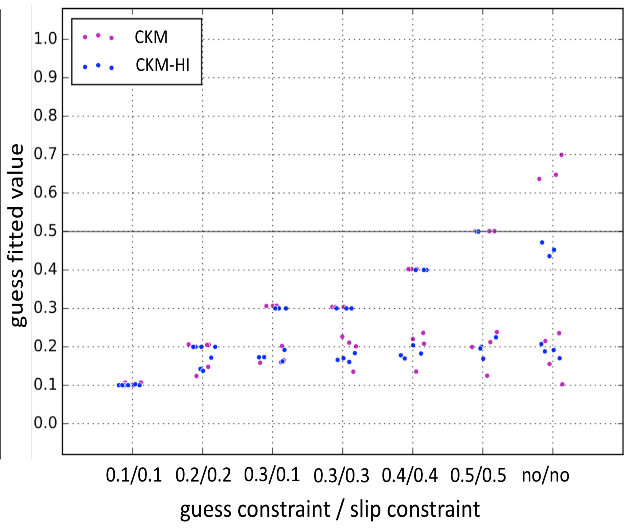


(d) True guess/slip=0.3/0.3

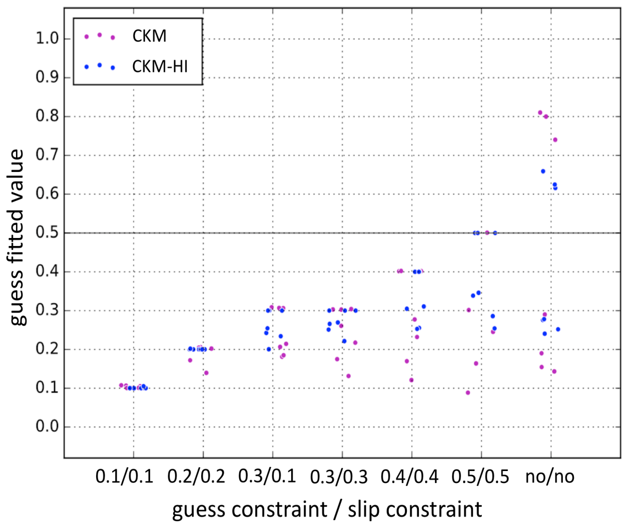
Figure 18: Comparison of RMSE values on test set prediction with datasets generated from different ground truth *guess* and *slip* parameters. In each setting of the ground truth parameters (i.e., in each subfigure), each learner model is fitted with different parameter constraints, ranging from the most strict (*guess*$\leq$0.1, *slip*$\leq$0.1) to the least strict (*guess*$\leq$0.5, *slip*$\leq$0.5), to no constraint.



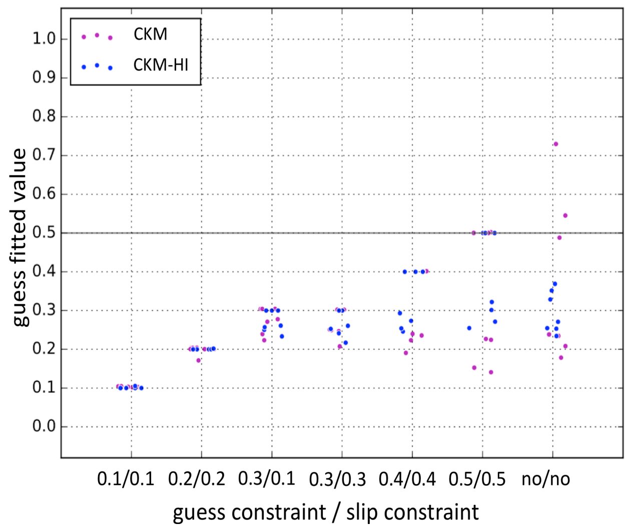
(a) True guess/slip=0.1/0.1



(b) True guess/slip=0.2/0.2

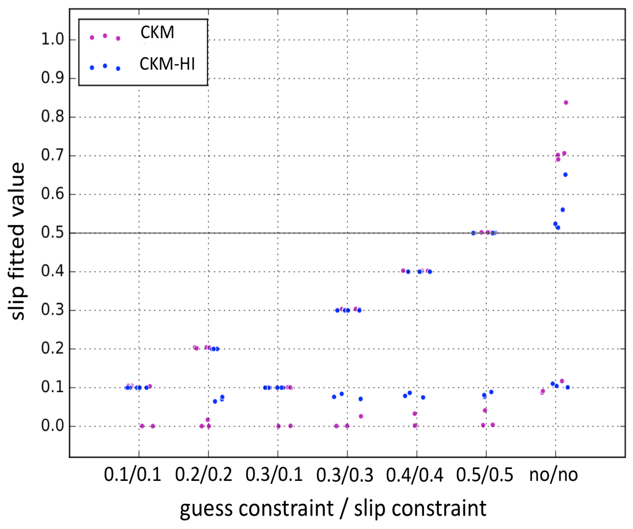


(c) True guess/slip=0.3/0.1

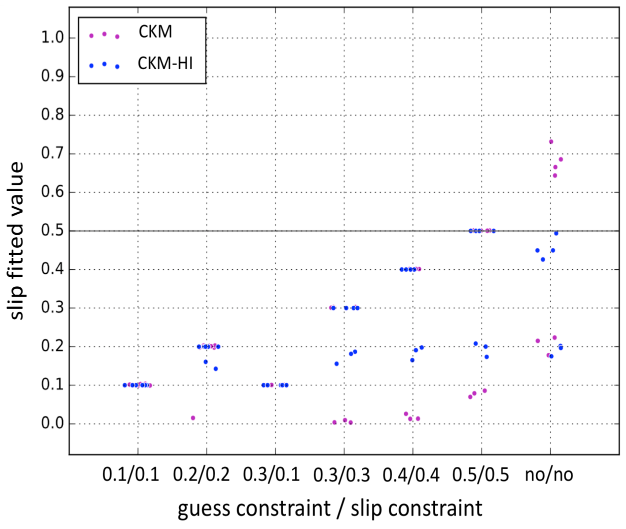


(d) True guess/slip=0.3/0.3

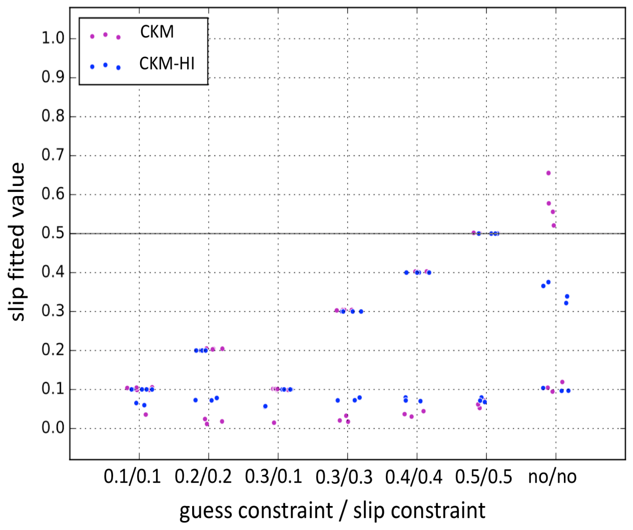
Figure 19: Comparison of *guess* values fitted on datasets generated from different ground truth *guess* and *slip* parameters. In each setting of the ground truth parameters (i.e., in each subfigure), each learner model is fitted from the most strict parameter constraints (*guess*$\leq$0.1, *slip*$\leq$0.1) to the least strict (*guess*$\leq$0.5, *slip*$\leq$0.5), to no constraint.



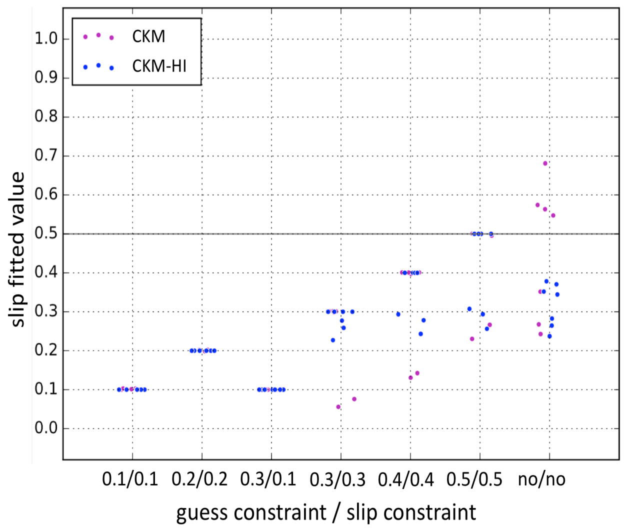
(a) True guess/slip=0.1/0.1



(b) True guess/slip=0.2/0.2



(c) True guess/slip=0.3/0.1



(d) True guess/slip=0.3/0.3

Figure 20: Comparison of *slip* values fitted on datasets generated from different ground truth *guess* and *slip* parameters. In each setting of the ground truth parameters (i.e., in each subfigure), each learner model is fitted from the most strict parameter constraints (*guess*$\leq$0.1, *slip*$\leq$0.1) to the least strict (*guess*$\leq$0.5, *slip*$\leq$0.5), to no constraint.

As more and more strict parameter constraints are added, CKM dramatically decreases its predictive performance while CKM-HI maintains it. Figure 19 and Figure 20 confirm that my method of imposing constraints using strong Bayesian priors has indeed constrained the parameters within each specified limit. As shown by the figures, without imposing any parameter constraints, CKM has consistently fitted higher *guess* and *slip* parameters than CKM-HI (particularly in the clearly implausible regions where *guess*$\geq$0.5 or *slip*$\geq$0.5) under different ground truth settings.

7.3.3 Discussion and Conclusions

In this section, I describe the methods I used to conduct a simulated experiment addressing the limited predictive improvement of CKM-HI over CKM in the real-world dataset, and found that the magnitude of the predictive improvement of CKM-HI over CKM increases after imposing parameter constraints on both models. After adding parameter constraints, CKM has dramatically decreased the predictive performance while CKM-HI maintains a similar predictive performance, in the simulated datasets.

These results suggest that the reason that CKM-HI has only a limited predictive improvement over CKM but a dramatic improvement on parameter plausibility on the real-world dataset is that CKM has fitted higher (less plausible) *guess* and *slip* parameters due to the fluctuations (irregularities) in problem performance caused by integration skills. Meanwhile, CKM-HI has explicitly specified integration skills to account for the fluctuations in the performance, and thus has a better modeling mechanism with a higher parameter plausibility without sacrificing predictive performance. High guess or slip parameters decrease the accuracy and reliability of the latent knowledge inference provided by learner models, since student performance is less attributed to the skills specified in the learner models, but more attributed to noise. Thus, imposing parameter constraints in the parameter fitting process is important.

The technique of configuring a strong Bayesian prior, to impose constraints is straightforward and effective on simulated experiments, yet it remains questionable as to how to set priors on the real-world dataset. Although several studies have imposed parameter constraints (e.g., disallowing negative learning rates) on logistic regression based models [Gong et al., 2010, Pavlik et al., 2009], adding parameter constraints to Bayesian network based learner models is still a challenging, less investigated topic. In future work, I will investigate adding parameter constraints to learner models when training on real-world datasets.

8.0 REAL-WORLD ADAPTATION EFFECTIVENESS EVALUATION OF THE PROPOSED INTEGRATION-LEVEL LEARNER MODEL

In this chapter, I explain the design and results of the Classroom Study 2 for investigating the adaptation effectiveness of the proposed integration-level learner model (CKM-HI) deployed in an ITS for teaching students program comprehension skills. This chapter aims at answering my RQ 5, *Is learner modeling for integration skills beneficial in terms of improving student learning when applied in an ITS?* In the following sections, I start by introducing the study design, and then report analyses from several aspects including posttest performance, time, performance under time constraints and the composition effects.

8.1 METHOD

This study compares the adaptation effectiveness of two versions of the program comprehension ITS, *T3* (introduced in Chapter 6), which are driven by two different learner models: the proposed model CKM-HI (introduced in Chapter 4) and the traditional learner model CKM (introduced in Section 7.1). There are three main reasons for choosing CKM over WKT as the baseline: 1) If I compare CKM-HI and WKT directly, it is not clear whether the benefit comes from the credit-blame assignment mechanism, or from incorporating the integration skills; 2) CKM is already one of the most widely used multi-skill knowledge tracing models, as shown in my literature review (Chapter 2), 3) my prior work [Huang et al., 2017] on a Java program comprehension dataset has shown that CKM is superior to WKT over a range of aspects; and 4) a more solid comparison requires a larger scale, more students to reach enough statistical power, but in my study, only a limited number of students were available.

I chose a between-subject design, where students were randomly assigned to one of the two versions of the ITS. I called the condition using the CKM-driven ITS the *control* group, and the one using the CKM-HI driven ITS the *experimental* group. Borrowing the terminologies from [Vanlehn, 2006], these two versions of ITS differ, but only in the outer-loop problem selection and the inner-loop hints (as a result of the differences in their learner models). They share the same set of problems and the same interface. Section 8.1.1 introduces their differences in problem selection, and Section 8.1.2 introduces their differences in hints. The remaining sections introduce study procedures, learner model parameters and initialization, practice problems and skill, and the pre/posttest.

8.1.1 Differences in Problem Selection

In both conditions, I employed the same problem selection algorithm. This algorithm is based on the generic algorithm introduced in Section 6.5, with a modification to its stop-practice criterion. The main reason for this modification is that the practice time in both conditions is now controlled to be identical, and the performance on posttests (and pretests) will be examined to compare the learner models. To achieve this, the algorithm gradually increases mastery thresholds to select unmastered skills, and switches to random selection when the maximum threshold is reached[1]. Since T3 is a new tutor with no prior data or knowledge to help set up the parameters, any fixed mastery threshold risks giving improper instructional decisions (the effect of mastery thresholds is discussed in Section 5.1.3.3). Also, if students in a condition are stopped later and also have higher posttest scores than those in the other condition, it will not be clear whether the benefits result from simply giving more practice, which could be achieved by a naive model, or result from the accuracy of knowledge inference. Moreover, both conditions also receive the same pool of problems and the same learner model parameters (for the overlapping parameters), so the selection of practice problems are only affected by the learner model and the student's own performance.

[1]The details of this algorithm are explained as follows: For each student, the initial mastery threshold m is set at 0.90. If there are no unmastered skills using this threshold, then the algorithm raises the mastery threshold m by one unit d (d=0.01 when $m \leq 0.99$ and d=0.001 when $m > 0.99$) and keeps doing so until there is at least one unmastered skill. If the mastery threshold has been raised to $m > 0.9999$ but there are still no unmastered skills, then an unattempted problem will be randomly selected. Note that there is a sufficient number of practice problems and none of the students has been able to finish even half of the problems.

Based on the different skill models and knowledge update mechanisms of CKM-HI and CKM, the two conditions can provide different practice experiences for students. If a student has already mastered basic skills but still has difficulty with the integration of basic skills, the CKM-HI condition could use the unmastered integration skill(s) to identify targeted integration problems, while the CKM group might still use basic skills to search within a mixed pool of integration problems and basic problems to select problems. In the result section, I provide validation that these two conditions indeed differ in their selection of problems.

