

Using the Student Model to Control Problem Difficulty

Joseph Beck, Mia Stern, and Beverly Park Woolf*

Department of Computer Science, University of Massachusetts, Amherst, MA, U.S.A.

Abstract. We have created a student model which dynamically collects information about a student's problem solving ability, acquisition of new topics and retention of earlier topics. This information is provided to the tutor and used to generate new problems at the appropriate level of difficulty and to provide customized hints and help. Formative evaluation of the tutor with 20 students provides evidence that the student model constructs problems at the correct level of difficulty. The problem generation technique is extensible for use in other problem-based domains. This paper describes the design and implementation of the student model and illustrates how the tutor adjusts the difficulty of a problem based on the student model.

1 Introduction

One-on-one human tutoring is much more effective than traditional classroom instruction (Bloom, 1984). We are building systems with the goal of emulating techniques that human tutors use, which can result in comparable learning gains (Shute et al., 1989). In order to do this, the tutor must maintain a representation of the student's knowledge and abilities as well as a set of teaching strategies. The student model keeps track of a student's *proficiency* on topics within the domain. Additionally, our model incorporates general factors of a student's ability, such as acquisition and retention. The model is then used for topic selection, problem generation, and hint selection. In this paper, we focus on the type of model we have implemented and how it is used to control the problem specifications. We also discuss the formative evaluation we have performed that provides preliminary evidence for the effectiveness of the student model.

The remainder of the paper is structured as follows. Section 2 discusses the domain. Section 3 describes the student model and section 4 discusses how the model is used for generating a problem. Section 5 describes how the student model is updated. Section 6 discusses the evaluation of the system. Section 7 concludes the paper.

2 The Domain

A developmental mathematics tutor, MFD (mixed numbers, fractions, and decimals), has been built and tested for fifth and sixth graders. The domain for the tutor is addition, subtraction, multiplication, and division of whole numbers, fractions, mixed numbers, and decimals. All of the problems in the tutor are presented as word problems. Because the system is being used by fifth and sixth graders, we present the problems using a compelling metaphor: endangered

* This work is supported through the National Science Foundation, under contract HRD-95555737. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

species. Students learn about, for example, the mating habits and the food requirements of various endangered animals, while solving mathematical word problems that deal with these issues.

Each type of problem in the domain is considered a *topic*. There are 14 topics in the tutor (there is no multiplication and division of mixed numbers). Each topic has an associated list of *pretopics*, which are themselves topics. These pretopics must be understood by a student before a problem of the topic can be given. For example, before a student can add mixed numbers, he must know how to add fractions and how to add whole numbers.

In addition to these pretopics, each topic has associated *subskills*, which are steps in the problem solving process for a given topic. For example, to add fractions, a student must find the least common multiple (LCM), convert the fractions to equivalent forms using that LCM, add the numerators, simplify the result, and convert the result to a proper form.¹

3 The Student Model

The student model contains topic information and material to encode acquisition and retention.

The student model in MFD records a *proficiency* for each topic within the domain. However, unlike many student models (e.g. Anderson and Reiser, 1985, and Eliot, 1996), ours does not use a simple number for the proficiency. A simple number, such as 0.4 (on a 0 to 1 scale) does not provide information about the context of the student's performance. Is the proficiency 0.4 because the student has just started on the topic, but is doing well? Or is the student having difficulty and his proficiency used to be higher? For these reasons, we use a history based model, so that the tutor can track the student's performance over time. This history is used in selection of high-level teaching strategies such as whether the student needs remediation or if he has forgotten a topic and needs a review.

However, even with a history model, simple numbers are too restrictive, since the tutor cannot express any degree of confidence in the proficiency. A 0.4 proficiency usually means the tutor is 100% sure that the proficiency is correct. However, this is very unlikely, since there is a significant probability that the student model is not completely accurate (Eliot, 1996). Furthermore, a single number does not provide sufficient information. A 0.4 usually means either that the student knows 40% of the material or that he has a 40% chance of knowing the material completely. Which of these choices is often left unspecified, and in any event it is often beneficial for the tutor to know both pieces of information.

