Int J Artif Intell Educ (2017) 27:5–36
DOI 10.1007/s40593-016-0112-1

ARTICLE

Evolution of an Intelligent Deductive Logic Tutor Using Data-Driven Elements

Behrooz Mostafavi[1] · **Tiffany Barnes**[1]

Published online: 19 April 2016

Abstract Deductive logic is essential to a complete understanding of computer science concepts, and is thus fundamental to computer science education. Intelligent tutoring systems with individualized instruction have been shown to increase learning gains. We seek to improve the way deductive logic is taught in computer science by developing an intelligent, data-driven logic tutor. We have augmented Deep Thought, an existing computer-based logic tutor, by adding data-driven methods, specifically; intelligent problem selection based on the student's current proficiency, automatically generated on-demand hints, and determination of student problem solving strategies based on clustering previous students. As a result, student tutor completion (the amount of the tutor the students completed) steadily improved as data-driven methods were added to Deep Thought, allowing students to be exposed to more logic concepts. We also gained additional insights into the effects of different course work and teaching methods on tutor effectiveness.

Keywords Deductive logic instruction · Intelligent tutoring systems · Data-driven methods

✉ Behrooz Mostafavi
bzmostafavi@gmail.com

Tiffany Barnes
tmbarnes@ncsu.edu

[1] North Carolina State University, Raleigh NC, USA

Introduction

Deductive logic is an important fundamental topic in computer science education. Meyers (1990) outlined this necessity very clearly. He noted that almost every aspect of computer science required logic as a foundational basis in order to understand it. In 2003 a study conducted by Page (2003) confirmed the necessity of discrete math, including deductive logic, to computer science education.

It is therefore in the interest of improving computer science education to improve how deductive logic is taught in the classroom. Individualized student instruction is one of the most effective educational strategies, and students who receive individualized instruction often perform significantly better on performance evaluations than students who receive classroom instruction (Ma et al. 2014; VanLehn 2011; Steenbergen-Hu and Cooper 2014). Computer learning environments that mimic aspects of human tutors have also been highly successful. Intelligent tutoring systems (ITS) have been shown to be highly effective in improving learning gains when used in combination with classroom instruction through scaffolding of domain concepts and contextualized, individual feedback (VanLehn et al. 2005). It is important for intelligent tutoring systems to present problems to students that are appropriate to the student's current level of proficiency in the subject matter. Students are more likely to perform better in-tutor when given problems in their zone of proximal development through scaffolding of major concepts (Murray and Arroyo 2002; VanLehn 2006). Proper scaffolding increases learning, and therefore increases the likelihood of further progression through the tutor and exposure to higher-level concepts. However, in complex domains such as deductive logic and programming, there is no defined breakdown of knowledge components, and traditional knowledge tracing models are therefore difficult to apply without developing an entire cognitive model and increasing the amount of development time and expert involvement required.

Research Question

Our research question is as follows: can data-driven methods be used to create an adaptive tutoring system in deductive logic that reduces student dropout and time needed to complete the tutor by selecting appropriate problems and feedback.

We have explored this research question through the development of Deep Thought, a computer-based logic proof tutor. Several versions of Deep Thought have been developed, with each new version augmented with intelligent, data-driven behavior in how problems were selected and feedback was provided. In the next section, we will examine previous work in using data-driven methods to develop intelligent tutoring systems, and intelligent tutors in deductive logic. Next we will describe the development of Deep Thought and outline in detail the intelligent, data-driven components that were added to each version. Then we will describe the studies

conducted to test the effectiveness of each version and the results gathered. Finally we will evaluate and discuss the results in the last section.

Background

Data-Driven Methods

The use of data-driven methods to develop intelligent tutoring systems is just starting to be explored in the field. An early example of a data-driven intelligent tutor is the Cognitive Algebra Tutor, first introduced by Ritter (2007). Here the authors introduce an algebra tutor which is fully driven by historical data, interpreted based on a cognitive theory and student data gathered from several previous studies. It took several years to develop the tutor, as well as several previous studies; the result is an example of a fully data-driven tutor that improved student performance. The authors noted that although it over-predicted student performance, it would be improved the more data was collected. There have also been several recent studies that demonstrate the potential for data driven methods to result in more effective tutors that accurately predict student behavior. Lee and Brunskill (2012) examined the benefits and drawbacks to basing model parameters on existing data from individual students in comparison to data from an entire population, specifically as it pertained to the number of practice opportunities a student would require (estimated) to master a skill. The authors estimated that using individualized parameters would reduce the number of practice opportunities a student would need to master a skill. Gonzalez-Brenes and Mostow (2013) demonstrated a data-driven model which automatically generates a cognitive and learning model based on previous student data in order to discover what skills students learn at any given time and when they use skills they have learned. The resulting model predicted student behavior without the aid of previous domain knowledge and performed comparably to a published model.

Data-driven intelligent tutors not only have the potential to more accurately predict student behavior, but interpret why it occurs. For instance, Elmadani et al. (2012) proposed using data-driven techniques to detect student errors that occur due to genuine misunderstanding of the concepts (misconception detection). They processed their data using frequent pattern growth (FP-Growth) in order to build a set of frequent itemsets which represented the possible misconceptions students could make. The authors were able to detect several misconceptions based on the resulting itemsets of student actions. Fancsali (2014) used data-driven methods to detect behaviors that usually detract from a student's experience with an ITS (off-task behavior, gaming the system, etc).

Overall the literature shows that incorporating data-driven methods have a great deal of potential to accurately adapt to and predict student behavior, and increase learning gains. However the focus has been on using data to develop or determine a

cognitive model, and predict student behavior based on how it fits with the model. Deep Thought does not attempt to develop a cognitive model, but a student model that is fully based on previous student behavior in the tutor.

Logic Tutors

Several computer systems have been built for instruction in solving logic proofs. However, many of these tools simply teach logic without any sort of adaptability. LogEx, for example, is a learning environment for propositional logic that logs student data, but doesn't currently use that data to enhance the tutor (Lodder et al. 2015). Sequent Calculus Trainer (Ehle et al. 2015), on the other hand, incorporates feedback only in the form of information presented on how to use each logic rule or an error if a rule is applied incorrectly, features that were present in the earliest version of Deep Thought. We are more concerned with work that uses data-driven methods incorporated into an intelligent tutor to improve its effectiveness. Logic is of particular interest because it is a complex problem solving domain that computer science students must learn (Meyers 1990; Page 2003). To that end, there are several logic tutors that incorporate data-driven methods, mainly to improve feedback. Logic-ITA (Merceron and Yacef 2005) provides students with performance feedback upon proof completion, but not much feedback beyond that. Association rule mining, clustering, and classification were performed on the data gathered to determine indicators for students completing the tutor successfully, such as how many problems the student was able to successfully solve, number of mistakes made (as well as what kind of mistakes), number of rules used, and number of steps taken. They later implemented proactive hints based on previous student behavior that the instructor could apply or alter. CPT (Sieg 2007) uses an automated proof generator to provide contextual hints. The proof generator specifically uses interaction calculus to automatically find a completed proof based on a set of premises. This search algorithm is used to generate hints for students based on the best possible path to completion from the current set of premises in that step of the problem. PLT (Lukins et al. 2002) uses a combination of dynamically generated hints and more general examples for rule applications and axiom generation based on previous student data. However, the authors determined that based on student responses, the tutor setup and design prevented students from gaining as much as they could have from a more effectively designed and user-friendly system. More recently, Gluz et al. implemented a dynamic and adaptive natural deduction proof tutor called Heraclito based on a dynamic student model that can automatically solve problems using strategies employed by instructors (Gluz et al. 2014). The authors argue that the Heraclito system is fully adaptive, conforming to the way students reason and solve logic proofs.

The Deep Thought logic tutor has been augmented for adaptive learning by providing hints and intelligent problem selection using data-driven methods. Unlike the previously listed tutors, which incorporate specific feedback generation methods, Deep Thought has been developed to flexibly provide students with a combination of hints and worked examples. Additionally while the previous logic tutors use data-based or data-driven methods to improve feedback to students, they do not make any

Table 1 Overview of Deep Thought versions 0-4

Deep Thought Version	Features
DT0	Basic Deep Thought Logic Tutor
DT1	Deep Thought with on-demand step level hints
DT2	Deep Thought with the DDML implemented for problem selection
DT3-NoHint	Identical to DT2-MP except for the new user interface
DT3-Random	Deep Thought with the DDML and on-demand step-level hints and/or worked examples (depending on features enabled by the instructor)
DT4	Deep Thought with the DDML for problem selection and the possibility for step-level hints or high-level hints in addition to worked examples

changes to the difficulty or type of problem presented to the students. Deep Thought uses a data-driven system to select the best next set of problems for students to solve.

Deep Thought Development

Deep Thought has gone through several iterations in its evolution as features have been added and improved upon. An overview of these versions and their distinctive features is outlined in Table 1.

Introduction to Deep Thought

Deep Thought in its original form (hereafter referred to as DT0) was a simple web interface for students to practice solving logic proof problems, with the only form of feedback consisting of a text box with an alert that appeared whenever a student made an error. Barnes and Stamper extended Deep Thought with a data-driven hint-generation system called Hint Factory to create Deep Thought 1 (also referred to as DT1) between 2006 and 2008 (Barnes and Stamper 2008). Hint Factory uses a Markov decision process (MDP) to automatically select on-demand hints for students upon request, based on their individual performance on specific problems. The MDP is data-driven, using actions logs from previous Deep Thought use in the classroom to assign weight to proof-state actions based on whether or not that action ultimately led to successful completion of the proof. The responses for the proof-state in Fig. 1 are shown in Table 2. The student is given a broader hint when they first request one, and given more direct hints the more they ask.

