

Using Evaluation to Shape ITS Design: Results and Experiences with SQL-Tutor

Antonija Mitrovic, Brent Martin, Michael Mayo

*Intelligent Computer Tutoring Group
Department of Computer Science, University of Canterbury
Private Bag 4800, Christchurch, New Zealand
{[tanja.bim20](mailto:tanja.bim20@cosc.canterbury.ac.nz),[mmayo](mailto:mmayo@cosc.canterbury.ac.nz)}@cosc.canterbury.ac.nz*

Abstract

This paper presents the results of three evaluation studies performed during 1998 and 1999 on SQL-Tutor, an intelligent tutoring system for the SQL database language. We have evaluated the system in the context of genuine courses, and used the results to further refine the system. The main goal of our research has been the exploration and extension of Constraint-Based Modeling (CBM), a student modeling approach proposed by Ohlsson (1994). SQL-Tutor provided us with experiences of using CBM, and we used it to extend the approach in several important ways. The main goal of all three evaluation studies was to determine how well CBM supported student learning. We have obtained positive results. The students who learnt with SQL-Tutor in the first study performed significantly better than those who did not when assessed by a subsequent classroom examination. Furthermore, the analysis of students' learning shows that CBM has a sound psychological foundation.

Besides the evaluation of CBM, we also evaluated the improvements in terms of student assessments of the usefulness of the system and evaluated various techniques used in SQL-Tutor. In the second study, we evaluated the effectiveness of feedback provided to the students. This study showed that high-level advice is most beneficial to students' learning. The focus of the third study was different. We extended CBM to support long-term modeling of student knowledge, and used this extension to develop an adaptive problem-selection strategy. The study revealed the benefits of this strategy in comparison with a simple heuristic strategy. We also reflect on our experiences in evaluating SQL-Tutor.

Keywords

Student modeling, constraint based modeling, evaluation, intelligent tutoring systems, probabilistic student model, pedagogical decision making

This paper has not been submitted elsewhere in identical or similar form, nor will it be during the first three months after its submission to UMUAI.

1. Introduction

Evaluating any AI-based project is a difficult task. Intelligent Tutoring Systems (ITS) fit this statement perfectly, being the application of AI methods to educational environments. The underlying theories are either new or still under development, and there is no widespread agreement as to how the fundamental tasks (student modeling, adaptive pedagogical decision making, generation of instructional dialogues etc.) should be performed. Since the area is quite young, it is logical for researchers to focus on establishing the basic methodology first. However, the importance of evaluation cannot be overlooked. Evaluation is fundamental in all stages of a research project, as it provides guidance for future work. It is not only within projects that evaluation can benefit research in ITS: across-system evaluation is also of extreme importance, because it allows for comparisons of effectiveness and suitability of various approaches. This type of evaluation provides us with the relative benefits of different approaches and identifies avenues for future research, thereby helping shape the whole research area.

ITSs may be evaluated with real or simulated students. The advantage of using simulated students is a higher level of control of the experimental setting. As (Millan & Perez-de-la-Cruz 2001) point out, such evaluation may provide useful insights, but should be just a first step towards full evaluation, as simulated students are too simplistic in comparison to real students in realistic environments.

Many reported evaluations with real students are performed in isolated, artificial situations, where subjects are paid to participate. We believe that such an approach to evaluation is inappropriate for ITSs, because the nature of the environment in which such systems will be used is not taken into account. Furthermore, subjects who participate in such evaluations are not representative of the target student groups. We firmly believe that it is important to perform evaluation in the context of genuine teaching activities. The benefits of using real classrooms, with students who are learning the subject material, are manifold. First, this kind of evaluation lowers the evaluation costs. More importantly, since the system is evaluated in an authentic situation rather than in an artificial setting, we can expect realistic behavior and data. However, evaluations in real classrooms face other problems, such as the timing of studies, unpredictable student behavior and others, which we discuss later in this paper.

We report on three evaluation studies performed in 1998 and 1999 on SQL-Tutor, an ITS for the SQL database language. All three studies involved elements of summative and formative evaluation and were performed in the context of courses we teach at the University of Canterbury, New Zealand. We present the system in section 2, followed by a brief overview of Constraint Based Modeling (Ohlsson 1994) in section 3. Section 4 describes the evaluation studies. We used the results of these studies to enhance the system, and also to evaluate the fundamental methodologies used in SQL-Tutor. The analysis of students' responses to the user questionnaire is given in section 5, and shows that the students perceived SQL-Tutor as user-friendly, and judged the feedback helpful, and the interface easy to learn.

The main goal of our research has been the exploration and extension of Constraint-Based Modeling. One of the aims of developing SQL-Tutor was to provide a test bed for the methodology in a fairly complex domain. Therefore, SQL-Tutor provides us with experiences of using CBM and also presents opportunities to extend the approach in several important ways. The main goal of all three evaluation studies was to determine how well CBM supports student learning. We have obtained positive results in all three evaluation studies, presented in section 6. Section 7 presents the analysis of students' performance in an examination. In the second study, discussed in section 8, we evaluated the effectiveness of feedback provided to the students. Section 9 presents the third study, which revealed the benefits of an adaptive problem-selection strategy based on a probabilistic student model. We present the conclusions in the final section.

2. SQL-Tutor

SQL-Tutor is a problem-solving environment intended to complement classroom instruction, for students that are already familiar with database theory and the fundamentals of SQL. The need for an intelligent tutoring system in the area of SQL is discussed elsewhere (Mitrovic 1998a). In SQL-Tutor, students work individually as much as possible, with the system only intervening when they are stuck or ask for help. There are three functionally identical versions of the system, for Solaris, MS Windows and the Web. Here we give only a brief description of the system, and the interested reader is referred to other papers (Mitrovic 1998b, Mitrovic & Ohlsson 1999) and the system's Web page¹ for details. The architecture of the stand-alone version² of the system is

¹ <http://www.cosc.canterbury.ac.nz/~tanja/sql-tut.html>

²For details of the architecture of the Web-enabled version, please see (Mitrovic & Hausler 2000).

illustrated in Figure 1. At the beginning of a session, SQL-Tutor selects a problem for the student to work on. When the student enters a solution, the pedagogical module sends it to the student modeler, which analyzes the solution, identifies mistakes (if there are any) and updates the student model appropriately. On the basis of the student model, the pedagogical module generates an appropriate pedagogical action (i.e. feedback). After the first attempt, the student is only told whether or not his/her solution is correct. The level of detail in feedback messages then increases if the student is still unable to correct the solution. We discuss different levels of feedback in section 8. When the current problem is solved, the user may log off, or go on to the next problem. There are two ways to select the next problem in SQL-Tutor: students can work through a pre-specified sequence of problems, or turn problem selection over to the system. In the latter case, the system will select an appropriate problem on the basis of the student model. At the moment, SQL-Tutor does not allow the student to select a problem directly from a list of all those available, but we plan to add this option soon. We discuss the problem-selection strategies used in more detail in section 9.

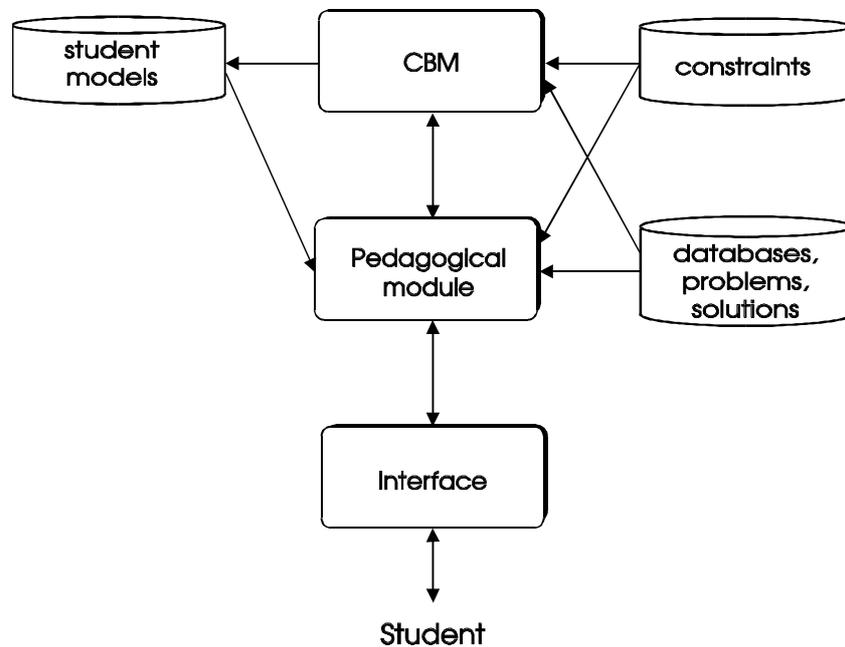


Fig. 1. Architecture of SQL-Tutor

The system contains definitions of several databases, a set of problems for each, and the ideal solutions to them. New databases can easily be added to SQL-Tutor, by supplying the same SQL files used to create the database in a database management system. Each problem is assigned a difficulty level, which depends on many features, such as the wording of the problem, the perceived complexity of the problem (which in turn depends on the number and nature of constructs needed for the solution), the number of required tables/attributes etc. Problem levels are assigned by a human expert, and are not related to constraints in any way. There are nine levels of problem complexity.

Each student is also assigned a level. At the beginning of the first session with SQL-Tutor, the student selects the appropriate initial level her/himself, from three possibilities: “novice”, “intermediate”, or “experienced”. This level is later updated in accordance with observations of the student’s behavior: it is incremented if he/she solves two or more problems consecutively at or above his/her current level, within three attempts each. Both problem and student levels are used for problem selection, as described in section 9.

The ideal solutions are necessary because SQL-Tutor has no problem solver, and therefore must evaluate student solutions by comparison to correct ones. There are two reasons for not having a problem solver. First, database queries are given in a natural language; however, the current state-of-the-art in Natural Language Processing (NLP) is still far from being able to handle various inherent problems, such as references and synonyms. There is a possibility to avoid NLP: the text of the problem may be represented not in its natural-language form, but in a form that could be the product of NLP, as used in (Anderson et al. 1995). However, it is difficult to avoid building parts of the solution into such a representation. Furthermore, even if we overlook the NLP problem, the knowledge required to write SQL queries is ill defined and it would be very difficult to develop a problem solver in this area.

Nevertheless, an ITS must be able to evaluate student answers. SQL-Tutor does this by comparing student solutions to the ideal ones. The system uses domain knowledge represented in the form of constraints to check correctness, described in more detail below. This constraint-based model handles differences between the student and ideal solutions, even major ones such as radically differing solution strategies.

The interface of SQL-Tutor, illustrated in Figure 2, has been designed to be robust, flexible, and easy to use and understand. It reduces memory load by displaying the database schema as well as the text of a problem, by providing the basic structure of the query, and by providing explanations of the elements of SQL. The main page is divided into four areas. The upper part shows the text of the problem being solved so students can remind themselves easily of the elements requested in queries. The middle left part contains the clauses of the SQL SELECT statement: students need not remember the exact keywords used and the relative order of clauses. The middle right section of the window is the feedback area. In Figure 2, the student has submitted an incorrect solution, and the current feedback informs the student that the solution is erroneous. The lowest part displays the schema of the currently chosen database. Schema