When deciding what type of student model to use in MFD, we were not specifically interested in low-level representation issues associated with student models. We were not concerned if the system's beliefs were generated using methods such as Bayesian networks or the Dempster-Shafer theory of evidence. Rather, we were more interested in how to use the model to alter interactions with students. Additionally, we are investigating how to use general characteristics of students, in addition to their performance on each topic.

For these reasons, we have decided to use a simple scheme for representing uncertainty on each topic: *belief vectors*. This representation is based on the "fuzzy" distributions first used in Sherlock II (Katz et al., 1993) and also used in (Gurer et al., 1995). Each vector contains seven points, with the values summing to 1. The value at each point indicate the approximate probability

¹ Not all subskills will apply for a given problem. A problem such as $\frac{1}{4} + \frac{1}{4}$ will not require finding an LCM, converting to equivalent fractions, or making the result proper.

that the student is at that level of knowledge. The lowest value for the vector is (1 0 0 0 0 0) and the highest value is (0 0 0 0 0 1). A vector of (0.14 0.28 0.4 0.18 0 0) means the tutor believes there is approximately a 14% chance the student is at level 1, a 28% chance he is at level 2, a 40% chance he is at level 3, and an 18% chance he is at level 4. There is no chance he is above level 4. Furthermore, maintaining a list of these vectors enables the tutor to evaluate the student in the context of his past work.

This representation has some drawbacks. First, it does not actually use probability theory, but is updated via heuristics. Therefore, the values in the vector are only approximate. Second, there is no formal understanding of how this mechanism works, so it must be validated empirically. The difficulty of deriving precise estimates via experiments led (Katz et al., 1993) to consider switching to a mechanism like Bayesian networks for future implementations of Sherlock. Finally, it is difficult to relate this mechanism to other, more formal theories of representing uncertainty.

We chose to use this representation as it is easy to work with and to quickly modify as the need arises. Given that we are exploring the effect of adding new, general factors to the student model, and that this ad hoc solution appears to work adequately, it suffices for our purposes. When we develop a better understanding of how to use general factors to update our model, and how to use our model to construct problems, we will use more established frameworks.

While this vector representation gives us more information about a student's ability, we are currently collapsing the vector to a single number for the tutor to use in its calculations. To determine the value of the vector, we use a weighted calculation. Each item in the vector is multiplied by its index, and these values are summed. For example, the value of the vector (0.14 0.28 0.4 0.18 0 0) is 2.62.² This value is used as the student's proficiency in a particular topic. In the future, we will analyze the vector itself rather than calculating a single number.

3.1 Acquisition and Retention

In addition to the student model storing information on each topic within the domain, we also maintain general factors concerning the student, specifically acquisition and retention. Prior research indicates that examining general factors such as acquisition and retention can be beneficial for student modeling. Work with the LISP tutor (Anderson, 1993) and with Stat Lady (Shute, 1995) indicates that general factors are predictive of overall learning and allow for a more accurate response to the idiosyncrasies of the student.

Acquisition records how well students learn new topics. When a new topic is introduced, the tutor views how the student performs on the first few problems. If the student performs well on these problems, then he is acquiring skills quickly, and his acquisition factor will reflect this. However, if a student requires many problems on a given topic before he illustrates that he understands it, his acquisition will be lower.

Retention measures how well a student remembers the material over time. This factor is updated when the student is presented with a problem on a topic that he has not seen for a given period of time. If he answers the problem correctly without requiring any hints, then he has retained the knowledge, so his retention factor will be high. On the other hand, if he needed many hints to answer the problem, and previously he did not, then his retention is poor.

² $1 * 0.14 + 2 * 0.28 + 3 * 0.4 + 4 * 0.18$

4 Using the Student Model

As we have described previously (Stern et al., 1996), the student model is used to select the topic, generate the problem, and provide appropriate feedback. To select the topic, the proficiencies of all topics are examined, as well as the student's acquisition and retention factors. The tutor first examines topics on which the student may need remediation or may have forgotten. If no such topics exist it then selects among those that are not mastered by the student, but that he is ready to attempt.