The CKM-HI condition could not only differentiate integration practice from basic practice better than the CKM condition could, but on a finer-grained level, could also further differentiate isolated integration practice from full integration practice, adapting to a student's level. According to the problem selection algorithm in Section 6.5, a student's stronger unmastered skill has a higher chance to be selected first and a problem with the highest focus on the chosen skill will be selected. According to the BN structure and parameters (explained later in Section 8.1.4) of CKM-HI, the conceptual part of an integration skill will always be stronger than the full integration skill (since the former is the prequisite of the latter). So when a student has already mastered basic skills but still doesn't know how the integration works, the conceptual part of an integration skill will be more likely to be selected first than the full integration skill, and the isolated integration problems will be selected (if not yet attempted) since they have the highest focus on the conceptual parts of integration skills compared to other integration problems.

8.1.2 Differences in Hints

Both conditions also differ in hint messages for cells labeled with conceptual parts of integration skills. As introduced in Section 6.3, hint messages are generated based on the underlying learner model (more specifically, the skill model used by the learner model). The learner models from two conditions differ in their skill labels for cells requiring conceptual parts of integration skills: such cells in the CKM group are labeled with basic skills, while such cells in the CKM-HI group are labeled with the conceptual parts of integration skills.

When a problem addressing an integration skill is selected to be presented, CKM is not aware of the integration skill underlying the problem, and gives hints explaining the basic skill, while CKM-HI in the experimental condition is aware of the underlying integration skill and presents hints explaining the conceptual parts of the integration skill. Note that both conditions differ in their first level hints (second level hints are not provided) but have the same bottom-out hints (which give the correct answers directly). In order to make the comparison fair, I made sure that the number of words in both conditions were similar with at most a one - three word count difference, in both English and Spanish. An example can be found in Table 15 (Chapter 6).

8.1.3 Procedure

This study was held as an in-class practice session in a computer lab, two weeks after the topic of *lists* was lectured on (the topic of *for* loops had already been lectured on several weeks before this) and students had already done some exercises on the *lists* topic. The session lasted for about 80 minutes. The procedure is as follows:

1. *Demo*: Firstly, students were given instructions on how to use the system and required to finish a very simple demo example within the system.
2. *Pretest*: Secondly, students were required to take a pretest.
3. *Practice*: After each student had finished the pretest, they were able to access the main practice link. Here students were given a sequence of program comprehension problems that were selected by the backend student engine.
4. *Posttest*: 15 minutes before the class ended, the posttest link was activated, and students were asked to exit the main practice window and attempt the posttest.

8.1.4 Learner Model Parameters and Initialization

Both learner models are constructed on the step level (since the fine-grained step level performance is available from T3), and perform a knowledge update after each problem submission. Learner models in both conditions have the same *guess* and *slip* probabilities (0.1) on step level items, and the same initial probability (0.5) for all skills (including basic skills and conceptual parts of integration skills) for all students. For CKM-HI, the conditional probability table of each integration skill node $K_{i\&j}$ given the values of its parents K_i^b, K_j^b and $K_{i\&j}^c$ is set to be {P(T|TTT)=0.999, P(T|TTF)=P(T|TFT)=P(T|FTT)=P(T|TFF)=P(T|FTF)= P(T|FFT)=P(T|FFF)=0.001}, indicating that a student has a very low probability (0.001) of knowing an integration skill if he/she doesn't know any of the prerequisite basic skills or conceptual parts of integration skills, and a very high probability (0.999), otherwise.

Both learner models use students' pretest scores to update their knowledge levels. Once a student attempts the pretest, the performance is used to update the student's learner model, so when each student starts his/her main practice, the student could be recommended problems based on his/her own learner model.

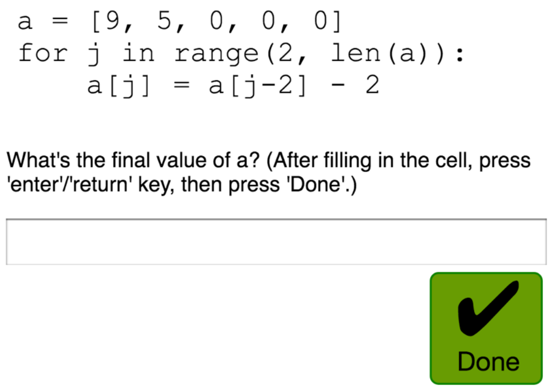
Note that I didn't use parameters trained from data because there was no past data on the same type of contents; nor did I use expert knowledge to set the initial knowledge probabilities in a way that differentiates different skills' intrinsic difficulties, because it may introduce bias and unfairness across the conditions.

8.1.5 Practice Problems and Skills

The main practice problems are trace table problems, as introduced in Section 6.1. The problem and skill focus is on the topic of *lists* and the topic of *for* loops. The problems designed in this study focused on five integration skills: *for&for, for&#=, for&if, for&a[i], a[i]&=* (the definitions for these are introduced in Section 4.1 Table 12). Only the CKM-HI condition is aware of these integration skills. At the same time, there are six basic skills involved: *for, #=, if, a[i], %, //* (definitions introduced in Section 4.1 Table 13). Both conditions are aware of these basic skills. Different variants of problems were created for each integration skill and each basic skill. They were designed in a way that didn't introduce new skills to be learned, but merely slightly changed the degree of overall difficulty. For example, variants within *for&#=* could be: getting the average of a sequence of numbers with a *for* loop and an addition assignment, or getting the sum of the even numbers with a *for* loop, an addition assignment, *if* and *modulus (%)*, etc. Each integration skill has one isolated problem (Chapter 6 Figure 12) designed for practicing the conceptual part of the integration skill. Each basic skill has at least two problems to practice on, and each integration skill has at least five problems to practice on. In total there are 66 problems, with 42 (64%) of the problems requiring integration skills.

8.1.6 Pretest and Posttest

The adaptation effectiveness of a learner model is measured by the learning effectiveness (considering both accuracy and speed in solving problems) for students, based on the pretests and posttests. In addition, the pretest was also designed for initializing the learner models. I designed six pretest problems each of which tested only one basic skill, and they are also referred to as *basic* problems. I didn't include integration skills in the pretest, due to practical constraints explained as follows. The TutorShop platform doesn't have an automatic time control; it depends on manual activation/deactivation of different study stages (assignments) by the teacher/administrator, or it automatically activates the next stage once a student finishes all the problems in the current stage. If students mostly arrive at the same time, manual activation will be the perfect choice for controlling time on pretest, yet according to the teachers, many students often arrive late, making manual time control difficult to accomplish. In addition, due to the implementation time limit and technical complications, I couldn't include an automatic time control in the backend service in time for deployment. So I decided to use the default access design in TutorShop for the pretest and practice stages and limited the pretests to only basic problems so that students could have more time for the main practice and the time they needed for the posttest. Meanwhile, learner models could still at least be aware of students' initial levels of basic skills based on students' performance on basic problems. The posttest included the same kind of basic problems (six of them) as in the pretest (with changes in the values of the literals), while adding five integration problems covering all the targeted integration skills. Each posttest problem focused on one to four targeted integration skills. These five integration problems are also referred to as *integration* problems. In each pretest or posttest problem, students were asked to give the printed output or final values of the key variables, and the trace table space was not provided in the problem. Figure 21 shows the interface for a pretest or posttest problem.



```
a = [9, 5, 0, 0, 0]
for j in range(2, len(a)):
    a[j] = a[j-2] - 2
```

What's the final value of a? (After filling in the cell, press 'enter'/'return' key, then press 'Done'.)

Figure 21: The interface for a pretest/posttest problem.

In terms of grading pretests and posttests, each problem was graded as correct(1) or incorrect(0). I refer to the proportion correct on the pretest/posttest as *pretest/posttest score*. I used the last attempt for each problem, and allowed only the key part matching, i.e., if the target-skill-related parts were correct, then the answer was graded as correct. This ignores most of the printing-format-related errors (except in nested loop, where the printing format might matter). For example, when students were asked to print a sequence of numbers in one line without spaces, answers that added spaces or commas as separators between numbers were considered as correct. In pretest and posttest phrases, all students were shown the problems in the same sequence, and students could only move to the next problem after submitting the previous problem.

8.1.7 Participants

I conducted the classroom study (Classroom Study 2) in an introductory Python programming course at the Austral University of Chile (Universidad Austral de Chile) in November of 2017. 80 students submitted at least one problem, and 74 students submitted at least one problem in each of these: pretest, main practice and posttest. These 74 students were used for the final analysis. Among the 74 students, 36 were assigned to the control condition and 38 were assigned to the experimental condition. All hints and instructions in the system were translated into Spanish since the students were Spanish speakers.

8.2 RESULTS

This section reports analyses from different aspects to compare the learning effectiveness between the control and experimental conditions, and thus compare the real-world adaptation effectiveness of CKM and CKM-HI learner models. To start with, there are three important aspects to examine, so that reliable conclusions about learner models can be drawn upon.

Firstly, I examined whether both conditions differ in the learning materials provided in the practice stage. As shown in Figure 22, both conditions indeed differ in the percentile of different types of problem selected by the learner models. Specifically, more than 50% of the selected problems were basic problems in the condition group, while more than 50% of selected problems were integration related problems in the experimental group. This difference also reflects the difference in hints the students have received, since integration skill related hints are only provided in integration related problems.

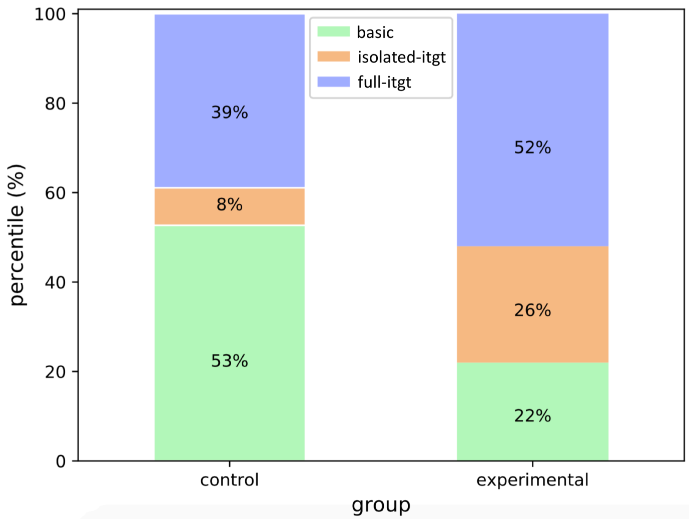


Figure 22: Percentile comparison of different types of problems (basic/isolated-integration/full-integration problems) between conditions.

Secondly, I examined total practice time and the number of problems in both conditions, to be sure that any advantage in learning effectiveness only results from the learner models (which provide different problem sequences and hints), rather than being due to a longer practice time or having a higher number of practice problems. According to my study design, students in both conditions should have spent the same total time in the practice stage (since learner models in both conditions kept selecting practice problems from a large pool of problems, until 15 minutes before the class ended). Indeed, a two-sample t-test (after confirming approximate normality) comparing the total practice time (minutes) in the control condition (M=41.0, SD=9.4) and experimental condition (M=44.1, SD=9.7) revealed no significant difference [t(72)=-1.37, p=0.17] between the conditions. Similarly, a two-sample t-test (after confirming approximate normality) comparing the number of practice problems in the control condition (M=12.9, SD=11.8) and experimental condition (M=13.2, SD=8.7) revealed no significant difference [t(72)=-0.10, p=0.92] between the conditions. Both conditions have statistically the same total practice time and number of problems.

Finally, I examined whether students in each condition have learned from the system in general. I only considered the basic problems in the posttest in comparison to the pretest, because the pretest only covers basic problems. In the control condition, a paired t-test (after confirming approximate normality) comparing the proportion correct of posttest basic problems (M=0.90, SD=0.15) to that of the pretest (M=0.82, SD=0.18) for control group students, revealed a significant difference [t(35)=2.41, p=0.02, d=0.44]; in the experimental condition, a paired t-test (after confirming approximate normality) comparing the proportion correct of posttest basic problems (M=0.88, SD=0.15) to that of the pretest (M=0.75, SD=0.21) for experimental group students, revealed a significant difference [t(37)=4.25, p<0.001, d=0.75]. So students did in fact learn from the system in each condition.

Next, I report the main results, which compare the learning effectiveness of two versions of ITSs in order to decide whether the CKM (control condition) or the CKM-HI (experimental condition) learner model has the best adaptation effectiveness.

8.2.1 Posttest Scores

To compare the learning effectiveness between the two educational interventions, one of the most common ways is to compare posttest scores (controlling for pretest scores), after students have finished the practice stage. To start with, I examined pretest scores to see whether it was necessary to account for pretest scores when comparing posttest scores. A two-sample t-test (after confirming approximate normality) revealed a marginally significant difference [t(72)=1.69, p=0.096, d=0.39] between the control condition (M=0.82, SD=0.18) and the experimental condition (M=0.75, SD=0.21). Thus, I found it necessary to take into account pretest scores when comparing posttest scores. The conventional learning gain (i.e., the increased scores from pretest to posttest) computation which accounts for pretest scores is not readily applicable here, since the posttest almost doubled the number of problems by adding the integration type of problems not included in the pretest.

Based on the above considerations, I have chosen a regression analysis that easily allows taking into account pretest scores, and in particular, I have chosen *generalized linear mixed models* (GLMM) which capture the dependency among data, allowing for the variances introduced by problems and students. I constructed GLMMs (by R package *lme4*) to predict posttest correctness per problem per student, given the continuous overall pretest score (fixed intercept), the binary condition indicator (fixed intercept), the problem indicator (random intercept) and the student indicator (random intercept). A positive significant coefficient of the condition indicator will suggest that students in the experimental group have significantly higher posttest scores than those in the control group, on average.

Also, in order to have a fine-grained understanding, I further split the integration problems into *easy integration* problems (*itgt-easy*) and *hard integration* problems (*itgt-hard*), depending on whether the proportion correct was higher than 0.35 or not. This resulted in placing three problems in the easy integration group, and two problems in the hard integration group. The hard integration group contains the only two nested loop related problems.

Then, I constructed separate GLMM models for the different posttest problem groups. As shown in Table 22, the coefficients for the condition indicators are all positive yet not significant (and same coefficient estimates were obtained for models without the random intercept of problems), with the easy integration group having the condition indicator close to marginally significant (p=0.13). Figure 23 visually displays the means and confidence intervals of the posttest scores of different problem groups within each condition. The posttest scores considering different problem groups between the two conditions are statistically the same, even though the pretest score difference is taken into account.