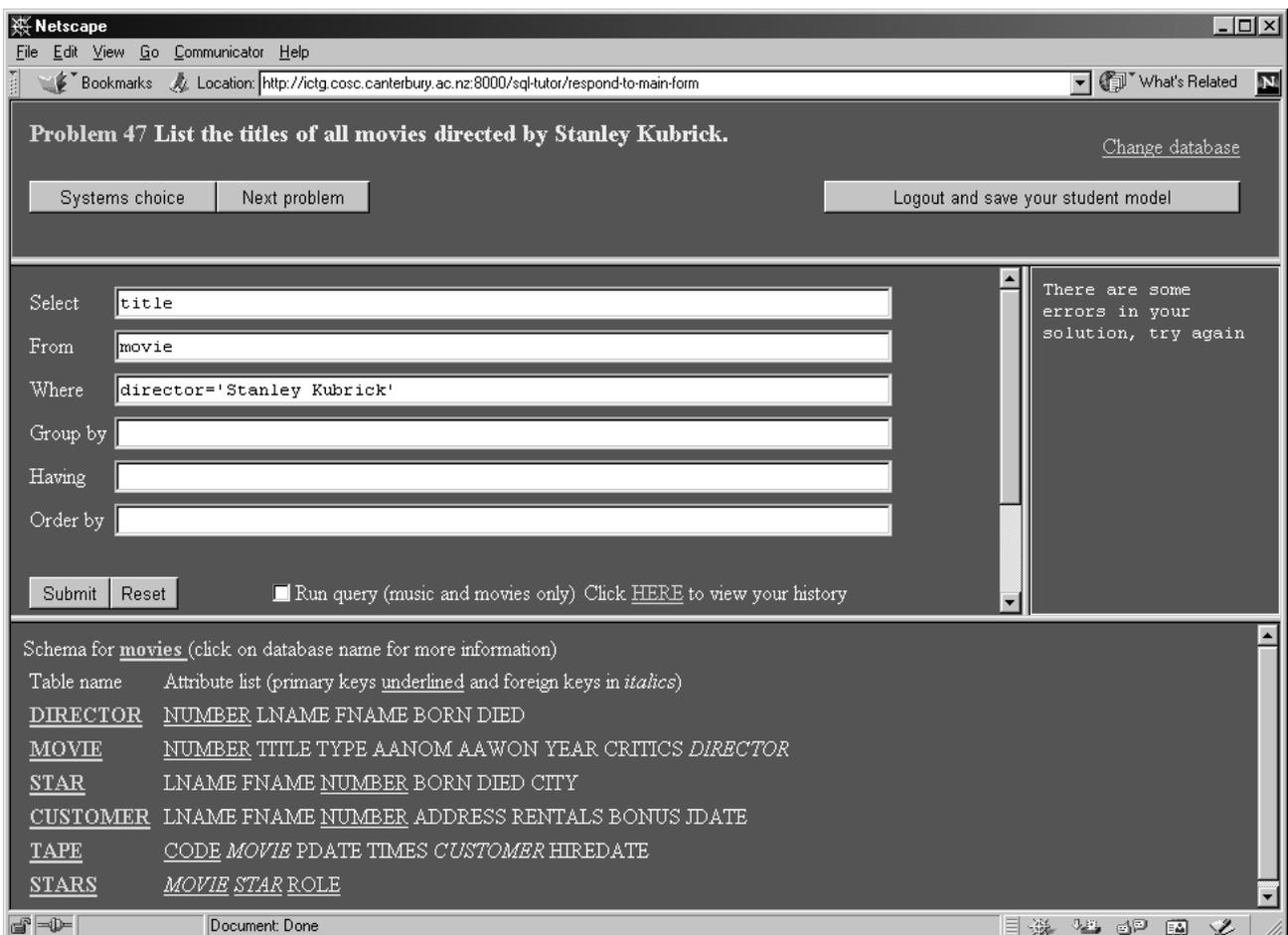


Fig. 2. Interface of the Web-enabled version of SQL-Tutor

visualization is very important: all database users are painfully aware of the constant need to remember table and attribute names, and the corresponding semantics. Students can ask the system for descriptions of databases, tables and attributes, as well as the descriptions of the SQL constructs. The motivation here is to remove from the student some of the cognitive load required for checking the low-level syntax, and to enable the student to focus on higher-level, query definition problems.

SQL-Tutor uses Constraint-Based Modeling (Ohlsson 1994) to diagnose students' solutions. The conceptual domain knowledge is represented in terms of over 500 constraints. We discuss CBM and give examples of constraints included in SQL-Tutor in the next section. A student's solution is matched to constraints to identify any that are violated. Initially we represented a student's long-term knowledge as an overlay upon the constraint set, in which a tally for each constraint shows the frequency of correct and incorrect use. We describe a more sophisticated probabilistic student model in section 9.

3. CBM

Constraint-Based Modeling is a student modeling approach proposed by Ohlsson (1994), as a way of overcoming the intractable nature of student modeling. CBM arises from Ohlsson's theory of learning from errors (1996), which proposes that we often make mistakes when performing a task, even when we have been taught the correct way to do it. According to this theory, we make mistakes because the declarative knowledge we have learned has not been internalized in our procedural knowledge, and so the number of decisions we must make while performing the procedure is sufficiently large that we make mistakes. By practicing the task, however, and catching ourselves (or being caught by a mentor) making mistakes, we modify our procedure to incorporate the appropriate rule that we have violated. Over time, we internalize all of the declarative knowledge about the task, and so the number of mistakes we make is reduced. Ohlsson describes the process of learning from errors as consisting of two phases: *error recognition* and *error correction*. A student needs declarative knowledge in order to detect an error. Only then can the error be corrected so that the solution used is applicable only in situations in which it is appropriate.

Procedure-tracing domain models (Anderson et al. 1995) check whether or not the student is performing correctly by comparing the student's procedure directly with one or more "correct"

ones. In CBM, we are not interested in what the student has done, but in what *state* they are currently in. As long as the student never reaches a state that is known to be wrong, they are free to perform whatever actions they please. Constraints define equivalence classes of problem states. An equivalence class triggers the same instructional action; hence all states in an equivalence class are pedagogically equivalent. It is therefore possible to attach feedback messages directly to constraints. A violated constraint signals an error, which translates to incomplete/incorrect knowledge. The domain model is therefore a collection of state descriptions of the form:

“If <relevance condition> is true, then <satisfaction condition> had better also be true, otherwise something has gone wrong.”

In other words, if the student solution falls into the state defined by the relevance condition, it must also be in the state defined by the satisfaction condition in order to be correct. An example³ of a constraint in the domain of SQL is given in Figure 3.

The first part of the constraint is a unique number, followed by the hint message that will be displayed if the constraint is violated. The relevance condition follows, which specifies that this constraint is important for solutions in which the WHERE clause is not empty, and contains a condition based on the ANY/ALL keyword. The satisfaction condition asserts that in such cases, the solution is correct if the SELECT clause of the nested query contains only one expression, which is an attribute of the same type as the attribute preceding the subquery. The last clause of the constraint identifies which part of the solution the constraint is dealing with (the WHERE clause in this example).

SQL-Tutor contains more than 500 constraints, and this number is likely to increase as new problems requiring new situations are added to the system. As can be seen, the relevance and satisfaction conditions are LISP clauses. They may contain any LISP predicate, but the most frequent predicate is *match*, which performs pattern matching.

A very important feature of CBM is its computational simplicity. Instead of using complex reasoning, as required by other student modeling approaches, CBM reduces student modeling to pattern matching. Conditions are combinations of patterns, and can therefore be represented in compiled forms, such as RETE networks (Forgy 1982), which are very fast, and for which off-the-shelf software is available. In the first step, all relevance patterns are matched against the

³ Annotations are shown in italic. For more examples of constraints see (Mitrovic & Ohlsson 1999)

problem state. In the second step, the satisfaction components of constraints that matched the problem state in the first step (i.e., the relevant constraints) are matched. If a satisfaction pattern matches the state, the constraint is satisfied. Otherwise, the constraint is violated. The student model consists of all violated constraints.

Unlike enumerative modeling (Anderson et al. 1995), CBM does not require extensive studies of student bugs. Furthermore, Ohlsson's approach is not sensitive to the *radical strategy variability phenomenon*, since it completely ignores the procedures used to solve the problem. It thus allows for inconsistencies in choosing a problem-solving strategy. CBM is neutral with respect to the pedagogy, since different pedagogical actions (immediate or delayed) may be generated on the basis of the model.

We believe that CBM is also neutral with respect to the domain. This paper describes a system that teaches a database language, but we have also developed CAPIT, a system that teaches the rules of punctuation and capitalization in English (Mayo et al. 2000), and KERMIT, a system for database design (Suraweera & Mitrovic 2001). We have experienced no problems expressing the knowledge of these various domains in terms of constraints. CBM is applicable both to procedural and declarative tasks. Currently, we are developing other ITSs in domains

(p 34

"If there is an ANY or ALL predicate in the WHERE clause, then the attribute in question must be of the same type as the only expression of the SELECT clause of the subquery."

(and (not (null (where ss)))

Non-empty WHERE clause?

(match '(?*d1 ?a (?or "<" ">" "=" "!=" "<>" "<=" ">=")

(?or "ANY" "ALL") "(" "SELECT" ?*l1 "FROM" ?*d2 ")" ?*d3)

(where ss) bindings))

Is there a condition based on the ANY/ALL predicate and a nested query in WHERE?

(and (equal (length ?l1) 1)

Single expression in the nested SELECT clause?

(equalp (find-type ?a) (find-type (car ?l1))))

That attribute of the same type as the attribute preceding the ANY/ALL predicate?

"WHERE")

Fig. 3. An example constraint

with different characteristics, to test the generality of CBM.

Another advantage of CBM is that it allows for a simpler architecture, since there is no need for a problem solver. CBM-based systems are able to generate instructional actions even without being able to solve problems on their own, by focusing on violated constraints. Of course, CBM does not prevent us from having a problem solver; on the contrary, the existence of such a component could be very beneficial to the student, since it might provide the answer to questions such as “What do I do next?”

CBM, as proposed by Ohlsson (1994), is a method for diagnosing a student’s solution. The approach identifies errors, which is extremely important for students lacking declarative knowledge, because these students are unable to detect errors themselves. As stated earlier, one of the goals of our research is to evaluate how well CBM supports learning. We also discuss in this paper how CBM can be extended to allow for long-term modeling of students’ knowledge, and alternatives for generation of pedagogical actions.

4. Evaluation Studies

Three evaluation studies of SQL-Tutor have been performed to date, with a new study planned for October 2000. We present the common details of all three studies here, with the specifics discussed in later sections. All studies were carried out at the University of Canterbury, using Computer Science students enrolled in database courses. Prior to using the system, the students had all attended six lectures about SQL, and completed at least eight hours of hands-on experience of query definition. They used SQL-Tutor in a two-hour session, during their normal lab time. All students' actions were recorded, and the students filled out a questionnaire at the end of the session. There were several observers present at each evaluation, who all reported that the students were quite interested in interacting with the system and exploring its various functions.

Study 1 was performed in April 1998. Participation was voluntary and anonymous. Twenty of the 49 students enrolled in the senior-year database course chose to participate. The goal of this study was twofold: to evaluate how well CBM supports student learning and to evaluate the interface and the constraint base of SQL-Tutor.

Study 2 was carried out in May 1999, and involved all senior-year students enrolled in a database course (33 students). The goal of this study was to evaluate the effectiveness of various

types of feedback provided by the system. Students were randomly allocated to one of two versions of the system: the first gave restricted feedback, while the second generated all levels of feedback.

Study 3 was performed in October 1999, and involved all second-year students taking an introductory database course (48 students). In addition to the questionnaire, study 3 students sat a pre- and post-test. Three versions of the system were used in the study: the basic version, a version which generated probabilistic student models and used them to select problems, and a version in which feedback was presented via an animated pedagogical agent. Students were randomly assigned to one of the versions. Since the evaluation of the pedagogical agent is irrelevant to this paper, we focus on the other two versions only. For details of the analysis of the pedagogical agent, please see (Mitrovic and Suraweera 2000). In this paper, we report on the evaluation of the probabilistic student model and the appropriateness of problems selected on the basis of such student models.

We present the results of subjective analysis first, by summarizing the responses to user questionnaires from all three studies. When questionnaires included specific questions relevant to a single study only, the responses are summarized in a section devoted to the corresponding evaluation study.

5. Subjective evaluation

This section presents a summary of the students' answers to the user questionnaire in all three studies. The purpose of the questionnaire given to students at the end of the session was to evaluate the students' perception of SQL-Tutor. The questionnaire, which is included in Appendix A⁴, consisted of 16 questions, most of which were based on the Likert scale with five responses ranging from *very much* (5) to *not at all* (1). Students were also able to give free-form responses.

Table I gives the number of students involved in each study, and their responses to whether they would recommend SQL-Tutor to other students. Please note that the percentages do not add up to 100%, because some students did not answer all questions. As illustrated there, students appreciated the style of learning with the system. Figure 4 further illustrates the students'

⁴ Appendix A contains the questionnaire used in Study 3. Some of the questions in the questionnaires for the previous two studies were slightly different.

impressions of the system. The majority of the students appreciated the learning experiences with SQL-Tutor and the feedback received, and liked the interface.

The interface has changed only slightly between studies: new options were added to the system, and some studies required some options to be hidden from students allocated to specific evaluation groups. The differences in the ratings of the interface (Figure 4.c) are therefore small. Although study 2 concentrated on the various kinds of feedback, the same feedback was generated in all studies. We believe that the apparent improvement in students' perception of SQL-Tutor, as seen in Table I and Figures 4.a, 4.b and 4.d, is attributable to the introduction of new databases and problems to the system. The version of the system used in the first study contained problems formulated in the context of two databases, both of which have been used for examples in lectures and labs. The students were therefore already familiar with those problems, a fact reflected frequently in their comments. For the later studies, we added new databases and problems, which proved to be much more interesting and challenging for the students.

