LEARNING FROM STUDENTS TO IMPROVE AN INTELLIGENT TUTORING SYSTEM

Eric Gutstein
Computer Sciences Department
University of Wisconsin-Madison
1210 W. Dayton St.
Madison, WI 53706
gutstein@cs.wisc.edu
608-233-6776

Abstract

SIFT, a Self-Improving Fractions Tutor, is an intelligent tutoring system which learns from its interactions with the students it tutors. Its learning module takes transcripts of tutoring sessions as input. It analyzes the results of its work and modifies its knowledge bases in domain-specific ways, creating new tutorial rules and extending its domain knowledge. To produce its transcripts, SIFT tutors *student-simulations* which interact with the tutor to solve problems.

SIFT generates a rule for each hypothesis that could possibly explain why it took inappropriate or wrong actions, although not all hypotheses are ultimately valid. It initially assumes that new rules have equal correctness probability. It then continues to tutor (using its probabilistic conflict-resolution rule-selection algorithm) and uses feedback from its rule applications to modify probabilities with the Dempster-Shafer theory of evidence. Initial results show that (1) SIFT learns more rules than it uses, but eventually converges on correct and minimal rule sequences, (2) it learns what one would want it to given the type of student who uses it, and (3) its learning patterns are intuitively plausible in terms of human tutors.

Keywords

Learning from the environment, intelligent tutoring systems, evaluation by simulation.

INTRODUCTION

The central question I address in this work is: how can an intelligent tutoring system (ITS) learn from its interactions with students to improve its tutoring over time? More generally, how can a machine learn through environmental interaction? Several approaches to these questions exist. In the area of ITS, a few self-improving systems have been built (e.g., Dillenbourg, 1988a, 1988b; Dillenbourg & Goodyear, 1989; Kimble, 1982; O'Shea, 1982). In machine learning, researchers have approached this question several different ways, including learning by discovery (Langley, 1981; Lenat, 1977), learning by experimentation (Carbonell & Gil, 1987; Mitchell, Utgoff, & Banjeri, 1983; Shen, 1989), and feedback-driven learning such as connectionist learning (Rumelhart, McClelland, et al., 1986). This has also been addressed in work on correcting imperfect domain theories (Mitchell, Keller, & Kedar-Cabelli 1986; Rajamoney, 1988; Rajamoney & DeJong, 1987).

My approach to this question is that a rule-based ITS should be able to examine the results of its tutoring and have criteria by which to judge their effectiveness. It should then alter its tutoring knowledge accordingly—learn new tutorial rules and extend its domain knowledge—and use these modifications in future tutoring. These modifications need to be made sufficiently general as to apply in (potentially) all future tutoring situations. To be effective, however, a systems also needs specialization operators as a complement, in order to discriminate between which one of the enlarged rule set to apply. Therefore, the system should continue to refine its knowledge by finely honing its rule selection based on further feedback as it tutors and should eventually converge on the appropriate rule sequences (of new *and* old rules) that work in most situations with most students.

In this paper, I describe SIFT, a Self-Improving Fractions Tutor, based on the above framework. SIFT is an implemented production system which combines both a scaled-down tutor and a learning module which improves its tutoring through practice. Its information flow is shown below in Figure 1. SIFT's knowledge bases and learning mechanisms (rules and domain knowledge) are derived from my analysis of some 50 actual transcripts of an expert tutoring children fractions, from my interviews with the tutor and other curriculum experts, and from my own knowledge as a math teacher for seven years (Gutstein, 1992).[1] Given the additional difficulties of designing an effective user interface, the students who use SIFT are production-system *student-simulations* to which I refer whenever I use the term "student" in this paper.

1. The expert tutor is Nancy Mack, a mathematics educator on the faculty at Northern Illinois University. She studied how children learn fractions with understanding for her dissertation research and has provided the transcripts (Mack, 1987, 1990). Much of my own teaching was also in the area of fractions.

These students interact with the tutor, solve or mis-solve problems, occasionally learn, and produce the transcripts with the tutor which the learning module uses. They are also used for evaluation purposes as I describe in the RESULTS section.