Table 22: Generalized linear mixed models predicting correctness per problem per student based on the formula: $posttest\_correctness \sim condition + pretest\_score + (1|problem) + (1|student)$. Likelihood Ratio Test statistics (χ^2) and p-values comparing each model with its counterpart (which removes the targeted fixed effect) are reported. Residuals are (approximately) normally distributed. For models with a random effect SD=0, results of models without the random factor are reported. (Sig. level ***:<.001, **:<.01, *:<.05.)

posttest	#obs	fixed effect								random effect	
prob.		condition(EP)				pretest_score				prob.	stu.
grp		Est.	SE	$\chi^2(1)$	p	Est.	SE	$\chi^2(1)$	p	SD	SD
basic	444	0.09	0.39	0.05	0.82	3.42	0.93	14.30	<0.001***	0.67	0.74
itgt-easy	222	0.88	0.59	2.32	0.13	8.08	1.96	24.3	<0.001***	–	1.68
itgt-hard	148	0.33	0.98	0.11	0.74	6.00	2.82	5.97	0.015*	–	3.03
itgt	370	0.91	0.79	1.40	0.24	8.92	2.39	19.1	<0.001***	0.13	2.66
all	814	0.40	0.48	0.67	0.41	5.34	1.22	17.91	<0.001***	2.35	1.77

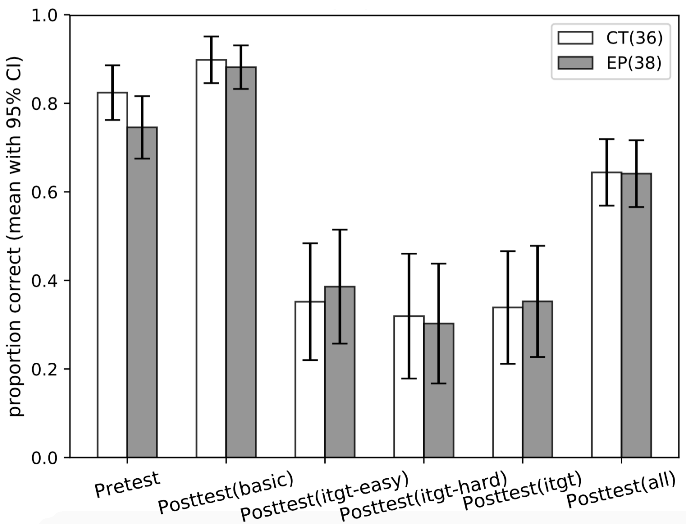


Figure 23: Comparison of the posttest scores over different sets of problems between the condition (CT) and experimental (EP) groups.

Is the effect on posttest scores from conditions dependent on the students' levels? To answer this, I added an interaction term between the pretest score and the condition indicator into each of the above models. Still, the coefficients of the condition indicator from all models were positive but not significant ($p>0.12$); the interaction terms (for the experimental condition) were all negative and not significant ($p>0.18$). The lack of significant interaction and the direction of the interaction terms is a bit unexpected, since the CKM-HI model should be able to help students with high level basic skills but low level integration skills learn more effectively, while behaving similarly to CKM for students with low level basic skills. One probable reason is that the pretest with only basic problems fails to reflect students' integration skill levels, and thus the pretest score is not representative of students' knowledge levels.

Overall, there is insufficient evidence showing that the experimental intervention (CKM-HI learner model) could significantly increase posttest scores compared with the control intervention (CKM learner model) in the current analysis setting. My following analysis continues to examine the effects of the experimental intervention from other aspects.

8.2.2 Posttest Time

Although there was no significant difference between posttest scores for the two conditions, students sometimes differed in the time they spent on posttest problems. In this section, I report three different ways to analyze this to help draw reliable conclusions.

Firstly, I conducted an initial analysis where I compared the students' median time over posttest problems in both conditions. For each student, I calculated the median time over the set of posttest problems, and then I compared the average value of such median time over students from two conditions. However, for students who failed to submit all posttest problems (as shown in Table 23), the median time over the submitted problems will introduce bias towards the time required on earlier problems. To address this, I conducted an imputation on missing problems: I computed the average value of time within a problem group (basic/integration), and filled the missing value with the corresponding average according to its group association. Note that before computing any median or average values of time of one condition group, I always conducted a 90% *winsorizing*, setting data below the 5th percentile to the 5th percentile, and data above the 95th percentile to the 95th percentile, if the distribution fails the Shaipiro-Wilk normality test (α=0.05) .

Table 23: Number of students who submitted each specified posttest problem, numbered 1 to 11, in the control (CT) and experimental (EP) conditions.

	total	basic problems						integration problems				
	#stu	1	2	3	4	5	6	7	8	9	10	11
CT	36	36	36	36	36	34	34	34	32	29	28	27
EP	38	38	38	38	38	38	38	37	37	37	36	35

Table 24 reports basic descriptive statistics and statistical test results. As shown in the table, both conditions don't have statistical difference in the average pretest median time, but students in the experimental condition spends significantly less median time on posttest basic problems (p<0.05) and hard integration problems (p<0.01) with around medium effect size (d≥0.49). Particularly, on posttest hard integration problems, the average median time has decreased by around 36 seconds (from 115.9 seconds on the condition group to 80.3 seconds on the experimental group). As a result of the effect from hard integration problems, viewing integration problems as a whole or all problems as a whole, the reduction in median time is still significant (p<0.05) with around medium effect size (d≥0.49). Similar conclusions can be drawn from data without imputation. Figure 24 displays the means and confidence intervals of median time over problem groups in each condition for better visual inspection.

Secondly, I conducted linear mixed modeling (LMM) which accounts for the individual time differences (or variances) on problems and students and allows modeling dependency among the data. The constructed linear mixed models (by R package *lme4*) predict time on each submitted posttest problem of a student, based on the following predictors: the binary condition indicator (fixed intercept), the continuous total time on pretest for this student (fixed intercept), the problem indicator (random intercept) and the student indicator (random intercept). I constructed each LMM based on only the data of a targeted set of

Table 24: Descriptive statistics and statistical tests based on the median time (sec) for different sets of problems, between the control (CT) and experimental (EP) conditions. The distributions are approximately normally distributed. A Welch's t-test was conducted for the *posttest(all)* group since Levene's test for equal variance failed. (Sig. level **:<.01, *:<.05.)

	CT (#obs=36)		EP(#*obs*=38)		two-sample t-test		
	M	SD	M	SD	t(72)	p	Cohen's d
pretest	71.7	29.5	67.0	25.3	0.73	0.469	–
posttest(basic)	37.9	15.2	31.4	11.2	2.08	0.041*	0.49
posttest(*itgt-easy*)	93.1	41.6	82.2	35.3	1.21	0.231	–
posttest(*itgt-hard*)	115.9	52.5	80.3	41.4	3.20	0.002**	0.74
posttest(*itgt*)	96.5	41.6	77.0	36.4	2.12	0.037*	0.49
posttest(all)	57.1	23.3	46.0	14.7	2.39	0.020*	0.56

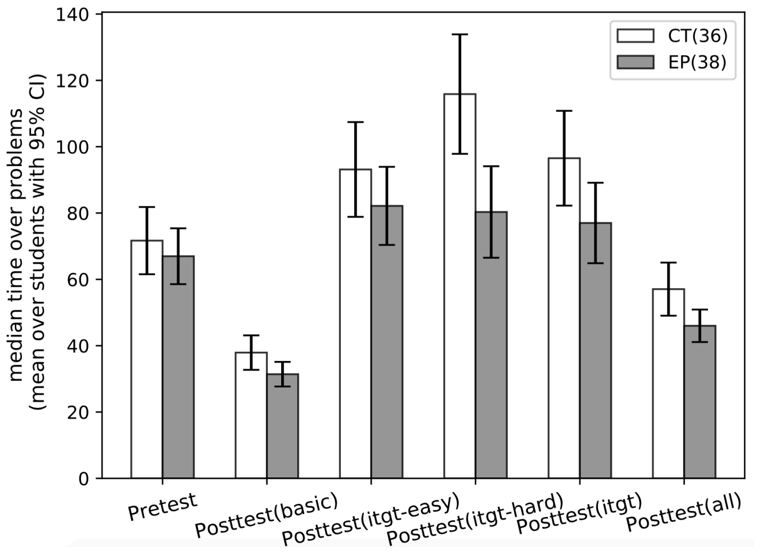


Figure 24: Comparison of the median time for different sets of problems, between the control (CT) and experimental (EP) groups.

posttest problems, without the time imputation, and with the winsorizing on pretest total time and posttest time within each subset of the data. The results are reported in Table 25.

As shown in the Table 25, students in the experimental condition used significantly less time ($p<0.01$) on the hard integration problems with an average of 36 seconds fewer than in the control group. As a result of less time being spent on hard integration problems, viewing integration problems or all problems as a whole, the reduction in time is still significant ($p<0.05$). On basic problems and easy integration problems, students in the experimental group also spent less time, although the difference was not significant ($p>0.1$). The same levels of significance for the condition indicator coefficients were also found for models without the non-significant pretest total time predictors (e.g., models for *itgt-hard* and *itgt* groups).

Table 25: Linear mixed models predicting time spent (sec) on each submitted problem by a student based on the formula: $posttest\_time \sim condition + pretest\_total\_time + (1|problem) + (1|student)$. Likelihood Ratio Test statistics (χ^2) and p-values comparing each model with its counterpart (which removes the targeted fixed effect) are reported. Residuals of all models are (approximately) normally distributed. (Sig. level ***:<.001, **:<.01, *:<.05, •:<.1.)

posttest	#obs	fixed effect								random effect	
prob.		condition(EP)				pretest_total_tme				prob.	stu.
grp		Est.	SE	$\chi^2(1)$	p	Est.	SE	$\chi^2(1)$	p	SD	SD
basic	440	-4.05	2.82	2.04	0.15	0.052	0.007	38.00	<0.001***	11.3	9.0
itgt-easy	199	-7.38	9.05	0.66	0.42	0.047	0.025	3.50	0.06•	17.6	24.6
itgt-hard	133	-35.74	10.76	10.18	0.0014**	-0.009	0.029	0.10	0.75	26.1	22.2
itgt	332	-18.78	8.62	4.58	0.03*	0.025	0.023	1.08	0.30	21.3	27.8
all	772	-10.38	3.95	6.59	0.0103*	0.044	0.011	15.64	<0.001***	30.7	12.2

A common way to compare time on task in educational research is to compare the time spent on correct answers. However, as shown in Table 26, the number of students who succeeded in posttest integration problems was very small, which significantly reduces the power of any analysis. So, the results here only serve as a supplement to the above time analysis. I conducted similar linear mixed effect modeling (LMM) investigating the effect of the experimental condition. The linear mixed models (Table 27) predict time on each correct posttest problem of a student, based on the binary condition indicator (fixed intercept), the continuous variable of total time on correct pretest problems of this student (fixed intercept), the problem indicator (random intercept) and the student indicator (random intercept).

Table 26: Number of students who succeeded in each specified posttest problem, numbered 1 to 11, in the control (CT) and experimental (EP) conditions.

	total	basic problems						integration problems				
	#stu	1	2	3	4	5	6	7	8	9	10	11
CT	36	34	35	32	34	33	26	11	12	14	11	13
EP	38	31	37	35	36	32	30	15	13	16	10	13

Table 27: Linear mixed models predicting time spent (sec) per correct problem by a student based on the formula: $posttest\_time\_on\_correct \sim condition + pretest\_total\_time\_on\_correct + (1|prob) + (1|stu)$. LRT statistics ($\chi^2$) and p-values comparing each model with its counterpart (which removes the targeted fixed effect) are reported. Residuals of all models are (approximately) normally distributed. For the model with a random effect SD=0, results of the model without the random factor is reported. (Sig. level ***:<.001, **:<.01, *:<.05.)

posttest	#obs	fixed effect								random effect	
prob.		condition(EP)				pretest_total_time_on_correct				prob.	stu.
grp		Est.	SE	$\chi^2(1)$	p	Est.	SE	$\chi^2(1)$	p	SD	SD
basic	395	-4.06	2.92	1.92	0.16	0.038	0.007	23.8	<0.001***	11.0	9.4
itgt-easy	82	1.80	14.13	0.02	0.90	0.039	0.036	1.18	0.28	10.5	32.2
itgt-hard	46	-31.15	15.34	3.93	0.047*	0.013	0.038	0.12	0.73	27.8	–
itgt	128	-9.46	14.02	0.45	0.50	0.042	0.036	1.41	0.24	16.8	36.5
all	523	-4.86	3.06	2.49	0.11	0.039	0.008	22.8	<0.001***	29.6	7.0

According to Table 27, students in the experimental group spent significantly less time (31 seconds on average) to correctly solve a hard integration problem (p<0.05), and the same coefficient estimate was obtained in models without the random intercept of students. On other problem groups (basic/*itgt-easy*/*itgt*/all), the differences are not significant. Note that the datapoints available to construct integration problem related mixed models are limited, and also the standard errors for the coefficients of the condition indicator are much higher than those of the mixed models predicting time on a submitted problem (Table 25). For better visual inspection, Figure 25 displays the means and confidence intervals of median time on correct answers over problem groups in each condition, and it shows a similar pattern as in Figure 24, but with larger variances over correct integration problems (particularly on hard integration problems) than those over submitted integration problems. I leave for future research the collection of more data to conduct a more reliable analysis.

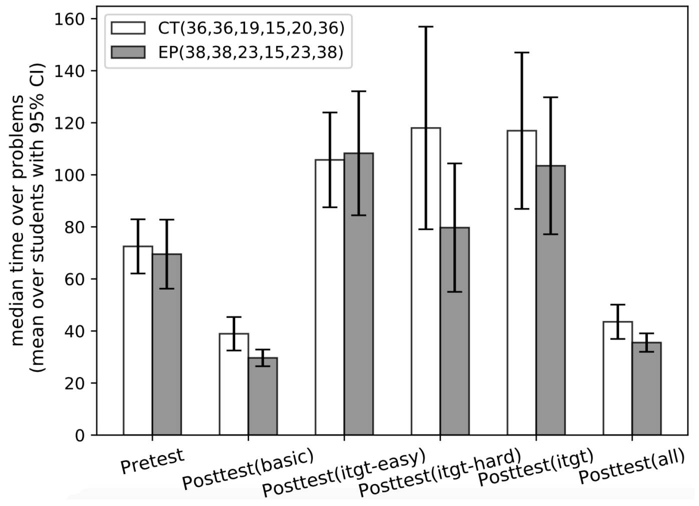


Figure 25: Comparison of the median time spent on correctly solved problems over different sets of problems, between the control (CT) and experimental (EP) groups.

The above posttest time related analyses demonstrated that the experimental group (CKM-HI group) helped students solve problems faster than the condition group, with a larger effect on integration problems than on basic problems and a consistently significant ($\alpha = 0.05$) effect on hard integration problems across the analyses, while reaching the same level of accuracy on each set of problems[2]. Such a reduction in problem solving time suggests that the learner model CKM-HI in the experimental group enables students to reach a higher fluency in skills given the same amount of practice time, compared with the learner model CKM in the control group.