Study	No of students	Recommend SQL-Tutor Yes/No (%)
1	20	75/0
2	33	84/3
3	48	94/0

Table I. Some statistics about the subjective evaluation

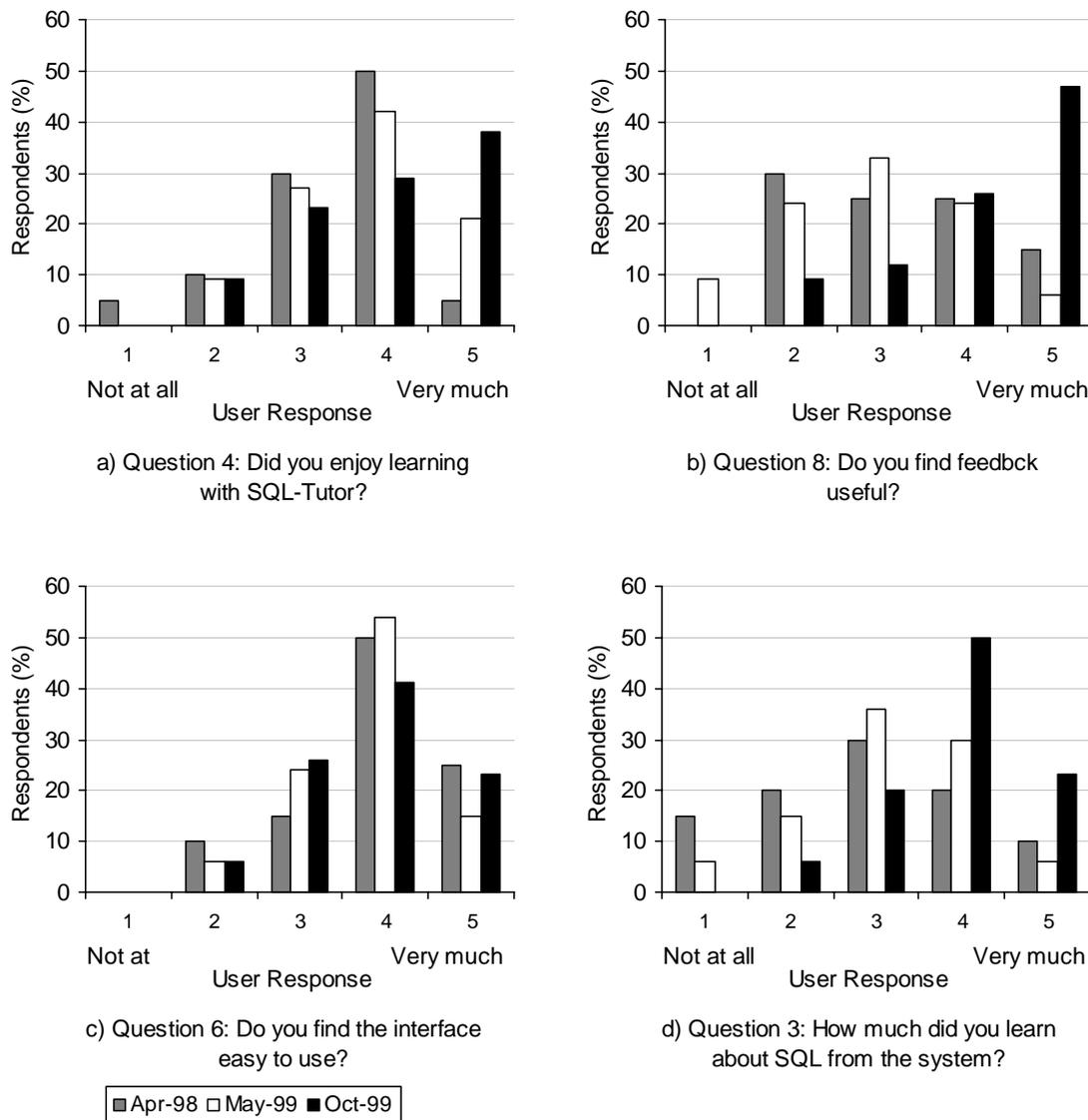


Fig. 4. Responses from the user questionnaire (in percentages)

6. Mastery of Constraints

In a previous paper (Mitrovic & Ohlsson 1999) we showed that state constraints represent psychologically appropriate units of knowledge. When students' learning is plotted in terms of constraints, we get a smooth curve that closely approximates a so-called power law (Newell & Rosenblum 1981). In other words, the degree of mastery of a given constraint is a function of the

amount of practice on that constraint. The evaluation data reported in (Mitrovic & Ohlsson 1999) was collected in study 1. The analysis of students' learning published in that paper concentrated on a subset of 100 randomly selected constraints, which were relevant at least once during the study. Here we perform the same type of analysis, but this time using all the constraints that were used by students, and we report on the findings for all three evaluations.

We first describe the procedure used to analyze the data. For each constraint, we identified all problems (and their initially submitted solutions) in which this constraint was relevant in a student's log, and rank ordered them from 1 through R. We refer to these as *occasions of application*; each entry is for a different problem, hence this analysis shows how practice with a constraint on one problem affects its use on subsequent problems. For each occasion, it was recorded whether the relevant constraint was violated or satisfied. This analysis was repeated for each constraint and each student. From this transformation of the computerized records we can

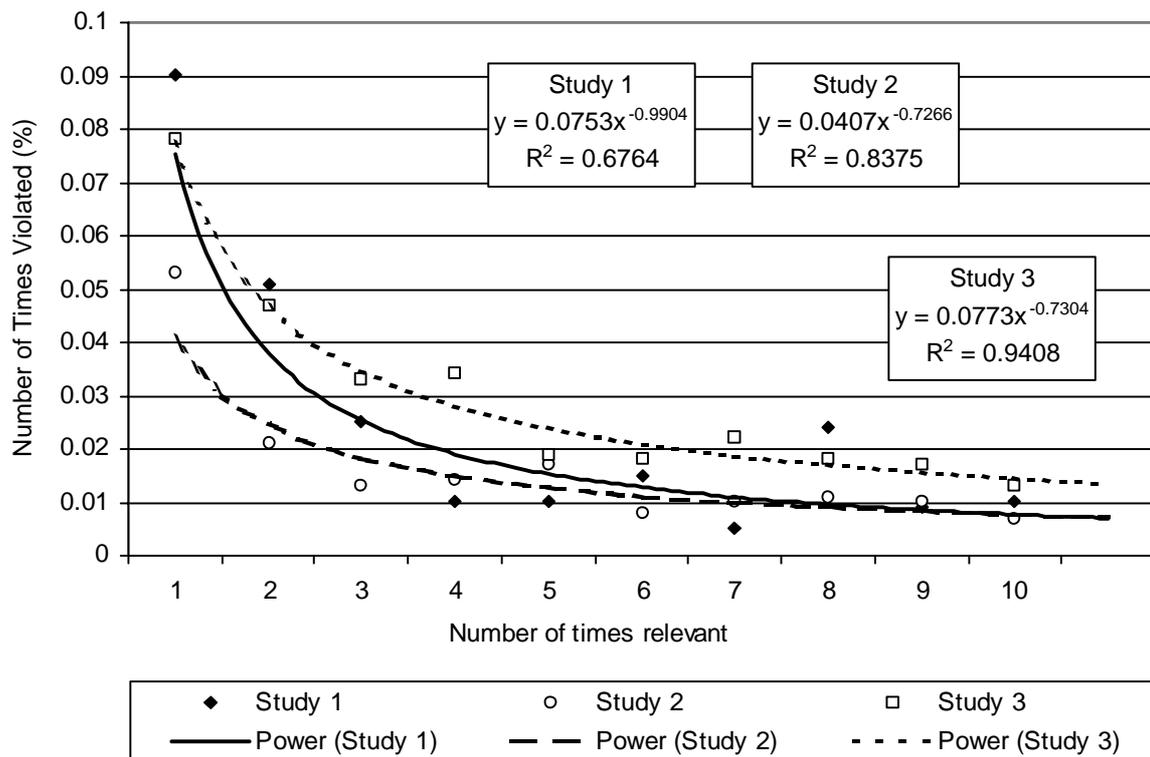


Fig. 5. Mastery of constraints

compute the probability of violating a given constraint C . To estimate this quantity, we computed, for each student, the proportion of constraints that he or she violated on the first occasion of application, the second occasion, and so on. These proportions were averaged across all students and plotted as a function of the number of occasions when C was relevant. Figure 5 shows the results of this analysis for all three evaluation studies.

As the number of uses increases, the set of constraints that were relevant for that number of times diminishes in size. At $n=10$, the constraint set has, on average, dropped to 32% of the original set, while at the end of each series (when the number of failures is zero) the set can be as low as 3% of the original. Hence, a single failure will have 30 times the impact on probability as at the start of the curve. We have arbitrarily chosen $n=10$ to reduce this effect. As can be seen, in all three studies we get a close fit to a power curve, which reinforces the conclusion stated in (Mitrovic & Ohlsson 1999): the probability of violating a constraint drops with the opportunity to practice it, and therefore students' knowledge of constraints does improve with practice. These results provide an additional proof of the sound foundations of CBM.

7. Study 1: Classroom performance

Study 1 included elements of formative and summative evaluation. As stated earlier, we wanted to evaluate some components of the system (the interface and the constraint base), and we also wanted to evaluate CBM. Students' reactions to the system were summarized in section 5, while the previous section presented an analysis of how well CBM supports learning. Here we report on the effect of learning with the system on subsequent classroom performance.

Study 1 was voluntary; out of the 49 students enrolled in the course, twenty chose to participate. Therefore, this study was not a controlled one. At the end of the course, students sat an examination, which contained questions relevant to the domain of SQL-Tutor. This allowed for a comparison of competence of the two groups of students. The students who used SQL-Tutor achieved higher marks than the students in the control group, as illustrated in Table II.

The difference in means is significant ($t= 2.908$, $p = .006$). We computed the values for the effect size and power, the two measures commonly used to determine the effects and validity of experiments. In the ITS world, the common way to calculate the effect size is to divide the increase in means by the standard deviation of the control group (Bloom 1984). Using this formula, we get an effect size of 0.66. This result is comparable to those published in (Albacete

Group	Mean	Standard deviation
Experimental	82.75	8.76
Control	71.23	17.56
Total	76.24	15.39

Table II. Competence of the two groups in study 1

& VanLehn 2000); they report on the effect size of 0.63 in a similar setting, with students using their system in a single, 2-hour session. An effect size of this magnitude is common in remedial tutoring (Albacete & VanLehn 2000), while better results are obtained in longer studies. For example, (Anderson et al. 1995) report an effect size of 1.0 in studies that lasted for one semester, and Bloom (1984) reports an effect size of 2.0 for one-on-one human tutoring. An effect size of 0.66 after only a single session is therefore remarkable, and we believe even better results would be obtained for a longer-term study.

Another way of measuring the effect size is advocated in Chin (2000): the effect size is computed as the omega squared value, which gives the magnitude of change in the dependent variable due to changes in the independent variables. In our case, we get an effect size of 0.141, which is quite large.

The other measure, power (or sensitivity), gives an indication of how repeatable the experiment is. It gives the percentage of repeated experiments for the same design, effect size and number of subjects that would produce the given significance. Chin (2000) recommends that researchers strive for power of 0.8. In our case, the power at a significance threshold of 0.05 is 0.75, and we get the power of 0.8 for a significance threshold of 0.08.

Because this experiment was not controlled, and because some student kept using the system after the study, we cannot claim to have a definite proof of the quality of the system from these results. However, the competence of the experimental group is significantly higher.

8. Study 2: Evaluating feedback

The goal of the second study was to evaluate the effect of feedback given to students. The level of feedback determines how much information is provided to the student. There are six levels of

feedback in SQL-Tutor: “positive/negative feedback”, “error flag”, “hint”, “all errors”, “partial solution”, and “complete solution”. At the lowest level (*positive/negative*), the message simply informs the student whether or not the solution is correct. This type of feedback was illustrated in Figure 2. An *error flag* message informs the student about the clause⁵ in which the error occurred. *Hint* gives more information about the type of error, by specifying the general principle that has been violated. This description is taken directly from the constraint. A message of type *all errors* presents the hint messages for all errors the student has made. *Partial solution* displays the correct content of the clause relevant to the first violated constraint, while *complete solution* simply displays the pre-specified ideal solution for the current problem. Table III illustrates the messages that the student would get in the situation illustrated in Figure 2.