Figure 1 shows the interface for Deep Thought, where students construct logic proofs by connecting displayed logical premises (seen at the top of the figure window) using buttons for logical rules (seen at the right of the figure) to derive a conclusion (seen at the bottom of the figure window). For example, students solving the proof in Fig. 1 use premises $A \rightarrow (B \wedge C)$; $A \vee D$; and $\neg D \wedge E$ to derive

Table 2 A solution to the proof shown in Fig. 1

#	Premise	Derivation
1	$A \rightarrow (B \wedge C)$	Given
2	$A \vee D$	Given
3	$\neg D \wedge E$	Given
4	$\neg D$	3/Simplification
5	A	2,4/Disjunctive Syllogism
6	$B \wedge C$	1,5/*Modus Ponens*
7	B	6/Simplification

conclusion B. An example solution to this proof is shown in Table 2. Each of the logic rule buttons has a rule-reference in graphical representation that could be seen by hovering the mouse pointer over the selected rule button for two seconds (see inset window in Fig. 1). As a student worked through a problem, each step was logged in a data file that recorded the current problem, the rule being applied, any errors made (such as attempting to use a rule that was logically impossible), completion of the problem, time taken per step, and elapsed time taken to solve the problem. Problem selection and window options in DT1 were accomplished through a drop-down window. Students could select the representation of the logical operators in proof expressions (and; or; not; implication; equivalence): either in English mode (as "and"; "or"; "not"; "if...then"; "iff"); or in symbolic mode (as "&"; "v"; ["∼"/"-"]; ["⊃"/"→"]; "="). Students had the option of working forwards (top down), by

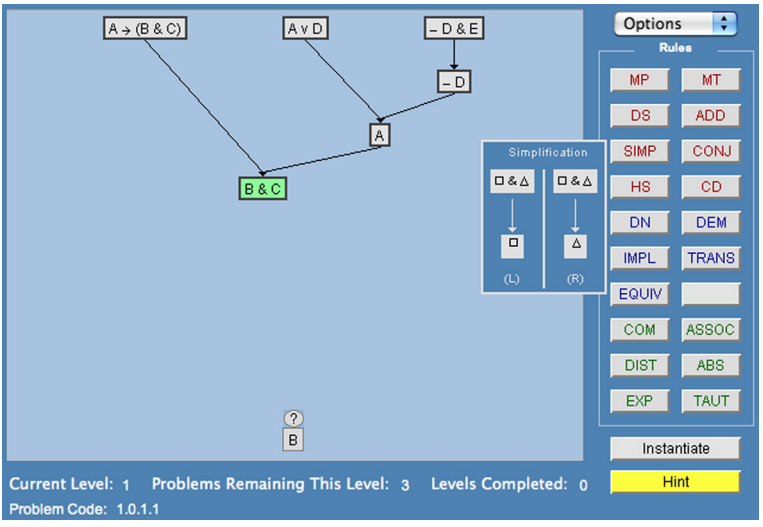

Fig. 1 A screen capture of the Deep Thought tutor, showing given premises at the *top*, conclusion at the *bottom*, and rules for application on the *right*. The *inset window* is an example of a rule-axiom reference that appears when a student hovers a mouse pointer over a rule button

selecting one or two expression nodes and applying a rule to derive a new expression,or by working backwards (bottom up) by selecting the "?" button above a conclusion block to create rule-axiom-structured parent blocks, that then must have their unknown variables instantiated (Croy 2000).

The proofs presented in Deep Thought were divided into three sets of problems. The first set contained problems requiring basic logical implication rules (e.g. Modus Ponens, Simplification), the second set contained problems expanding the rule requirements to include more advanced logical implication rules (e.g. Hypothetical Syllogism, Constructive Dilemma), and the third set contained problems that required logical equivalence rules in addition to implication rules (e.g. DeMorgan's, Contrapositive). In DT1 students were allowed to solve assigned problems at will, in any order, though assignments were ordered. There were a total of thirteen available problems in the tutor for the duration of the studies referenced.

Hint Factory was able to provide a hint when requested by a student 72 % of the time on average using one semester of performance data, 79 % when using two semesters, and 82 % of the time when using three semesters of performance data (Barnes et al. 2008). Eagle and Barnes later showed these hints were effective in reducing the time taken to complete the tutor by 55 %, and increasing retention in the tutor by 300 % (Eagle and Barnes 2014a, b) (Table 3).

Deep Thought 2: Data-Driven Mastery Learning (DDML) System

After DT1 was tested in classroom use we found significant numbers of students that did not complete the tutor. This trend needed to be addressed because students gain the most exposure to, and practice with, higher level concepts in Deep Thought when they finish the entirety of the tutor. To address this issue, a novel data-driven mastery learning system (DDML) was developed to improve learning in complex procedural problem solving domains, particularly deductive logic. The DDML system was integrated into DT1 in 2013 as Deep Thought 2 (also referred to as DT2). It was hypothesized that the DDML system developed for DT2 would result in higher percentage completion and lower student dropout rates than DT1.

The user interaction of DT2 was identical to DT1, except that instead of allowing students to select a proof problem from a list in the "Options" menu, DT2 selected problems based on that student's measured proof-solving proficiency. Students were allowed to skip a tutor-selected problem using the drop-down menu; the

Table 3 A generated on-demand hint sequence for the proof-state in Fig. 1

Hint #	Hint Text	Hint Type
1	Try to derive "B" working forward	Indicate goal expression
2	Highlight "B & C" to derive it	Indicate premises to select
3	Click on the rule Simplification (SIMP)	Indicate the rule to use
4	Highlight "B & C" and click on Simplification (SIMP) to derive "B"	Bottom-out hint

problems populated in the drop-down menu were dependent on that student's current progression through the tutor.

DT2 divided the problem sets of DT1 into more levels. Each of the three sets of problems in DT1 was split into two sets in DT2 to allow assessment of student rule application at regular intervals. Since the students are directed through the tutor via the DDML, the feedback for DT2 was restricted to alerts for errors in the text box. No additional hints or problem-solving feedback were provided so that the DDML could accurately assess student proficiency without the proficiency being affected by additional feedback. DT2's data-driven mastery learning system consists of two major components: a mastery learning leveling component; and an assessment component. Their processes are described in the following subsections.

Mastery Learning Leveling Component

To provide sufficient student-performance benchmarks for regular assessment, DT2 breaks the problem sets of DT1 into more levels. The new DT2 problem sets maintain the same rule applications and the same difficulty level of problems in DT1. Each of the three sets of problems in DT1 is split into two sets in DT2 to allow regular assessment of student rule application at the end of each level. DT2 Level 1 and 2 problems require basic logical implication rules, Level 3 and 4 problems expand the rule requirements to include more advanced logical implication rules, and Level 5 and 6 problems require logical equivalence rules in addition to implication rules. The new problem sets were made and tested by two domain experts to ensure consistency between DT1 and DT2 problem rule requirements and problem difficulty. The DT2 DDML system strictly orders levels and their problem sets to ensure consistent, directed practice using increasingly difficult logic rules. Problems sets are presented at two degrees of difficulty (higher and lower). As opposed to traditional mastery learning, where students are required to complete a number of training tasks until they pass a mastery test, the DDML system requires a minimum number of completed problems in a level. Their cumulative performance on target-rule actions at the end of a level determines whether they attempt the next level at higher or lower proficiency. An overview of the assessment algorithm is shown in Fig. 2.

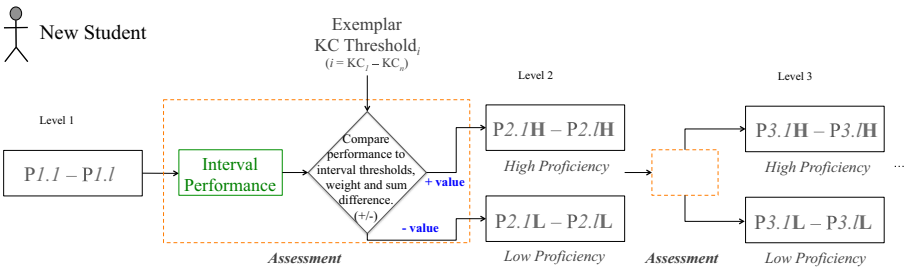

Fig. 2 The data-driven mastery learning system. At each level interval, new student knowledge component (KC) scores are calculated and then compared to exemplar thresholds for corresponding problem sets (see Fig. 5). The +/− threshold sign difference is weighted by KC priority and summed. The sign of the result determines whether students are assigned problem sets of high or low proficiency in the next level

In DT2, evaluation of student performance is performed at the beginning of each level of problems. Level 1 of DT2 contains three problems common to all students who use the tutor, and provides initial performance assessment data to the DDML model. Levels 2–6 of DT2 are each split into two distinct sets of problems, labeled higher and lower proficiency, by domain experts who had previously taught the course based on their hands-on experience with students. The problems in the different proficiency sets prioritize the same rules judged by domain experts to be important for solving the problems in that level. However, the degree of problem solving difficulty between proficiency sets is different, with problems in the low proficifency set requiring fewer numbers of steps for completion (1 or 2 fewer steps on average than a high proficiency problem), lower complexity of logical expressions, and lower number of rule applications than problems in the high proficiency set. An example of a high proficiency set problem and a low proficiency set problem, and the steps to complete them, are listed in Table 4. These two problems both start with four premises and prioritize the correct usage of the same two rules – Conjunction and Constructive Dilemma - but there are a couple of differences. The lower proficiency problem (top) requires five steps to reach the conclusion, while the higher proficiency problem (bottom) requires six. Additionally, it is much more readily obvious where to apply one of the prioritized rules - Constructive Dilemma - in the low proficiency problem as opposed to the high proficiency problem. In the low

Table 4 An example of lower and higher proficiency set problems from DT2 requiring the same target-rules: Level 4 Problem 3 from the lower proficiency set (top); Level 4 Problem 2 from the higher proficiency set (bottom)

#	Premise	Derivation
	$(A \rightarrow B) \wedge (\neg D \rightarrow F)$	Given
2	$A \vee \neg D$	Given
3	$\neg A \rightarrow (D \vee G)$	Given
4	$\neg A$	Given
5	$B \vee F$	1,2/Constructive Dilemma
6	$\neg D$	2,4/Disjunctive Syllogism
7	$D \vee G$	3,4/*Modus Ponens*
8	G	6,7/Disjunctive Syllogism
9	$(B \vee F) \wedge G$	5,8/Conjunction
1	$Z \rightarrow (\neg Y \rightarrow X)$	Given
2	$Z \wedge \neg W$	Given
3	$W \vee (T \rightarrow S)$	Given
4	$\neg Y \vee T$	Given
5	Z	2/Simplification
6	$\neg W$	2/Simplification
	$\neg Y \rightarrow X$	1,5/*Modus Ponens*
8	$T \rightarrow S$	3,6/Disjunctive Syllogism
9	$(\neg Y \rightarrow X) \wedge (T \rightarrow S)$	7,8/Conjunction
10	$X \vee S$	4,9/Constructive Dilemma

The prioritized rules required for these problems are Conjunction and Constructive Dilemma

proficiency problem the student is able to apply Constructive Dilemma directly from the premises, which would resemble the types of examples the student would likely have seen in a classroom setting. In the high proficiency problem on the other hand, the student must derive a few additional logic statements before they are able to use either of the prioritized rules.

