A Comparison of Model-Tracing and Constraint-Based Intelligent Tutoring Paradigms

Viswanathan Kodaganallur, Rob R. Weitz, David Rosenthal, *Department of Computing and Decision Sciences, Stillman School of Business, Seton Hall University, South Orange, NJ 07079, USA*
weitzrob@shu.edu

Abstract. Two approaches to building intelligent tutoring systems are the well-established model-tracing paradigm and the relatively newer constraint-based paradigm. Proponents of the constraint-based paradigm claim that it affords performance at levels comparable to that of model-tracing tutors, but with significantly less development effort. We have built both a model-tracing and constraint-based tutor for the same problem domain (statistical hypothesis testing) and report on our findings with the goals of evaluating proponents' claims, more generally contrasting and comparing the two approaches, and providing guidance for others interested in building intelligent tutoring systems. Principally we conclude that two characteristics of the problem domain are key in distinguishing the appropriateness of the approaches for a given problem domain. First, the constraint-based paradigm is feasible only for domains in which the solution itself is rich in information. There is no such restriction for model tracing. Second, model tracing demonstrates superiority with respect to the ability to provide targeted, high-quality remediation; this superiority increases with the complexity of the solution process goal structure. Finally, we observe that the development effort required to build a model-tracing tutor is greater than that for building a constraint-based tutor. This increased effort is a function of additional design requirements that are responsible for the improved remediation.

Keywords: Intelligent Tutoring Systems, Model-tracing Tutors, Cognitive Tutors, Constraint-based Modelling, Constraint-based Tutors, Comparison of Intelligent Tutors, Knowledge-based Systems

INTRODUCTION

There are many types of intelligent tutoring systems (Shute & Psotka, 1996). The differences between them, and in particular, their relative advantages and disadvantages are important to anyone interested in developing and/or fielding Intelligent Tutoring Systems (ITSs). This research evaluates and contrasts two prominent tutoring approaches. The first is the well-established model-tracing (MT) or cognitive tutor approach and the second is the relatively newer constraint-based modelling (CBM) paradigm.

ITSs are generally set in a problem solving context; the student is given a problem to solve and the tutor provides remediation as the student works on the problem. MT and CBM are based on fundamentally different assumptions about tutoring. MT is a *process*-centric approach wherein the tutor tries to infer the process by which a student arrived at a solution, and bases remediation on this. CBM, on the other hand, can be considered to be *product*-centric in that remediation is based solely on the solution state that the student arrived at, irrespective of the

steps that the student took to get there. Consequently, there are significant differences in how the two paradigms represent and use domain knowledge.

We have built both a model-tracing tutor (MTT) and a constraint-based model tutor (CBMT) for the same problem domain (statistical hypothesis testing) and report on our findings. Our focus is on 1) contrasting and comparing the two approaches and 2) furnishing guidance for others interested in building intelligent tutoring systems. Our perspective is that of practitioners, interested in building and deploying ITSs in the real world, and that of researchers interested in extending what is known about ITSs.

We address the following questions. Are there specific problem domain characteristics that make one or other paradigm more suitable? How exactly does the development effort differ between the two? Does one or other paradigm inherently provide scope for better remediation? How do tutors built using the two paradigms differ in terms of the content of their knowledge bases, and what are the implications of these differences?

The main contributions of this paper include:

- A comparison of the two paradigms based on implementations of tutors employing each approach for the same problem domain.
- A fine-grained analysis of the content of the knowledge bases of tutors built using the two paradigms.
- Identification of the key dimensions that help in characterizing problem domains suited for each approach.

Successful MTTs and CBMTs have been developed primarily, if not exclusively, through two research groups: MTTs at the Pittsburgh Advanced Cognitive Tutor Center at Carnegie Mellon University and CBMTs at the Intelligent Computer Tutoring Group at the University of Canterbury, New Zealand. (Detailed references are provided below.) We are not affiliated with either group and believe that our experiences can provide valuable insights to others interested in building ITSs.

A conference paper (Mitrovic, Koedinger & Martin, 2003) comparing the two paradigms appeared while this paper was under initial review. In addition to the point made in the previous paragraph, our work differs in the detail of analysis, the choice of attributes on which to focus, and in some fundamental conclusions.

Principally, we conclude that two characteristics of the problem domain are key in distinguishing the appropriateness of the approaches for a given problem domain. First, the constraint-based paradigm is feasible only for domains in which the solution itself is rich in information. There is no such restriction for model tracing. Second, model tracing demonstrates superiority with respect to the ability to provide targeted, high-quality remediation; this superiority increases with the complexity of the solution process goal structure. Finally, we observe that the development effort required to build a model-tracing tutor is greater than that for building a constraint-based tutor. This increased effort is a function of additional design requirements that are responsible for the improved remediation.

This paper is organized as follows. We first briefly describe the two intelligent tutoring paradigms and review their attributes. We then describe our problem domain and provide details of our implementations of the two tutors. Next, we compare the two paradigms. We conclude with a summary and suggestions for future research.

MODEL-TRACING TUTORS

MTTs have been fielded in a variety of domains including:

- College-level physics (Gertner & VanLehn, 2000; Shelby et al., 2001).
- High school algebra (Koedinger & Anderson, 1997; Heffernan & Koedinger, 2002; Heffernan, 2001).
- Geometry (Anderson, Boyle & Yost, 1985; Wertheimer, 1990).
- Computer programming (Corbett & Anderson 1993; Corbett, Anderson & O'Brien, 1995; Corbett & Bhatnagar, 1997).