The level of feedback is adjusted in the following way. When a student starts working on a new problem, he/she receives only feedback of the *positive/negative* type. If the student goes through several unsuccessful solution attempts, the feedback is upgraded to the *error flag* level and then to the *hint* level. The system never volunteers more than a *hint*, but the student can ask for *partial* and *complete* solutions by clicking on the *feedback* button and selecting the desired option.

The mechanism of selecting feedback described is overly simple: it adapts to the student’s performance only in a rudimentary way. One of our goals is to develop a truly adaptive mechanism for selecting feedback types, which would tailor feedback to the constraint mastery. As an initial step towards this goal, we performed an evaluation of the effectiveness of various types of feedback available to students. Our initial hypothesis was that constraint-level feedback (*hint* or *all-errors*) would be most effective (that is, best support student learning). We hypothesized that *positive/negative* and *error-flag* feedback would be too general to be informative for students, and that *partial-solution* and *complete-solution* feedback would be counter-productive in many cases. When presented with the solution, the student might directly copy it, which results in a correct solution in the next submission. However, this would not stimulate students’ thinking, which is necessary to correct misconceptions in the long term.

⁵ If there are several mistakes in various clauses, the pedagogical module selects one of them.

Feedback type	Message
Error flag	Almost there – a few mistakes though. One of them is in the WHERE clause.
Hint	Make sure that you have listed all the necessary tables for this query. Consider all the attributes necessary in join conditions, search conditions, expressions to be retrieved, grouping and restricting grouping, and sorting.
All errors	<ol style="list-style-type: none"> 1. You have to use another table in this query! 2. When you compare the value of an attribute to a constant in WHERE, they must be of the same type. 3. You need another search condition, using a string constant! 4. Check whether you are using the appropriate string constants in WHERE! 5. Check that you have all the necessary string constants in WHERE! You need to specify more. 6. Check that you have all the necessary string constants in WHERE! You need to specify more.
Partial solution	WHERE DIRECTOR=(SELECT NUMBER FROM DIRECTOR WHERE FNAME='Stanley' AND LNAME='Kubrick')
Complete solution	<p>The correct solution of this problem is:</p> <pre> SELECT TITLE FROM MOVIE WHERE DIRECTOR=(SELECT NUMBER FROM DIRECTOR WHERE FNAME='Stanley' AND LNAME='Kubrick') </pre>

Table III. Examples of various types of feedback messages

Study 2 involved two versions of the system. The first provided feedback of type *positive/negative* and *error flag* only (we refer to this version as *limited*), and the second

generated all levels of feedback (*full version*). We analyzed the logs collected during the evaluation session in several ways, and present the results in the following subsections.

8.1. Probability of constraint violation

The first analysis we performed focused on the learning performance. Study 1 showed that the degree of mastery of a given constraint is a function of the amount of practice on that constraint. We also wanted to determine whether feedback would influence mastery of constraints, and so analyzed, for the two feedback groups, the probability of violating a given constraint C for the n^{th} problem for which the constraint is relevant. To estimate this quantity, we computed, for each student, the proportion of all constraints that he/she violated in the first problem, the second problem, and so on. These proportions were averaged across all subjects and all constraints. This is the same type of analysis as presented in Figure 5.

Figure 6 illustrates the learning performances of the two groups of students. As in section 6.1, we have used a cutoff of $n=10$ to reduce the statistical effects that result from the number of constraints in each set becoming very small. When the *full* group is compared to *limited*, the latter has the higher learning rate. However, the existence of the two groups does not allow us to evaluate our hypothesis, because the *full* group received *partial/complete* solutions (the detrimental feedback according to our hypothesis) as well as the “good” feedback (*hints/all-errors*). We therefore post hoc split the *full* group into two groups: a *solution* group who used *partial* and *complete* feedback predominantly, and a *hint* group who used mainly *hint* and *all-errors*. Only students who used the desired feedback type more than 85% of the time when requesting further feedback were included; in fact, most of the students in these groups requested the required feedback type 100% of the time. Of 32 students in the *full* group, 16 were placed into one or other of these two groups, with the remainder being discarded because they used a mixture of feedback types.

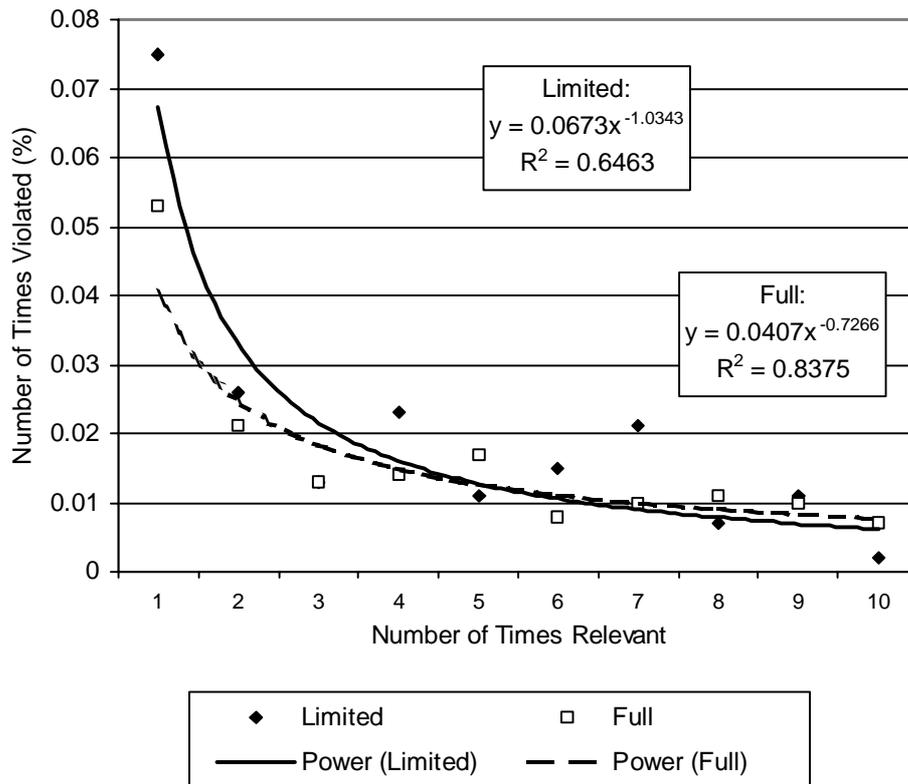


Fig 6. The probability of constraint violation for the *full* and *limited* groups

An analysis of the rate of learning of the three groups (*limited*, *hint* and *solution*) is given in Figure 7. The *solution* group has the poorest fit to the power curve, which might be explained by a lot of copying going on, which does not initiate deep thinking and learning. This suggests that being shown a solution is detrimental to the rate of learning.

However, it is important here to consider possible sources of extraneous effects. In most cases, the group that begins with the highest error rate also has the highest learning rate. The group with the highest initial error rate (which is independent of feedback) will therefore display the highest initial learning rate.

Further, for the *hint* and *solution* groups, the students chose the feedback level, while for the *limited* group it was artificially determined. It is possible that any trends observed are not because of the effects of feedback, but reflect a characteristic of the students that choose that feedback level.

We also computed a few statistics on the three groups, given in Table IV. The *solution* group solved most problems on average; however, this may be because the students in this group predominantly selected *partial* and *complete solution* (witnessed in their logs), which enabled them to copy correct solutions that they may not otherwise have arrived at. It is much more important that the students in the *hint* group needed only 2.17 attempts per problem, compared to 2.21 and 2.25 attempts on average for the *solution* and *limited* groups. Also, the amount of time per attempt is shortest for the *hint* group, which supports our hypothesis. The difference in times for the *limited* and *hint* groups is significant ($t=1.87$, $p=0.08$), with an effect size of 0.063, and a power of 0.33. This suggests that the “good” feedback, i.e. the feedback messages provided from the constraints, was easier to absorb, and so the time required to understand the feedback and make the necessary changes was substantially reduced. The *variation* in time required is also

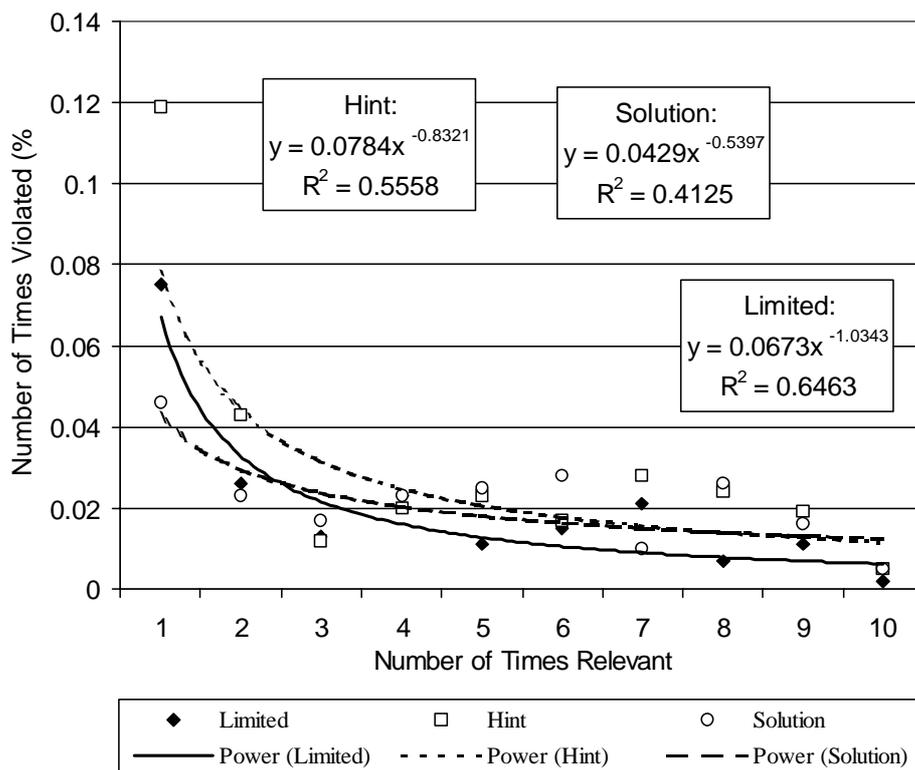


Fig. 7. The probability of constraint violation for the three groups

Group	Population	Solved	Number of attempts		Time/attempt	
			Mean	SD	Mean	SD
Solution	9	87.07%	2.21	0.49	65.26s	66.45
Hint	7	83.49%	2.17	0.65	47.81s	11.20
Limited	21	84.10%	2.25	1.38	78.06s	64.31

Table IV. Statistics for the three groups

heavily reduced, so the worst examples from both the *limited* and *solution* groups lie many standard deviations outside the distribution for the general group.

8.2. Effect of the feedback on violated constraints

Our hypothesis has a corollary that effective feedback on a violated constraint will increase the chance of that constraint being used successfully the next time. We therefore analyzed the effect of feedback received for a violated constraint on the *next* attempt/problem for which the same constraint is relevant. If a particular type of feedback is better than another, we expect to see an increase in the probability that the constraint is used correctly the next time, because the student is more likely to have learned the constraint.

We determined the frequency of a constraint being used successfully after being violated, with respect to a particular level of feedback received on it. Because some feedback types are intended to refer only to the first violated constraint (*error flag, hint, partial solution*), the other violated constraints were treated as having received a level of feedback higher than positive/negative, but lower than any of the other feedback types. This is because the other constraints may indirectly receive feedback (e.g. if they relate to the same clause, and so the same partial solution applies), but at a level which is unknown and variable.