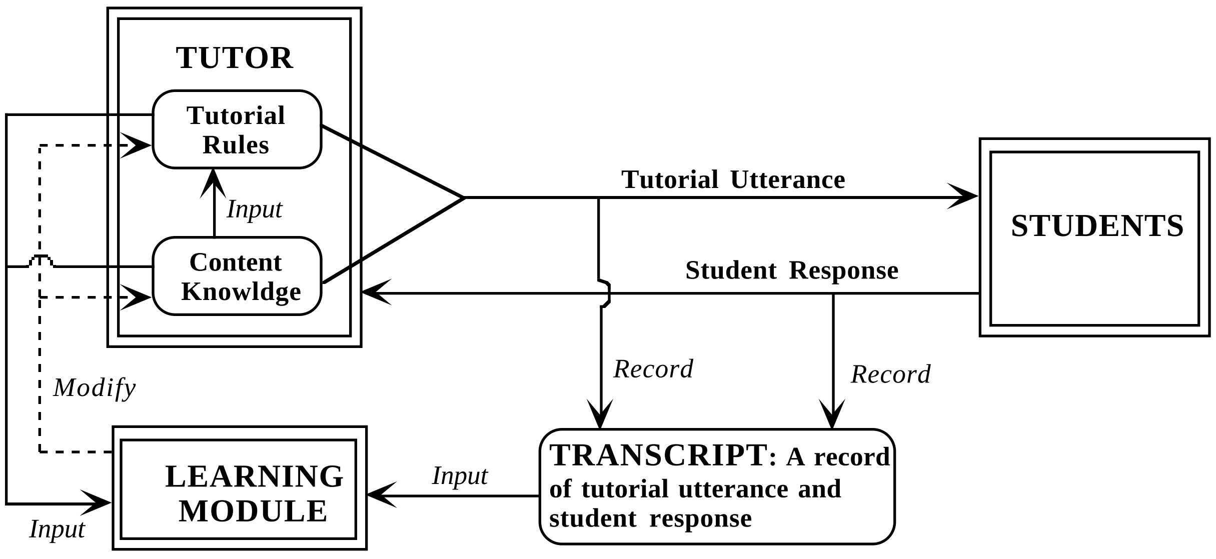

Figure 1: SIFT's information flow. Students and tutor together produce transcripts which are input to the learning module. The learner subsequently modifies tutorial rules and content knowledge.

LEARNING IN SIFT

SIFT learns through experience, by interacting with its environment, where its environment may be considered the students who use it. Specifically, it learns by analyzing transcripts of sessions created by its tutor and students. The inputs to the learning module are the transcripts, tutor's rules and underlying content (domain) knowledge which is encoded as a semantic net (*rational task analysis* in education terminology), and the results of previous learning. Its learning goal is to produce knowledge modifications that improve its tutoring.

Tutoring is a complex task, just recently becoming understood at the fine-grained level necessary to fully implement successful ITS (Glades, 1990; Lewis, McArthur, Stasz, & Zmuidzinas, 1990; McArthur, Stasz, & Zmuidzinas, 1990; Ohlsson, 1986; Putnam, 1985, 1987; Shavelson, Webb, Stasz, & McArthur, 1988). The findings of these researchers indicate that tutoring is a knowledge-intensive task demanding much domain expertise.[2] Therefore, it follows that any attempt to improve a ITS must depend heavily on domain knowledge.

2. Research into teaching, in addition to tutoring, also supports this contention (Corno & Snow, 1986; Shulman, 1986).

Examples of how SIFT relies on this knowledge are how it creates new tutorial rules which alter the actions taken by the tutor and how it extends its content knowledge.

However, it is also possible for an intelligent tutoring system to improve itself in a more general way, such as by evaluating empirically the effectiveness of its knowledge-base modifications. As an example, SIFT's uses a conflict-resolution algorithm to choose which tutorial rule to execute when more than one is enabled. While the algorithm includes other mechanisms, the primary choice method is based on rule probabilities. These are updated when SIFT is learning based on the success or failure of a particular rule application. While the criteria for success and failure of a rule application is clearly a domain-dependent issue, updating probabilities is a more general method of modifying the knowledge bases, which I describe more fully below. Table 1 below gives an abstract description of SIFT's learning module.

Table 1: An abstract description of SIFT's learning module.

INPUTS TO THE LEARNING MODULE:
• A recording of interactions between (1) a goal-directed, rule-based agent, with a probabilistic conflict-resolution algorithm, and (2) the external world.
• The agent's knowledge base, consisting of:
  a. Rules controlling its actions,
  b. A semantic net specifying other domain knowledge.

LEARNING SYSTEM'S ACTIONS:
• Analyze the record and evaluate both individual and sets of actions with respect to how they contribute to or detract from the agent's goals.
• Modify the agent's knowledge bases in domain-dependent ways:
  a. Create new rules where appropriate,
  b. Extend the agent's semantic net.
• Empirically evaluate prior knowledge modifications and, in a domain-independent way, update rule probabilities using Dempster-Shafer reasoning (Gordon & Shortliffe, 1984).

SIFT learns new rules in a variety of situations. The general precondition for SIFT to learn is for it to judge that its tutor took some inappropriate tutorial action(s). This is not the only situation from which it learns, but it is the primary one. I list below the different instances in which it learns.

• Evaluation of *opportunistic digressions*. A digression is when a tutor decides that another subject should be taught and therefore switches topics.[3] SIFT evaluates whether digressions are successful or not and determines whether the tutor should take the same departure in the future.

3. Lewis et al. (1990) discuss this in some depth; digressions are obvious in Mack's transcripts.

• <u>The student's response is not in the tutor's *expectation-set*</u>. The expectation-set is a set of incorrect, variableized answers SIFT stores with each *chunk* (i.e., problem-type). These are student responses for which the tutor believes it understands the underlying misconception. For instance, suppose the tutor knows the common error of adding both numerator and denominator when adding fractions. It will consider *3/10* to be within its expectation-set when a student gives that as an answer to the problem, *1/5 + 2/5 = ?* The premise here is that a tutor is better able to help students learn when it understands the student's error. During learning, SIFT extends its expectation-set when an answer is not in it, in a way similar to *model-tracing* (Anderson, 1984).