8.2.3 Posttest Scores with Time Constraints

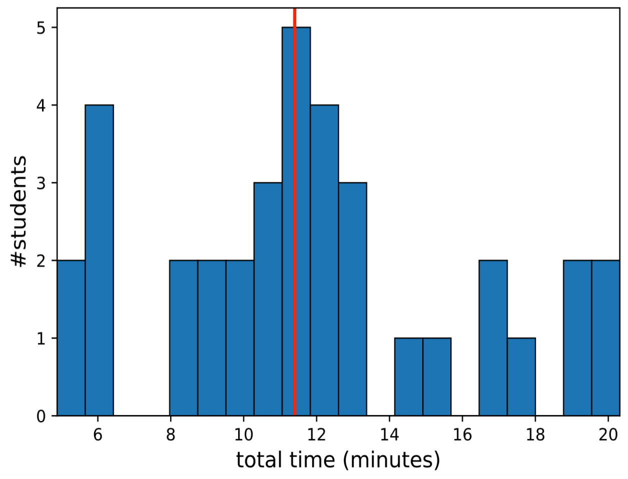
Inspired by the conclusions drawn from the above posttest scores and posttest time analyses, this subsection provides a further analysis attempting to jointly analyze posttest scores and time. To start with, I examined the posttest total time distributions in each condition, as shown in Figure 26. Specifically, a non-trivial number of students actually spent more than 15 minutes on the posttest in both conditions[3]. The choice of 15 minutes, in the study design, was based on an heuristic estimation. If we had given them different time limits, the students' scores on the posttest might have been different. Indeed, I examined the posttest scores with time constraints from 10 minutes to 15 minutes, treating answers as incorrect for problems submitted after the specific time constraint. I focused my analysis on the easy integration problems, since they showed the biggest contrast between posttest scores in the two conditions under time constraints among basic/*itgt-easy*/*itgt-hard*/all problem groups. Figure 27 displays the means and confidence intervals of posttest scores, given a range of strict time constraints on easy integration problems in each condition.

I conducted generalized linear mixed modeling which takes into account the variances introduced by problems and students for predicting posttest correctness per problem per student, given the continuous overall pretest score (fixed intercept), the binary condition indicator (fixed intercept), the problem indicator (random intercept) and the student indicator (random intercept). The results are reported in Table 28.

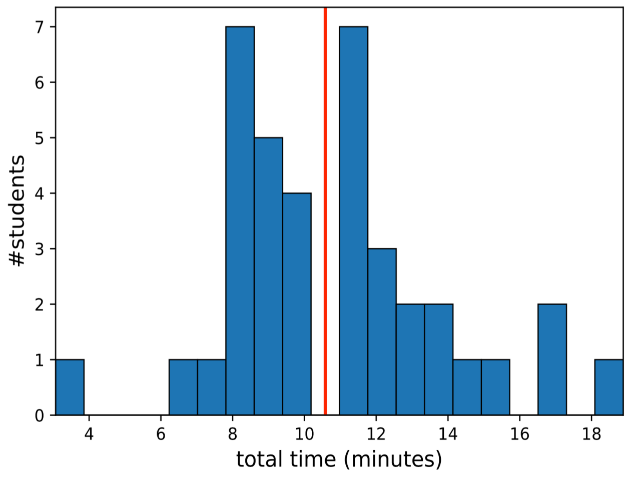
As shown in Table 28, the coefficients for the experimental condition are significant ($p<0.05$) with 10 and 11 minutes time constraints, and are consistently marginally significant

[2]In Section 8.2.1, it was found that students in both conditions had statistically the same proportion correct in basic/*itgt-easy*/*itgt-hard*/all posttest problems.

[3]Although we had deactivated the main practice component in the system 15 minutes before the class ended, some students insisted on finishing the posttest even though the class had finished, so we waited for these students.



(a) CT (N=36, M=12, SD=4, min=5, max=20)



(b) EP (N=38, M=11, SD=3, min=3, max=19)

Figure 26: Posttest total time (minutes) distributions for the control (CT) and experimental (EP) groups. The red lines indicate median values.

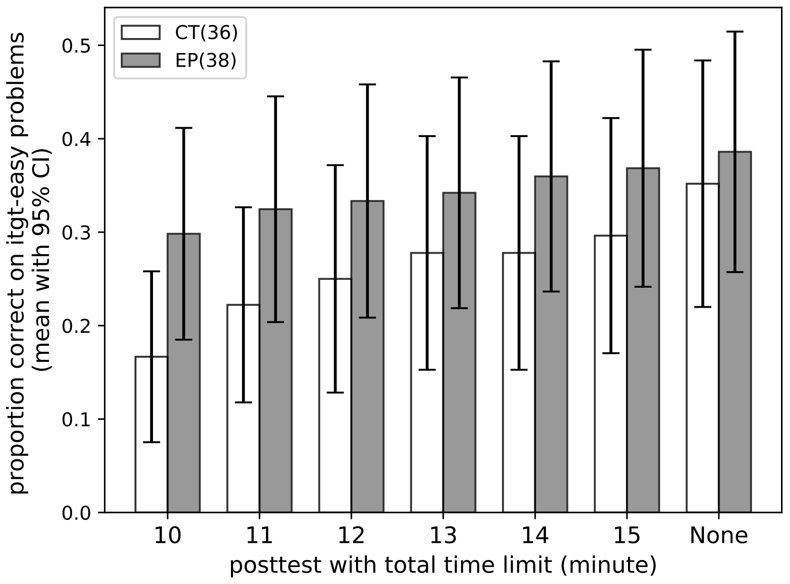


Figure 27: Comparison of posttest scores on easy integration problems with different time constraints between the control (CT) and experimental (EP) conditions.

Table 28: Generalized linear mixed models predicting posttest correctness per problem per student within easy integration problems (*itgt-easy*) with time constraints, based on the formula: $posttest\_correctness \sim condition + pretest\_score + (1|problem) + (1|student)$. Likelihood Ratio Test statistics (χ^2) and p-values comparing each model with its counterpart (which removes the targeted fixed effect) are reported. Residuals of all models are (approximately) normally distributed. (Sig. level ***:<.001, **:<.01, *:<.05, •:<.1.)

time	#obs.	fixed effect								random effect	
constr.		condition(EP)				pretest_score				prob.	stu.
		Est.	SE	$\chi^2(1)$	p	Est.	SE	$\chi^2(1)$	p	SD	SD
10 min	222	1.51	0.67	5.70	0.017*	4.26	1.83	6.43	0.011*	0.80	1.78
11 min	222	1.27	0.65	4.25	0.039*	5.42	1.88	10.52	0.0012**	0.59	1.80
12 min	222	1.29	0.76	3.11	0.078•	6.35	2.20	10.68	0.0011**	0.37	2.27
13 min	222	1.06	0.68	2.63	0.105	6.47	2.03	13.36	<0.001***	0.22	2.00
14 min	222	1.16	0.64	3.54	0.060•	6.49	1.94	14.84	<0.001***	0.15	1.85
15 min	222	1.14	0.65	3.38	0.066•	7.15	2.01	17.55	<0.001***	0.18	1.86

(p<0.11) with 12 to 15 minutes time constraints. I repeated the above analysis for the hard integration problems and basic problems, and the coefficients for the experimental condition were not significant, although they were all positive. Note that the non-significant coefficients on the hard integration problems were not due to positioning them in the end of quizzes, because the easy integration problems were evenly distributed in the second half of the posttest, being interwoven with hard integration problems. This set of results shows that given the same limited total time on task after practicing, students in the experimental condition have a significantly higher proportion correct on the easy integration problems. The effect is larger when the time constraint is more strict.

8.2.4 The Composition Effect (CE)

In this section, I revisit the issue of the composition effect, thanks to new data from this study, and also provide a novel way to evaluate learning effectiveness, by examining the strength of the composition effect in the posttest.

8.2.4.1 Existence of CEs on the Pretest and Practice Problems

In Chapter 3, the DFA classroom study (Classroom Study 1) with students from an introductory Python course in Spring 2017 provided evidence of the existence of composition effects and integration skills. Here, the intervention study with a program comprehension ITS was run in the same introductory Python course with a new population of students and different teachers than in Fall 2017. This new set of data enabled me to again investigate the composition effect: Do the composition effects of the identified integration skills still exist among new students? Also, do the two hypothesized new integration skills identified by the integration difficulty factors indeed show composition effects? To answer these questions, I gathered pretest and practice problems from both groups, with a total of 74 students. Following the methods used in Chapter 3 to make the analysis as focused as possible, I picked practice problems that focused on only one integration skill without basic skills, not as components of the integration skill; yet for some cases, where an integration skill always occurred with basic skills but not as components of it, I picked corresponding integration problems with a minimum of extra basic skills. In the pretest, there were only basic problems, so they naturally were used for estimating the basic skill levels (together with other basic problems from the practice stage). I removed isolated integration problems from this analysis, because the definition of integration skills corresponds to the full version, and isolated integration problems, as observed from the data, are mostly much easier than full integration problems, although with some exceptions (e.g., on *for*&#= they are harder than the full counterparts). Table 29 shows the results with the column of *pM-pI* denoting the existence and strength of the composition effect. Note that *for* corresponds to a version of *for* loop with the start and end values explicitly specified, e.g., range(1, 3), range(2, k); for_{v2}[4] It corresponds to a much

[4]The subscript *v2* in *for* denotes a meaning different from that in Chapter 3.

harder version (observed from the data) with only the end value explicitly specified and it only occurs in nested loop integration problems, e.g., range(3), range(k). Also, the joint probability pM in the case of $for\&for_{v2}$ equates to the smaller probability of being correct between the two *for* instances, based on the assumption that different instantiations of the same basic skill should be considered as highly dependent (same as Chapter 3).

Table 29: Examining the composition effect on pretest and practice problems from both conditions (74 students). For each integration skill under investigation, pK_x denotes the proportion correct among students on the corresponding basic problem; pM denotes the estimated proportion correct of the sequential problem computed by multiplying pK_xs (or picking the smallest in the cases of $for\&for_{v2}$); pI denotes the proportion correct on the integration problem. *pM-pI* denotes the strength of a composition effect.

itgt. skill	K_1	K_2	K_3	pK_1	pK_2	pK_3	pM	pI	pM-pI
a[i]&=	a[i]	=	#=	0.81	1.00	0.93	0.75	0.28	0.47
$for\&for_{v2}$	for	for_{v2}		0.70	0.51		0.51	0.05	0.46
for&a[i]	for	a[i]		0.70	0.81		0.57	0.13	0.43
for&#=	for	#=		0.70	0.93		0.65	0.49	0.16
for&if	for	if	a[i]	0.70	0.80	0.81	0.45	0.35	0.10
avg	–	–	–	–	–	–	0.59	0.26	0.32

Table 29 provides consistent results as the ones obtained from Table 4 in Chapter 3[5]. In particular, the new hypothesized integration skill *a[i]&=* identified by the *state update* integration difficulty factor and generalized from *for&a[i]* discovered in Chapter 3 has a very high *pM-pI* (0.47), suggesting a strong composition effect. The newly hypothesized integration skill *for&#=* extended from *for&x=x+i* also has a high *pM-pI* (0.16) suggesting

[5]Note that the Wilcoxon signed-rank test or paired t-tests used for each hypothesized integration skill in Table 4 in Chapter 3 is not readily applicable here, because different problems used for the calculation had different pools of students, due to adaptive practice (except for the pretest, which all students attempted), and some problems used for calculating pI had only a small number of students. More complex statistical analyses could be considered (e.g., mixed effect modeling), and I leave this for future research. Also, I used *pS-pI* rather than *pM-pI* in Table 4 to compare with the composition effect strength *pM-pI* here, because *pS-pI* was a more accurate estimation of the composition effect in Table 4 for Classroom Study 2.

the existence of the composition effect, consistent with the significant composition effect for *for*&+= in Table 4. Meanwhile, although *for*&*a[i]* received mixed results in terms of the significance of the composition effect in Table 4, the data here clearly shows a very high *pM-pI* (0.43), suggesting a strong composition effect. Regarding *for*&for_{v2}, the strength of the composition effect is consistent and even larger than that in Table 4[6] (0.46 vs. 0.38). Regarding *for*&*if*, the strength of the composition effect is similar to the one which is significant in Table 4 (0.10 vs. 0.18).

To get an idea of the overall existence of the composition effect in the targeted topics (*for* and *lists*), I also ran nonparametric tests on the five values of *pM-pI*. A one-sided exact Wilcoxon signed-rank test and a one-sided sign test (binomial test) both show that the overall composition effect is significant (p=0.03) with a large effect size (1). Admittedly, the sample size here is quite small, yet the tests provides some additional information to interpret the results.

Overall, this section of analyses provides additional support for the existence of the composition effects and integration skills for the overlapping ones investigated in Chapter 3, and provides evidence for the generalizability of composition effects to new students and new contexts (problems). It also shows a successful example of using the *state update* integration difficulty factor to identify new integration skills (e.g., *a[i]*&=, *for*&#=), providing additional support for the generalizability of the difficulty factor.

8.2.4.2 Comparison of CEs on Posttest Problems

The T3 tutor is designed for teaching both basic and integration skills in program comprehension under both conditions. As shown in Table 23, students from both conditions have reached a high level of basic skills, as suggested by the high average score on the posttest basic problems ($\geq$0.88) in each condition, and they are not statistically different. Thus, a lower composition effect (*pM-pI*) should indicate a higher level of integration skills, and overall, better learning by students. This serves as a new method to compare the learning effectiveness of the two conditions grounded on different learner models.

[6]Although for_{v2} has a meaning here different from that in Table 4, in both cases, *for*&for_{v2} denotes a nested loop where the outer loop iteration variable appears as the argument of the range function of the inner loop.

Similar to the method used above (Section 8.2.4.1), the main idea is to compare performance differences between matching non-integration and integration problems. Here, posttest basic problems are used for getting individual basic skill knowledge estimates, and ultimately getting the estimated proportion correct on the sequential problem (pM). Posttest integration problems are used as the units to examine composition effects. Different from Section 8.2.4.1 where each integration problem only involves one integration skill, a posttest integration problem here can involve multiple integration skills (and also basic skills that are not components of the integration skills), so the composition effect here could be an overall effect from multiple integration skills in complex integration contexts.