Table V presents the frequencies of successful application of a constraint to the next attempt in solving the current problem, after receiving feedback of a specific type. We have used all available data to produce this table. The *Const* column gives the total number of constraints that

Feedback	Const	Success	Failure	Success%	Learned
Pos/neg	436	254	636	29%	78.0%
Error flag	116	98	126	44%	81.8%
Hint	43	33	43	43%	74.4%
All errors	64	72	81	47%	80.0%
Partial sol	26	22	10	69%	91.6%
Full sol	18	6	16	27%	44.2%

Table V. The effect of feedback on whole sessions

were violated and followed by feedback of a certain type (i.e., the number of different constraint/feedback pairs). In some cases, the same pair appears several times in a log, indicating that the constraint was violated on more than one problem attempt, for which the same feedback type was requested. *Const* gives the number of different pairs, not the number of their appearances in the logs, and so is a measure of the constraint population size for this feedback type. *Success* is the number of successful applications of the same constraint in the next attempt, while *Failure* specifies the number of times the same constraint was violated following the feedback. Therefore, the total number of pairs for a specific feedback type can be obtained as the sum of *Success* and *Failure*. The most frequent type of feedback was positive/negative (a total of 890 messages), while full solution was only given on 22 occasions.

Success% gives the percentage of successful applications of the constraint to the next attempt following the feedback. The highest value of *Success%* is obtained for *partial solution*; however, this does not necessarily mean that the students have learnt the constraint from such a feedback message. Instead, students may simply retype the given solution fragment and submit it. Therefore, there may not be any real learning involved. After *partial solution*, the next best feedback type is *all errors*, followed closely by *error flag* and *hint*. However, these three types of feedback were offered in very different proportions, with 224 *error flag* messages, 153 messages of the *all errors* type, and only 76 *hint* messages. Only 27% of the solutions made in the attempt following *full solution* are correct, so this type of feedback is counter-productive. The

last column (*Learned*) gives the percentage of correct applications of the constraint following the feedback in *any* future problem. *Partial solution* again has the highest percentage here, but it has only been offered 32 times, which is much less than the number of messages generated for the other types of feedback. Most significantly, *full solution* leads to a substantial drop in subsequent long-term performance.

8.3. Focusing on single feedback type

In the previous subsection, we divided all the students logs into three groups (*limited*, *hint* and *solution*) according to the predominant type of feedback used. The three groups were then compared. We cannot reach definite conclusions from such an analysis because of problems in the experimental design, since the students in the *hint* and *solution* groups received messages of other types in addition to the predominant ones. Brevity of the sessions was another problem, resulting in a relatively small number of occasions where the same constraint was used more than once.

In this subsection we report on another kind of analysis, performed on the level of individual attempts. Instead of taking a whole student session as the unit of analysis, we now take each *constraint* as the unit, classified each according to the type of feedback obtained for it. Therefore, the set containing all instances of *hint* messages consists of attempts made by various students, with no regard to the version of the system they used in the study, where all constraints in the set had *only* the desired feedback requested for them. Constraints that received mixed feedback were discarded. Hence, the datasets are very small.

We performed the same kind of analysis reported in section 6: we analyzed the probability of violating a constraint each time that it was relevant, for different types of feedback obtained on the constraint on previous occasions. Some feedback types (*error flag*, *hint*, *partial solution*) apply only to a subset of the constraints, and are intended to target the first constraint failed. In such cases, any other constraints failed during this attempt are given a feedback level of *positive/negative*. We report results only up to the point where the number of instances being considered is still at least 33% of the starting size of the set.

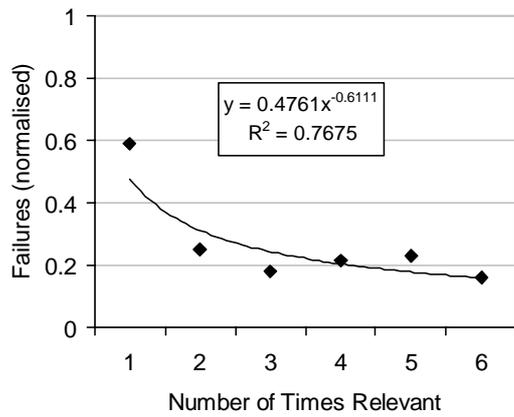
For the curves shown in Figure 8, the initial learning rate (i.e. the slope at $n=1$) is highest for *all errors* (0.44) and *error flag* (0.40) messages, closely followed by *positive/negative* (0.29) and

hint (0.26). The learning rates for *partial* (0.15) and *full solution* (0.13) are low. This supports our hypothesis that our CBM-based general feedback is superior to offering a correct solution. However, it is important to remember that the number of instances in each set was very small.

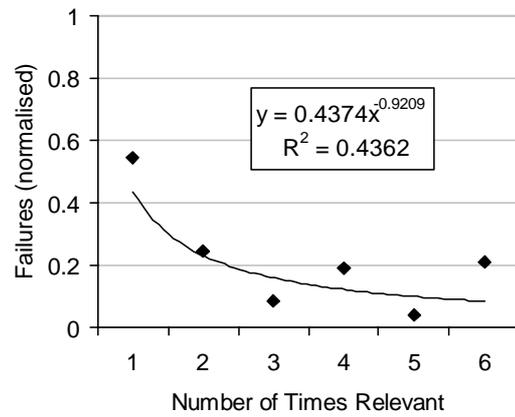
8.4. Discussion

The analysis of the data gathered in study 2 suggests feedback that presents information about general domain principles (e.g., *hint* and *all errors*) is preferable. These feedback types yield the shortest time per attempt and the fewest attempts per solved problem. These two feedback levels also give the highest rate of learning when analyzing the individual attempts.

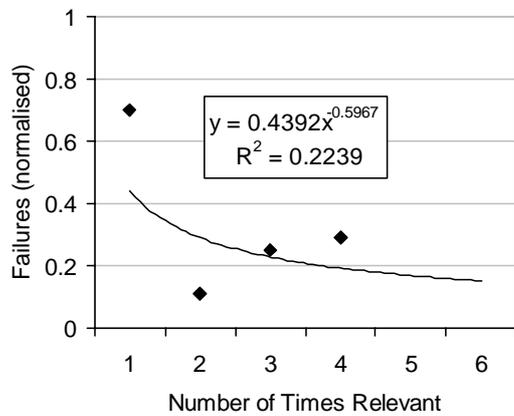
Because of the problems in experimental design previously discussed, and the high level of uncertainty inherent in all projects dealing with human subjects, our conclusions are not irrefutable. However, we believe that it is absolutely critical to perform evaluations of this kind in all ITS-related projects and that we have made a small contribution in identifying and dealing with some of the caveats that await researchers.



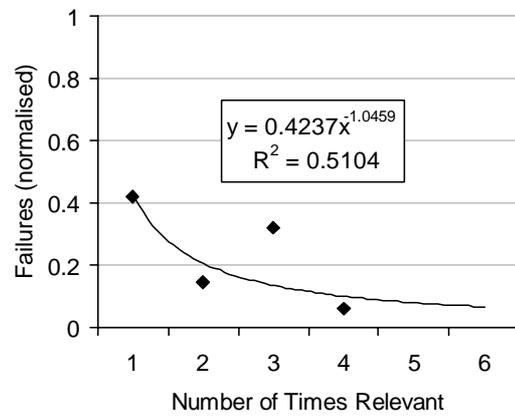
a) Positive/Negative



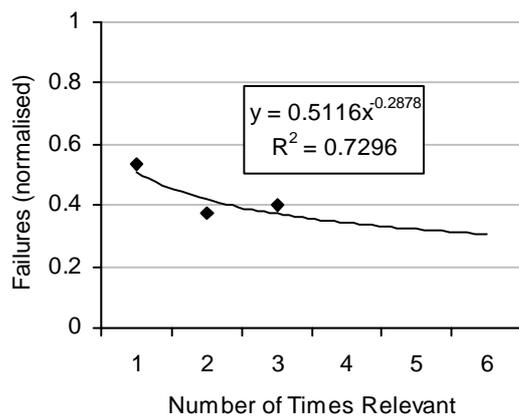
b) Error Flag



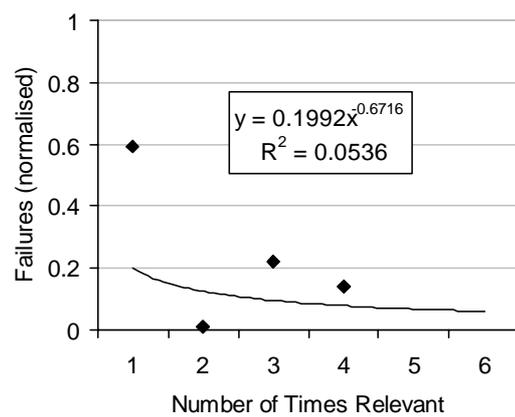
c) Hint



d) All Errors



e) Partial Solution



f) Complete Solution

Fig. 8. The probability of constraint violation after feedback

9. Study 3: Evaluating a probabilistic student model

As described previously, the initial extension of CBM for use in SQL-Tutor involved a long-term model of student's knowledge expressed as an overlay over the constraint base. This simple model was used to develop a heuristic problem-selection strategy: when the student asks the system to select a problem, SQL-Tutor examines the student model, identifies the focus constraint (that which has been violated most often) and selects a problem that is relevant to the focus constraint from the pool of unsolved problems whose level is within +1 or -1 of the student's current level.

This problem selection strategy is overly simple. In studies 1 and 2, selected problems were often either too complex or too simple for the student, or they jumped to another part of the domain seemingly not connected to the previous problem. We therefore wanted to explore other approaches to long-term modeling of student knowledge, and to develop new strategies for generating pedagogical actions based on such models.

The solution we have developed is a probabilistic model of students' long-term knowledge. Bayesian networks (Charniak 1991, Pearl 1988) are tools for representing and reasoning about uncertain knowledge using Bayesian probability theory. In the next subsection, we describe the probabilistic student model developed for SQL-Tutor. On the basis of this model, we have designed a problem selection strategy, which is described in the rest of this section.

9.1. The probabilistic student model

Before Bayesian networks could be applied to the task of problem selection, SQL-Tutor's student model had to be reformulated in probabilistic terms. The new student model consists of a set of binary variables $Mastered_1, Mastered_2, \dots, Mastered_n$, where n is the total number of constraints. Each variable can be in the state *YES* or *NO* with a certain probability, indicating whether or not the student has mastered the constraint.

Initial values for $P(Mastered_c = YES)$ were determined by analyzing the student logs from the previous study. We identified how many times each constraint was relevant and how many times it was satisfied, the quotient of the two numbers giving the initial probability. The logs were only analyzed up to the point where the user gets the first constraint-specific feedback about c . This ensured that the effects of learning did not bias the initial probabilities. Some

constraints did not appear in the past SQL-Tutor logs, because they were either too new or had never been used. For these constraints, $P(\text{Mastered}_c = \text{YES})$ was initialised to 0.5.

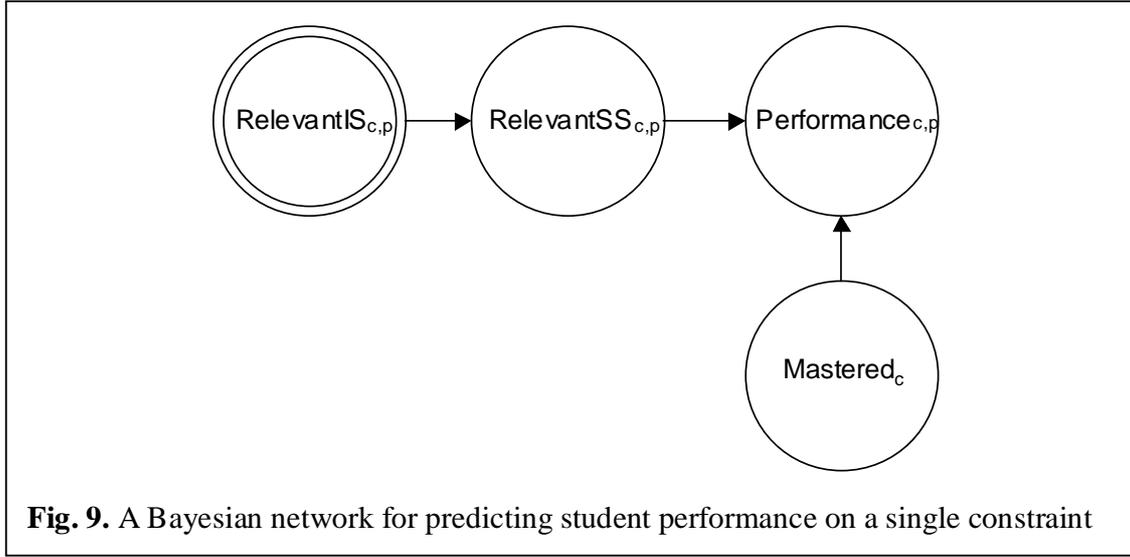
The student model is updated after the student submits his/her solution to a problem and receives feedback. The system currently uses the heuristics in Table VI to update the probabilities. The reason for manipulating the probabilities by percentages rather than real values is that percentage changes are discounted near the extremes of the probability interval. For example, if $P(\text{Mastered}_c = \text{YES}) = 0.4$, then satisfying c would result in a much more significant change to the student model than if $P(\text{Mastered}_c = \text{YES})$ had been, for example, 0.95.