Figure 4 shows the possible path progressions of students using DT2. Students cannot progress from one level to the next in DT2 without showing an expected level of proof solving mastery in the current level by completing the problems in their proficiency path. Students on the higher proficiency path for a given level are required to solve two proof problems of higher difficulty to continue to the next level, while students on the lower proficiency path are required to solve three problems of lower individual difficulty with identical target-rules to continue to the next level. This difference in the number of problems is compensated for by the length of the problems. Harder problems are longer and easier problems are shorter, allowing for similar numbers of opportunities to apply each target-rule. An example of the difference between proficiency sets in a given level is shown in Fig. 3, where students solve 3 lower proficiency problems in about 15 cumulative steps or 2 higher proficiency problems in about 15 cumulative steps. Individual problems in the lower proficiency set each require fewer rule applications than individual problems in the higher proficiency set; however, by giving an additional problem in the lower proficiency set, students receive more opportunity to practice in applying specific target-rules and problem solving skills.

In DT1, students were allowed to solve problems in any order, giving flexibility in how students chose to approach the assignment, and preventing a student from becoming stuck on any individual proof. Because the problem sets in DT2 are completely ordered instead of random access, DT2 allows students to temporarily skip problems within a level, to allow that flexibility. Depending on the state of progression in the current level, skipping a problem has several outcomes. Students who skip once in the higher proficiency set (with two problems) are given the next unsolved problem in the set. Skipping twice in the higher proficiency set will drop students to

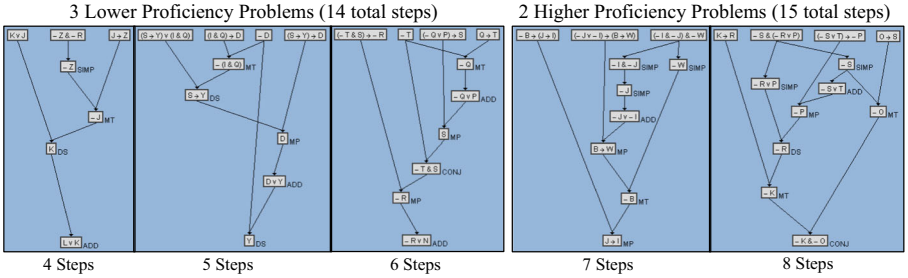

Fig. 3 An example of lower and higher proficiency set problems from DT2, Level 3. Lower proficiency proofs contain fewer rule application steps so students do one more problem to have more target-rule opportunities. The target-rules for this level are Modus Tollens, Addition, Conjunction (high priority) and Modus Ponens, Disjunctive Syllogism, Simplification (low priority)

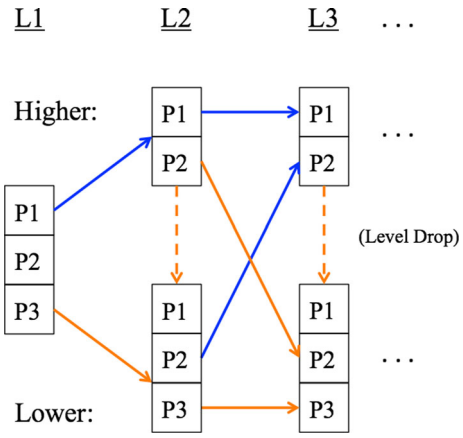

Fig. 4 DT2 path progression. At each level, students are evaluated and provided either the higher or lower proficiency problem sets. Students can also be switched from the higher to lower proficiency set within a level

the lower proficiency problem set in the same level, regardless of how many problems they solved in the higher proficiency set (see Fig. 4). Students in the lower proficiency set (with three problems) who skip a given problem are first offered an alternate version of the same problem with different ordering of required rule applications in the proof, before moving them to the next unsolved problem in the set (see Table 5 for an example of a problem and its given alternate). This gives students who get stuck on a problem the opportunity to approach the problem in a different manner. Students who repeatedly skip problems in the lower proficiency set will cycle through the unsolved problems in the set until three assigned problems have been solved, before moving on to the next level in the tutor.

Table 5 An example of regular and alternate problems from a lower proficiency set in DT2: Level 2 Problem 2 (top); Level 2 Problem 2-Alt (bottom)

#	Premise	Derivation
1	$(\neg K \vee L) \rightarrow (M \wedge N)$	Given
2	$K \rightarrow O$	Given
3	$\neg O$	Given
4	$\neg K$	2,3/*Modus Tollens*
5	$\neg K \vee L$	4/Addition
6	$M \wedge N$	1,5/*Modus Ponens*
7	N	6/Simplification
1	$(\neg O \vee L) \rightarrow (M \wedge \neg N)$	Given
2	$\neg O$	Given
3	$K \rightarrow N$	Given
4	$\neg O \vee L$	2/Addition
5	$M \wedge \neg N$	1,4/*Modus Ponens*
6	$\neg N$	5/Simplification
7	$\neg K$	3,6/*Modus Tollens*

Since there are more problems per level in the lower proficiency track than the higher proficiency track, we had to ensure that being dropped from the higher to lower path would not require the student to solve more problems than would be otherwise required. Therefore, the system maintains a maximum of three required solved problems to continue to the next level, taking into account any problems that may have been solved in the higher proficiency set, and ensures that all rules required for demonstration are presented to the student regardless of skipped status. The purpose of the skipped problem sub-system is to compensate for students who may have shown higher proficiency in a previous level, but have a harder time with the current set of problems. Students given the higher proficiency set are expected to have satisfactory mastery needed for the next set of problems, without the need for alternate problems that reorder rule applications. If students have difficulties with the harder set, they are given the opportunity to work through a greater number of easier problems in order to practice the required rules before moving on to the next level.

At the beginning of the next level, accumulated data from the student's performance in the tutor is used with the data-driven knowledge tracer (DDKT) presented in the next section to determine the proficiency set given to the student. Between the two proficiency sets, and the alternate problems offered in the lower proficiency set, there are 43 problems in the DT2 problem set between Levels 1–6. Students are required to solve at least 13 problems for full tutor completion if they are always placed on the higher proficiency path, and are required to solve at most 18 problems if they are always placed on the lower proficiency path.

DT2 saves student progress in the tutor, placing the student at the beginning of their most recent non-completed problem each time the student logs into the system, as opposed to DT1, where students select a problem to work on from a list. This prevents the ability to repeatedly complete the same problem, and ensures that the student stays on track to complete the tutor.

By limiting the number of problems students are required to solve and by not requiring specific mastery tests to progress to the next level, the DDML system ensures that students do not spend an unreasonable amount of time solving problems in the current level – allowing them to move forward in the tutor and be exposed to new concepts – yet still receive practice with problems that have difficulty relative to their performance. The difficulty of the problem sets that students are given is dependent on how their performance compares to past exemplars, which is described in the next section.

Assessment: DDKT Rule Score Updating

The instructor who developed Deep Thought chose proof problems that were deemed necessary for students to solve to demonstrate sufficient skill in propositional logic by the end of the course curriculum. It is hypothesized that students who have completed Deep Thought in the past have acquired a coherent skill set for proof problem solving and rule application. By splitting the DT1 problem set into levels, DT2 allows regular evaluation of new students compared to the standard of what exemplars were able to accomplish at corresponding points in the tutor. An overview of the threshold calculation system is shown in Fig. 5.

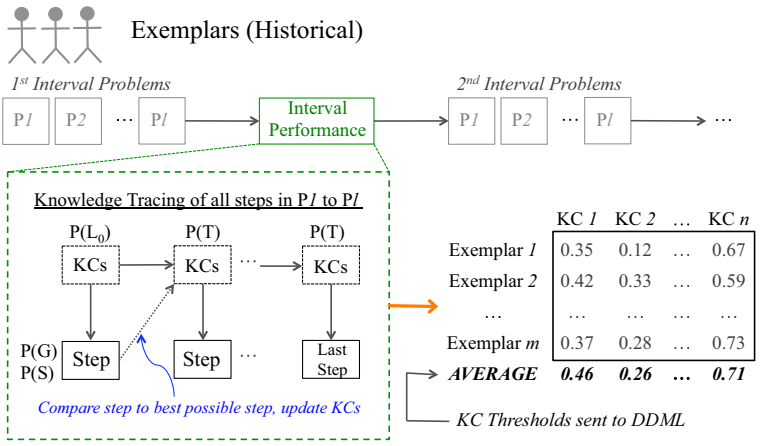

Fig. 5 The data-driven threshold builder. Knowledge components (KCs) for each exemplar are updated using action steps from an interval set of tutor problems. The KC score averages at each interval are used as thresholds in the DDML system

DT2 uses data-driven knowledge tracing (DDKT) of past and current students to evaluate student performance, and specifically the way students apply logic rules. Knowledge tracing and the parameters defined below were first outlined by Corbett and Anderson (1995). The knowledge tracing calculations require the following parameters to be defined; the learning value $p(L_0)$, which is the probability that a rule was learned prior to its application; the acquisition value $p(T)$, the probability that the rule will have been learned after the student has been given the opportunity to apply it; the guess value $p(G)$, the probability that the student will guess the correct application of the rule before it has been learned; and slip value $p(S)$, the probability