• <u>Evaluation of rule sequences in which a tutoring goal is not achieved</u>. An implicit tutoring goal is that students solve every problem correctly, or at least make progress toward solutions. These break down into the following sub-categories:

1. The tutor attempts to get a student to generalize a concept or procedure, as in Table 2 below. However, the *inductive leap* may be too large or too early, and the student may need to be given a particular sequence of problems which helps him or her to generalize (Gutstein & Shavlik, 1990).
2. The problem sequence varies along too many dimensions at once, and the student is unable to handle the changes. For example, in one of Mack's transcripts, the student correctly solved a problem of type *integer-minus-fraction* (e.g., *3 – 3/4*). The next problem Mack gave him was of the type, *mixed-minus-mixed* (e.g., *3 1/2 – 1 1/4*). The changes the form of both operands in the problem; the first changes from an integer to a mixed number, the second from a fraction to a mixed number. The added complexity with insufficient *scaffolding* (Cog apprenticeship ref) may be too much for a student to handle.
3. The tutor presents a problem in symbolic terms before the student's informal or intuitive knowledge of the concepts is sufficiently developed. This is a well recognized problem in mathematics education (Hiebert & Carpenter, 1992).
4. The student may have insufficiently learned some prerequisite knowledge.
5. The tutor presents a *compound* problem type (one with more than one operator), but the student needs to have it broken down into its component parts.

I describe these learning mechanisms more fully in Gutstein (1993). Below I detail how and what SIFT learns from a transcript with a compound problem. Following that, I discuss results of some learning tests with this type of problem.

Table 2 contains a transcript portion in which Mack was tutoring a compound problem. From this example, I derived the preconditions of some of SIFT's rules and the actions it takes after it learns. I call the problem type, *compound-divide-then-mult.* The particular problem that Mack gives the student is encoded as *(/ * 1 4 4)* in SIFT, or, "take one whole, divide it into *n* parts, and then take all *n* of the parts. What do you have?" In this case, Mack was looking for the answer, *4/4.*

When SIFT reviews a transcript such as this, it notices that the same problem has been given three times with increasingly explicit hints.[4] This causes several learning rules to fire. What these rules do depends partly on SIFT's current student-model in which the tutor rates the student's knowledge of the different prerequisite chunks and partly on the sequence of taught chunks shown below in Figure 2. Each rule corresponds to a different hypothesis about why the student failed to answer the problem correctly. In this example, SIFT generates three hypotheses:[5]

(1) the student insufficiently learned the immediate prerequisite that was taught *longest* ago, implying perhaps a memory lapse or decaying factors,

(2) the student insufficiently learned the immediate prerequisite that was taught *most recently* , implying perhaps that it was insufficiently encoded,

(3) the student insufficiently learned the *lowest-rated* prerequisite chunk, and

(4) the student needed to have the problem decomposed in order to understand and solve it.

Each hypothesis is reasonable, in general, but all depend on the particular student and the subject matter being taught. In the absence of other information, nothing determines unequivocally which is correct. Furthermore, there may be another unhypothesized reason, or the tutor's actions may have been appropriate, but appeared otherwise for some anomalous reasons. In fact, this happens in SIFT, as I discuss in the RESULTS section.

For the *problem-decomposition* hypothesis, SIFT creates three new rules. These rules are based on the compound problem, in this case, *(/ * 1 4 4)*. The first new rule creates a problem

4. SIFT would see that the number of hints increases and that the final time the problem is given, the hint number is at the threshold of allowable hints. This is what actually triggers the learning rules.

5. I do not claim these to be an exhaustive hypothesis set. Mathematics educations have to describe such a set. Furthermore, such a scenario does not lend itself easily to a buggy diagnosis (Burton & Brown, 1978). This is because, although one can write production rules (mal-rules) to produce the student errors at each stage, they will hardly explain the content of this interaction, which may depend on language, situation (context), prior knowledge, or other factors. In addition, the intent of a buggy diagnosis is to provide remediation, at least in the framework of Burton and Brown (1978). My claim here is that these are possible inferences, which has been confirmed by experts in mathematics learning. In any event, to add additional hypotheses to SIFT is simple due to its production system knowledge representation.

Table 2: A tutoring interaction with tutorial goal of solving a *compound-divide-then-multiply* problem. Angle bracketed statements are my comments, not Mack's.

Tutor:	***Suppose you're going to get a pizza and it's cut into 4 pieces when you get it. Can you tell me the name for how much you get?*** <question has 2 parts--(1) how big is each part, and (2) how many pieces altogether. Mack was looking for the answer, 4/4.>
Student:	**1/4.**
Tutor:	***If you get the whole pizza?*** <re-ask problem, focuses on part that would seem to clarify--ie, gives a hint>
Student:	**A fourth.**
Tutor:	***Okay, they're all fourths and how many do you have?*** <re-ask yet again, increases hint level--actually gives half the answer>
Student:	**4, so 1/4 (pause) 4 ones...**
Tutor:	***Almost...(asked student if he wanted to use manipulatives, which he did... he got out the 1/4 sized pieces) Okay, so if you had one piece of the pizza, how much would you get?*** <decomposes original problem and asks first part>
Student:	**1/4.**
Tutor:	***If you got all 4 pieces, you'd get?*** <uses answer to first part of original problem to ask question about second part of original problem>
Student:	**(pause) 4 fourths, 'cause it's 4 pieces and I get all of them, so 4 fourths, there are 1/4 and you put 'em in all at once and you get 4 fourths.**
Tutor:	***(asked student if pizza is cut into 5 pieces and get whole pizza, how much get)*** <inductive step--asks student to begin generalizing the solution>
Student:	**5 fifths, because if I get it all, there's 5 pieces in it.**
Tutor:	***(asked about cut into 8 pieces)*** <continues to ask student to generalize, inductive leap>