<p>If constraint c is satisfied, then $P(\text{Mastered}_c = \text{YES})$ increases by 10% of $(1 - P(\text{Mastered}_c = \text{YES}))$.</p>
<p>If constraint c is violated and no feedback about c is given, then $P(\text{Mastered}_c = \text{YES})$ decreases by 20%.</p>
<p>If constraint c is violated but feedback is given about c, then $P(\text{Mastered}_c = \text{YES})$ increases by 20% of $(1 - P(\text{Mastered}_c = \text{YES}))$.</p>

Table VI. Heuristics used for updating the student model

9.2. Predicting student performance on single constraints

We use a simple Bayesian network given in Figure 9 to predict the performance of a student on a single constraint c , while solving a problem p . This network uses several variables (shown as the nodes in the network) to predict a student's performance $\text{Performance}_{c,p}$. The $\text{RelevantIS}_{c,p}$ variable specifies whether the given constraint is relevant to the problem's ideal solution. Since relevant constraints for the ideal solution are fixed, $\text{RelevantIS}_{c,p}$ is always known with certainty (hence the double circle).



$RelevantSS_{c,p}$ specifies whether the student has used the given constraint in his/her solution. Very often, students use the same approach to solving the current problem as that used in the ideal solution. In other cases, different approaches to solving the same problem are related to each other. Therefore, $RelevantSS_{c,p}$ depends on $RelevantIS_{c,p}$, which is illustrated in Figure 9 by an arc connecting these two nodes.

$Mastered_c$ comes from the student model, and represents the system's estimate of the student's knowledge of a particular constraint. This variable determines (along with the relevance of the constraint to the student's solution) the student's performance on the constraint. $Performance_{c,p}$ is a three-valued node taking values *SATISFIED*, *VIOLATED* or *NOT-RELEVANT*.

A full specification of this Bayesian network requires prior and conditional probabilities. $P(Mastered_c)$ and $P(RelevantIS_{c,p})$ are the prior probabilities, which are already available from the student model and problem database respectively. Table VII specifies the conditional

$RelevantSS_{c,p}$	$RelevantIS_{c,p} = \text{YES}$	$RelevantIS_{c,p} = \text{NO}$
YES	α_c	β_c
NO	$1 - \alpha_c$	$1 - \beta_c$

Table VII. Distribution of the conditional probability $P(RelevantSS_{c,p}|RelevantIS_{c,p})$

probability distribution over RelevantSS, which is the probability of constraint c being relevant to problem p 's student solution, when the constraint is/is not relevant to p 's ideal solution. In this table, α_c and β_c are properties of the constraint c . α_c (β_c) is the probability of a constraint being relevant to the student's solution if it is (not) relevant to p 's ideal solution. Effectively, α_c and β_c provide a measure of the “predictive usefulness” of the ideal solution. For example, when $\alpha_c = \beta_c = 0.5$, the relevance of c to the ideal solution tells us nothing about the relevance of c to a potential student solution. However, if $\alpha_c = 0.9$ and $\beta_c = 0.1$, there is a high probability that constraints relevant to the ideal solution will also be relevant to a student solution, and vice versa.

Like the initial probabilities of mastery, we determined values for α_c and β_c from past SQL-Tutor logs. However, these conditional probabilities were not available *directly* from the data. All that can be determined from the logs was the frequencies with which constraints were relevant to the ideal solution (IS), the student solution (SS) or both. Derivation (1) shows how α_c was calculated using the chain rule. A similar calculation was done for β_c . For new or previously unused constraints, α_c and β_c were initialized to 0.5.

$$\begin{aligned}
 \alpha_c &= P(\text{RelevantSS}_{p,c} = \text{YES} \mid \text{RelevantIS}_{p,c} = \text{YES}) & (1) \\
 &= P(\text{RelevantSS}_{p,c} = \text{YES} \ \& \ \text{RelevantIS}_{p,c} = \text{YES}) / P(\text{RelevantIS}_{p,c} = \text{YES}) \\
 &= \# \text{ times } c \text{ is relevant to both SS and IS in the logs} / \# \text{ times } c \text{ is relevant to IS}
 \end{aligned}$$

Table VIII gives the conditional probability distribution of $Performance_{c,p}$ given its parent variables $RelevantSS_{c,p}$, and $Mastered_c$. $Slip_c$ ($Guess_c$) are defined as the probability of a student who has mastered (not mastered) c slipping (guessing) and violating (satisfying) the constraint. In the third and fourth columns of Table VIII, $P(Performance_{c,p} = NOT-RELEVANT) = 1.0$ and the other entries are 0, because these represent the two scenarios where $RelevantSS_{c,p} = NO$ (i.e. c is not relevant to the student solution). The four columns represent situations where the values of the parent nodes are known with certainty. In practice, the values of the parents will not be known with certainty.

	Relevant/Mastered			
$Performance_{c,p}$	YES/YES	YES/NO	NO/YES	NO/NO
SATISFIED	$1-Slip_c$	$Guess_c$	0	0
VIOLATED	$Slip_c$	$1-Guess_c$	0	0
NOT-RELEVANT	0	0	1	1

Table VIII. Conditional Probability Distribution of $P(Performance_{c,p}/RelevantSS_{c,p}, Mastered_c)$

The Bayesian network is used to predict the probabilities of the student violating, satisfying or not using c in his/her solution to p . A simple example will illustrate the evaluation process. Let us take the following constants: $\alpha_p = 0.9$, $\beta_p = 0.1$, $Slip_c = 0.3$, $Guess_c = 0.05$. Now, suppose that c is relevant to problem p 's ideal solution (i.e. $P(RelevantIS_{c,p} = YES) = 1$) and the student is not likely to have mastered c (e.g. $P(Mastered_c = YES) = 0.25$). An evaluation of the network yields the probability distribution [$P(Performance_c = VIOLATED) = 0.709$, $P(Performance_c = SATISFIED) = 0.191$, $P(Performance_c = NOT-RELEVANT) = 0.1$].

9.3. Selecting problems

A single problem requires mastery of many constraints before it can be solved. The number of relevant constraints per problem ranges in SQL-Tutor from 78 for the simplest problems, to more

than two hundred for complex ones. It is therefore necessary to select an appropriate problem for a student on the basis of his or her current knowledge.

We determine the value of a problem by predicting its effect on the student. If the student is given a problem that is too difficult, he/she will violate many constraints. When given a simple problem, they are not likely to violate any constraints. A problem of appropriate complexity is the one that falls into *the zone of proximal development*, defined by Vigotsky (1978) as “the distance between the actual development level as determined by independent problem solving and the level of potential development as determined through problem solving under adult guidance or collaboration with more capable peers”. Therefore, a student should be given a problem that is slightly above their current level but not so difficult as to discourage the student.

Let us discuss the strategy we propose for selecting problems. Each violated constraint triggers a feedback message. If the system poses a problem that is too difficult, there will be many feedback messages coming from various violated constraints, and it is unlikely that the student will be able to cope with them all. If the problem is too easy, there will be no feedback messages, as all constraints will be satisfied. A problem of appropriate complexity will generate an optimal number of feedback messages. This is the basis of the evaluation function we propose.

The algorithm for evaluating problems is given in Figure 10. The function takes two parameters, the problem p to be evaluated and an integer, *OptimalFeedback*. It returns the value of p . *OptimalFeedback* is an argument specifying the optimal number of feedback messages the

```
int Evaluate(problem  $p$ , int OptimalFeedback) {
    int Feedbacks:=0;
    For every constraint  $c$  {
        Evaluate the Bayesian network;
        If  $P(\text{Performance}_{c,p} = \text{VIOLATED}) > 0.45$ 
            Then  $\text{Feedbacks} := \text{Feedbacks} + 1$ ; }
    Return (- |OptimalFeedback - Feedbacks|); }
```

Fig. 10. The problem evaluation function.

student should see regarding the current problem. Its value is currently set to the student's *level* + 2, reflecting the fact that novices are likely to cope well with a small number of messages at a time, while advanced students are able to resolve several deficiencies in their solutions simultaneously.

The evaluation function assumes that feedback will be generated for every constraint where $P(\text{Performance}_{c,p} = \text{VIOLATED}) > 0.45$. This heuristic is used because it is intractable to calculate the exact probability of a problem producing the optimal number of feedback messages. The value 0.45 was chosen because initial tests showed that it gave best the results. The problem with the highest value is selected from the pool of unsolved problems within 1 level of the student's level.

9.4. Results

All actions students performed in the study were logged, and later used to analyze the effect of the proposed problem-selection approach on learning. Both groups of students had two ways of selecting problems; they could go through all problems in order, or they could ask the system to select an appropriate problem based on the student model. In the case of the control group, the system first identifies the focus constraint (as described at the beginning of this section), and then selects a problem for which that constraint is relevant. The selected problem must be one of the unsolved problems, and its level of complexity must be within +1 or -1 of the student's current level. The Bayesian approach was used to select the best problem for the experimental group.

In order to evaluate the proposed problem selection method, we identified the logs of students who used *system's choice* in both groups. Six students from the experimental group attempted 36 problems selected by *next problem* and 38 problems selected by *system's choice* using the new Bayesian approach. Thirteen students from the control group worked on 106 and 79 problems selected by *next problem* and the original *system's choice* respectively.

Both problem selection strategies try to keep the student focused on a concept they are having difficulty with. However, as stated earlier, the focus of the heuristic-based problem selection strategy is too narrow, and it often selects problems which are either of the wrong difficulty, or which introduce new concepts as well as the target concept. We would therefore expect it to cause an increase in the problem solving effort, when compared to "next problem", since the default ordering is of (roughly) increasing difficulty, and introduces new concepts in a fairly orderly fashion. Conversely, we would expect a decrease in required effort when students use the Bayesian problem selection strategy, because it overcomes these difficulties, while introducing concepts in an order appropriate to the student. The results in Table IX illustrate this: for the students in the control group, the problems selected by the heuristic problem selector ("system's choice") are significantly more difficult than those selected by the "next problem" option ($t=2.69$, $p=.015$, power is 0.73 for the significance threshold of 0.05). Conversely, the effort required by the students in the experimental group when using the Bayesian method decreases by a small, not significant, amount (0.47). Table IX also reports the "gain" caused by using "system choice", i.e. the decrease in effort that resulted. When we compare this for the two groups, we get a significant difference ($p=.007$, $t=3.08$). This statistic has a large effect size (0.36) and power of 0.83 at significance threshold of 0.05. Therefore, the results strongly suggest that the new problem solving strategy is superior in its ability to select problems of appropriate difficulty.

Group	Exper. (Bayes) (N=6, P=36)		Control (Heuristic) (N=13, P=106)	
	Mean	SD	Mean	SD
Average attempts				
Next problem	3.96	1.5	2.43	0.93
System's choice	3.49	1.12	3.96	1.82
Gain	0.47	1.26	-1.53	1.33

Table IX. Average number of attempts per solved problem

N -the number of students; P – the number of problems

The advantages of the Bayesian approach are clearer when we observe what happens during the problem solving session. The students start with simple problems, and progress to more complex ones. Figure 11 illustrates the average number of attempts students took to solve the i^{th} problem, for the experimental group. It can be seen that the initial problems selected by the *next problem* option are easier for students than those selected by the Bayesian approach. This fact is easily explained by the fact that the Bayesian network progresses faster to more complex problems. However, later problems selected by the Bayesian approach are more adapted to the student and therefore require fewer attempts to be solved. Figure 12 illustrates the number of attempts per problem for the students in the control group. The opposite trend is obvious here: students find the system-selected problems (which are selected by using the simple heuristic discussed at the beginning of this section) more challenging than those visited in turn.