Table 6 List of rules required for completion of DT2

Rule	
MP	*Modus Ponens*
DS	Disjunctive Syllogism
SIMP	Simplification
MT	*Modus Tollens*
ADD	Addition
CONJ	Conjunction
HS	Hypothetical Syllogism
CD	Constructive Dilemma
DN	Double Negation
DEM	DeMorgan's
IMPL	Implication
TRANS	Transportation (Contrapositive)
EQUIV	Equivalence

that a student will make a mistake in applying the rule if it has been learned. For each student, a knowledge tracing (KT) score $ruleScore_i$ for each logical rule i in the tutor is created when a student first logs into DT2, and these scores are maintained and updated at each rule application made by the student. Table 6 shows the list of rules required for completion of the tutor. The $ruleScore_i$ for a given rule i is initialized with $p(L_0) = 0.01$, $p(T) = 0.01$, $p(G) = 0.3$, and $p(S) = 0.1$. After each observed apptempted application of rule i, $ruleScore_i$ is updated using Bayesian knowledge tracing equations for inference and prediction of individual skill knowledge (Eagle and Barnes 2012), shown below. Both of the equations are the sums of two probabilities; the posterior probability that rule i was already learned based on correct or incorrect application of the rule, and the probability that the rule will be learned if it had not already been learned. Equation (1) is used for rule i when that rule is correctly applied in the current state, reflecting the probability of the student knowing and correctly applying that rule. Equation (2) is used for rule i when that rule is not correctly applied in the current state. Applying a rule correctly in the context of the current problem shows evidence that the student recognizes the applicability of that rule, while an incorrect rule application implies lack of recognition of a better option.

$$p(L_n) = \frac{p(L_{n-1})(1 - p(S))}{p(L_{n-1})(1 - p(S)) + (1 - p(L_{n-1}))p(G)} + (p(1 - L_{n-1}))p(T) \quad (1)$$

$$p(L_n) = \frac{p(L_{n-1})p(S)}{p(L_{n-1})p(S) + (1 - p(L_{n-1}))(1 - p(G))} + (1 - p(L_{n-1}))p(T) \quad (2)$$

Assessment: Proficiency Determination

By the time the student has completed a level of the tutor, that student will have accumulated a set of rule scores for each rule i calculated based on their performance. These scores can then be used to determine the student's current level of mastery, as well as what set of problems to complete next. At the end of each level, the DDKT scores for each rule $ruleScore_i$ are compared to a threshold value for that same rule. This threshold value, $ruleThreshold_i$, was calculated using data-driven knowledge tracing of DT0 tutor logs from six 2009 Deductive Logic course sections via (1) and (2). Only students who completed the entire DT0 assignment ($n = 302$) were used for threshold calculation, since these exemplars demonstrated proficiency for proof problem solving using all required rules. DDKT student scores were computed for problems in DT0, and mapped to the corresponding levels in DT2. The scores from DT0 for each rule score were averaged at each of these break points and set as $ruleThreshold_i$. Students using DT2 are therefore judged to have proficiency on a rule based on how their performance compares to average student usage from previous use of Deep Thought by past exemplars.

For $i = 1 : n$

$$scoreSign_i = \mathbf{sign}(ruleScore_i - ruleThreshold_i) \quad (3)$$

For each student in DT2, every $rule_i$ is assigned a positive $scoreSign_i$ value if the rule score is above $ruleThreshold_i$, the prior student performance for students who

completed the whole tutor. Each $rule_i$ receives a negative $scoreSign_i$ value if the score is below the $ruleThreshold_i$, meaning that they have not yet shown the same level of proficiency in deciding which rules to use as past students who successfully completed all the problems in DT0. This positive or negative value is weighted based on the priority of rule i in the current level. Rule priority is determined by the set of rules deemed necessary by domain experts to best solve the current proof problem.

Rules with high priority (rules required for completion of the proof problems in that level) are weighted by 1. Rules with low priority (rules the students are not expected to demonstrate but may still be necessary for the problem solving process) are weighted by 0.5. Rules with 0 priority (rules that are not required for completion of the proof problems in that level) are weighted by 0. Low priority rule scores are included in calculation of student proficiency because it is assumed that every legal or illegal rule application, regardless of level rule priority, is an indication of a student's knowledge (or lack thereof), and should be considered when determining a student's overall proof-solving proficiency at any given point.

$$Proficiency = \mathbf{sign}\left[\sum_{i=rule_0}^{rule_n} scoreSign_i \times \begin{cases} 1 & \text{if priority}_i = \text{high} \\ 0.5 & \text{if priority}_i = \text{low} \\ 0 & \text{if priority}_i = 0 \end{cases}\right] \tag{4}$$

The weighted $scoreSign_i$ values are summed, and the sign of the sum determines whether a student is sent to the higher proficiency path ($Proficiency = +$ value) or the lower proficiency path ($Proficiency = -$ value) in the next level, as shown above.

The DDML system measures proof-solving proficiency by aggregating the scores of individual tutor actions. This focus on individual student actions rather than domain-based knowledge concepts is what allows the DDML system to be domain independent. The DDKT method used in the DDML system can be applied to any subject matter where actions taken by students can be prioritized in the context of the current problem set. The list of rules or operations required for the tutor are dependent by the domain; all other operations are only dependent on prior student data.

Deep Thought 3: Data-Driven Hints and Worked Examples

Deep Thought was redesigned and rebuilt between 2013 and 2014. In addition to expanding the Deep Thought window and removing menus to improve accessibility of options and relevant information, the redesign includes the ability to develop and manage system features based on course and research requirements, and provides comprehensive step-level logs. Deep Thought 3 (also referred to as DT3) is designed to be a modular instructional and research tool, where features or modules can be included or excluded for particular classroom or research settings. See Fig. 6 for an example of the interface.

DT3's new features include a traditional text-based list of proof steps that reflect the proof state in the graphical window, the ability to work a problem indirectly by

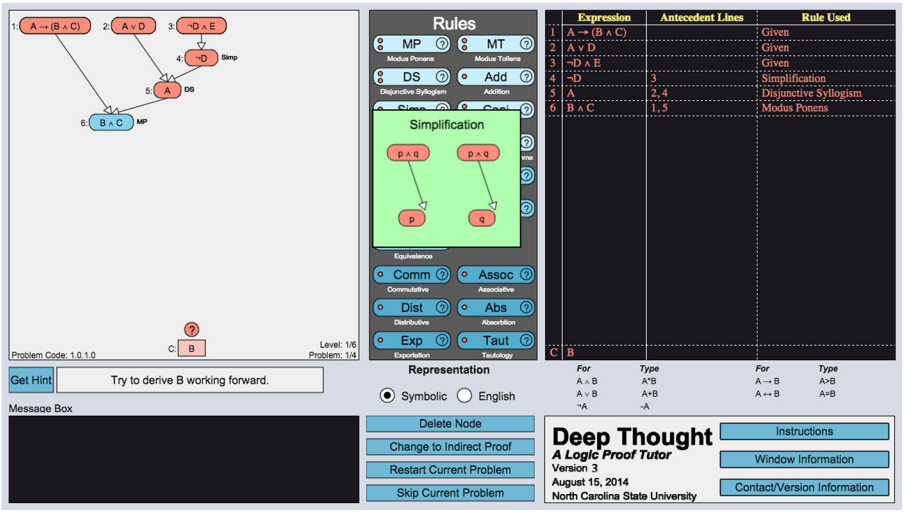

Fig. 6 A screen capture of the Deep Thought tutor redesign, mirroring the proof-state in Fig. 1

negating the conclusion and searching for a logical contradiction, and more detailed feedback. DT3 also includes support for hints while students work backwards, which was facilitated by a complete redesign of the data logs and hint functionality. DT3 is also fully usable with mobile-web devices.

DT3 was specifically designed to add on-demand hints and worked examples to the DDML problem set. As with DT2, the problem set consists of 6 strictly ordered levels, each split into a higher proficiency track with a lower number of more complex problems, and a lower proficiency track with a greater number of less complex problems. Evaluation of student performance occurs at the beginning of each level of problems, with the first level consisting of common problems to collect initial data for assessment.

DT3 has an updated DDML system that allows hints, and a hint policy that selects hint types based on instructor parameters. The hypothesis after testing and evaluating DT2 was that the addition of hint systems would significantly improve student retention. DT3's hint policy can present problems with no hints, next-step hints, or worked examples.

DT3 has the option of selecting any combination of no-hint, on-demand next-step hint, and worked example problems as determined by the course instructor. For the experiment in this study, two versions of DT3 were created. The first version is identical to the set-up of DT2, using the same problem set, the same number of low- and high-proficiency track problems, and no on-demand hints (DT3-NoHint). The second version adds on-demand hints to the DT2 problem set, as well as worked examples of the same problems, conditionally chosen as described below (DT3-Random). To ensure the worked examples did not greatly reduce the amount of practice students

received solving problems, this version of DT3-Random adds an additional problem in each level (one in each proficiency track), required to solve, and with no on-demand hints. This also provides a built-in post-test on each level of the tutor.

Next-Step Hints

As with DT1, hints were created using Hint Factory. If hints are available for a given proof problem, clicking the "Get Hint" button with the mouse pointer will result in one of four possible feedback responses for any individual proof-state, displayed on successive "Get Hint" button presses for that same step. The hints for DT3-Random were created using student log data from DT2, and are available for every problem in the DT3-Random data set, with the exception of additional problems added because of worked examples. Instructors may select which problems will have available hints for students; this is helpful for ensuring we periodically collect diverse data that is not influenced by hint usage for problems.