Hints in MFD have *levels* indicating how much information they provide to the student. If the student needs a hint while solving the problem, the tutor finds a hint of the appropriate level on the skill that the student probably performed incorrectly. Within this domain, the tutor cannot be certain about which step was performed incorrectly. A set of heuristics is used to determine where the student made a mistake, and if two or more skills may be at fault, the one that must be performed earliest in the problem solving process is tagged as the likely culprit.

In the remainder of this section, we discuss how the student model is used to generate the problem for the student.

4.1 Problem Difficulty

One of the main goals of an intelligent tutoring system is to tailor instruction to the needs of each student. MFD provides problems that are at the correct level of challenge and difficulty for the student through the algorithm described in Section 4.2.

The general philosophy is that the more subskills required to solve a problem, the harder the problem. For example, when adding fractions, a problem such as $\frac{1}{3} + \frac{1}{3}$ requires few subskills, whereas $\frac{2}{3} + \frac{5}{8}$ requires many. Table 1 shows the subskills required for some sample problems.

Table 1. Three sample add-fractions problems.

	Find LCM	Equivalent Fractions	Add Numerators	Simplify	Make Proper
$\frac{1}{3} + \frac{1}{3}$	no	no	yes	no	no
$\frac{1}{3} + \frac{1}{4}$	yes	yes	yes	no	no
$\frac{2}{3} + \frac{5}{8}$	yes	yes	yes	yes	yes

Additionally, the topic itself may put certain constraints on the problem which affect its difficulty. For example, students conceptually have an easier time adding fractions that start in simplified form. Therefore these topic constraints must be taken into account when determining problem difficulty.

Given this philosophy, the goal of the tutor in generating problems is to determine how many and which subskills are needed when solving the problem, as well as which topic constraints should be used. For each problem generated, the tutor dynamically makes these decisions, and produces a problem to fit the chosen criteria.

How many subskills? As we have noted, the more subskills required in solving a problem, the harder the problem. The question is, therefore, one of determining how many subskills should be used on each problem.

In MFD, the number of subskills needed is based on two factors: topic ability and acquisition. If the student is doing well on the topic itself, then he should be challenged, and thus the problems should be more difficult. In addition to overly simple problems being an inefficient use of time, there is evidence (Kashihara et al., 1994) that for learning to occur students must be challenged mentally. On the other hand, if the student has been performing poorly on this topic, then the problems should be easier, with fewer subskills involved.

However, a student's acquisition factor should also play a role in problem difficulty. If his acquisition factor is high, then he should also be challenged, since he is able to learn the new material quickly. So even if the topic ability is low, when a student starts a new topic, the problems should not be the simplest possible.

Which subskills? The topic's proficiency affects how many subskills are needed to solve the problem. Just as with topic selection, we want students to practice those skills that they have not mastered. Therefore, the system picks which subskills the student must use based on his ability on each subskill.

To decide which subskills will be used in solving the problem, each skill is given a *priority*, which is a function of its proficiency. The lower the proficiency, the higher the priority; the higher the proficiency, the lower the priority.

Furthermore, even the subskills have levels of difficulty. For example, finding the LCM of two numbers that are multiples is much easier than finding the LCM of two numbers that are not multiples. When choosing the level of difficulty of the subskill, its proficiency is used. The higher the proficiency, the higher the level of difficulty.

It may be the case that two subskills suggest different properties for the problem. For example, the "find LCM" skill may suggest that the denominators are not multiples, but the "equivalent fraction" skill indicates that the denominators be multiples. When this occurs, the one with the lower proficiency is given the higher priority.

4.2 The Algorithm

To determine which subskills should be used for the current problem, a line is dynamically constructed on an x-y coordinate system before a problem is presented. The x-axis represents the student's ability at individual skills and the y-axis is the probability that those skills will be required to solve the problem.

The y-coordinate for the line is determined by the student's proficiency on the current topic. The slope is a function of the student's acquisition factor, and is always negative. The general equation for the line is:

$$y = f(\text{acquisition})x + g(\text{topic ability}) \quad (1)$$

The priorities for all the subskills are the x-coordinates that are plugged into the equation. For topic constraints, the topic proficiency is used as the x-coordinate. Each subskill's and constraint's y-coordinate is calculated based on the line and its x-coordinate. The higher the proficiency, the higher the x-value, and thus the lower the probability the skill will be selected.