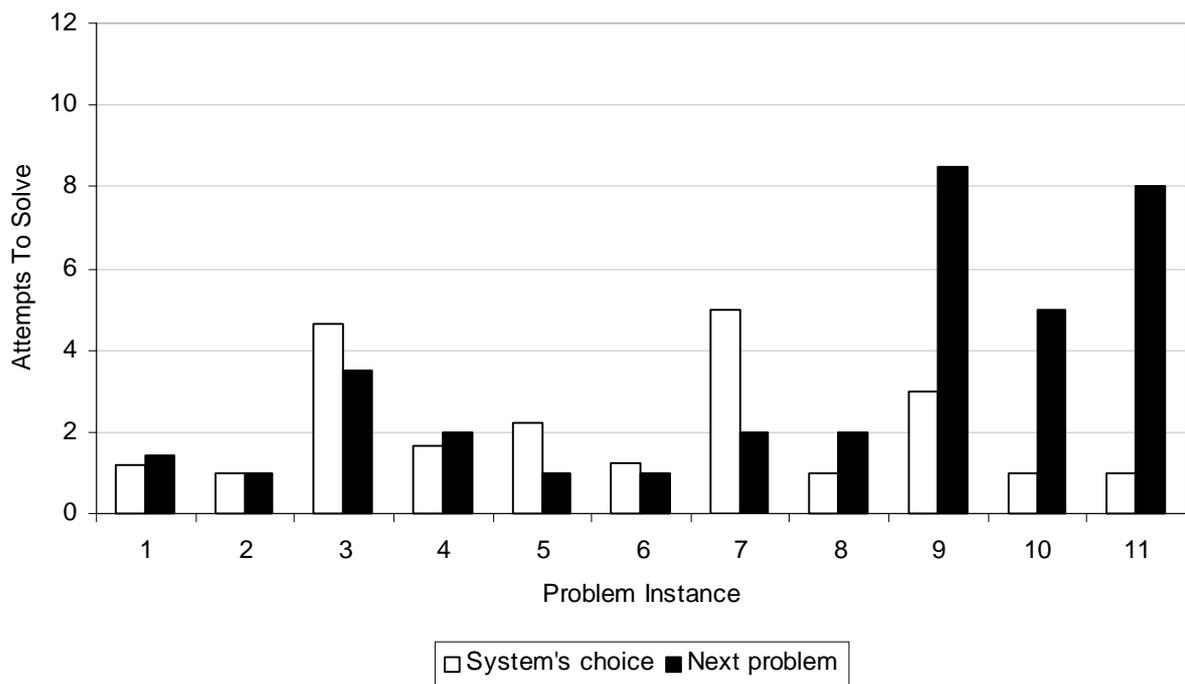


Fig. 11. Average number of attempts per problem for the experimental group

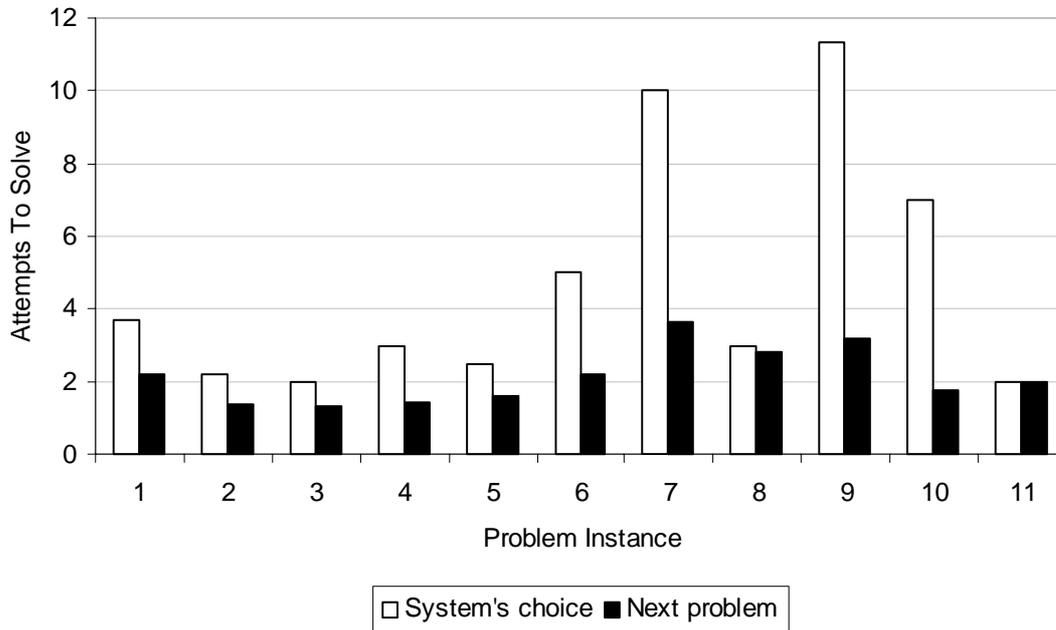


Fig. 12. Average number of attempts per problem for the control group

9.5. Pre/post tests

The pre- and post-tests, given in Appendices B and C, consisted of three multi-choice questions each, of comparable complexity. The marks allocated to the three questions were 1, 5 and 1 respectively. Nine out of fourteen students in the experimental group and sixteen out of eighteen in the control group submitted valid pre-tests, the results of which are given in Table X. There is no significant difference between the mean scores of the two groups ($t=0.656$, $p=.52$), showing that the control and experimental groups contained a comparable cross-section of students. However, a number of factors, such as the short duration of the user study, the holding of the study during the last week of the year etc, conspired to result in a very small number of post-tests being completed. Because some students did not log off, they did not sit the post-test that was administered on a separate Web page. Only one student from the control group and four from the experimental group sat the post-test. As the result, we can draw no conclusions from the post-test results.

Question	Exper. group	Control group
1	0.20	0.25
2	3.20	2.73
3	0.60	0.73
Total	4.00	3.50

Table X. Means for the pre-test

9.6. Related Work

Other researchers have proposed the use of Bayesian networks in ITSs. ANDES (Conati et al. 1997, Gertner 1998), an ITS for teaching Newtonian physics, uses Bayesian networks for predicting student performance and problem solving behavior. The ANDES network has a dynamic component, comprising nodes specific to the current problem, and a static component, comprising nodes representing the student's knowledge. The dynamic component is constructed on-line when a new problem is started. However, this approach relies on the system knowing *a priori* which rules can be relevant to the problem's solution. This is not the case in the SQL domain where the correct solution known by the system is only one example of a correct solution. The usefulness of the ideal solution in predicting the student solution is determined by the α_c and β_c parameters. Thus, in the SQL domain, we would be forced to model the entire domain for each problem.

One approach that does model the entire domain is Collins et al.'s (1996) hierarchical Bayesian network model for student modeling and performance prediction on test items. A similar hierarchical model was initially intended for our probabilistic student model. However, the key difference between our domain and Collins' example is that SQL-Tutor contains more than 500 constraints whereas Collins' example consists of only 50 questions. Initial investigations showed that it was infeasible to evaluate a traditional Bayesian network modeling more than 500 constraints on-line. Furthermore, Collins' example domain of elementary arithmetic divides neatly into 10 categories (e.g. addition theory, subtraction theory etc) whereas

in SQL there is no such simple classification of constraints, because they can be partitioned in many different ways.

Finally, Reye (1998) proposes a dynamic Bayesian network model for student modeling. Each variable, corresponding to a single knowledge item, is dynamically updated over time using Bayesian probability theory as the student's performance is observed. Again, this is a similar scheme to our student model where single constraints are represented by single nodes. However, Reye's model makes each knowledge item probabilistically independent. This simplification makes Bayesian student modeling tractable, but for solving decision tasks such as problem selection the probabilities do need to be combined. Reye does not show how this can be done, whereas this is the main emphasis of our approach.

9.7. Discussion

One of the vital tasks an ITS has to perform is to provide problems that are of appropriate complexity for the student's current knowledge. In our approach, a probabilistic, long-term student model is used to predict student performance on candidate problems. The value of a problem depends on the predicted number of errors the student is likely to make. Each error results in a feedback message. Novices are unable to deal with many feedback messages, while advanced students are, and therefore an optimal number of feedback messages can be established based on the current student's level. Of all available problems, we select the problem that generates the optimal number of feedback messages.

Initial evaluations indicate that the proposed solution is promising. However, we implemented several heuristics due to the inefficiencies of evaluating large Bayesian networks on-line. For example, both Table VI and Figure 10 depict heuristics used by the system. Ideally the system should use theoretically sound rules based on probability theory and/or decision theory. Future work will look at developing this further. Use of new technologies such as qualitative Bayesian networks (Chao-Lin & Wellman 1998), which are known to be much faster in their evaluation time than traditional Bayesian networks, may also make the development of large-scale Bayesian networks feasible.

Future research will also focus on other decision tasks that an ITS must solve. Problem selection is only one such task; others include topic selection, adapting feedback, hint selection,

and selective highlighting of text. We are working towards a general framework for solving these types of problems (Mayo 2000, Mayo & Mitrovic 2001).

10. Conclusions

In this paper we reported on three evaluation studies performed using SQL-Tutor. These studies included elements of formative evaluation, which enabled us to improve the system over time, and summative evaluation. We analyzed Constraint Based Modeling as a student modeling approach, and showed that CBM has sound foundations, and that it can be extended successfully to provide for long-term modeling of student's knowledge and to support pedagogical decision-making.

As Chin (2000) points out, there are many obstacles to evaluating user-adaptive systems. Even when the factors to be studied are identified, and the study is designed properly, there are numerous outside factors that can still influence the outcomes. All systems intended for humans suffer from problems with using different groups of users, who may have different backgrounds, (dis)abilities, learning styles, motivations etc, which are very difficult to account for. Chin also identifies other problems, such as the practice effect, problems caused by the experimental set up and the others. Stern and Sterling (1997) point out the enthusiasm factor with the opposite effect: they note that the enthusiasm of researchers and students involved in the evaluation of an experimental learning environment will contribute to the results of learning. Students generally react well to changes in classroom routine.

As we pointed out in the introduction, we believe that ITSs, as well as other educational systems, need to be evaluated in the context of genuine teaching activities. This is because such systems are intended for use in real classrooms, and so we need to evaluate them in their natural environment, in order to establish the real effects on students' learning. However, such evaluations face various problems, such as constraints on the timing of the study, unpredictable student behavior, and others. We have experienced many of these problems, as well as the problems identified by Chin (2000). In study 1, we reported on a significant improvement in performance of the experimental group in comparison to the control group. However, this study was not controlled, because we asked for volunteers. Furthermore, we cannot be absolutely sure that the students in the control group abstained from using the system; they might have heard about it from their peers. In study 2, we experienced problems with the experimental design,

which offered feedback of mixed types to the experimental group. It was therefore necessary to post-hoc split the experimental group into two subgroups, which had further negative effects on the validity of the results.

We appreciate Chin's (2000) effort to propose guidelines for designing evaluation studies and believe that many of them are generally applicable. However, in our experience, some of them are difficult or even impossible to apply to educational systems. For example, Chin suggests that subjects are put into groups randomly, with random allocation of times. We were constrained to scheduled lab times, and it was not an option to change the allocation of times to students. Furthermore, since the studies were carried out in departmental labs, we had no control of the environment, which is another of Chin's guidelines. In study 3, timing was a constraint, because students needed to get an overall understanding of databases prior to using the system. The only possible time for this study was the last week of lectures, which meant the very last day (Friday afternoon) of the school year for one set of students. This had a negative effect on the number of participating students, and on their motivation.

Another illustration of possible problems comes from study 3. We did not anticipate students logging off before filling out the post-test and, as a result, collected very few post-tests. We cannot, therefore, compare the performances of students on pre- and post-test. Also, we interacted with the students during the sessions, giving additional explanations, because our main goal was to help students to learn better. Our primary role as teachers is to help students learn, and that interferes with our goals as researchers.

All three studies involved a small number of students, and consequently we did not have enough data to thoroughly evaluate the system. The studies were short, consisting of a single 2-hour session per student, so a big improvement in students' performance cannot be expected. Further, we do not get many opportunities for evaluation, because there are only one or two relevant courses each year. The existence of a Web-enabled version of the system will provide us with more data, but it will be harder to analyze it, since little or nothing is known about the background of the users. If a Web-based version of the system is to be used in a study, then it will be necessary to identify critical variables and design a pre-test or a questionnaire on these variables.

Appendix B: Pre-test

The questions are based on the MOVIE table. Each movie has a unique number, which is the primary key of the table. Additionally, we have the title, the year of production (YEAR), the critic's rating (CRITICS), the type of movie, the number of Academy awards the movie was nominated for (AANOM) and has won (AAWON), and the number allocated to the director of the movie (DIRECTOR).