Worked Examples

Worked examples incorporated into intelligent tutoring systems have the potential to increase learning gains and learning efficiency (Hilbert and Renkl 2009; McLaren et al. 2008). Figure 7 shows a worked example in DT3. Students are given an entire

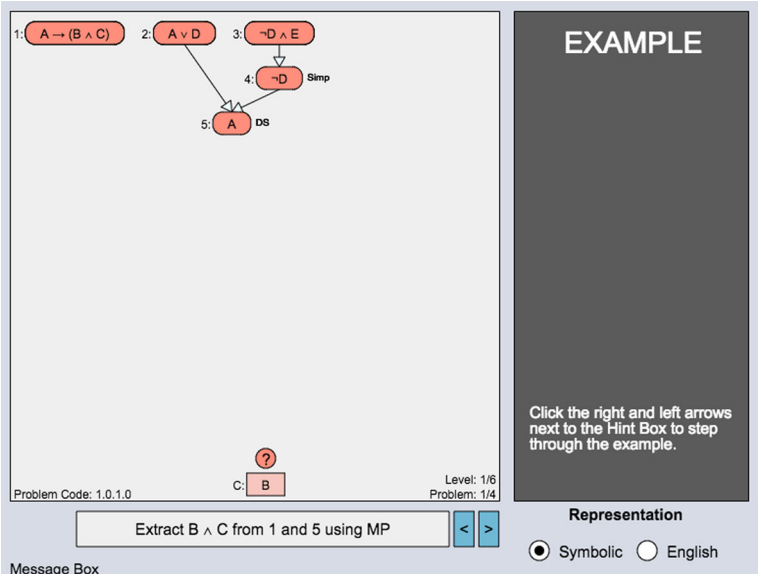

Fig. 7 A worked example for the same problem in Fig. 6. A problem step-state is displayed on the screen, with an annotation in the box below describing the next step. Students can move between worked example steps using the arrow buttons next to the annotation

solution to a proof problem step-by-step, with each step annotated with the next action. Students can move backward and forward between steps at leisure.

Once a student attempts to continue past the final example step, the next problem in the level is given. Each problem in the DT3-Random data set has a corresponding worked example, which was created from past student proof solutions. For each problem, the solution chosen for the worked example was the one that had the lowest number of steps, and included all the primary expected rules for the given problem. In the event that a past student solution did not meet these criteria, an expert solution was used for the worked example. The annotations for the worked examples were procedurally generated from the worked example steps.

Hint Policy

To gather information on how student behavior in-tutor is affected by worked examples, the hint-based version of DT3-Random randomly assigns the next problem selected by the DDML as either a problem with on-demand hints to solve, or as a worked example of that same problem. To ensure students receive enough practice solving problems, and are assigned enough worked examples to collect sufficient data, a hint policy was created to control the balance of instruction through example and the opportunity for proof-solving practice.

An additional problem was added to each level in the problem set (one in each proficiency track). These problems were designed to mirror the rules and problem solving strategies as the other problems in the respective levels and proficiency tracks. Unlike the other problems in the DT3-Random data set, the additional problems are presented without hints, and with no corresponding worked example. The purpose of these problems is three-fold. The first purpose is to ensure that students receive sufficient practice solving proof problems. With some of the problems presented as worked examples, students will solve fewer problems over the course of the tutor than students using DT2. The addition of a non-example problem adds an additional problem to solve. The second purpose of the problems is to act as a demonstration of learning. Worked examples have shown to be effective in improving learning in previous work (Sweller and Cooper 1985). However, Deep Thought requires demonstration of proof-solving ability in order to assess learning between levels.

As bottom-out hints are themselves a type of worked example (Shih et al. 2008), which DT3-Random allows for with next-step hints, the inclusion of a required no-hint proof problem forces students to demonstrate what they have learned before allowing them to continue forward. Third, for the purpose of understanding the difference in learning gains between problem solving and worked examples for a later study, a comparison of performance of the same problem set with and without worked examples is needed. Within a level, the number of worked examples is randomly decided. By having a problem required for all students, the effect of worked examples can be determined by studying how the number of worked examples presented in a level affect student performance metrics in the additional "post-test" problem (such as time taken, rules applied, and any errors made).

For the hint-based version of DT3, there are a total of three problems in the high proficiency track in each level, four problems in the initial level (Level 1) and four problems in the low proficiency track in each level. The hint policy ensures that for each level, students will receive at least one, and no more than two, worked examples and solvable problems in the high proficiency track, and at least two, and no more than three, worked examples and solvable problems in the initial level (Level 1) and low proficiency track.

Because of the reduction in the number of problems required to complete each level (1-2 for the high proficiency track and 2-3 for the low proficiency track), the rule threshold scores used for assessment by the DDML system will not be effective for students who solve fewer problems than in DT2. Since there is less opportunity to demonstrate rule application, individual rule scores for a student solving fewer problems will be lower than the threshold for that level, even with ideal proof solving strategy. To counteract this effect, the rule scores for each student are updated when students receive a worked example with the rules and number of rule application in the worked example, as if the student had solved the problem themselves. Shih et al. (2008) found that students who go through a worked example learn as much as if they had worked through the problem themselves. This is expected to put most students on the high proficiency path in each new level, assuming their cumulative performance up to that point is satisfactory. However, as with DT2, students have the option of skipping problems in the high proficiency track if they are having difficulty, resulting in a drop to the low proficiency path in the same level. Also as with DT2, the number of problems and worked examples required for a single level are maintained, regardless of a level drop.

Deep Thought 4: Data-Driven Proficiency Calculation

The DDML system in DT3 uses expert-decided priorities for each of the rule application actions when calculating a student's proficiency, so any new problems or levels added to the system would require expert involvement to determine which rules were prioritized in each new or altered level. Deep Thought 4 (referred to as DT4) was created by extending DT3 to replace the expert authored mastery settings with a data-driven system. We created a data-driven problem profiler (DDPP), which uses the clustering of exemplar scores at each level interval for each rule, weighted by primary component importance, to classify exemplars into types of student progress through the tutor (see Fig. 8). New student performance is compared to exemplars of the same classification, accounting for different problem solving strategies. This is expected to improve proficiency classification of students who are using Deep Thought over the expert-authored system in DT2. In addition, this would allow Deep Thought to be used in other classrooms where the pedagogical method and problem-solving ability of the class may be disparate from the current exemplar data from Deep Thought. This would eliminate the potential "cold start" in using the system in a different classroom setting before data from that class could be gathered. Other work has involved attempting to solve the "cold start" problem in data-driven feedback generation (Rivers and Koedinger 2015), but not when determining student proficiency.

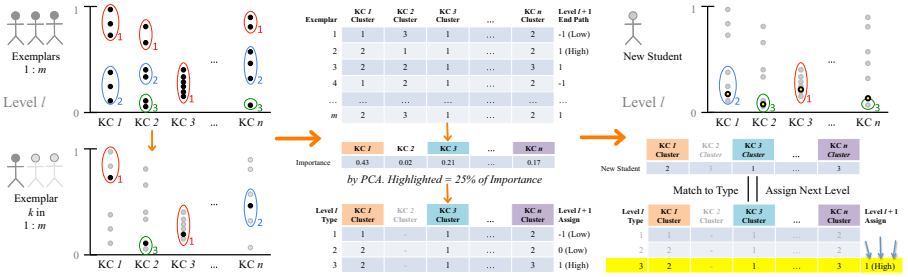

Fig. 8 The data-driven proficiency profiler. (*Left*) At each level interval, exemplar KC scores are clustered, and exemplars are assigned a cluster for each KC Score. (*Center*) KCs that make up 25 % of importance in the current level are used to assign exemplars to types. (*Right*) New student scores are assigned to clusters, and compared to existing types to determine next level path

In addition to the hint types from DT3, high-level hints were available for use by the hint policy.

Cluster-Based Classification

Cluster-based classification has several advantages when applied to data-driven tutoring. New educational technologies may reveal unexpected learning behaviors, which may not yet be incorporated in expert-decided classification processes. For example, Kizilcec et al. (2013) clustered MOOC learners into different engagement trajectories, and revealed several trajectories that are not acknowledged by MOOC designers. Second, experts classify with their perceptions on the average students' performance (Perez et al. 2012; Watering and Rijt 2006). This perception may be different from the actual participants group. Cluster-based classification, however, is able to classify and update classifications based on actual student behaviors.

Moreover, previous studies have shown that personalized tutoring based on cluster-based classification not only helps learning, but improves users' experience. Klasnja-Milicevic et al. (2011) gave students different recommendations on learning content based on their classified learning styles. As a result students who used hybrid recommendation features completed more learning sessions successfully, and perceived the tutor as more convenient. Despotovic-Zrakic et al. (2012) adapted different course-levels, learning materials, and content in Moodle, an e-learning platform, for students in different clusters. Results showed that students with adapted course design had better learning gain, and more positive attitude towards the course.

However, the majority of previous work clustered students solely on their overall performance statistics. In contrast, our method clusters students based on their application of specific knowledge components throughout the tutor.

Data-Driven Problem Profiler (DDPP)

In the DDML system used in DT2, between-level assessment is performed by comparing rule-application scores for an individual student to threshold values for those rules, and then weighing the positive or negative differences by rule-priorities for

the given level. These rule-priorities are determined by domain experts for DT2 and DT3 NoHint and Random. The priorities are weighted at three discrete levels: full priority (1), half priority (0.5), and no priority (0). At the conclusion of the previous semester's use of the tutor, enough performance data from DT3-NoHint and DT-Random exemplars (students who successfully completed the entire tutor) was available to determine the importance of required rules in a level for improved student performance. This data was used to develop the DDPP system shown in Fig. 8. We used unsupervised learning methods to learn different strategies for solving particular problems. Clustering only the exemplars allows us to determine successful problem-solving strategies, as in the problem-solving strategies that lead to a student successfully completing the tutor, while avoiding the least successful strategies. This allows us to direct new students along a path that will most likely lead to those students successfully completing the tutor, as well as preventing the system from suggesting a worse path simply because more students overall used a particular problem-solving method.