using the *first* operator of the compound problem (division) applied to its *first two* operands (*1* and *4*). This rule's preconditions are the same as those of the rule which *first* gave the compound problem. This ensures that this new rule will be enabled at the same time as the original rule in a similar situation. The new problem is *(/ 1 4)* and is of the type, *divide-units*., (e.g., "if you have a pizza and it's cut into 4 pieces, how big is each piece?") SIFT then takes the first problem and simulates working memory contents as if an ideal student were to solve it. These become the preconditions for the second rule created, which produces a problem using the *second* operator, (*), and whose two operands are the result of the first problem correctly solved, (i.e., *1/4*), and the *last* operand in the original problem (i.e., *4*). This problem, *(* 1/4 4),* is of the type, *mult-inverse*. (e.g., "if you a piece of pizza that's one fourth of a pizza and you have 4 pieces like that, how much pizza do you have altogether?") SIFT simulates working memory after solving this one as well, which becomes the preconditions for the third (and final) rule

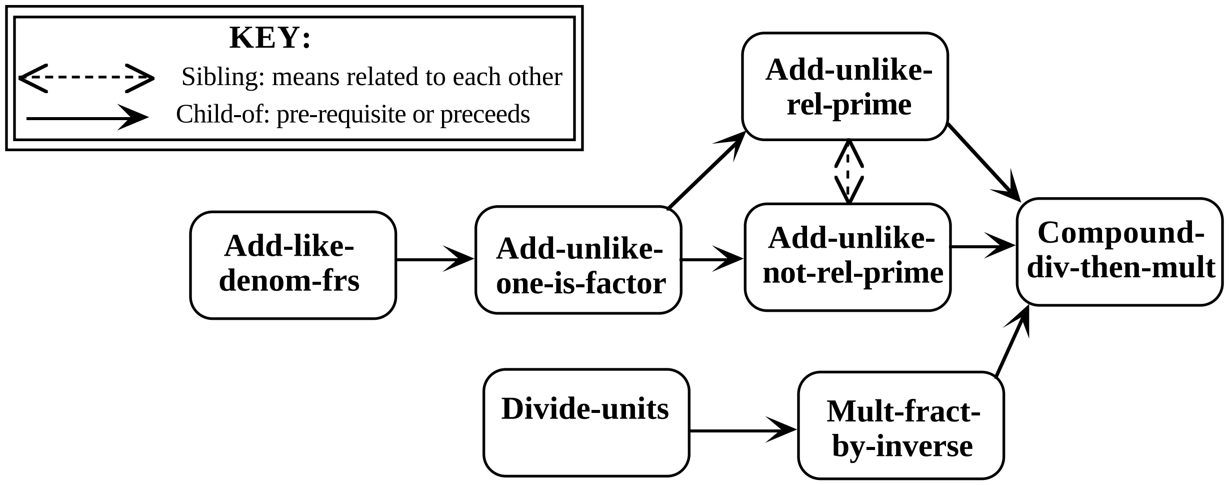

Figure 2: A portion of the task analysis through which SIFT tutors a *compound-divide-then-mult* problem, starting from *add-like-denom-frs*. SIFT may go either way from *add-unlike-one-is-factor*.

created. This rule gives the *original* compound problem if and only if the two immediately preceding problems are considered *building-block* chunks of the compound one and the student correctly solved both. This building-block chunk information is encoded as part of the task analysis. Table 3 below summarizes each new rule created with its preconditions and actions.

Table 3: A summary of rules created for a *compound-divide-then-mult* problem, e.g., (/ * 1 4 4).

Rule	**Preconditions**	**Action**
1	Same as the original rule which first created the problem.	Take first operator and first two operands, and produce a new problem, *Problem-1*.
2	Simulates solving *Problem-1*. Contents are new LHS.	Take second operator, *Problem-1*'s solution, and third operand; produce *Problem-3*.
3	Simulates solving *Problem-2*. Contents are new LHS and must have problem-types of *Problem-1 & 2* as immediate prerequisites.	Give original problem.

However, in this example, two other new rules are created as well, corresponding to the other three hypotheses listed above.[6] All five rules are added to SIFT's tutorial knowledge base, though only the first of the three rules created for the third hypothesis above is enabled at the same time as the other two new rules. Together they make up the *conflict-set* —the tutorial rules enabled given an isomorphic situation. Table 4 below lists the three new rules and the original rule of the conflict-set for the above example.

Table 4: The original rule and three newly created rules making up the *conflict-set* in a learning instance.