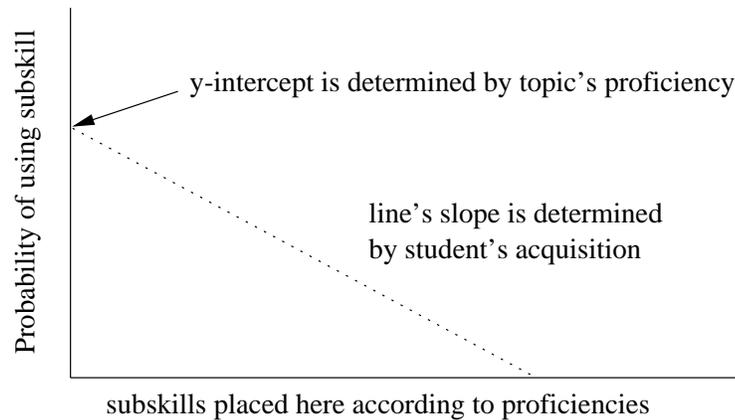


Figure 1. Basic structure of problem construction.

Next a random number is generated, and if the number is less than the y-coordinate, the skill or constraint is chosen as a candidate to be used for generating the problem. The randomness is to provide for challenge and/or review. Some subskills for which the student is not ready may be chosen, and some subskills the student has mastered may also be chosen. Also, the randomness helps control for potential inaccuracies in the student model.

Once the subskills and constraints have been chosen, their levels of difficulty must be decided upon. This is determined by looking at the proficiency of the skill. If it is very low, then the easy version is selected. If it is high, the hard version is selected. Otherwise, the medium version is chosen.³

At this point, the problem itself can be generated. Generic problems for the chosen topic are generated, and each one is evaluated for its “goodness”. This goodness factor is determined by examining how many of the chosen subskills must be applied in solving this problem, as well as how many of the topic constraints apply. Each subskill or constraint that applies contributes its priority to the goodness value. When a problem’s goodness is greater than 80% of the maximum goodness,⁴ then that problem is selected to be given to the student.

4.3 An Example

First consider Figure 1; this demonstrates graphically the main components of the model. The slope is a function of acquisition, the y-intercept is based on the topic’s proficiency, and the subskills are arranged according to their proficiencies on the x-axis. For simplicity, this example assumes that subskills only have two levels of difficulty, and students’ abilities are represented as *Excellent* (1), *Good* (0.75), *OK* (0.5), *Fair* (0.25), or *Poor* (0). The example will be constructed for a student working on a problem that requires adding fractions, and who has the following set of abilities:

³ We are omitting the details of the thresholds for easy, medium, and hard versions.

⁴ When all subskills chosen apply and the problem satisfies all constraints.

-
- *Add Fraction*: Good
 - *Simplify Fraction*: Good
 - *Make Proper*: Fair
 - *Find LCM*: Poor
 - *Equivalent Fractions*: OK
 - *Acquisition*: Good

Determining features of the line. The first step is to transform the problem's topic's (Add Fraction) proficiency into a y-intercept. Currently MFD simply uses the proficiency, with a maximum score corresponding to a y-intercept of 1 and a minimum score corresponding to a 0. In this case, the y-intercept will be 0.75.

Similarly, the line's slope is determined by the acquisition factor, which is a number between 0 and 1. In this case the acquisition is 0.75. However, for this model the desired slope is negative. Therefore this number is changed as follows:

$$\text{slope} = \frac{-0.5}{\text{acquisition}} \quad (2)$$

So, in this case the slope is $\frac{-0.5}{0.75} = -0.67$. This simple mathematical formulation was tried as a first approximation. If more complexity is needed to account for users' behavior there is no reason a more complex mechanism cannot be substituted in the future.

The y-intercept and slope have been computed, and the equation for the resulting line can be written as:

$$y = -0.67x + 0.75 \quad (3)$$

The y-value is the probability that the subskill will be required to solve the problem. All that remains now is to determine where each subskill is placed on the x-axis. To determine where to place a skill on the x-axis, the system simply uses its proficiency. Since the line that maps x-coordinates to probabilities has a negative slope, a subskill with a large x-value has a small chance of being used. Therefore, the tutor generates problems that require the use of subskills on which the student is not proficient.