Table 30: Comparing composition effects on posttest problems between the control group (CT) with 36 students and the experimental group (EP) with 38 students. For each problem, pK_x denotes the proportion correct on the corresponding basic problem; pM denotes the estimated proportion correct on the sequential problem by multiplying pK_xs; pI denotes the proportion correct on the integration problem. *pM-pI* denotes the strength of the composition effect.

id	itgt. skill(s)	K1,K2,	pK1 pK2 (pK3 pK4)		pM		pI		pM-pI	
		(K3,K4)	CT	EP	CT	EP	CT	EP	CT	EP
8	for&a[i],a[i]&=	for,a[i],=	.94 .92 1	.95 .84 1	.87	.80	.31	.39	.56	.40
9	for&for,for&a[i] a[i]&=,for&#=	for,a[i],#=,=	.94 .92 .97 1	.95 .84 .97 1	.84	.78	.33	.34	.51	.43
10	for&if	for,if,a[i]	.94 .89 .92	.95 .92.84	.77	.73	.39	.42	.38	.31
11	for&for	for,a[i]	.94 .92	.95 .84	.87	.80	.31	.26	.56	.53
12	a[i]&=,a[i]&a[i]	for,a[i],//,=	.94 .92 .94 1	.95 .84 .82 1	.82	.65	.36	.34	.46	.31

Table 30 reports the analysis of the composition effect. Figure 28 gives a graphical summary. As shown by the table and the figure, the experimental group consistently has smaller values of *pM-pI*, compared to the control group, suggesting smaller composition effects over the set of posttest integration problems. Further, a one-sided exact Wilcoxon

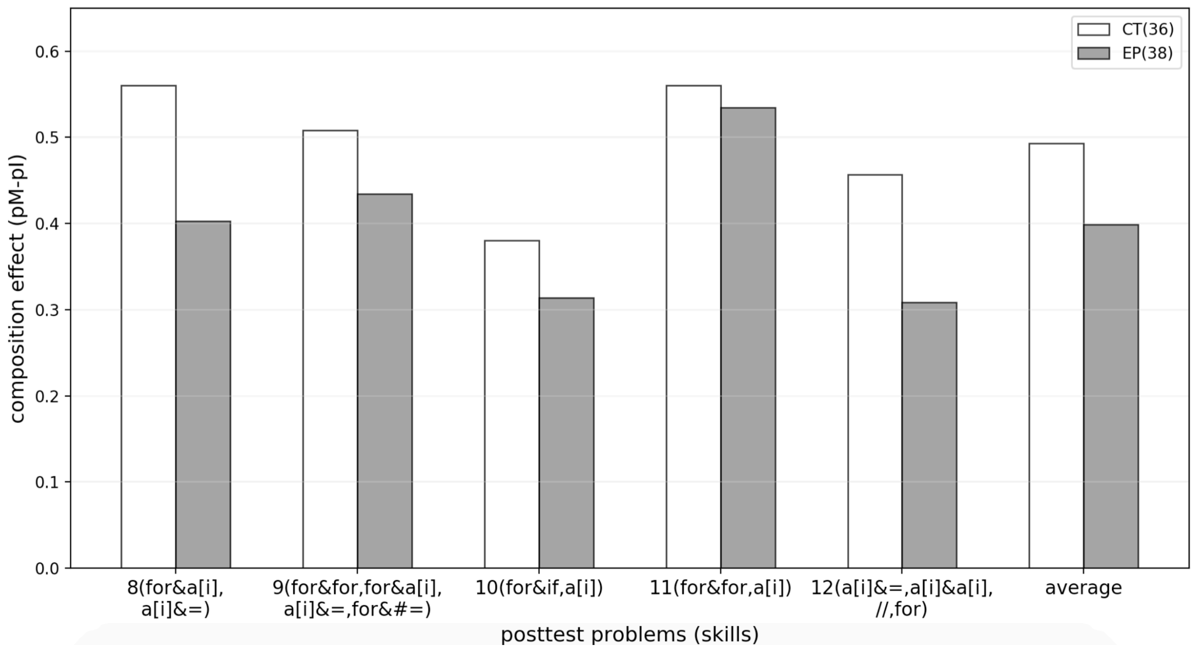


Figure 28: Comparison of composition effects (*pM-pI*) on posttest problems, between the control (CT) and experimental (EP) groups.

signed-rank test and a one-sided sign test (binomial test) both show that the reduction of composition effects from control group to experimental group is significant (p=0.03) with a large effect size (1). The small sample size here requires caution to interpret the results. Another caution when interpreting these results is that the initial composition effects before practicing in the system is unknown, since the pretest doesn't include integration problems (due to practical constraints, as explained in Section 8.1.6). However, student performance on basic problems might positively correlate with that on integration problems. According to Table 22, students' pretest scores significantly positively correlate with the performance on easy/hard/all integration problems (controlled for other variables). As analyzed in Section 8.2.1, the control condition had marginally significantly higher pretest scores (p=0.096), so it is possible that students in the control condition also had (slightly) higher integration skill levels, which could lead to similar initial composition effects as the experimental group. However, a rigorous, thorough investigation is needed for drawing a reliable conclusion. So far, the results provide promising initial evidence that the experimental group reduces the composition effect to a larger extent than the control group.

8.3 DISCUSSION AND CONCLUSIONS

Combining the above analyses, a clear advantage of the experimental group intervention with the integration-level learner model CKM-HI over the control group intervention with the basic-level learner model CKM, has been demonstrated: without strict total time constraints on the posttest, students in the CKM-HI group were significantly faster on posttest difficult integration problems, reaching the same level of accuracy; with a range of strict total time constraints on posttest, students in the CKM-HI group were significantly more accurate on posttest easy integration problems. In addition, an analysis of composition effects on posttest problems shows that the composition effects are much weaker in students in the CKM-HI group compared to those in the CKM group.

There are several aspects that could have hindered revealing a larger effect for the CKM-HI experimental condition. Firstly, the trace table working space was not provided in the posttest, and during this study we didn't find students using a paper or editors on the computer to help them make drafts to solve the posttest problems. Students had to mentally trace the programs and give the final answers. As a result, even if students in the experimental condition had conceptually understood how integration works and could solve integration problems using trace tables in the practice stage, they might still lack enough fluency to be able to solve problems mentally without the assistance of the trace tables. This might account for them having a similar level of accuracy as the control group. To address this, a future study should include matching conditions where either the trace tables are removed from the practice problems so that students would be trained to trace programs mentally; or an optional trace table working space should be included in the posttest problems.

Secondly, the total practice time for both conditions was still quite limited (with a mean of 41 minutes in the control condition and a mean of 44 minutes in the experimental condition). With a longer span study, it's highly likely that the experimental intervention (with CKM-HI learner model) could have a larger impact on student learning compared to the control condition, e.g., enabling higher accuracy on the hard integration problems and shortening the time to correctly solve easy integration problems, based on the trend found in the current study. I leave a longer span study for further investigation to future research.

Finally, the lack of integration problems in the pretest (due to study time constraints) limited conducting a more accurate and deeper learning-gain analysis, and failed to inform the CKM-HI learner model the initial levels of integration skills of students. A future study should include integration problems in the pretest (and also lengthen the total study time), so that the CKM-HI learner model could utilize its advantage to help students with high basic skill levels to focus on practicing their weak(est) integration skills.

9.0 CONCLUSIONS, DISCUSSION AND FUTURE WORK

In this chapter, I first summarize the conclusions from all studies and analyses, then discuss the implications, limitations and future work. Finally, I state the contributions of my dissertation.

9.1 CONCLUSIONS

In this dissertation, I have proposed a learner modeling approach for integration skills and have systematically demonstrated its grounding and value through a set of empirical classroom studies and data-driven analytical studies.

To provide solid grounds and generalizable insights for integration skill modeling, I conducted a classroom study following a Difficulty Factors Assessment (DFA) approach. The data consistently showed that students have significant difficulties in integrating the basic skills found in common basic programming patterns, even if they have already learned these basic skills by following a standard curriculum. A drill-down integration error analysis and correlation analysis investigating these integration skills further revealed that students lacked these skills due to wrong/missing conceptual understanding about how components work together or due to procedural cognitive load during integration. Students exhibited individual differences in integration skills due to both integration difficulty factors and topics (main programming constructs). (Chapter 3)

I have developed a general algorithm called Automated Potential Integration Skill identification (APIS), based on conceptual integration difficulty factors and the aspects for describing (potential) integration skills. In the context of my dissertation, I obtained these inputs through the aforementioned DFA study. I also introduced a new type of knowledge graph, an integration graph, which shows how component skills progressively integrate and form integration skills. Based on this skill model, I constructed an integration-level learner model, CKM-HI, using a Bayesian network, which incorporates integration skills in a hierarchical structure. It learns parameters from student data, and gives dynamic, individualized knowledge estimates when deployed in a tutoring system. (Chapter 4)

Being aware of the challenges in evaluating such multi-skill practice learner models, I proposed a multifaceted evaluation framework for learner models (Chapter 5). Under this framework, I conducted data-driven evaluations based on a real-world Python program comprehension dataset, and demonstrated that my proposed learner model, CKM-HI, is superior to two popular multiple-skill knowledge tracing models that do not incorporate integration skills, WKT and CKM. I considered a range of aspects: predictive performance, parameter plausibility, and expected instructional effectiveness. Further, I conducted a simulated study and discovered that larger improvements in predictive performance can take place if parameter constraints are imposed on the noise parameters. (Chapter 7)

As the last important aspect in my evaluation framework, I designed and implemented a Python program comprehension ITS (T3), to evaluate the real-world impact of CKM-HI on student learning. T3 has assistance-enhanced trace tables to address the integration difficulties I discovered in my earlier composition effect studies (Chapter 6). Although having limited time on practice during the classroom study, students in the CKM-HI intervention group were significantly faster on difficult posttest integration problems than the CKM group, while reaching the same level of accuracy, without strict time constraints on the posttest. When there was a range of strict time constraints on the posttest, students in the CKM-HI group were significantly more accurate on easy posttest integration problems than the CKM group. Also, composition effects on posttest problems were weaker in students in the CKM-HI group than with those in the CKM group. Additionally, this study also provides more evidence for the existence and generalizability of composition effects and integration skills to a new population of students and new problems. This classroom study evaluation, together with the data-driven evaluation give a comprehensive picture of the value of learner modeling for integration skills. (Chapter 8)

9.2 DISCUSSION, LIMITATIONS AND FUTURE WORK

The implications, limitations and future work of my dissertation are outlined as follows.

Firstly, I have only investigated learner modeling for integration skills in one Python programming dataset on selected basic topics. To reliably conclude the effectiveness of the idea and approach for integration-level learner modeling, broader contexts in terms of topics, programming languages, and domains should be examined.

To consider broader contexts, my approach of building a Bayesian network based on a skill model with integration skills is readily applicable to other contexts, yet building the skill model in new contexts remains a challenge. The general automated potential integration skill identification algorithm (APIS) I created could be considered to construct the item-to-skill mapping. APIS is based on conceptual integration difficulty factors and aspects for describing integration skills. These two inputs can be obtained by directly consulting domain experts, or by conducting DFA studies. In less understood contexts, DFA studies might be needed to help reduce expert bias or expert blind spots (as compared with directly consulting domain experts), while maintaining the interpretability of the resulting skill model, and enabling generalizable insights (compared with machine learning approaches). However, a considerable amount of time is required to design, deploy a DFA study and analyze its data.

On the other hand, machine learning approaches could be explored, when there is a sufficient amount of student performance data available. They eliminate the time required by DFA studies, at the potential cost of losing interpretability. Specifically, data from multiple MOOCs or courses with a large number and high variety of problems and students could be good inputs for developing such approaches. Also, parameter constraint optimization might be necessary to address the challenge of having a flexible parameter space in the context of multi-skill practice problems. My preliminary approach [Huang et al., 2016] demonstrated an initial step in this direction. More advanced data-driven Bayesian network structure learning or causal discovery methods could be explored here, inspired by successful methods for prerequisite structure discovery [Desmarais et al., 2006, Chen et al., 2016]. To increase the interpretability of such machine learning approaches, direct expert knowledge on potential integration difficulty factors and aspects for describing integration skills could be applied (as in the APIS algorithm) to narrow down the search space, or could be utilized to conduct a post-hoc analysis on discovered skill models. Concerning the complexity and the potentially high number of integration skills in domains like programming, direct expert knowledge may potentially be considered to be approximated by crowdsourced knowledge from experienced programmers.

Another limitation is that my dissertation hasn't thoroughly investigated the generalization of integration skills, integration difficulty factors, nor the proper representation abstraction level of integration skills. Due to an insufficient number of problems and the fixed order imposed in my DFA study, I didn't examine the correlation between the levels of integration skills of related topics under the same integration difficulty factor (e.g., *for&for* with *while&while*) or different integration difficulty factors (e.g., *for&x=x+i* with *while&x=x+i*). Thus it is not clear 1) whether the level of an integration skill in one topic can be generalizable to its counterpart in another topic[1], and further, 2) whether the acquisition (learning) of one integration skill can be transferred to the acquisition (learning) of other integration skills. An additional limitation is that I have only investigated three topics in Python program comprehension, and used a subset of problems across the three topics as an approximation of new problems in Section 3.2.3. It is not clear whether the identified integration difficulty factors and aspects for describing integration skills are readily applicable to new topics.

The limited exploration on the generalization of integration skills and integration difficulty factors also results in insufficient grounds to represent integration skills on the integration difficulty factor level. Since there is no evidence supporting that the integration difficulty or learning from an integration skill in one topic can be generalized to its counterpart in another topic under the same integration difficulty factor, representing integration skills on integration difficulty factor level risks losing a more varied practice for students, which is necessary for them to reach mastery in a set of topics. However, the answers to both the generalization and representation questions may depend on the instruction, e.g., whether the teacher has introduced an integration skill in a more abstract way or not, as

[1]On a group level, integration skills on a set of topics can be generalized to other integration skills in the same set of topics under the same integration difficulty factor, as shown by Table 9.

well as the level of students. For example, higher level students may find it easier to transfer from one integration skill to another with similar constructs. Thus, representing integration skills on a more abstract level may be sufficient. Data-driven evaluations (e.g., applying the framework in Section 5) comparing a set of learner models varying the integration skill representation levels could shed light on selecting the proper representation level; DFA studies with more problems and topics as well as intervention studies teaching integration skills could help provide solid answers to these two questions.

My proposed integration-level learner model incorporates integration skills in a hierarchical structure, based on a Bayesian network, but it is not clear whether and how much the benefit over baseline models without integration skills comes from incorporating integration skills, or from the hierarchical structure, or from both. Although the hierarchical structure allows knowledge inference to propagate across nodes, which can improve the accuracy of inference, it increases the overall complexity of the model and increases the difficulty of introducing learning dynamics. On the other hand, flat structures assuming independence among skills allow for easy incorporation of learning dynamics, but sacrifice knowledge inference propagation across skills. Such flat-structured integration-level learner models should be added into the comparison as an immediate next step of my dissertation.

One interesting direction is to extend the Bayesian network of the integration-level learner model into a decision network (equivalently, an inference diagram). A decision network would provide a mechanism to make adaptive tutorial decisions based on the network's beliefs of a student's anticipated states (e.g., knowledge, focus of attention) after a tutorial action, and multiple competing objectives [Howard, 1983, Murray, 2005].

My dissertation also provides some implications for teaching integration skills. The existence and nature of conceptual errors that occurred during integration but not sequential applications of basic skills suggest that additional, new skills arise when basic skills interweave together. Even though students followed the standard curriculum and knew how basic skills work, a significant number of the students still failed in integration. These results suggest that the acquisition of integration skills may not be achieved by merely teaching basic skills. Including explicit conceptual explanations and the practice of integration skills could be beneficial for curriculum/course design and ITSs. Prior work has provided strong evidence of the effectiveness of teaching integration skills in the algebra domain [Heffernan and Koedinger, 1997], and it would be interesting to conduct further studies to see whether the same effect holds true in the programming domain following initial attempts [Proulx, 2000, Muller et al., 2007, de Raadt, 2008, Lane and VanLehn, 2005].