- T-RULE-5 ;; an original rule which shifts chunks after a student solves 3 problems of the same type.
- #decomposing-rule-1302 ;; the <u>first</u> of the 3 rules learned for the *problem-decomposition* hypothesis.
- #least-recently-taught-chunk-1305
- #lowest-rated-chunk-1306

A human tutor may prefer one hypothesis for some reason, however, without other knowledge, SIFT initially assumes equal probabilities for each hypothesis. How can it know which rule is the appropriate one to use in the situation? SIFT does this by empirical verification. It tries each new rule in an isomorphic situation at least once, and uses the feedback to alter its rule probabilities via the Dempster-Shafer theory of evidence (Gordon & Shortliffe, 1984).

The Dempster-Shafer theory is a mathematically-based generalization of Bayesian probability theory. Its purpose, as proposed for use in *MYCIN*, was to incrementally narrow a hypothesis set over time by gathering evidence and altering hypotheses' probabilities.[7] In SIFT, each rule corresponds to a hypothesis. I consider an appropriate rule firing as evidence of the hypothesis' correctness. Thus, the evidence gathering process in SIFT is the accumulation of the results of rule firings. In the above example, the hypothesis set is represented by the enabled rules in Table 4; note that this includes the original rule. SIFT alters the rule probabilities using the Dempster-Shafer updating rule. If a rule correctly fires (e.g., a student solves the problem the rule produced), SIFT increases its probability and lowers that of the other rules within the

6. In this particular scenario, it turns out that the two hypotheses, *reteaching the most recently taught chunk*, and *reteaching the lowest rated prerequisite chunk*, create identical rules. SIFT does not create identical rules, thus only two other rules are created, despite three hypotheses.:

7. It was never actually implemented because *MYCIN* was no longer in use by the time these authors proposed its application.

conflict-set. The reverse takes place if a rule fires incorrectly. When a rule probability in a hypothesis set reaches 0.99, or *converges*, SIFT stops altering the probabilities of that set.

An important aspect of the Dempster-Shafer theory is that confidence can be expressed in a non-singleton hypotheses set. For example, a tutor may believe that a student insufficiently knows *some* prerequisite, without being able to know initially which one. Thus a non-singleton set of hypotheses may be given a probability, which is important to SIFT's rule selection process. This is shown below in Figure 3. Furthermore, if a rule fires incorrectly, when SIFT increases the probabilities of the other rules within the hypothesis set, it does so in aggregate without differentiating between them, since it has no way to do so without further results. However, in the real world, a human tutor may be able to more appropriately assign changing probabilities based on the specific situation.

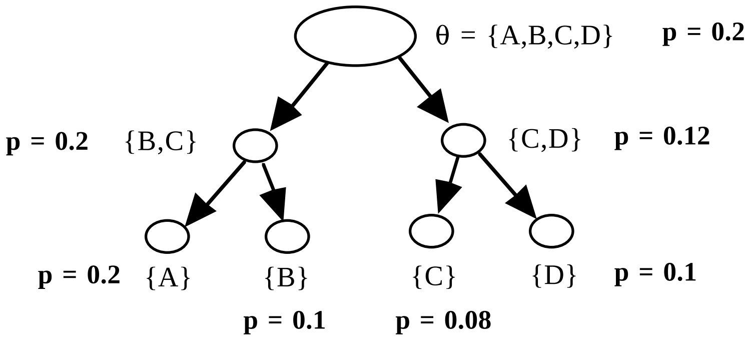

Figure 3: A sample hypothesis space with probabilities assigned to non-singleton subsets. The term, θ, is used to represent the set of all hypotheses.

In addition, Dempster-Shafer theory defines *plausibility* as the probability of a hypothesis set plus the amount of certainty not assigned to the hypothesis' negation—its *doubt*. These ideas—assigning probabilities to non-singleton hypothesis-sets and plausibility—affect how SIFT's tutor chooses which rule to fire. To do this, SIFT generates a random number between 0 and 1, which it maps to a linear sequence of the hypothesis sets as shown in Figure 4 below. If the random number falls within a singleton subset, SIFT fires the corresponding rule, but if it falls in one with more than one rule, it chooses the rule with the maximum plausibility. Note that this may *not* be the same as the rule with the maximum probability. For example, if the random number generated were 0. 634, SIFT would fire the more plausible of Rule B or C, which is C in this case, even though B's rule probability is higher.

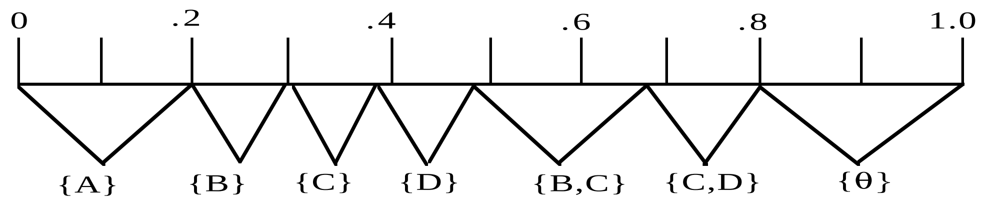

Figure 4: The hypothesis space of Figure ? presented linearly. With a random number of 0.634, SIFT would fire rule C, which is the more plausible of rules B or C, even though B's raw score is higher.

RESULTS

To evaluate a self-improving ITS, one can test it before learning, let it learn, and then test it after learning. Probably the best measure of its learning is how well human students learn when using it. However, since SIFT is not yet usable by people, I use the *student-simulations* to evaluate it.[8] I evaluate SIFT by having it tutor a set of students and record the results. I then have SIFT learn from these transcripts, have it re-tutor an isomorphic set of students probabilistically varied from the original, and measure its effectiveness. I define below some evaluation methods and discuss one learning example.