Selecting subskills to use. Now the subskills to be used must be chosen. To determine which subskills to use, the system simply calculates the y-value of equation 3 using the proficiency as the x-value in each case. Table 2 shows the result of these steps: based on the subskills' proficiency, its priority and probability of use are calculated.

Table 2. Information about subskills from example.

	Proficiency	P(using subskill)	Priority
Simplify Fraction	0.75	0.25	0.25
Make Proper	0.25	0.58	0.75
Find LCM	0.00	0.75	1.00
Equivalent Fractions	0.50	0.42	0.50

At this stage in the algorithm, randomness is used to decide which subskills will be required of the student in order to solve this problem. For this example, assume the tutor wants to build a problem that will require the student to make the problem's answer proper and to find equivalent fractions, but the student will not have to simplify the result or find a least common multiple (i.e. the result will already be simplified, and both operands have the same denominator).

There is a contradiction between not having to find an LCM and requiring the student to make use of equivalent fractions. Rather than encode this knowledge explicitly, problems are generated and their usefulness is evaluated based on the summed priority of the subskills with which the problem agrees. Priority is the importance of requiring the student to use a subskill in the course of solving the problem, and is computed by calculating $(1 - \text{proficiency})$. Therefore it is possible for a skill that has a high priority to be ignored if several others "vote" in the opposite direction. The tutor generates several problems and picks the one with the highest value. Table 3 provides examples of problem goodness. The first problem matches for Simplify Fraction, Make Proper, and Equivalent Fractions, while the second problem only matches for Find LCM. Thus the first problem has a goodness of 1.50, while the second has a goodness of 1.

5 Updating the Student Model

Once the student solves a problem correctly, the student model must be updated. This is accomplished by examining the hints the student needed to be able to solve the problem, the student's current ability at the topic, and his acquisition and retention factors. The mechanism is similar to that used in (Gurer et al., 1995):

$$\text{Upgrade rule: } p_i = p_i - p_i c + p_{i-1} c \quad (4)$$

$$\text{Downgrade rule: } p_i = p_i - p_i c + p_{i+1} c \quad (5)$$

where c is a constant that controls the rate of updating, and p_i is the value of slot i in the "belief vector" (Section 3.1).

Table 3. Value computation for sample problems.

	Priority	Desired?	$\frac{2}{3} + \frac{1}{2}$	$\frac{1}{4} + \frac{1}{4}$
Simplify Fraction	0.25	No	Match	-
Make Proper	0.75	Yes	Match	-
Find LCM	1.00	No	-	Match
Equivalent Fractions	0.50	Yes	Match	-
Problem goodness			1.50	1.00

This formula is used to update the beliefs at each slot in the vector. If a student gets an item correct, the upgrade rule is used; an incorrect response results in the downgrade rule being

applied. The result of this is to shift the belief distribution either upwards towards mastery (the student probably knows the material) or downwards.

We have extended this model in two ways. First, instead of simply shifting the entire belief distribution upwards or downwards, it is shifted towards the student's most likely level of ability. Second, instead of using a constant term c , MFD uses parameters that vary as a function of the student's acquisition and retention.

5.1 Changing the Update Parameter

Rather than using a simple constant to update the vector values, MFD calculates a parameter based on the student's acquisition and retention factors. The rationale for this is the relation between how quickly values move towards the "right" of the vector and how quickly the student learns, while how quickly the student moves to the "left" is primarily a function of how likely some of the information is forgotten. If a student has been learning material quickly, his proficiency at a skill should increase more quickly than if he generally required much practice to master a skill.

5.2 Target for Updating Values

Rather than simply shifting the distribution one way or the other, the system moves the belief distribution towards what it infers the student's level of ability to be. The mechanism for computing this is similar to that described in (Stern et al., 1996): The system tracks the levels of the hints presented to the student. A hint's "level" refers to how much information it provides the student. A low-level hint may be a simple prompt such as "Recheck your work" while a high-level hint might say "The answer is 21."

The current implementation first considers the highest level hint the student needed to solve the problem, and then takes its complement⁵. The update rules given previously are used with a slight modification: For vector indices less than the hint level, the upgrade rule is used, while for indices above the hint level the downgrade rule is used.