Expert weighting was replaced by principal component analysis (PCA) of the frequency of the rules used for each exemplar for each level, accounting for 95 % variance of the results. PCA is typically used to reduce the dimensionality of a data set by determining the most influential factors in the data set. The influence of a given factor is based on how much that factor contributes to the variability in the data. In our analysis, we use PCA analysis on the Deep Thought dataset to determine which rules were most important to success in the tutor at each level. Rules which account for 25 % of importance and higher are considered most important for completing a level. This percentage was determined through incremental testing at 5 % intervals. From the results generated, 25 % is the percentage that maximized accuracy. For each rule, its PCA importance value is the new weight for that rule score. Unlike expert authored weights, these rule score weights are based on each rule's importance as determined by the data. New students in the system will have their KC scores labeled by the clusters their scores fall into (distance to closest centroid). New students will then be compared to exemplars to find which Type (cluster based on problem solving strategy) the new student has the most clusters in common. The new students will then be assigned the proficiency track in the next level in Deep Thought of that Type. In the case of multiple matching Types, the new students will be assigned the Type that is weighted higher by frequency. To classify a student an existing Type, they must match at least half the clusters of any one Type. Otherwise, the student is considered to have no match, and will be given the low proficiency track in the next level.

In the original system, the student proficiency was determined based on one set of rule thresholds and a set of expert authored weights. However this means the system did not take varying student strategies into account. The data is based on students who completed the tutor, who have therefore shown the level of mastery required to successfully complete Deep Thought, but the scores are averaged over all the students at the end of each level. By taking the average of these student scores at this point, we are still assuming only one successful problem solving strategy for completing each level in the tutor. However while most strategies might be the same for earlier levels, there may be a variety of strategies in later levels that can still result in successful completion.

The DDPP method accounts for that possible variety in problem solving methods. In using an unsupervised clustering method, we are able to account for different clusters while not knowing how many clusters there are. By clustering the scores, we are essentially looking for different strategies that utilize particular rules and determining these strategies based on the student data. Once we determine which strategy a new student is utilizing, we can look to the data again to see how exemplars who employed a similar strategy were placed in the tutor and how they performed, thus determining the best way for the tutor to react to that particular student. Finally, using PCA based weights allows us to weigh rule scores based on rule importance as determined by previous students who completed the tutor, rather than expert determination.

Studies and Results

Throughout its development, Deep Thought in each of its iterations was tested in Philosophy and Computer Science classroom settings as a mandatory homework assignment in order to determine the effectiveness of each version and its newly added data-driven features. The study conditions, results, and conclusions for each version of Deep Thought are discussed in the following subsections (Table 7).

Deep Thought 2

Study

DT2 was used as a mandatory homework assignment by students in a philosophy deductive logic course in the Spring 2013 semester (DT2-DDML group, $n = 47$). Students were allowed to work through the problem sets at their own pace for the entire 15-week semester. Problem Levels 1–6 were assigned for full completion of the tutor, totalling 13–18 problems depending on proficiency path progression. Class credit was given at completion of each level. Data from DT2 was compared to data collected in two prior semesters (Spring and Fall 2009) of the same philosophy course using DT1 and DT0, respectively. Both courses were taught by the same instructor as the DT2-DDML group, and were assigned the same 13 problems in DT0 and DT1 for full completion of the tutor. The problems in DT0 and DT1 cover the three main logical rule sets (basic implication rules; advanced implication rules; implication and equivalence rules), and correspond to problems in DT2. The Spring 2009 students used DT1 with on-demand hints system (DT1-Hint group, $n = 48$). The Fall 2009

Table 7 Experiment conditions for each version of Deep Thought

Deep Thought version	Conditions
Deep Thought 2	DDML in a philosophy deductive logic course
Deep Thought 3	Hints and worked examples in two computer science courses
Deep Thought 4	Comparison of DDPP to manual path prediction

students used the original unaltered DT0 (DT0-Control group, $n = 43$) with no hints. Class credit was given at the completion of each assigned problem.

The DT2-DDML group were only provided hints in problems that were identical to hint-problems from DT1. In total there were 8 problems with available hints out of the 43 problems in the DT2-DDML group problem set, spread between proficiency paths and alternate problems. Considering the cumulative probabilities of encountering a hint problem in each level on each path, the odds of a student encountering more than three of the hint-problems in the tutor was less than 1 in 32. No significant useage of hints were found. For every problem that had a hint available, on average less than 1 student requested any hints. Therefore, the DDML group is considered a non-hint group.

Results: Data-Driven Problem Selection Reduces Dropout and Improves Tutor Completion

Percentage completion of the tutor is used as a measure of overall success for each group of students; however, because of differences in assignment structure, calculation of percent completion for DT0/DT1 is not the same as DT2. For the DT1-Hint and DT0-Control group, class credit was given per-completed problem. Because problem selection was un-ordered, and the number of problems assigned per problem set was not uniform, the percentage completion of the total tutor is calculated as:

$$pctComplete = \frac{\text{assigned problems solved}}{\text{total assigned problems}} \tag{5}$$

For the DT2-DDML group, class credit was given per-level completed. In DT2, the number of problems necessary for problem completion is dependent on individual student path progression. Students who remain strictly in the high proficiency path work a total of 13 problems, and students who remain strictly in the low proficiency path work a total of 18 problems; students who move between proficiency paths can work any number of problems between the two limits. Therefore, since path-progression for an individual student is not known unless the entire assignment is completed, percentage completion is calculated as:

$$pctComplete = \frac{\text{assigned levels completed}}{\text{total assigned levels}} \tag{6}$$

Table 8(a) shows the number of total assigned problems solved in tutor for the three groups. Students in the DT2-DDML group solved 13 problems on average – the minimum required for completion of the tutor – while students in the DT1-Hint group completed 8 assigned problems out of 13, and students in the DT0-Control group completed 6 out of 13.

Table 8(b) shows that students from the DT2-DDML group complete 33 % more of the tutor on average than the DT0-Control group, and 18 % more of the tutor than the DT1-Hint group, with lower variance in the final scores. The median score of 100 % is due to the fact that over half (55 %) of the DT2-DDML group completed the entire tutor (see Table 9 for tutor completion by group).

Table 8 (a) Number of total assigned problems solved in tutor for the three groups

(a)	DT2-DDML	DT1-Hint	DT0-Control
Mean	13.09 of 13–18	7.98 of 13	6.07 of 13
StDev	4.94	4.78	5.20

(b)	DT2-DDML	DT1-Hint	DT0-Control
Mean	79.79*	61.38	46.69
Median	100.0	61.54	46.15
StDev	29.88	36.78	40.02

(b) Percentage of tutor completion. An * indicates significance over the control

A one-way ANOVA test on percent completion difference was performed on tutorial completion across all three test groups, showing significance ($F(2, 120) = 7.559$, $p = 0.001$). A Tukey post-hoc comparison shows a significant improvement of the DT2-DDML group ($M = 0.80$, 95 % $CI[0.70, 0.90]$) over the DT0-Control group ($M = 0.51$, 95 % $CI[0.40, 0.62]$, $p < 0.001$, *Cohen's d* $= 0.84$). Although the DT2-DDML group had a higher mean percentage completion than the DT1-Hint group, the results were not significant ($p = 0.138$, *Cohen's d* $= 0.35$). However, this still indicates that DT2's data-driven knowledge tracing combined with individualized problem set selection can improve student percent completion as much as on-demand hints (DT1).

Deep Thought 3

Study

DT3-Random, DT3 with on-demand hints and data-driven worked examples, was implemented and tested in Fall 2014 to determine its effect on tutor completion, student dropout, and how the order of worked examples affects student performance. DT3-Random was used as a mandatory homework assignment by students in two computer science discrete mathematics courses taught by the same instructor (Worked Example (WE) group, $n = 261$). Problem Levels 1–6 were assigned for full completion of the tutor, totalling 6–18 problems depending on proficiency path progression and number of worked examples given. Class credit was given at completion of each level.

Data from the DT3-Random was compared to data collected from DT2 for tutor completion and dropout comparison, since DT2 is the only previous version of Deep

Table 9 Student completion of the tutor by group

Group	Completed	Dropped	Total
DT0-Control	8 (18.6 %)	35 (81.4 %)	43
DT1-Hint	15 (31.3 %)	33 (68.7 %)	48
DT2-DDML	26 (55.3 %)	21 (44.7 %)	47
Total	49 (35.5 %)	89 (64.5 %)	138

Dropped indicates that the student did not complete Deep Thought

Thought that utilizes the data-driven mastery learning system. Deductive logic is relevant to computer science and philosophy, and comparing the two classes gives an indication of how well the DDML translates to different topic areas. DT3-NoHint, DT3 without hints or worked examples, was used as a mandatory homework assignment by students in a philosophy deductive logic course in Spring 2013 (NoWE group, $n = 47$). The DT3-NoHint problem set was identical to DT3-Random with the exception of the additional problems. Problem Levels 1–6 were assigned for full completion of the tutor, totalling 13–18 problems depending on proficiency path progression. Class credit was given at completion of each level. DT3-Random also differs from DT3-NoHint in that it provides on-demand next step hints for each problem in the DT2 data set. However, since in the DT3-Random data set there are only 98 instances of students requesting hints out of 500,000 total actions, we consider the effect of hints on total student performance negligible, and these data points were removed for the worked example specific test conditions.

Results: Addition of Worked Examples improves Tutor Completion and Time in Tutor

Table 10 shows the number of problems solved, worked examples received, and total time spent in tutor for the DT3-Random and DT3-NoHint groups. Students in both groups solved the same number of problems in the tutor on average with no significant difference between them ($p = 0.327$, $power = 1$). However, the DT3-Random group spent 27 % less time in tutor on average than the NoWE, even with the added worked examples. This was marginally significant ($p = 0.063$).

We use percent completion as a measure of overall success for each group of students. As with the DT2 study in the previous section, percent completion is a measure of how far students progress through the tutor, on average. Table 11(a) shows the average percentage of tutor completion by group.

As before, student dropout is defined as the termination of tutor activity at any point in the tutor prior to completing all of the assigned problems. Table 11(b) summarizes the number of students who completed and dropped out of the tutor across both groups. From this data, it is evident that the worked examples are dramatically improving retention and tutor completion.