The following results are from SIFT tutoring the chunks shown above in Figure 2. Before SIFT learns, all the sessions end with the students unsuccessfully solving a *compound-divide-then-mult* problem. These students are unable to solve this problem type even when SIFT gives them as many hints as its hint threshold allows (currently 2). The learning module takes the transcripts and creates the new tutoring rules in Table 4 above. SIFT continues to tutor, with both the new rules and the originals, and gradually converges on the correct rules using Dempster-Shafer probability updating.

In this test, I have categorized the students into two groups. SIFT tutors the students together, randomly choosing a student from either group. G*roup A* students correctly solve a *compound-divide-then-mult* problem if (and only if) they are tutored in a way similar to Mack's tutoring in Table 2—by being given a *divide-units* problem, immediately followed by a *mult-inverse* problem, immediately followed by a *compound-divide-then-mult* problem, where the *divide-units* and *mult-inverse* problems are derived from the *compound-divide-then-mult* problem. *Group B* students solve the problem if and only if they receive sufficient practice in

8. The student-simulations contain from one to seventeen rules. These rules both solve problems correctly, as well as incorrectly. If a student contains or more two rules applicable to a given problem type, they are assigned probabilities which sum to 1.

one of the prerequisite chunks, although for this example, it does not matter which one.[9] Students from both groups solve the compound problem correctly with a probability of less than 1, even after SIFT learns to give them appropriate extra practice or divide-and-conquering assistance.[10] Initially, both groups solve compound problems wrong. The reason is that before it learns, SIFT tutors by giving more explicit hints when a student gets a problem wrong and give up after a set number of hints. It does not initially know to give extra practice in a prerequisite chunk nor break down a problem into parts—it must learn these things.

When SIFT first tutors a student who cannot solve a *compound-divide-then-mult* problem, it learns the rules as described in Table 3 above. It adds these to its rule base and continues teaching. However, as shown in Figure 2 above, there are two paths to reach *compound-divide-then-mult,* starting from *add-like-denom-frs.* Thus, SIFT learns <u>different</u> sets of rules depending from which path it approaches *compound-divide-then-mult* and depending on which type of student it tutors. Each rule set is enabled in different situations as the main distinction between the rule sets is in the conditions of applicability.

For *Group A* students, SIFT converges on the appropriate sequences after tutoring an average of 24.13 students over 25 trials. In two cases of 25, SIFT converged on incorrect rules, but this only happened when the probability of the students correctly solving the compound problem was 50% after being presented with a *divide-units* and then a *mult-fraction-by-inverse* problem. For any probabilities greater than 50%, the rule sequences on which SIFT converged were always appropriate for *Group A* students.

For *Group B*, the results are complicated by what SIFT regards as extra practice for a particular chunk and by the interplay between new rules when tutoring more than one student group. Before learning, SIFT's tutor switches problem types if a student answers three consecutive problems in a row of the same type. This is its T-RULE-5 mentioned in Table 4. If the tutor is providing extra practice in a prerequisite chunk, it gives five problems in a row before switching. However, this leaves more room for the tutor to explore different problem combinations when the final chunk (e.g., *compound-divide-then-mult*) has more than one path from which it can be reached, as in Figure 2 above. Because of this, SIFT takes a longer time to converge on the correct rule sequence. The average number of students to converge for students of *Group B* was 46.36 students over 11 trials. Table 5 below summarizes these results.

9. The chunks, *add-unlike-relatively-prime* and *add-unlike-not-relatively-prime* are not necessarily the actual prerequisite chunks of *compound-divide-then-mult,* however, I consider them to be for the purposes of this evaluation.

10. I have experimented with probabilities ranging from 50% to 90%.

Table 5: Learning results for tutoring a *compound-divide-then-mult* problem. SIFT tutors both groups of students together.

Student	**Average number of students taught to convergence**	**Number Trials**
Group A	24.13	25
Group B	46.36	11

In one learning test SIFT learned the following two rule sequences for *Group B* students:

- *least-recently-taught-rule-2178, lowest-rated-rule-2084, T-RULE-5, T-RULE-5.*
- *lowest-rated-rule-2084, lowest-rated-rule-2084, T-RULE-5.*

T-RULE-5 is an original rule and the other two are new. However, SIFT converged quite slowly on the second sequence. This is because the students it tutored had a 10% probability of incorrectly solving the compound problem after receiving extra practice in the preceding chunks. At one point, SIFT gave a student the appropriate preceding problems, but the student failed to correctly solve the compound problem. This caused additional new rules to be learned, which delayed convergence because they were based on situations occurring only 10% of the time. However, since SIFT did not know this and treats all new rules equally, it had to learn through experience that these new rules were not applicable. Furthermore, these new rules caused SIFT to learn other new and ultimately inappropriate rules, slowing down convergence even more.

Intuitively this makes sense for people. When a human tutor has a student who does things different from the norm, the tutor should learn to do something different from what has worked with the majority. When SIFT learns from students with a 100% probability of correctly solving the compound problem, this does not occur, convergence is faster, and SIFT learns less rules. However, this is less natural, as it assumes too much conformity than found in people. The more variance we have of students, the more rules one might be expected to learn to account for them.