As a result of the two changes described, the update rule takes the form:

$$\text{For } i \text{ less than the hint level: } p_i = p_i - p_i A + p_{i-1} A \quad (6)$$

$$\text{For } i \text{ greater than the hint level: } p_i = p_i - p_i B + p_{i+1} B \quad (7)$$

where A is a function of the acquisition factor, and B is a function of the retention factor. For example, assuming A and B are both 0.5:

$$\begin{aligned} T_0 &= (0.14 \ 0.14 \ 0.14 \ 0.14 \ 0.14 \ 0.14 \ 0.14); \text{ default proficiency} \\ T_1 &= (0.07 \ 0.14 \ 0.14 \ 0.14 \ 0.14 \ \mathbf{0.28} \ 0.07); \text{ after a level "2" hint is presented} \\ T_2 &= (0.04 \ \mathbf{0.25} \ 0.14 \ 0.14 \ 0.21 \ 0.18 \ 0.04); \text{ after a level "6" hint is presented} \\ T_3 &= (0.02 \ 0.14 \ 0.19 \ 0.14 \ \mathbf{0.37} \ 0.11 \ 0.02); \text{ after a level "3" hint is presented} \end{aligned}$$

The bold items indicate the most likely level of skill based on the hint given.

This sequence results in the system being able to reason that there is little chance the student knows nothing (or everything) about the domain while it is most likely his level of ability corresponds to "5". Adding this information to the update rule is justified, because of the level of the hint the student requires is closely related to his degree of proficiency (Shute, 1995).

⁵ This is done by taking $(8 - \text{hint level})$ and is performed simply as a notational convenience

6 Formative Evaluation

In order to test the usefulness of the modeling techniques described, the MFD system was tested on 20 fifth graders with an average age of 10.5 years (one subject's data were discounted due to a learning disability). As this system was used on a limited basis (3 to 4 hours per student), learning gains compared to a traditional class were not the focus of the evaluation. Rather we were interested in how the subjects perceived the level of problem difficulty and the amount of help they required at the beginning of their work with the tutor, and at the end of their time with it.

If the system is adjusting problem difficulty correctly, students will almost certainly need some help from the tutor; otherwise it is probably not being assertive enough at providing difficult problems. Of course, if too much help is required, it is possible that students are having considerable difficulty and may be getting frustrated with the system.

An analysis was performed of how much help students needed as their interactions with the system progressed. For the first third of the problems students required about 0.01 hints per problem, for the medium third 0.17 hints per problem, and for the final third 1.1 hints per problem. The final number of hints per problem seems slightly high, but the first two numbers are extremely low. A reasonable conclusion is that problems should be more difficult initially, but the system was somewhat too aggressive in providing difficult problems (either through overestimating their abilities or simply constructing problems that are too difficult for a given ability).

Table 4. Student responses about problem difficulty.

Student's belief	Beginning	End
Too easy	53%	11%
Too hard	11%	58%
Just right	37%	32%

However, as Table 4 shows, examining qualitative data gathered from questionnaires gives a different perspective. Many students thought the initial problems were too easy; however for others the problems were at the right level of difficulty or too hard. Initial problems are heavily based on the default student model, which attempts to capture the "average" student. Based on student responses, it has done a good job at this. However a mechanism for quickly moving the default values towards a student's actual ability is needed. Previous researchers (Shute, 1995) have used methods such as pretests to initialize the default student model. We would prefer a more integrated mechanism that would automatically update the tutor's representation of a student's ability more quickly when the tutor's estimate is uncertain (e.g. when the student first starts working with the tutor). Alternately, if a few common types of users can be distinguished, stereotypes (Kay, 1994) can be used to provide superior default reasoning about users.

Additionally, students thought the later problems were more difficult than the analysis of hints required indicated. Students claimed the problems were too difficult, but the number of hints they

required wasn't that high (we were trying to have problems that required one hint per problem). It is possible that students are underestimating their abilities as they are not needing too much help on later problems. In future studies, we will examine both the number of hints required and qualitative feedback from students in order to evaluate the system.