Our results show that students using Deep Thought with worked examples (DT3-Random) completed more of the assigned problems and are less likely to drop out. There is a statistically-significant difference between the groups in terms of percent

Table 10 The number of problems solved (not including worked examples), number of worked examples received, and total tutor time for the DT3-Random and DT3-NoHint groups

Group	# Solved problems			# Worked examples			Total tutor time (mins)		
	Mean	Median	StDev	Mean	Median	StDev	Mean	Median	StDev
DT3-Random	14.12	13	5.09	7.53	8	1.35	224.4	97.7	346.5
DT3-NoHint	13.08	13	4.94	–	–	–	307.2	244.4	263.4

Table 11 (a) Percentage of the tutor completed by group

(a)	Mean	StDev	(b)	Completed	Dropped	Total
DT3-Random	94.02*	21.07	DT3-Random	236 (90.4 %)	25 (9.6 %)	261
DT3-NoHint	79.79	29.88	DT3-NoHint	26 (55.3 %)	21 (44.7 %)	47
			Total	262	46	308

(b) Student completion of the tutor by group. Dropped indicates that the student did not complete Deep Thought. An * indicates significance

completion (one-way ANOVA: $F(2, 308) = 6.38$, $p = 0.014$). The average percentage of tutor completion was significantly greater for the WE group than for NoWE (94 % vs. 79.8 %), an improvement of 14.2 %. Thus adding worked examples to Deep Thought enabled students to complete more of the tutor.

Deep Thought 4

Comparison to Other Methods

The DDPP method is compared to results from the thresholds set to the average rule score and minimum rule score, weighted based on PCA scores. The average rule scores are the set of average scores for each rule in each level, while minimum scores are the smallest scores for each rule. Comparing the current DDML system with expert-decided rule priorities and average score or minimum score based proficiency calculation weighted by PCA score offers insight into the effect of introducing PCA based rule weighting to students' performance baseline on the prediction accuracy. From here, the methods will be distinguished as follows: the DDML method in DT2 will be referred to as MP (MP in this case would stand for Manual Proficiency determination, to distinguish it from the data-driven proficiency calculation in the DDPP); the proficiency calculation based on average scores will be referred to as AEP (for Average Exemplar Proficiency); the minimum score method will be referred to as MEP (for Minimum Exemplar Proficiency); finally, the DDPP calculation as CEP (for Clustered Exemplar Proficiency).

Study

This study was based on the population of students who used versions of Deep Thought with the unaltered DDML system (DT2) in order to compare the data-driven methods for proficiency calculation (DT2 from Spring 2013 $n = 47$, DT3-NoHint from the Fall 2014 philosophy course, $n = 47$).

The first stage of the experiment is to compare the results of calculating student proficiency using the CEP, MP, AEP, and MEP methods with existing data from DT2 and DT3-Random. Exemplars will be chosen with the requirement that the students should have completed any given level of Deep Thought on the proficiency track that they were assigned. To ensure no other additional factors, only data will be used from

Table 12 Path prediction accuracy of the original DDML system (MP), the DDPP system (CEP), average score assessment (AEP), and minimum score assessment (MEP), for both Philosophy and CS students at the end of each level

	MP	CEP	AEP	MEP
Lvl 1	**88.2 %**	65.8 %	65.8 %	35.5 %
Lvl 2	**85.5 %**	67.1 %	73.7 %	18.4 %
Lvl 3	**75.0 %**	63.2 %	60.5 %	69.7 %
Lvl 4	**78.9 %**	61.8 %	64.5 %	40.8 %
Lvl 5	**78.9 %**	64.5 %	59.2 %	59.2 %

Bold values are the maximum percentage accuracy of each system, per level

students using the unaltered DDML system (DT2 Spring 2013 $n = 47$, DT3-NoHint Fall 2014 $n = 47$). The CEP, AEP, and MEP was run on the student actions from these exemplars, and the results were compared to the actual students results with 10-fold cross validation.

Evaluation: Path-Prediction Accuracy

The DDML system allowed for students to change proficiency paths mid-level in a situation where they found the problems assigned to them too difficult. The accuracy of the original system can therefore be determined by how often students who completed the entire tutor changed proficiency paths in the middle of a level throughout the course of the tutor. This calculation tells us, for students who completed the entire tutor, how well the original system predicted the paths for them to continue on. If a student had to change proficiency paths, it is likely the system assigned them to the wrong proficiency path. This serves as a basis of performance comparison between the DDPP and the proficiency calculation in the original DDML system.

Results: Data-Driven Proficiency Determination predicts Best Paths as well as Expert-Based Determination

Table 12 shows the path prediction accuracy of the MP, CEP, AEP, and MEP assessments across all the students in the Philosophy and CS courses. The MP accuracy

Table 13 Path prediction accuracy of the original DDML system (MP), the DDPP system (CEP), average score assessment (AEP), and minimum score assessment (MEP), for Philosophy students

	MP	CEP	AEP	MEP
Lvl 1	76.9 %	**80.8 %**	**80.8 %**	23.1 %
Lvl 2	65.4 %	69.2 %	**76.9 %**	19.2 %
Lvl 3	50.0 %	**84.6 %**	80.8 %	38.5 %
Lvl 4	65.4 %	69.2 %	**76.9 %**	30.8 %
Lvl 5	**53.8 %**	46.2 %	46.2 %	26.9 %

Bold values are the maximum percentage accuracy of each system, per level

was very high, ranging from 75 % at the end of level 3 to 88.2 % at the end of level 1. CEP was somewhat accurate, ranging from 61.8 % path prediction accuracy at the level 4 interval to 67.1 % path prediction accuracy at level 2. While these accuracies are not nearly as high as in the original system, they are acceptable considering that, unlike the original system, path prediction in the DDPP is entirely data-driven. It should also be noted that the DDPP was more consistent in its accuracy, only varying by at most 5 % between levels (in comparison to the original DDML system, which ranged in accuracy by 9.3 %).

Overall the original system (MP) predicted paths more accurately than the CEP, AEP, or MEP methods across all levels. The MEP method was least accurate across all levels. This makes sense considering that the MEP method is based on the minimum thresholds, so several students were assigned to the higher proficiency track when it is likely they should have been assigned to the lower proficiency track. In comparison to the AEP method, the CEP was more accurate than the average method on level 3, and level 5. The CEP was equally as accurate as the average method at the end of level 1, and less accurate at the end of levels 2 and 4. However, some of the lower accuracy was likely due to the distribution of exemplars across the two courses. Recall that the CS students made up a higher proportion of the analyzed exemplars than the Philosophy students.

Analyzing the path prediction accuracy by the individual course reveals more detail on the path prediction accuracy. In the case of the Philosophy students, where proportionally fewer of the students were selected as exemplars, the CEP method was more accurate than the original system on every set of levels except for level 5 (see Table 13). In comparison to the average calculation method, the CEP method was only more accurate on level 3. In level 1 and 5, the CEP was as accurate as the AEP method, and in levels 2 and 4 the CEP was less accurate.

In the CS course, where proportionally more of the students were selected as exemplars, not only was the original system far more accurate than it was for the entire set of students overall, but the DDPP path accuracy was much worse in some places. However, in comparison to the average method, the DDPP method was only less accurate in level 2. In all other levels the DDPP was either more accurate than the average method (levels 3 and 5) or equally as accurate (levels 1 and 4) (Table 14).

Table 14 Path prediction accuracy of the original DDML system (MP), the DDPP system (CEP), average score assessment (AEP), and minimum score assessment (MEP), for Computer Science students

	MP	CEP	AEP	MEP
Lvl 1	**94.0 %**	58.0 %	58.0 %	42.0 %
Lvl 2	**96.0 %**	66.0 %	72.0 %	18.0 %
Lvl 3	**88.0 %**	52.0 %	50.0 %	86.0 %
Lvl 4	**86.0 %**	58.0 %	58.0 %	46.0 %
Lvl 5	**92.0 %**	74.0 %	66.0 %	76.0 %

Bold values are the maximum percentage accuracy of each system, per level

Discussion and Conclusion

Each version of Deep Thought was augmented with data-driven, intelligent features in order to improve tutor performance. This improvement is evident in the results of each study. In DT1, DT2, and DT3 tutor performance was determined principally based on changes to the tutor completion rate of previous classes. With each augmentation to Deep Thought, this rate was steadily improved.

DT1 added on-demand data-driven hints to DT0, resulting in a logic tutor where students could select problems at will and prompt the system for hints if needed. The addition of on-demand hints improved tutor retention by 300 % and reduced the time it took for students to complete the tutor by 55 %, as shown in previous studies (Eagle and Barnes 2014a, b). This indicated that the addition of intelligent data-driven feedback could improve learning gains for students in a logic proof tutor. However, students using DT1 were still only completing 61 % of the tutor on average. This was due to the undirected nature of problem selection in DT1; students could still select any problem to solve in any order, whether the problem fit their current proficiency level or not. As a result, students who selected a problem that was too difficult for them, or incorporated concepts they had not been exposed to yet, would have been stuck on that problem. This problem existed in DT0 as well, as it was also undirected in its problem assignments; however based on the improvement to tutor completion (47 % in DT0, 61 % in DT1), it is clear that the addition of hints in DT1 did somewhat improve the effect this problem had on tutor completion.

Problem selection in DT2 was changed so that the problems students solved were determined by the tutor via the DDML system rather than the students themselves. As a result, students were given problems to solve based on the skills they possessed and their current proficiency level. This change further improved the tutor completion percentage. On average students using DT2 completed 79.9 % of the tutor, an 18 % improvement over the average tutor completion for students using DT1. As with the addition of hints, adding intelligent problem selection based on student proficiency and skill level improved the likelihood of students completing a higher percentage of the tutor and being exposed to more logic concepts than the previous group of students using DT1.

DT3 added on-demand hints and worked examples to DT2, resulting in a tutor that showed hints and worked examples in addition to intelligently selecting problems for the student to solve. DT3 was also the earliest opportunity to test Deep Thought in a Computer Science class, whereas previous versions of Deep Thought were tested in Philosophy Deductive Logic courses. Average tutor completion in DT3 with worked examples and hints increased by 15 % as a result of these changes (79.9 % in DT2, 94 % in DT3). Additionally, students who used DT3 with worked examples spent less time in the tutor on average by 27 %. This indicates that students who used DT3 not only completed more of the tutor, but did so in much less time than students using DT2.