Another future direction would be to improve the instructional methods in the program comprehension ITS (T3) and explore the effect of different instructional methods on learning integration skills. Program tracing, although helpful to program comprehension, doesn't equate to program comprehension [Lister et al., 2009]. Many participants in my studies have commented that it would be even more beneficial to let them know or reflect on what the goal of a program is. This goal abstraction process would encourage a deeper understanding of integration skills, enabling a more accurate and faster application of them as well as transferring them to new contexts. In addition, some participants suggested that they would like to see concrete, straightforward examples given while T3 is explaining the conceptual parts of skills, indicating the potential use of program examples for teaching integration skills. Moreover, different kinds of integration skills may require different instructional methods. To investigate this issue, intervention studies could be conducted to compare the reduction of the composition effect (or the increase of integration skill levels) under the same instructional methods for different (kinds of) integration skills, and under different instructional methods for the same (kind of) integration skill, controlling for other factors. My current Classroom Study 2 (Section 8.2.4.2) is insufficient to investigate this issue, since the posttest problem composition effects have been affected by other factors (e.g., practice opportunities, whether the trace table space was provided or not, and the number of integration skills).

A fruitful application of the integration-level skill model, particularly the integration graph, would be to guide the evaluation and construction of learning content. As shown in Chapter 3, students following a standard programming course curriculum still exhibit significant integration difficulties, even after lectures and assignments about each topic, indicating the need for improving learning materials. An integration-level skill model sheds light on the design of courses and ITSs from several aspects: 1) it could be beneficial or even necessary to explicitly teach the conceptual parts of integration skills, 2) sufficient diverse exercises following the skill model might be needed for reaching fluency in basic and integration skills within each topic and across topics, and 3) a progressive exercise design starting from simple integration to more complex integration, mixing multiple integration skills or involving new (basic) skills following the integration graph might be needed for helping students to develop integration skills to a suitable difficulty level. Meanwhile, datasets collected from such courses or ITSs would also help to validate or improve integration-level skill models.

Another fruitful future direction would be to understand, improve and enrich the multi-faceted learner model evaluation framework, such as investigating how parameter plausibility relates to expected instructional effectiveness and whether it matters in the real world, addressing the problem of having insufficient "mastery" data when computing expected instructional effectiveness in a principled probabilistic way. Moreover, more dimensions or metrics could be considered to enhance this framework in other ways, such as the use of the Brier score [Pelánek, 2015], parameter consistency (used in my prior work) [Huang et al., 2015] and having a predictive similarity policy [Rollinson and Brunskill, 2015].

A natural extension to my dissertation would be to investigate learner modeling for program construction. Here, skill modeling is more challenging, since students can use different skills to write a program that achieves the same functionality. Also, it is not straightforward how to evaluate the correctness of applying a skill, and it is not clear what an assessment unit (or a step) is in the forward-backward process of developing a program. However, since program comprehension and construction share similar conceptual knowledge, the integration difficulties/skills identified here and the learner model constructed here might be helpful to program construction. If learner models for program construction are built, they could be combined with automated hint generation [Rivers and Koedinger, 2017], and potentially enhance the overall effectiveness of programming ITSs.

9.3 CONTRIBUTIONS

The contributions of my dissertation work are summarized as follows.

Firstly, my dissertation brings a new perspective to the construction of learner models—skill integration—which raises the cognitive level that ITSs can diagnose and act on. I created a new type of knowledge graph that I call an integration graph, which shows how basic component skills progressively integrate and form new skills that are essential to describing domain expertise. I also created a new learner model, CKM-HI, that incorporates integration skills as explicit nodes in a hierarchical structure in a Bayesian network. Although numerous cognitive science research and educational practices have repeatedly claimed the existence of integration skills in expertise and the importance of practicing integration skills [Chase and Simon, 1973, De Groot, 1978, Egan and Schwartz, 1979, Larkin et al., 1980, Soloway and Ehrlich, 1984, Koedinger and Anderson, 1990], very little effort has been made to incorporate the skill integration viewpoint into learner modeling in modern ITSs[2]. If integration-level learner models are built and are used to inform the content design and adaptation decision making in ITSs, ITSs could have a greater chance of preparing students to be robust learners who have a deeper, conceptual understanding or higher fluency in solving complex tasks in varied contexts. My classroom study described in Chapter 8 provides evidence for the positive impact of incorporating integration skills into an ITS on learning.

Secondly, my dissertation research provides a multifaceted evaluation framework and showcases a comprehensive evaluation example which contributes to several communities. For the educational data mining (EDM) community, my framework provides metrics for parameter plausibility and expected instructional effectiveness. It also alerts the community to the limitations of predictive performance and thus demonstrates the value of looking at learner models from new aspects: it's important for learner models to be predictive, but it's at least equally important for them to be plausible, interpretable, and useful for instructional decisions, in order to reach the ultimate goal of improving student learning. In addition, a multifaceted perspective (such as the joint examination of parameter plausibility and predictive performance in Chapter 7) also helps to explain why a certain model is not predicting as it is expected to, and provide insights to improve the model or raise caution when applying the model. Moreover, I have shown how a real-world classroom study can reveal a learner model's actual effect on student learning, the ultimate goal of ITSs, which adds to the growing literature by advocates for close-the-loop studies in EDM. For the ITS,

[2]Prior work which has incorporated the skill integration viewpoint into learner modeling used heuristic approaches [Brusilovsky, 1992, Weber, 1996a, Kumar, 2006, Mathews, 2006, Chrysafiadi and Virvou, 2013], and my work is the first to apply a formal probabilistic approach.

AIED and UMAP community, my framework provides multiple data-driven metrics for the separate evaluation of the learner modeling component, as a supplement to a holistic system-level evaluation.

Thirdly, my dissertation work is also the first work to apply an empirical cognitive task analysis method (specifically, a DFA) to computer science education research to systematically study integration skills in program comprehension. One of the biggest challenges in programming education is the lack of unified, clear, fine-grained skill definitions, such as the those defined in the math domain. The DFA method from cognitive science research could be a helpful way to develop interpretable, generalizable techniques, effective instructional policies and also better learning materials. It provides a way to unravel the underlying skills and difficulty factors that are contained in integration skills within program comprehension. Hopefully, my dissertation work can inspire more efforts to applying empirical methods to collect and analyze student data, in order to understand the skills and processes underlying programming tasks.

Fourthly, my dissertation work is also among the first steps to bringing recent ITSs infrastructure and techniques into programming education. I built a program comprehension tutor (T3) with a common ITS infrastructure [Ritter et al., 2007, Vanlehn et al., 2005], based on the learning-by-doing philosophy. It contains both knowledge tracing (achieved by a probabilistic learner model) and model tracing (achieved by developing an underlying solution graph), with both inner loop (e.g., hints) and outer loop (e.g., problem selection) adaptation techniques that are driven by a learner model. The learning gain observed in both conditions and the highly positive feedback from students and teachers in Classroom Study 2 (Chapter 8) have indicated the promise of such a programming ITS. On top of this, such a benefit is not at the price of a high engineering effort, thanks to existing, mature ITSs infrastructures CTAT[3] [Aleven et al., 2006, Koedinger et al., 2003] and automated methods to analyze programs. Hopefully, my dissertation can spark more endeavors in this direction.

Last but not least, my dissertation work contributes an example to EDM and ITS research of taking an interdisciplinary approach (combining analytic methods from computer science and empirical methods from cognitive science) to tackle a topic which is ultimately

[3]https://github.com/CMUCTAT/CTAT/wiki

aimed at the practical value of improving student learning. A cognitive science approach (a DFA) has been used to provide the foundation for building a machine learning model and a real-world ITS; the machine learning model and real-world ITS have been constructed which could hopefully enable scalable ways to expand research efforts in this field and beyond. As shown by my dissertation, cognitive science methods can be of benefit when machine learning methods for skill discovery/refinement fall short. Machine learning methods for skill discovery/refinement [Barnes, 2005, Winters et al., 2005, Cen et al., 2006, Desmarais, 2012, Lan et al., 2014, González-Brenes, 2015] have only shown effectiveness on data where each assessment unit maps to one to three skills, yet a program comprehension problem or a complex integration problem may involve a larger number of skills mapping to each assessment unit, raising significant challenges for such methods. On the other hand, an empirical method from cognitive science research, the DFA, which utilizes experts' hypotheses (knowledge), provides a systematic way to generate and analyze data, enabling a much more reliable analysis to draw insights as the foundation for building machine learning models.

APPENDIX

INTEGRATION ERROR ANALYSIS

Table 31: The integration error analysis for the *nested loop integration type*, among students who succeeded in the sequential problem but failed in the matching integration problem. (Sig. level ***:<.001, **:<.01, *:<.05, $^{\bullet}$:<.1; effect size $^{++}$:>.3, $^{+}$:>.1).

Hypothesized integration skills (f:for, w:while)	w&w$_{v2}$	f&f$_{v2}$	w&w	f&f	w&f
Topic (F:For, W:While, L:List)	W	F	W	F	W
Problem set id	8	3	6	1	11
Composition effect (pS-pI) sig. level and effect size	*** ++	*** ++	*** ++	*** +	• +
#stu. with seq. prob. correct but itgt. prob. wrong	24	26	20	26	8
Nested loop conceptual errors	91%	58%	65%	58%	76%
When outer loop iter. var. is used in inner loop condition,					
missed the middle repeated numbers	33%	31%			
used outer loop bound as the bound for all inner loops	25%	4%			
Treated inner(outer) loop as outer(inner) loop		4%		27%	13%
Only did one outer iter.(with all inner iter.) and it was correct	33%	19%	65%	31%	63%
Nonconceptual errors (conceptually knew how itgt. works)	0%	4%	0%	12%	13%
Made errors on basic skills due to cognitive load					
Did first two outer iter. correctly, added 1 more outer iter.				4%	13%
Added one more inner iter. in the last outer iter.		4%			
Did inertial thinking slip by printing outer not inner iter. var.				8%	
State update conceptual errors	0%	0%	0%	0%	0%
Uncategorized errors	9%	38%	35%	30%	11%
Made errors on basic skills				4%	
Empty		15%		4%	
Other errors	9%	23%	35%	22%	11%

Table 32: The integration error analysis for the *state update integration type*, among students who succeeded in the sequential problem but failed in the matching integration problem. (Sig. level ***:<.001, **:<.01, *:<.05, $^{\bullet}$:<.1; effect size $^{++}$:>.3, $^{+}$:>.1).

Hypothesized integration skills (f:for, w:while)	f&w$_{v3}$, a[i]$_{v4}$&a[i]$_{v5}$	f&a[i]$_{v2}$	f&x=x+4	f&x=x+i	f&a$_1$=a$_1$+a$_2$	f&a[i]$_{v3}$
Topic (F:For, W:While, L:List)	L	L	F	F	L	L
Problem set id	16	17	4	2	12	15
Composition effect (pS-pI) sig. level and effect size	*** ++	*** +	*** +	* +	$\bullet$ +	
#integration errors	22	16	16	11	8	11
State update conceptual errors	23%	44%	25%	64%	43%	45%
Failed to update both list elem. in a swap pattern	23%					
Used initial list (which should be updated) in all iter.		44%			43%	45%
Used initial value of the sum variable in all iter.			25%	64%		
Nested loop conceptual errors	23%	0%	0%	0%	0%	0%
Only did 1st outer iter. and it was correct	23%	0%	0%	0%	0%	
Nonconceptual errors(conceptually knew how itgt. works)	0%	0%	13%	9%	14%	0%
Made errors on basic skills due to cognitive load						
Did first two iter. correctly, but added 1 more iter.				9%	14%	
Did inertial thinking slip by doing x=x+i after 1st iter.			13%			
Uncategorized errors	54%	56%	62%	27%	43%	55%
Only did 1st iteration and it was correct	0%	25%	50%	9%	14%	18%
Made errors on basic skills					14%	
Empty	36%	6%	6%		15%	9%
Other errors	18%	25%	6%	18%		28%

Table 33: The integration error analysis for the *nonconceptual integration type*, among students who succeeded in the sequential problem but failed in the matching integration problem. (Sig. level **:<.01, *:<.05, $^{\bullet}$:<.1; effect size $^{++}$:>.3, $^{+}$:>.1).

Hypothesized integration skills (f:for, w:while)	f&x=x+i	w&x=4+i	f&if	w&x=x+i	w&x=x+4	f&x=4+i
Topic (F:For, W:While, L:List)	L	W	L	W	W	F
Problem set id	14	10	13	7	9	5
Composition effect (pS-pI) sig. level and effect size	$\overset{**}{+}$	$\overset{**}{+}$	$\overset{*}{+}$	$\overset{*}{+}$	$+$	
#stu. with seq. prob. correct but itgt. prob. wrong	12	14	9	10	10	11
Nonconceptual errors (conceptually knew how itgt. works)	73%	79%	44%	43%	40%	54%
Made errors on basic skills due to cognitive load						
Used iter. var. i not list elem. a[i] to do sum in some iter.	37%					
Used the wrong list elem. index (e.g., by starting with 1)	18%		33%			
Did one more or one fewer iteration	9%		11%		40%	9%
Forgot the add the initial value of the sum var.	9%					
Used initial value of the iteration variable in all iter.				43%		
Slipped due to inertial thinking						
Did x=4+i in 1st iter. but did x=4+x or x=x+i afterwards		79%				45%
State update conceptual errors	9%	0%	22%	43%	20%	0%
Used initial value of the sum variable in all iter.	9%			43%	20%	
Didn't update the max variable or updated it using iter. var.			22%			
Nested loop conceptual errors	0%	0%	0%	0%	0%	0%
Uncategorized errors	18%	21%	34%	14%	40%	46%
Only did 1st iteration and it was correct	9%	7%	11%	14%	20%	9%
Made errors on basic skills			11%			
Empty		7%				9%
Other errors	9%	7%	12%		20%	28%

BIBLIOGRAPHY

[Adelson, 1981] Adelson, B. (1981). Problem solving and the development of abstract categories in programming languages. *Memory & cognition*, 9(4):422–433.

[Aleven et al., 2006] Aleven, V., McLaren, B. M., Sewall, J., and Koedinger, K. R. (2006). The cognitive tutor authoring tools (ctat): preliminary evaluation of efficiency gains. In *International Conference on Intelligent Tutoring Systems*, pages 61–70. Springer.

[Ambrose et al., 2010] Ambrose, S. A., Bridges, M. W., DiPietro, M., Lovett, M. C., and Norman, M. K. (2010). *How learning works: Seven research-based principles for smart teaching*. John Wiley & Sons.

[Anderson et al., 2004] Anderson, J. R., Bothell, D., Byrne, M. D., Douglass, S., Lebiere, C., and Qin, Y. (2004). An integrated theory of the mind. *Psychological review*, 111(4):1036.

[Anderson et al., 1995] Anderson, J. R., Corbett, A. T., Koedinger, K. R., and Pelletier, R. (1995). Cognitive tutors: Lessons learned. *The journal of the learning sciences*, 4(2):167–207.

[Baker et al., 2008] Baker, R., Corbett, A., and Aleven, V. (2008). More accurate student modeling through contextual estimation of slip and guess probabilities in bayesian knowledge tracing. In Woolf, B., Aïmeur, E., Nkambou, R., and Lajoie, S., editors, *ITS*, volume 5091 of *Lecture Notes in Computer Science*, pages 406–415. Springer.