SIFT learns about 60 new rules when tutoring the above two groups of students, starting from ten initial rules.[11] Many of the rules are similar to each other, varying mainly in the student model preconditions. It learns most of its rules early on when tutoring. This also corresponds with what a human tutor might do—learn a lot of rules (i.e., new tutoring procedures) early on, differing mainly in their applicability conditions, then refine them through practice to settle on a minimal, but sufficient set. Figure 5 below shows a graph with the number of rules learned per ten students tutored. Once SIFT converges, it almost always chooses a rule sequence it found to

11. SIFT selects the next problem to give to a student from a stored list, thus eliminating complicated problem generation rules. This is one reason it can start off with such a small set of tutoring rules.

be empirically valid based on the students it is teaching. If SIFT subsequently starts tutoring additional students with different learning patterns, it learns more new rules to account for them and eventually settles on the proper rule sequences for these additional students as well. Thus, despite learning many rules, once it converges, SIFT ultimately uses very few and these are almost always correct.

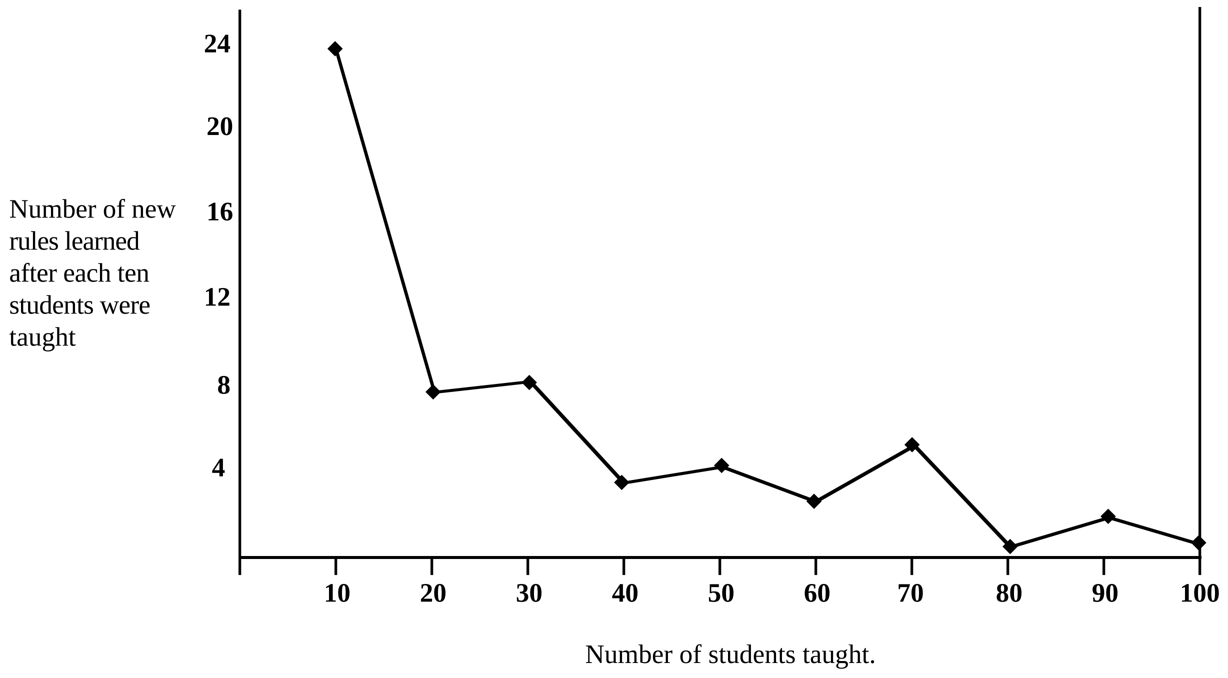

Figure 5: Average rate at which SIFT learns rules when teaching *compound-divide-then-mult* problems.

CONCLUSION

Expert systems, including ITS, with fixed knowledge bases are inherently limited. All human experts learn as they perform their craft. This learning comes in many forms, but a reflective analysis of the results of one’s work is clearly an important one. Furthermore, theories of tutoring are being developed which will be the foundations of future ITS as researchers acknowledge the expertise and knowledge involved in tutoring (e.g., McArthur et al., 1990). It thus becomes important to integrate machine learning theories with educational ones to build teaching systems that improve with age. As an ITS based on the knowledge and actions of an expert human tutor and which uses machine learning techniques and methods, SIFT represents one attempt to bridge these fields.

REFERENCES

Anderson, J. R. (1984). Cognitive psychology and intelligent tutoring. *Proceedings of the Cognitive Science Society Conference*, Boulder, CO.

Carbonell, J. G., & Gil, Y. (1987). Learning by experimentation. *Proceedings of the 4th International Machine Learning Workshop.* Irvine, CA: University of California, Irvine.

Corno, L. & Snow, R. E. (1986). Adapting teaching to individual differences among learners. In M. C. Wittrock (Ed.), *Handbook of research on teaching*. New York: MacMillan Press.

Dillenbourg, P. (1988a). A pragmatic approach to student modeling: Principles and architecture of PROTO-TEG. *Proceedings of the Second European Seminar on ITS*. Lemans, France.