Please answer ALL questions:

1. We need to find the titles of all movies other than comedies. Will the following SQL statement achieve that? Yes/No

```
SELECT TITLE
FROM MOVIE
WHERE TYPE = NOT('comedy')
```

2. Now, we need to find the title of the movie that has won the most awards. Select ALL correct answers:

a)

```
SELECT TITLE
FROM MOVIE
WHERE AAWON = MAX(AAWON)
```

b)

```
SELECT TITLE
FROM MOVIE
GROUP BY NUMBER
HAVING AAWON = MAX(AAWON)
```

c)

```
SELECT TITLE
FROM MOVIE
WHERE AAWON = (SELECT MAX(AAWON) FROM MOVIE)
```

d)

```
SELECT TITLE
FROM MOVIE
GROUP BY TITLE
```

HAVING AAWON = (SELECT MAX(AAWON) FROM MOVIE)

- e) SELECT TITLE
FROM MOVIE
WHERE AAWON >= ALL (SELECT AAWON FROM MOVIE WHERE
AAWON IS NOT NULL)
3. We need to find the total number of awards won by comedies in 1983. Which of the following statements will achieve that?
- a) SELECT SUM(AAWON)
FROM MOVIE
GROUP BY TYPE
HAVING TYPE IN ('comedy') AND YEAR=1983
- b) SELECT SUM(AAWON)
FROM MOVIE
WHERE TYPE='comedy' AND YEAR=1983
- c) SELECT SUM(AAWON)
FROM MOVIE
WHERE TYPE='comedy' AND YEAR=1983
GROUP BY NUMBER

Appendix C: Post-test

The questions are based on the MOVIE table. Each movie has a unique number, which is the primary key of the table. Additionally, we have the title, the year of production (YEAR), the critic's rating (CRITICS), the type of movie, the number of Academy awards the movie was nominated for (AANOM) and has won (AAWON), and the number allocated to the director of the movie (DIRECTOR).

Please answer ALL questions:

1. We need to find the titles of all comedies or dramas. Is the following SQL statement correct? Yes/No

```
SELECT TITLE
FROM MOVIE
WHERE TYPE='comedy' OR 'drama'
```

2. What is the type of movie that had the highest number of movies made in 1980? Select ALL correct answers:

- a.

```
SELECT TYPE
FROM MOVIE
WHERE YEAR=1980
GROUP BY TYPE
HAVING MAX(COUNT(*))
```

- b.

```
SELECT TYPE
FROM MOVIE
WHERE YEAR=1980
GROUP BY TYPE
HAVING COUNT(*)>=ALL (SELECT COUNT(*) FROM MOVIE WHERE
YEAR=1980 GROUP BY TYPE)
```

- c.

```
SELECT TYPE
FROM MOVIE
```

```
WHERE YEAR=1980 AND COUNT(*) = (SELECT MAX(COUNT(*)) FROM
MOVIE WHERE YEAR=1980)
GROUP BY TYPE
```

- d.

```
SELECT TYPE, MAX(COUNT(*))
FROM MOVIE
WHERE YEAR=1980
GROUP BY TYPE
```
- a.

```
SELECT TYPE
FROM MOVIE
WHERE YEAR=1980 AND NUMBER = MAX (COUNT(*))
```
3. We need to find the total number of awards won by comedies in 1983. Select all of the following statements that will achieve that?
- a)

```
SELECT COUNT(*)
FROM MOVIE
WHERE YEAR IN (1981, 1982, 1983) AND TYPE='drama'
```
- b)

```
SELECT COUNT(*)
FROM MOVIE
WHERE TYPE='drama'
GROUP BY YEAR
HAVING YEAR=1983 OR YEAR=1982 OR YEAR=1981
```
- c)

```
SELECT COUNT(*)
FROM MOVIE
WHERE YEAR>=1981 AND YEAR<=1983
```

Acknowledgements

The work presented here was supported by the University of Canterbury research grant U6242. We thank Stellan Ohlsson for all the support and ideas he provided us with. The comments from the anonymous reviewers are most appreciated. We also thank our students, for putting their time and effort into trying out SQL-Tutor and commenting on it. This research could not have been done without the support of other past and present members of ICTG.

References

- Albacete, P. L. and K. VanLehn: 2000, 'The Conceptual Helper: an Intelligent Tutoring System for Teaching Fundamental Physics Concepts'. *5th International Conference on Intelligent Tutoring Systems*, Montreal, pp. 564-573.
- Anderson, J. R, Corbett, A. T., Koedinger, K. R., and R. Pelletier: 1995, 'Cognitive Tutors: Lessons Learned'. *Journal of the Learning Sciences*, **4**(2), 167-207.
- Bloom, B.S.: 1984, 'The 2 Sigma Problem: the Search for Methods of Group Instruction as Effective as one-to-one Tutoring'. *Educational Researcher* **13**, 4-16.
- Chao-Lin Liu and M. Wellman: 1998, 'Incremental Tradeoff Resolution in Qualitative Probability Networks'. *14th Conference on Uncertainty in Artificial Intelligence*, Madison, WI, pp. 346-353.
- Charniak E.: 1991, 'Bayesian Networks without Tears'. *AI Magazine*, Winter 1991, 50-63.
- Chin, D. N.: 2000, 'Empirical Evaluation of User Models and User-adapted Systems'. *User-Modeling and User Adapted Interaction*, **10**(4) (to appear).
- Collins, J. A., Greer, J. and S. X. Huang: 1996, 'Adaptive Assessment Using Granularity Hierarchies and Bayesian Nets'. *3rd International Conference on Intelligent Tutoring Systems*, Montreal, pp. 569-577.
- Conati C., Gertner, A., VanLehn, K. and M. J. Druzdzel: 1997, 'On-Line Student Modeling for Coached Problem Solving Using Bayesian Networks. *6th International Conference on User Modeling*, Sardinia, pp. 231-242.
- Forgy, C.L.: 1982, 'Rete: a Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem'. *Artificial Intelligence* **19**, 17-37.

- Gertner A. S.: 1998, 'Providing Feedback to Equation Entries in an Intelligent Tutoring System for Physics'. *4th International Conference on Intelligent Tutoring Systems*, San Antonio, TX, pp. 254-263.
- Mayo. M.: 2000, 'A Normative Framework for Intelligent Tutoring Systems', *5th International Conference on Intelligent Tutoring Systems*, Montreal (poster).
- Mayo, M., Mitrovic, A. and J. McKenzie: 2000, 'CAPIT: an Intelligent Tutoring System for Capitalization and Punctuation'. *International Workshop on Advanced Learning Technologies*, Palmerston North, NZ, pp. 151-154.
- Mayo, M. and A. Mitrovic: 2001, 'Optimising ITS Behavior with Bayesian Networks and Decision Theory'. Accepted for *International Journal on Artificial Intelligence in Education*.
- Millan, E. and J. L. Perez-de-la-Cruz: 2001, 'A Bayesian Diagnostic Algorithm for Student Modeling and its Evaluation'. *UMUAI*, this volume.
- Mitrovic, A.: 1998a, 'A Knowledge-Based Teaching System for SQL'. *ED-MEDIA'98*, Freiburg, pp. 1027-1032.
- Mitrovic, A.: 1998b, 'Experiences in Implementing Constraint-Based Modeling in SQL-Tutor'. *4th International Conference on Intelligent Tutoring Systems*, San Antonio, TX, pp. 414-423.
- Mitrovic, A. and K. Hausler: 2000, 'Porting SQL-Tutor to the Web'. *ITS'2000 workshop on Adaptive and Intelligent Web-based Education Systems*, Montreal, pp. 37-44.
- Mitrovic, A. and S. Ohlsson: 1999, 'Evaluation of a Constraint-Based Tutor for a Database Language'. *Int. Journal. on Artificial Intelligence in Education* **10**(3-4), 238-256.
- Mitrovic, A. and P. Suraweera: 2000, 'Evaluating an Animated Pedagogical Agent'. *5th International Conference on Intelligent Tutoring Systems*, Montreal, pp. 73-82.
- Newell, A. and P.S. Rosenblum: 1981, 'Mechanisms of skill acquisition and the law of practice'. In: J.R. Anderson (ed.): *Cognitive skills and their acquisition*, pp. 1-55.
- Ohlsson S.: 1994, 'Constraint-based Student Modeling'. In: J.E. Greer and G.I. McCalla (eds.): *Student Modeling: the Key to Individualized Knowledge-based Instruction*, pp. 167-189.
- Ohlsson, S.: 1996, 'Learning from Performance Errors'. *Psychological Review* **103**(2) 241-262.
- Pearl J.: 1988, 'Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference'. Morgan Kaufmann.
- Reye, J.: 1998. 'Two-Phase Updating of Student Models Based on Dynamic Belief Networks'. *4th International Conference on Intelligent Tutoring Systems*, San Antonio, TX, pp. 274-283.

Stern, L. and L. Sterling: 1997, 'Teaching AI Algorithms using Animations Reinforced by Interactive Exercises'. *Australian Computer Science Education Conference*, Sydney, pp. 78-83.

Suraweera, P. and A. Mitrovic: 2001, 'Designing a Constraint-based Tutor for Database Design'. To be presented at the 9th Int. Conf. on Human-Computer Interaction, New Orleans.

Vigotsky L.S.: 1978, '*The development of higher psychological processes*'. Cambridge, MA: Harvard University Press.

Content

1.	Introduction.....	2
2.	SQL-Tutor.....	3
3.	CBM.....	7
4.	Evaluation Studies.....	10
5.	Subjective evaluation.....	11
6.	Mastery of Constraints.....	13
7.	Study 1: Classroom performance.....	15
8.	Study 2: Evaluating feedback.....	16
9.	Study 3: Evaluating a probabilistic student model.....	27
10.	Conclusions.....	39
	Appendix A: User questionnaire.....	41
	Appendix B: Pre-test.....	43
	Appendix C: Post-test.....	45
	Acknowledgements.....	47
	References.....	47

List of Figures

Fig. 1 Architecture of SQL-Tutor

Fig 2. Interface of the Web-enabled version of SQL-Tutor

Fig. 3. An example constraint

Fig. 4. Responses from the user questionnaire

Fig. 5. Mastery of constraints

Fig. 6. The probability of constraint violation for the *full* and *limited* groups

Fig. 7. The probability of constraint violation for the three groups

Fig. 8. The probability of constraint violation after feedback

Fig. 9. A Bayesian network for predicting student performance on a single constraint

Fig. 10. The problem evaluation function

Fig. 11. The average number of attempts per problem for the experimental group

Fig. 12. The average number of attempts per problem for the control group

List of Tables

Table I. Some statistics about the subjective evaluation

Table II. Competence of the two groups in study 1

Table III. Examples of various types of feedback messages

Table IV. Statistics for the three groups

Table V. The effect of feedback on whole sessions

Table VI. Heuristics used for updating the student model

Table VII. $P(\text{RelevantSS}_{c,p} / \text{RelevantIS}_{c,p})$

Table VIII. $P(\text{Performance}_{c,p} / \text{RelevantSS}_{c,p}, \text{Mastered}_c)$

Table IX. Average number of attempts per solved problem

Table X. Means for the pre-test

VITAE

(1) Dr A Mitrovic:

Dept. of Computer Science, University of Canterbury, Private Bag 4800, Christchurch, New Zealand

Dr. A. Mitrovic is a Senior Lecturer at the Computer Science Department, the University of Canterbury, New Zealand. She received her B.E. degree in Electronic Engineering, M.Sc. and PhD degrees in Computer Science from the University of Nis, Yugoslavia, in 1987, 1991 and 1994 respectively. Dr. Mitrovic has worked in artificial intelligence in education, student modeling, machine learning and geographic information systems. She has authored over seventy technical papers.

(2) Brent Martin:

Dept. of Computer Science, University of Canterbury, Private Bag 4800, Christchurch, New Zealand

Brent Martin is a Ph.D. candidate in Computer Science at Canterbury University. He received his BSc in Computer Science and Physics from the University of Waikato in 1986, and his MSc in Machine Learning from the same university in 1995. His primary interests lie in the areas of intelligent tutoring systems and machine learning. He also has ten years industry experience as a software solution architect.

(3) Dr M Mayo:

Dept. of Computer Science, University of Canterbury, Private Bag 4800, Christchurch, New Zealand

Dr M. Mayo is a staff member of the Department of Computer Science at the University of Canterbury. He received a BA(Hons) in Computer Science from the University of Otago, New Zealand, in 1994, and a PhD in the same field from the University of Canterbury in 2001. His research interests lie in Intelligent Tutoring Systems and Artificial Intelligence. Parts of this paper summarise work done by Dr Mayo for his PhD thesis.