As shown in the results, as more aspects of Deep Thought were made to be intelligently data driven students were able to complete a higher percentage of Deep

Thought in less time. With this in mind, the DDPP was developed for DT4 so that the student proficiency calculation could be intelligently data driven rather than based on expert determination. In this case, DT4 with the DDPP was evaluated using previous student data rather than a new classroom condition. There were still significant results in terms of improving how accurately the system could determine student proficiency based on previous data, in this case via clustering exemplars, based on path prediction accuracy. The DDPP in its current form could not predict student proficiency more accurately than proficiency based on expert determination overall. However, when examining path prediction accuracy for the Philosophy and Computer Science classes separately, there were significant differences in how accurately the DDPP predicted student proficiency. The original proficiency calculation based on expert determination was far more accurate for Computer Science students than for Philosophy students, while the DDPP predicted student proficiency more accurately for Philosophy students. This difference was due to the fact that there were more Computer Science students chosen as exemplars than Philosophy students, and there was a higher variety of problem-solving methods as a result. Deep Thought as a fully-intelligent logic tutor therefore has a very different effect on tutor performance based on the student population and the previous data, and further study is necessary to quantify these differences and use them to improve Deep Thought as an intelligent logic tutor in other classroom conditions. One current limitation to the DDML and the DDPP is that student assessment is only done at level end as opposed to problem end. This results in the student only getting two binary options for any level, when in reality they should be able to move to different proficiency paths within a level. However, adding this extra process to the end of every problem would decrease the extent to which Deep Thought could scale. As is, the scalability in Deep Thought is limited only by the problem set. This article described the development of the logic tutor Deep Thought into a fully intelligent, data-driven tutor that steadily improved tutor completion and time in tutor. We also described each intelligent component, and how these components worked to improve student experience in the tutor. As Deep Thought was improved to be more intelligent, tutor completion improved overall from 47 % in DT0, then 61 % in DT1, to 79.9 % in DT2, and finally to 94 % in DT3. While the addition of data-driven student proficiency calculation in the DDPP did not improve how accurately student proficiency was calculated in DT4, its evaluation led to additional insights into the effects of student population and previous data on Deep Thought's potential learning gains. Future work will involve using additional student data to further improve on Deep Thought and realize its potential as an intelligent, fully data-driven logic tutor. Once this potential is realized, Deep Thought could be used to improve how deductive logic is taught to Computer Science students.

Acknowledgments This material is based on work supported by the National Science Foundation under Grants 1432156 and 0845997. Thanks to Dr. Amanda Cohen Mostafavi for assistance in editing of the manuscript.

References

Barnes, T., & Stamper, J. (2008). Toward automatic hint generation for logic proof tutoring using historical student data. In *Proceedings of the thirteenth international conference on intelligent tutoring systems (ITS '08)* (pp. 373–382).

Barnes, T., Stamper, J., Lehmann, L., & Croy, M.J. (2008). A pilot study on logic proof tutoring using hints generated from historical student data. In *Proceedings of the first international conference on educational data mining (EDM '08)* (pp. 197–201).

Croy, M.J. (2000). Problem solving, working backwards, and graphic proof representation. *Teaching Philosophy*, *23*(2), 169–187.

Corbett, A.T., & Anderson, J.R. (1995). Knowledge Tracing: Modeling the Acquisition of Procedural Knowledge. *User Modeling and User-Adapted Interaction*, *4*, 253–278.

Despotovic-Zrakic, M., Markovic, A., Bogdanovic, Z., Barac, D., & Krco, S. (2012). Providing adaptivity in Moodle LMS courses. *Educational Technology Society*, *15*(1), 326–338.

Eagle, M., & Barnes, T. (2012). Data-Driven Method for assessing Skill-Opportunity recognition in open procedural problem solving environments. In *Proceedings of the fifteenth international conference on Intelligent Tutoring Systems (ITS '12)* (pp. 615–617).

Eagle, M., & Barnes, T. (2014a). Modeling student dropout in tutoring systems. In *Proceedings of the twelfth international conference on Intelligent Tutoring Systems (ITS '14)* (pp. 676–678).

Eagle, M., & Barnes, T. (2014b). Survival analysis on duration data in intelligent tutors. In *Proceedings of the twelfth international conference on intelligent tutoring systems (ITS '14)* (pp. 178–187).

Ehle, A., Hundeshagen, N., & Lange, M. (2015). The sequent calculus trainer - helping students to correctly construct proofs. In *Proceedings of the fourth international conference on tools for teaching logic (TTL '15)* (pp. 35–44).

Elmadani, M., Mathews, M., & Mitrovic, A. (2012). Data-driven misconception discovery in constraint-based intelligent tutoring systems. In *Workshop proceedings of the twentieth International Conference on Computers in Education (ICCE '12)*.

Fancsali, S.E. (2014). Causal discovery with models: behavior, affect, and learning in cognitive tutor algebra. In *Proceedings of the seventh international conference on educational data mining (EDM '14)* (pp. 28–35).

Gluz, J.C., Penteado, F., Mossmann, M., Gomes, L., & Vicari, R. (2014). A student model for teaching natural deduction based on a prover that mimics student reasoning. In *Proceedings of the twelfth international conference on intelligent tutoring systems (ITS '14)* (pp. 482–489).

Gonzalez-Brenes, J.P., & Mostow, J. (2013). What and when do students learn? Fully data-driven joint estimation of cognitive and student models. In *Proceedings of the sixth international conference on educational data mining (EDM '13)* (pp. 236–239).

Hilbert, T.S., & Renkl, A. (2009). Learning how to use a computer-based concept-mapping tool: self-explaining examples helps. *Computers in Human Behavior*, *25*(2), 267–274.

Kizilcec, R.F., Piech, C., & Schneider, E. (2013). Deconstructing disengagement: analyzing learner subpopulations in massive open online courses. In *Proceedings of the third international conference on Learning Analytics and Knowledge. (LAK '13)* (pp. 170–179).

Klasnja-Milicevic, A., Vesin, B., Ivanovic, M., & Budimac, Z. (2011). E-learning personalization based on hybrid recommendation strategy and learning style identification. *Computers Education*, *56*(3), 885–899.

Lee, J., & Brunskill, E. (2012). The impact on individualizing student models on necessary practice opportunities. In *Proceedings of the fifth international conference on Educational Data Mining (EDM '12)* (pp. 118–125).

Lodder, J., Heeren, B., & Jeuring, J. (2015). A pilot study of the use of LogEx, lessons learned. In *Proceedings of the fourth international conference on tools for teaching logic (TTL '15)* (pp. 94–100).

Lukins, S., Levicki, A., & Burg, J. (2002). A tutorial program for propositional logic with human/computer interactive learning. *ACM SIGCSE Bulletin*, *34*(1), 381–385. New York, NY: ACM.

Ma, W., Adesope, O., Nesbit, J.C., & Liu, Q. (2014). Intelligent tutoring systems and learning outcomes: a meta-analysis. *Journal of Educational Psychology*, *106*(4), 901–918.

McLaren, B., Lim, S., & Koedinger, K. (2008). When and how often should worked examples be given to students? New results and a summary of the current state of research. In *Proceedings of the thirtieth conference of the cognitive science society* (pp. 2176–2181).

Merceron, A., & Yacef, K. (2005). Educational data mining: a case study. In *Proceedings of the twelfth international conference on artificial intelligence in education (AIED '05)* (pp. 467–474).

Meyers, J.P. Jr. (1990). The central role of mathematical logic in computer science, In Miller, J.E., & Joyce, D.T. (Eds.) *Proceedings of the twenty-first SIGCSE technical symposium on computer science education (SIGCSE '90)* (pp. 22–26). New York, NY: ACM.

Murray, T., & Arroyo, I. (2002). Toward measuring and maintaining the zone of proximal development in adaptive instructional systems. In *Proceedings of the tenth international conference on intelligent tutoring systems (ITS '02)* (pp. 289–294).

Page, R.L. (2003). Software is discrete mathematics. In *Proceedings of the eighth ACM SIGPLAN international conference on functional programming (ICFP '03)* (pp. 79–86). New York, NY: ACM.

Perez, E.V., Santos, L.M.R., Perez, M.J.V., de Castro Fernandez, J.P., & Martin, R.G. (2012). Automatic classification of question difficulty level: teachers' estimation vs. students' perception. In *Proceedings of the IEEE frontiers in education conference* (pp. 1–5).

Rivers, K., & Koedinger, K.R. (2015). Data-driven hint generation in vast solution spaces: a self-improving python programming tutor. *International Journal of Artificial Intelligence in Education*, 1–28.

Ritter, S. (2007). Cognitive tutor: applied research in mathematics education. *Psychonomic Bulletin Review*, *14*(2), 249–255.

Shih, B., Koedinger, K.R., & Scheines, R. (2008). *A response time model for bottom-out hints as worked examples*.

Sieg, W. (2007). The AProS project: strategic thinking & computational logic. *Logic Journal of IGPL*, *15*(4), 359–368.

Steenbergen-Hu, S., & Cooper, H. (2014). A meta-analysis of the effectiveness of intelligent tutoring systems on college students' academic learning. *Journal of Educational Psychology*, *106*(2), 331.

Sweller, J., & 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.

VanLehn, K. (2006). The behavior of tutoring systems. *International Journal of Artificial Intelligence in Education*, *16*(2), 227–265.

VanLehn, K. (2011). The relative effectiveness of human tutoring, intelligent tutoring systems, and other tutoring systems. *Educational Psychologist*, *46*(4), 197–221.

VanLehn, K., Lynch, C., Schulze, K., Shapiro, J.A., Shelby, R., Taylor, L., Treacy, D., Weinstein, A., & Wintersgill, M. (2005). The Andes physics tutoring system: lessons learned. *International Journal of Artificial Intelligence in Education*, *15*(3), 147–203.

Watering, G., & Rijt, J. (2006). Teachers and students perceptions of assessments: a review and a study into the ability and accuracy of estimating the difficulty levels of assessment items. *Educational Research Review*, *1*(2), 133–147.