Dillenbourg, P. (1988b). Self-improving tutoring systems. *International Journal of Educational Research, 12,* 851-862.

Dillenbourg, P. & Goodyear, P. (1989). Towards reflective tutoring systems: self-representation and self-improvement. In D. Bierman, J. Breuker, & J. Sandberg (Eds.), *Proceedings of the 4th International Conference on AI and Education*. Amsterdam: IO.

Gordon, J. & Shortliffe, E. H. (1984). The Dempster-Shafer theory of evidence. In B. Buchanan & E. Shortliffe (Eds.), *Rule-based expert systems: The MYCIN experiments of the Stanford Heuristic Programming Project.* Reading, MA: Addison-Wesley.

Gutstein, E. (1992). Using expert tutor knowledge to design a self-improving intelligent tutoring system. In C. Frasson, G. Gauthier, & G. I. McCalla (Eds.), *Lecture notes in computer science: Proceedings of the Second International Conference on Intelligent Tutoring Systems*. Berlin, Germany: Springer-Verlag.

Gutstein, E. (1993). *SIFT: A self-improving fractions tutor.* Doctoral dissertation, in preparation. Madison, WI: University of Wisconsin-Madison .

Gutstein, E. & Shavlik, J. W. (1990). Choosing the right problem to evoke changes in students' strategies. *Working Notes, AAAI Spring Symposium on Knowledge-Based Environments for Learning and Teaching*. Stanford, CA:Stanford University.

Hiebert, J. & Carpenter, T. P. (1992). Learning and teaching with understanding. In D. A. Grouws (Ed.), *Handbook of research on mathematics teaching and learning*. New York: McMillan.

Kimble, R. (1982). A self-improving tutor for symbolic integration. In D. Sleeman & J. S. Brown (Eds.), *Intelligent tutoring systems.* London: Academic Press.

Langley, P. (1981). Data-Driven discovery of physical laws. *Cognitive Science, 5*(1).

Lenat, D. B. (1977). The ubiquity of discovery. *Artificial Intelligence*, *9*, 257-285.

Lewis, M. W., McArthur, D., Stasz, C., & Zmuidzinas, M. (1990). Discovery-based tutoring in mathematics. In *Working Notes, AAAI Spring Symposium on Knowledge-Based Environments for Learning and Teaching*. Stanford, CA: Stanford University.

Mack, N. K. (1987). *Learning fractions with understanding*. Unpublished doctoral dissertation. Madison, WI: University of Wisconsin.

Mack, N. K. (1990). Learning fractions with understanding: Building on informal knowledge. *Journal of Research in Mathematics Education*, *21* (1).

McArthur, D., Stasz, C., & Zmuidzinas, M. (1990). Tutoring techniques in algebra. *Cognition and Instruction, 7*(3).

Mitchell, T. M., Keller, R. M., & Kedar-Cabelli S. T. (1986). Explanation-based generalization: A unifying view. *Machine Learning, 1*(1).

Mitchell, T. M., Utgoff, P. E., & Banjeri, R. (1983). Learning by experimentation: Acquiring and refining problem-solving heuristics. In R. S. Michalski, J. G. Carbonell, & T. M. Mitchell (Eds.), *Machine learning: An artificial intelligence approach*. Palo Alto, CA: Morgan Kaufmann.

O'Shea, T. (1982). A self-improving quadratics tutor. In D. Sleeman & J. S. Brown (Eds.), *Intelligent Tutoring Systems.* London: Academic Press.

Ohlsson, S. (1986). Some principles of intelligent tutoring. *Instructional Science*, *14*, 296-326.

Putnam, R. T. (1985). *Teachers' thoughts and actions in live and simulated tutoring of addition.* Unpublished doctoral dissertation. Stanford, CA: Stanford University.

Putnam, R. T. (1987). Structuring and adjusting content for students: A study of live and simulated tutoring of addition. *American Educational Research Journal, 24*(1).

Rajamoney, S. A. & DeJong, G. A. (1987). The classification, detection, and handling of imperfect theory problems. In *Proceedings of the Tenth IJCAI Conference*. Milan, Italy.

Rajamoney, S. A. (1988). *Explanation-based theory revision: An approach to the problems of incomplete and incorrect theories.* Unpublished doctoral dissertation. Champaign, IL: University of Illinois.

Rumelhart, D. E., McClelland, J. L., & the PDP Research Group (1986). *Parallel distributed processing: Exploring the microstructure of cognition*. Cambridge, MA: MIT Press.

Shen, W. (1989). *Learning from the environment based on percepts and actions*. Unpublished doctoral dissertation. Pittsburgh, PA: Carnegie-Mellon University.

Shavelson, R. J., Webb, N. M., Stasz, C., & McArthur, D. (1988). Teaching mathematical problem solving: Insights from teachers and tutors. In R. I. Charles & E. S. Silver (Eds.), *The teaching and assessing of mathematical problem solving*. Reston, VA: Erlbaum Associates & NCTM.

Shulman, L. S. (1986). Paradigms and research programs in the study of teaching: A contemporary perspective. In M. C. Wittrock (Ed.), *Handbook of research on teaching*. New York: MacMillan Press.