[Baker et al., 2007] Baker, R. S., Corbett, A. T., and Koedinger, K. R. (2007). The difficulty factors approach to the design of lessons in intelligent tutor curricula. *International Journal of Artificial Intelligence in Education*, 17(4):341–369.

[Baker et al., 2011] Baker, R. S., Goldstein, A. B., and Heffernan, N. T. (2011). Detecting learning moment-by-moment. *International Journal of Artificial Intelligence in Education*, 21(1-2):5–25.

[Barnes, 2005] Barnes, T. (2005). The q-matrix method: Mining student response data for knowledge. In *American Association for Artificial Intelligence 2005 Educational Data Mining Workshop*, pages 1–8.

[Beck and Chang, 2007] Beck, J. and Chang, K.-m. (2007). Identifiability: A fundamental problem of student modeling. In Conati, C., McCoy, K., and Paliouras, G., editors, *User Modeling 2007*, volume 4511 of *Lecture Notes in Computer Science*, pages 137–146. Springer Berlin / Heidelberg.

[Berges and Hubwieser, 2015] Berges, M. and Hubwieser, P. (2015). Evaluation of source code with item response theory. In *Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education*, pages 51–56. ACM.

[Brusilovsky, 1992] Brusilovsky, P. (1992). Intelligent tutor, environment and manual for introductory programming. *Educational and Training Technology International*, 29(1):26–34.

[Brusilovsky et al., 2004a] Brusilovsky, P., Karagiannidis, C., and Sampson, D. (2004a). Layered evaluation of adaptive learning systems. *International Journal of Continuing Engineering Education and Life Long Learning*, 14(4-5):402–421.

[Brusilovsky and Millán, 2007] Brusilovsky, P. and Millán, E. (2007). User models for adaptive hypermedia and adaptive educational systems. In *The adaptive web*, pages 3–53. Springer-Verlag.

[Brusilovsky et al., 1996] Brusilovsky, P., Schwarz, E., and Weber, G. (1996). Elm-art: An intelligent tutoring system on world wide web. In *International conference on intelligent tutoring systems*, pages 261–269. Springer.

[Brusilovsky et al., 2004b] Brusilovsky, P., Sosnovsky, S., and Shcherbinina, O. (2004b). Quizguide: Increasing the educational value of individualized self-assessment quizzes with adaptive navigation support. In *E-Learn: World Conference on E-Learning in Corporate, Government, Healthcare, and Higher Education*, pages 1806–1813. Association for the Advancement of Computing in Education (AACE).

[Carmona et al., 2005] Carmona, C., Millán, E., Pérez-de-la Cruz, J.-L., Trella, M., and Conejo, R. (2005). Introducing prerequisite relations in a multi-layered bayesian student model. In *User Modeling*, pages 347–356. Springer.

[Cen, 2009] Cen, H. (2009). *Generalized learning factors analysis: improving cognitive models with machine learning.* Carnegie Mellon University.

[Cen et al., 2006] Cen, H., Koedinger, K., and Junker, B. (2006). Learning factors analysis: A general method for cognitive model evaluation and improvement. In Ikeda, M., Ashley, K., and Chan, T.-W., editors, *Intelligent Tutoring Systems*, volume 4053 of *Lecture Notes in Computer Science*, pages 164–175. Springer Berlin / Heidelberg.

[Cen et al., 2007] Cen, H., Koedinger, K. R., and Junker, B. (2007). Is over practice necessary?-improving learning efficiency with the cognitive tutor through educational data mining. *Frontiers in artificial intelligence and applications*, 158:511.

[Chase and Simon, 1973] Chase, W. G. and Simon, H. A. (1973). Perception in chess. *Cognitive psychology*, 4(1):55–81.

[Chen et al., 2005] Chen, C.-M., Lee, H.-M., and Chen, Y.-H. (2005). Personalized e-learning system using item response theory. *Computers & Education*, 44(3):237–255.

[Chen et al., 2016] Chen, Y., González-Brenes, J. P., and Tian, J. (2016). Joint discovery of skill prerequisite graphs and student models. In *EDM*, pages 46–53.

[Chrysafiadi and Virvou, 2013] Chrysafiadi, K. and Virvou, M. (2013). Persiva: An empirical evaluation method of a student model of an intelligent e-learning environment for computer programming. *Computers & Education*, 68:322–333.

[Clark et al., 2007] Clark, R. E., Feldon, D. F., van Merrinboer, J. J. G., Yates, K. A., and Early, S. (2007). Cognitive task analysis. *Handbook of research on educational communications and technology (3rd ed.)*, pages 577–593.

[Collins et al., 1996] Collins, J. A., Greer, J. E., and Huang, S. X. (1996). Adaptive assessment using granularity hierarchies and bayesian nets. In *International Conference on Intelligent Tutoring Systems*, pages 569–577. Springer.

[Conati et al., 2002] Conati, C., Gertner, A., and Vanlehn, K. (2002). Using bayesian networks to manage uncertainty in student modeling. *User Modeling and User-Adapted Interaction*, 12(4):371–417.

[Corbett and Anderson, 1995] Corbett, A. T. and Anderson, J. R. (1995). Knowledge tracing: Modeling the acquisition of procedural knowledge. *User Modeling and User-Adapted Interaction*, 4(4):253–278.

[De Groot, 1978] De Groot, A. D. (1978). *Thought and choice in chess*, volume 4. Walter de Gruyter GmbH & Co KG.

[de Raadt, 2008] de Raadt, M. (2008). *Teaching programming strategies explicitly to novice programmers*. PhD thesis, University of Southern Queensland.

[Desmarais, 2012] Desmarais, M. C. (2012). Mapping question items to skills with non-negative matrix factorization. *ACM SIGKDD Explorations Newsletter*, 13(2):30–36.

[Desmarais and Baker, 2012] Desmarais, M. C. and Baker, R. S. (2012). A review of recent advances in learner and skill modeling in intelligent learning environments. *User Modeling and User-Adapted Interaction*, 22(1-2):9–38.

[Desmarais et al., 2006] Desmarais, M. C., Meshkinfam, P., and Gagnon, M. (2006). Learned student models with item to item knowledge structures. *User Modeling and User-Adapted Interaction*, 16(5):403–434.

[Druzdzel, 1999] Druzdzel, M. J. (1999). Smile: Structural modeling, inference, and learning engine and genie: a development environment for graphical decision-theoretic models. In *Aaai/Iaai*, pages 902–903.

[Du Boulay, 1986] Du Boulay, B. (1986). Some difficulties of learning to program. *Journal of Educational Computing Research*, 2(1):57–73.

[Egan and Schwartz, 1979] Egan, D. E. and Schwartz, B. J. (1979). Chunking in recall of symbolic drawings. *Memory & Cognition*, 7(2):149–158.

[Ekanadham and Karklin, 2017] Ekanadham, C. and Karklin, Y. (2017). T-skirt: Online estimation of student proficiency in an adaptive learning system. *arXiv preprint arXiv:1702.04282*.

[Embretson, 1997] Embretson, S. E. (1997). Multicomponent response models. In *Handbook of modern item response theory*, pages 305–321. Springer.

[Embretson and Reise, 2013] Embretson, S. E. and Reise, S. P. (2013). *Item response theory*. Psychology Press.

[Fancsali et al., 2013] Fancsali, S., Nixon, T., and Ritter, S. (2013). Optimal and worst-case performance of mastery learning assessment with bayesian knowledge tracing. In *Educational Data Mining 2013*.

[Feldon et al., 2010] Feldon, D. F., Timmerman, B. C., Stowe, K. A., and Showman, R. (2010). Translating expertise into effective instruction: The impacts of cognitive task analysis (cta) on lab report quality and student retention in the biological sciences. *Journal of research in science teaching*, 47(10):1165–1185.

[Gerdes et al., 2012] Gerdes, A., Jeuring, J., and Heeren, B. (2012). An interactive functional programming tutor. In *Proceedings of the 17th ACM annual conference on Innovation and technology in computer science education*, pages 250–255. ACM.

[Gong et al., 2010] Gong, Y., Beck, J. E., and Heffernan, N. T. (2010). Comparing knowledge tracing and performance factor analysis by using multiple model fitting procedures. In *Proc. 10th Int. Conf. Intelligent Tutoring Systems*, pages 35–44. Springer.

[González-Brenes, 2015] González-Brenes, J. (2015). Modeling skill acquisition over time with sequence and topic modeling. In *Artificial Intelligence and Statistics*, pages 296–305.

[González-Brenes and Huang, 2015] González-Brenes, J. P. and Huang, Y. (2015). Your model is predictive but is it useful? theoretical and empirical considerations of a new paradigm for adaptive tutoring evaluation. In *Proc. 8th Intl. Conf. Educational Data Mining*, pages 187–194.

[González-Brenes et al., 2014] González-Brenes, J. P., Huang, Y., and Brusilovsky, P. (2014). General features in knowledge tracing: Applications to multiple subskills, temporal item response theory, and expert knowledge. In *Proc. 7th Int. Conf. Educational Data Mining*, pages 84–91.

[Guo, 2013] Guo, P. J. (2013). Online python tutor: embeddable web-based program visualization for cs education. In *Proceeding of the 44th ACM technical symposium on Computer science education*, pages 579–584. ACM.

[Heffernan and Heffernan, 2014] Heffernan, N. T. and Heffernan, C. L. (2014). The assistments ecosystem: Building a platform that brings scientists and teachers together for minimally invasive research on human learning and teaching. *International Journal of Artificial Intelligence in Education*, 24(4):470–497.

[Heffernan and Koedinger, 1997] Heffernan, N. T. and Koedinger, K. R. (1997). The composition effect in symbolizing: The role of symbol production vs. text comprehension. In *Proc. 19th Annual Conf. Cognitive Science Society*, pages 307–312.

[Holland et al., 2009] Holland, J., Mitrovic, A., and Martin, B. (2009). J-latte: a constraint-based tutor for java.

[Hosseini and Brusilovsky, 2013] Hosseini, R. and Brusilovsky, P. (2013). Javaparser: A fine-grain concept indexing tool for java problems. In *CEUR Workshop Proceedings*, volume 1009, pages 60–63. University of Pittsburgh.

[Howard, 1983] Howard, R. A. (1983). *Readings on the principles and applications of decision analysis*, volume 1. Strategic Decisions Group.

[Hsiao et al., 2010] Hsiao, I.-H., Sosnovsky, S., and Brusilovsky, P. (2010). Guiding students to the right questions: adaptive navigation support in an e-learning system for java programming. *Journal of Computer Assisted Learning*, 26(4):270–283.

[Huang et al., 2015] Huang, Y., González-Brenes, J. P., Kumar, R., and Brusilovsky, P. (2015). A framework for multifaceted evaluation of student models. In *Proc. 8th Int. Conf. Educational Data Mining*, pages 203–210.

[Huang et al., 2017] Huang, Y., Guerra-Hollstein, J., Barria-Pineda, J., and Brusilovsky, P. (2017). Learner modeling for integration skills. In *Proceedings of the 25th Conference on User Modeling, Adaptation and Personalization*, pages 85–93. ACM.

[Huang et al., 2016] Huang, Y., Guerra-Hollstein, J. D., and Brusilovsky, P. (2016). Modeling skill combination patterns for deeper knowledge tracing. In *UMAP (Extended Proceedings)*.

[Huang et al., 2014] Huang, Y., Xu, Y., and Brusilovsky, P. (2014). Doing more with less: Student modeling and performance prediction with reduced content models. In *the 22nd Conf. User Modeling, Adaptation and Personalization*.

[Johnson and Soloway, 1985] Johnson, W. L. and Soloway, E. (1985). Proust: Knowledge-based program understanding. *IEEE Transactions on Software Engineering*, (3):267–275.

[Junker and Sijtsma, 2001] Junker, B. W. and Sijtsma, K. (2001). Cognitive assessment models with few assumptions, and connections with nonparametric item response theory. *Applied Psychological Measurement*, 25(3):258–272.

[Kahneman, 1973] Kahneman, D. (1973). *Attention and effort*, volume 1063. Prentice-Hall Englewood Cliffs, NJ.

[Käser et al., 2014] Käser, T., Klingler, S., Schwing, A. G., and Gross, M. (2014). Beyond knowledge tracing: Modeling skill topologies with bayesian networks. In *International Conference on Intelligent Tutoring Systems*, pages 188–198. Springer.

[Käser et al., 2017] Käser, T., Klingler, S., Schwing, A. G., and Gross, M. (2017). Dynamic bayesian networks for student modeling. *IEEE Transactions on Learning Technologies*, 10(4):450–462.

[Kasurinen and Nikula, 2009] Kasurinen, J. and Nikula, U. (2009). Estimating programming knowledge with bayesian knowledge tracing. In *ACM SIGCSE Bulletin*, volume 41, pages 313–317. ACM.

[Khajah et al., 2014] Khajah, M., Wing, R., Lindsey, R., and Mozer, M. (2014). Integrating latent-factor and knowledge-tracing models to predict individual differences in learning. In *Educational Data Mining 2014*. Citeseer.

[Klingler et al., 2015] Klingler, S., Käser, T., Solenthaler, B., and Gross, M. (2015). On the performance characteristics of latent-factor and knowledge tracing models. *International Educational Data Mining Society*.

[Koedinger and McLaughlin, 2010] Koedinger, K. and McLaughlin, E. (2010). Seeing language learning inside the math: Cognitive analysis yields transfer. In *Proceedings of the Annual Meeting of the Cognitive Science Society*, volume 32.

[Koedinger et al., 2003] Koedinger, K. R., Aleven, V., and Heffernan, N. (2003). Toward a rapid development environment for cognitive tutors. In *Artificial Intelligence in Education: Shaping the Future of Learning through Intelligent Technologies, Proceedings of AI-ED*, pages 455–457.

[Koedinger and Anderson, 1990] Koedinger, K. R. and Anderson, J. R. (1990). Abstract planning and perceptual chunks: Elements of expertise in geometry. *Cognitive Science*, 14(4):511–550.

[Koedinger et al., 2012a] Koedinger, K. R., Corbett, A. T., and Perfetti, C. (2012a). The knowledge-learning-instruction framework: Bridging the science-practice chasm to enhance robust student learning. *Cognitive Science*, 36(5):757–798.

[Koedinger et al., 2012b] Koedinger, K. R., Corbett, A. T., and Perfetti, C. (2012b). The knowledge-learning-instruction framework: Bridging the science-practice chasm to enhance robust student learning. *Cognitive Science*, 36(5):757798. Robust learning: sts, transfers, accelerates future learning.

[Koedinger and McLaughlin, 2016] Koedinger, K. R. and McLaughlin, E. A. (2016). Closing the loop with quantitative cognitive task analysis. In *EDM*, pages 412–417.

[Koedinger et al., 2012c] Koedinger, K. R., McLaughlin, E. A., and Stamper, J. C. (2012c). Automated student model improvement. *Proc. 8th Intl. Conf. on Educational Data Mining*, pages 17–24.