Students were generally enthusiastic about using the system. On a scale of 1 (don't want to work with the system again) to 7 (would like to work with the system very much) students gave the tutor a 5.5. Similarly, when asked how much of an effect the system had on their mathematics skills students reported that it had a positive effect (5.5 out of 7). An interesting item is that initially boys and girls rated problems at the same level of difficulty, but by the end the girls rated problems as being more difficult than the boys did (6.1 to 4.7). This difference was statistically significant, and troublesome as one of the system's goals is to help increase girls' confidence in their mathematics abilities. Additionally, this indicates that a tutor using gender as a variable is better able to customize its interactions to fit the user. However, this is a rather sensitive issue and altering instruction in this manner may not be appropriate. This remains a difficult balancing act between providing difficult problems vs. not allowing students to become frustrated or discouraged.

7 Conclusions and Future Work

With little extra effort, it is possible to collect additional information about how the student is currently performing and how his performance is changing over time. These data can be used to directly affect teaching strategy selection and problem generation. We have derived an algorithm, based on our model of problem difficulty, that uses the student model to adjust the characteristics, and thus the difficulty, of problems such that they are appropriate for the student's abilities. This mechanism is clearly applicable to other domains involving mathematics. With alteration, it can also be applied to areas that have procedural skills in which the steps to be performed can be considered to be subskills. For example, medical training has problems that can be categorized by topics and has distinct substeps that must be performed.

We are also interested in constructing general factors that provide an overall summary of a student's progress. While our metrics are not as accurate as those derived for Stat Lady (Shute, 1995), which used a lengthy test to derive one general factor, the factors here are easily obtainable and have a clear fit into the tutor's model of the student. Furthermore, all systems must make some assumptions about what constants to use. Rather than assuming one number will work for every student, MFD uses what data it has to get a better approximation. So even if the estimates of acquisition and retention are not precise, they are better than just using the same default value for each student.

Future research goals include finding better methods of computing general factors about the student to build a more accurate model. Additionally, finding a better use for the vector based model rather than collapsing it to a simple number is a priority. We are also investigating more formal approaches to probabilistic student models. To address these issues, the system is being tested in several local fifth grade classrooms for one month in the Spring of 1997.

References

- Anderson, J. (1993). *Rules of the Mind*. Hillsdale, NJ: Lawrence Erlbaum Associates.
- Anderson, J., and Reiser, B. (1985). The LISP tutor. *Byte* 10(4):159–175.
- Bloom, B. (1984). The 2 sigma problem: The search for methods of group instruction as effective as one-to-one tutoring. *Educational Researcher* 13:3–16.
- Eliot, C. (1996). *An Intelligent Tutoring System Based Upon Adaptive Simulation*. Ph.D. Dissertation, University of Massachusetts.
- Gurer, D., desJardins, M., and Schlager, M. (1995). Representing a student's learning states and transitions. Presented at the 1995 American Association of Artificial Intelligence Spring Symposium on Representing Mental States and Mechanisms.
- Kashihara, A., Sugano, A., Matsumura, K., Hirashima, T., and Toyoda, J. (1994). A cognitive load application approach to tutoring. In *Proceedings of the Fourth International Conference on User Modeling*, 163–168.
- Katz, S., Lesgold, A., Eggan, G., and Gordin, M. (1993). Modelling the student in Sherlock II. *Journal of Artificial Intelligence in Education* 3(4):495–518.
- Kay, J. (1994). Lies, damned lies and stereotypes: Pragmatic approximation of users. In *Proceedings of the Fourth International Conference on User Modeling*, 175–184.
- Shute, V. (1995). Smart evaluation: Cognitive diagnosis, mastery learning and remediation. In *Artificial Intelligence in Education: Proceedings of AI-ED 95*, 123–130.
- Shute, V., Glaser, R., and Raghaven, K. (1989). Inference and discovery in an exploratory laboratory. In Ackerman, P., Sterberg, R., and Glaser, R., eds., *Learning and Individual Differences*, 279–326.
- Stern, M., Beck, J., and Woolf, B. (1996). Adaptation of problem presentation and feedback in an intelligent mathematics tutor. In *Proceedings of Intelligent Tutoring Systems*, 605–613.