[Koedinger and Nathan, 2004] Koedinger, K. R. and Nathan, M. J. (2004). The real story behind story problems: Effects of representations on quantitative reasoning. *The journal of the learning sciences*, 13(2):129–164.

[Koedinger et al., 2011] Koedinger, K. R., Pavlik Jr, P. I., Stamper, J. C., Nixon, T., and Ritter, S. (2011). Avoiding problem selection thrashing with conjunctive knowledge tracing. In *Proc. 7th Int. Conf. Educational Data Mining*, pages 91–100.

[Koedinger et al., 2013] Koedinger, K. R., Stamper, J. C., McLaughlin, E. A., and Nixon, T. (2013). Using data-driven discovery of better student models to improve student learning. In *International Conference on Artificial Intelligence in Education*, pages 421–430. Springer.

[Kumar, 2006] Kumar, A. N. (2006). Using enhanced concept map for student modeling in programming tutors. In *FLAIRS Conference*, pages 527–532.

[Lan et al., 2014] Lan, A. S., Waters, A. E., Studer, C., and Baraniuk, R. G. (2014). Sparse factor analysis for learning and content analytics. *The Journal of Machine Learning Research*, 15(1):1959–2008.

[Lane and VanLehn, 2005] Lane, H. C. and VanLehn, K. (2005). Teaching the tacit knowledge of programming to noviceswith natural language tutoring. *Computer Science Education*, 15(3):183–201.

[Larkin et al., 1980] Larkin, J., McDermott, J., Simon, D. P., and Simon, H. A. (1980). Expert and novice performance in solving physics problems. *Science*, 208(4450):1335–1342.

[Le and Menzel, 2009] Le, N.-T. and Menzel, W. (2009). Usingweighted constraints to diagnose errors in logic programming–the case of an ill-defined domain. *International Journal of Artificial Intelligence in Education*, 19(4):381–400.

[Lee and Brunskill, 2012] Lee, J. I. and Brunskill, E. (2012). The impact on individualizing student models on necessary practice opportunities. In *Proceedings of the 5th International Conference on Educational Data Mining*, pages 118–125, Chania, Greece. www.educationaldatamining.org.

[Lesgold et al., 1988] Lesgold, A., Lajoie, S., Bunzo, M., and Eggan, G. (1988). Sherlock: A coached practice environment for an electronics troubleshooting job. *Computer-assisted instruction and intelligent tutoring systems: shared goals and complementary approaches.*

[Lister et al., 2004] Lister, R., Adams, E. S., Fitzgerald, S., Fone, W., Hamer, J., Lindholm, M., McCartney, R., Moström, J. E., Sanders, K., Seppälä, O., et al. (2004). A multinational study of reading and tracing skills in novice programmers. In *ACM SIGCSE Bulletin*, volume 36, pages 119–150. ACM.

[Lister et al., 2009] Lister, R., Fidge, C., and Teague, D. (2009). Further evidence of a relationship between explaining, tracing and writing skills in introductory programming. In *Acm sigcse bulletin*, volume 41, pages 161–165. ACM.

[Lister et al., 2006] Lister, R., Simon, B., Thompson, E., Whalley, J. L., and Prasad, C. (2006). Not seeing the forest for the trees: novice programmers and the solo taxonomy. *ACM SIGCSE Bulletin*, 38(3):118–122.

[Liu and Koedinger, 2017] Liu, R. and Koedinger, K. R. (2017). Closing the loop: Automated data-driven cognitive model discoveries lead to improved instruction and learning gains. *Journal of Educational Data Mining*, 9(1):25–41.

[Martin et al., 2011] Martin, B., Mitrovic, A., Koedinger, K. R., and Mathan, S. (2011). Evaluating and improving adaptive educational systems with learning curves. *User Modeling and User-Adapted Interaction*, 21(3):249–283.

[Mathews, 2006] Mathews, M. (2006). Investigating the effectiveness of problem templates on learning in intelligent tutoring systems.

[Mayo and Mitrovic, 2000] Mayo, M. and Mitrovic, A. (2000). Using a probabilistic student model to control problem difficulty. In *International Conference on Intelligent Tutoring Systems*, pages 524–533. Springer.

[Mayo and Mitrovic, 2001] Mayo, M. and Mitrovic, A. (2001). Optimising its behaviour with bayesian networks and decision theory.

[Millán and Pérez-De-La-Cruz, 2002] Millán, E. and Pérez-De-La-Cruz, J. L. (2002). A bayesian diagnostic algorithm for student modeling and its evaluation. *User Modeling and User-Adapted Interaction*, 12(2-3):281–330.

[Miller, 1956] Miller, G. A. (1956). The magical number seven, plus or minus two: Some limits on our capacity for processing information. *Psychological review*, 63(2):81.

[Mislevy and Gitomer, 1995] Mislevy, R. J. and Gitomer, D. H. (1995). The role of probability-based inference in an intelligent tutoring system. *ETS Research Report Series*, 1995(2).

[Mitrovic, 2012] Mitrovic, A. (2012). Fifteen years of constraint-based tutors: what we have achieved and where we are going. *User modeling and user-adapted interaction*, 22(1-2):39–72.

[Mitrovic et al., 2001] Mitrovic, A., Mayo, M., Suraweera, P., and Martin, B. (2001). Constraint-based tutors: a success story. In *International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems*, pages 931–940. Springer.

[Morales et al., 2006] Morales, R., Van Labeke, N., and Brna, P. (2006). Approximate modelling of the multi-dimensional learner. In *International Conference on Intelligent Tutoring Systems*, pages 555–564. Springer.

[Muller et al., 2007] Muller, O., Ginat, D., and Haberman, B. (2007). Pattern-oriented instruction and its influence on problem decomposition and solution construction. *ACM SIGCSE Bulletin*, 39(3):151–155.

[Murray, 2005] Murray, R. C. (2005). *An evaluation of decision-theoretic tutorial action selection*. PhD thesis, University of Pittsburgh.

[Murray, 1985] Murray, W. R. (1985). *Heuristic and formal methods in automatic program debugging*. Computer Science Department, University of Texas at Austin.

[Newell and Rosenbloom, 1981] Newell, A. and Rosenbloom, P. S. (1981). Mechanisms of skill acquisition and the law of practice. *Cognitive skills and their acquisition*, 1(1981):1–55.

[Ohlsson, 1994] Ohlsson, S. (1994). Constraint-based student modeling. In *Student modelling: the key to individualized knowledge-based instruction*, pages 167–189. Springer.

[Paas and Van Merriënboer, 1993] Paas, F. G. and Van Merriënboer, J. J. (1993). The efficiency of instructional conditions: An approach to combine mental effort and performance measures. *Human factors*, 35(4):737–743.

[Paramythis et al., 2010] Paramythis, A., Weibelzahl, S., and Masthoff, J. (2010). Layered evaluation of interactive adaptive systems: framework and formative methods. *User Modeling and User-Adapted Interaction*, 20(5):383–453.

[Pardos and Heffernan, 2010a] Pardos, Z. A. and Heffernan, N. T. (2010a). Modeling individualization in a bayesian networks implementation of knowledge tracing. In *International Conference on User Modeling, Adaptation, and Personalization*, pages 255–266. Springer.

[Pardos and Heffernan, 2010b] Pardos, Z. A. and Heffernan, N. T. (2010b). Navigating the parameter space of bayesian knowledge tracing models: Visualizations of the convergence of the expectation maximization algorithm. *EDM*, 2010:161–170.

[Pardos and Yudelson, 2013] Pardos, Z. A. and Yudelson, M. (2013). Towards moment of learning accuracy. In *AIED Workshops*.

[Pavlik et al., 2009] Pavlik, P., Cen, H., and Koedinger, K. (2009). Performance Factors Analysis–A New Alternative to Knowledge Tracing. In *Proceeding of the 2009 conference on Artificial Intelligence in Education: Building Learning Systems that Care: From Knowledge Representation to Affective Modelling*, pages 531–538. IOS Press.

[Pearl, 2014] Pearl, J. (2014). *Probabilistic reasoning in intelligent systems: networks of plausible inference.* Elsevier.

[Pelánek, 2015] Pelánek, R. (2015). Metrics for evaluation of student models. *Journal of Educational Data Mining*, 7(2):1–19.

[Pelánek and Řihák, 2017] Pelánek, R. and Řihák, J. (2017). Experimental analysis of mastery learning criteria. In *Proceedings of the 25th Conference on User Modeling, Adaptation and Personalization*, pages 156–163. ACM.

[Proulx, 2000] Proulx, V. K. (2000). Programming patterns and design patterns in the introductory computer science course. In *ACM SIGCSE Bulletin*, volume 32, pages 80–84. ACM.

[Ritter et al., 2007] Ritter, S., Anderson, J. R., Koedinger, K. R., and Corbett, A. (2007). Cognitive tutor: Applied research in mathematics education. *Psychonomic bulletin & review*, 14(2):249–255.

[Rivers et al., 2016] Rivers, K., Harpstead, E., and Koedinger, K. R. (2016). Learning curve analysis for programming: Which concepts do students struggle with? In *ICER*, pages 143–151.

[Rivers and Koedinger, 2017] Rivers, K. and Koedinger, K. R. (2017). Data-driven hint generation in vast solution spaces: a self-improving python programming tutor. *International Journal of Artificial Intelligence in Education*, 27(1):37–64.

[Rollinson and Brunskill, 2015] Rollinson, J. and Brunskill, E. (2015). From predictive models to instructional policies. *International Educational Data Mining Society.*

[Sfard and Linchevski, 1994] Sfard, A. and Linchevski, L. (1994). The gains and the pitfalls of reification?the case of algebra. In *Learning mathematics*, pages 87–124. Springer.

[Shneiderman, 1976] Shneiderman, B. (1976). Exploratory experiments in programmer behavior. *International Journal of Computer & Information Sciences*, 5(2):123–143.

[Soloway and Ehrlich, 1984] Soloway, E. and Ehrlich, K. (1984). Empirical studies of programming knowledge. *IEEE Trans. Software Engineering*, SE-10(5):595–609.

[Sweller, 1988] Sweller, J. (1988). Cognitive load during problem solving: Effects on learning. *Cognitive science*, 12(2):257–285.

[Sweller and Cooper, 1985] Sweller, J. and Cooper, G. A. (1985). The use of worked examples as a substitute for problem solving in learning algebra. *Cognition and Instruction*, 2(1):59–89.

[Taylor, 1977] Taylor, R. P. (1977). Teaching programming to beginners. In *ACM SIGCSE Bulletin*, volume 9, pages 88–92. ACM.

[Tulving and Craik, 2000] Tulving, E. and Craik, F. I. (2000). *The Oxford handbook of memory*. Oxford: Oxford University Press.

[Van de Sande, 2013] Van de Sande, B. (2013). Properties of the bayesian knowledge tracing model. *JEDM-Journal of Educational Data Mining*, 5(2):1–10.

[Van de Sande, 2016] Van de Sande, B. (2016). Learning curves for problems with multiple knowledge components. In *EDM*, pages 523–526.

[VanLehn, 1990] VanLehn, K. (1990). *Mind bugs: The origins of procedural misconceptions*. MIT press.

[Vanlehn, 2006] Vanlehn, K. (2006). The behavior of tutoring systems. *International journal of artificial intelligence in education*, 16(3):227–265.

[Vanlehn et al., 2005] Vanlehn, K., Lynch, C., Schulze, K., Shapiro, J. A., Shelby, R., Taylor, L., Treacy, D., Weinstein, A., and Wintersgill, M. (2005). The andes physics tutoring system: Lessons learned. *International Journal of Artificial Intelligence in Education*, 15(3):147–204.

[VanLehn et al., 1998] VanLehn, K., Niu, Z., Siler, S., and Gertner, A. S. (1998). Student modeling from conventional test data: A bayesian approach without priors. In *International Conference on Intelligent Tutoring Systems*, pages 434–443. Springer.

[Velmahos et al., 2004] Velmahos, G. C., Toutouzas, K. G., Sillin, L. F., Chan, L., Clark, R. E., Theodorou, D., and Maupin, F. (2004). Cognitive task analysis for teaching technical skills in an inanimate surgical skills laboratory. *The American Journal of Surgery*, 187(1):114–119.

[Vesin et al., 2012] Vesin, B., Ivanović, M., KlašNja-MilićEvić, A., and Budimac, Z. (2012). Protus 2.0: Ontology-based semantic recommendation in programming tutoring system. *Expert Systems with Applications*, 39(15):12229–12246.

[Wang et al., 2017] Wang, S., Han, Y., Wu, W., and Hu, Z. (2017). Modeling student learning outcomes in studying programming language course. In *Information Science and Technology (ICIST), 2017 Seventh International Conference on*, pages 263–270. IEEE.

[Weber, 1996a] Weber, G. (1996a). Episodic learner modeling. *Cognitive Science*, 20(2):195–236.

[Weber, 1996b] Weber, G. (1996b). Individual selection of examples in an intelligent learning environment. *Journal of Interactive Learning Research*, 7(1):3.

[Weber and Brusilovsky, 2001] Weber, G. and Brusilovsky, P. (2001). Elm-art: An adaptive versatile system for web-based instruction. *International Journal of Artificial Intelligence in Education*, 12(4):351–384.

[Weng et al., 2018] Weng, R. C.-H., Coad, D. S., et al. (2018). Real-time bayesian parameter estimation for item response models. *Bayesian Analysis*, 13(1):115–137.

[Whalley et al., 2006] Whalley, J. L., Lister, R., Thompson, E., Clear, T., Robbins, P., Kumar, P., and Prasad, C. (2006). An australasian study of reading and comprehension skills in novice programmers, using the bloom and solo taxonomies. In *Proceedings of the 8th Australasian Conference on Computing Education-Volume 52*, pages 243–252. Australian Computer Society, Inc.

[Winters et al., 2005] Winters, T., Shelton, C., Payne, T., and Mei, G. (2005). Topic extraction from item-level grades. In *American Association for Artificial Intelligence 2005 workshop on educational datamining, Pittsburgh, PA*, volume 1, page 3.

[Xu and Mostow, 2012] Xu, Y. and Mostow, J. (2012). Comparison of methods to trace multiple subskills: Is LR-DBN best? In *Proc. 5th Intl. Conf. Educational Data Mining*, pages 41–48, Chania, Greece.

[Yudelson et al., 2014] Yudelson, M., Hosseini, R., Vihavainen, A., and Brusilovsky, P. (2014). Investigating automated student modeling in a java mooc. *Educational Data Mining 2014*, pages 261–264.

[Yudelson and Koedinger, 2013] Yudelson, M. and Koedinger, K. (2013). Estimating the benefits of student model improvements on a substantive scale. In *Educational Data Mining 2013*.

[Yudelson et al., 2013] Yudelson, M. V., Koedinger, K. R., and Gordon, G. J. (2013). Individualized bayesian knowledge tracing models. In *International Conference on Artificial Intelligence in Education*, pages 171–180. Springer.

[Zapata-Rivera and Greer, 2001] Zapata-Rivera, J.-D. and Greer, J. (2001). Smodel server: Student modelling in distributed multi-agent tutoring systems. In *Proceedings of AIED*, pages 446–455.