A number of studies have assessed the impact on student performance of MTTs. In their evaluation of the PUMP algebra tutor, Koedinger and Anderson (1997) report improvement in student performance of one standard deviation as compared to students receiving standard classroom instruction. Students who used the Andes physics tutor performed 1/3 of a letter grade better on average than students who did not use Andes (Gertner & VanLehn, 2000). Anderson, Corbett, Koedinger and Pelletier (1995) indicate that students using an MTT for geometry improved their performance by more than one standard deviation. They also report on two evaluation studies of an MTT for the LISP programming language. In one study, "Students using the tutor completed the exercises 30% faster and scored 43% better on a post-test" than did students who didn't use the tutor. In the other, students completed the exercises 64% faster and scored 30% higher (though they indicate some caveats for these results). Other evaluation studies are reported in Koedinger and Anderson (1993) and Schofield, Evans-Rhodes and Huber (1990). Generally, the claim for MTTs is that their use results in as much as a one standard deviation improvement in student performance beyond standard classroom instruction.

The underlying paradigm has its origins in the ACT theory (Anderson, 1983; 1993). According to ACT, "acquiring cognitive knowledge involves the formulation of thousands of rules relating task goals and task states to actions and consequences." This paradigm can be seen as taking a "process centric" view since it tries to fathom the process that a student employs in arriving at a solution. An MTT is composed of expert rules, buggy rules, a model tracer and a user interface. Expert rules model the steps that a proficient individual might take to solve the problem in question. These include rules for decomposing a problem into subproblems (or "planning" rules) and rules that address the solution of atomic subproblems ("operator" or "execution" rules). Planning rules embody procedural domain knowledge and operator rules embody declarative domain knowledge.

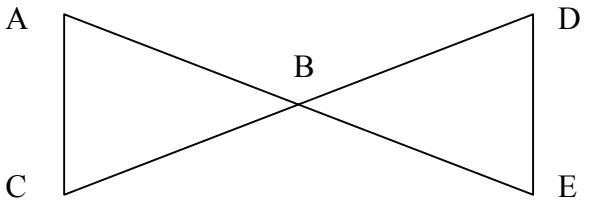
Two sample expert rules for the domain of geometry, drawn from Anderson et al. (1995) and Anderson and Pelletier (1991) are provided below:

IF The goal is to prove two triangles are congruent.

THEN Set as a subgoal to prove that corresponding parts are congruent.

IF The current goal is to prove that corresponding parts of triangle ABC and triangle DBE are congruent, and AB and BE are collinear and CB and BD are collinear (see diagram).

THEN Infer that angle ABC is congruent to angle DBE because they are vertical angles.



The first rule is a planning rule while the second is an operator or execution rule.

The following are example expert rules for the hypothesis testing domain. Hypothesis testing is a fundamental topic in inferential statistics; a later section provides more details on this problem domain.

IF	The goal is to solve the problem	IF	The goal is to determine the critical value
THEN	Set subgoals to determine: The critical value, and The sample statistic value.	THEN	Set subgoals to determine: The value of alpha, and The test statistic to use.

IF The appropriate test statistic to be used is the z statistic, and
The population standard deviation is unknown.

THEN The value of the test statistic is $z = \frac{\bar{x} - \mu}{s_{\bar{x}}}$, where $s_{\bar{x}} = s/\sqrt{n}$.

Here, the first two are planning rules and the third is an operator rule.

In MT, domain knowledge is captured in the form of many such rules. The crux of an MTT is to "trace" the student's input, where tracing consists of finding a sequence of rule executions whose final result matches the student's input. If the tutor is able to do this, it is taken to mean that the tutor has understood the process by which the student arrived at an answer. In order to identify student errors an MTT has a set of "buggy" rules that reflect common student misperceptions. If the tutor's trace of a student solution contains the application of one or more of these buggy rules, then the tutor provides the remediation associated with the buggy rule(s).

A sample buggy rule in our tutor follows:

IF The population standard deviation is unknown, and
The sample size < 30.

THEN The appropriate test statistic is the z statistic.

This rule models incorrect reasoning; for sample sizes less than 30 the appropriate test statistic is the t statistic. If the student's behaviour matches this buggy rule, the system concludes that the student does not understand this piece of declarative knowledge, and provides the remediation associated with the rule. Since MTTs can provide well-targeted remediation only

when one or more buggy rules are used in a successful trace, their tutoring capabilities depend on how well they capture the corpus of mistakes made by students.

While the ideal situation is that the tutor is able to trace all student inputs, this might not always be practical. Tutors should generally be able to trace all student inputs that are correct. For incorrect student input that the tutor is unable to trace, the best remediation might be a general response that something is wrong.

In general, there could be several alternate strategies to solve a problem. A particular tutor might choose to allow only one strategy, or support multiple strategies. In the latter case, the tutor must have expert rules and buggy rules for each strategy, and the model-tracing process should be able to map the student's input to a particular strategy (Shultze et al., 2000).

CONSTRAINT-BASED MODEL TUTORS

CBMTs have their theoretical underpinnings in the works of Ohlsson (1992; 1994) whose theories advocate relying on a student's errors in order to build a student model as well as to provide remediation. "The basic assumption, in stark contrast to philosophy of the model-tracing paradigm, is that diagnostic information is not hidden in the sequence of student's actions, but in the problem state the student arrived at" (Mitrovic, Mayo, Suraweera & Martin, 2001). Application areas for which CBMTs have been constructed and evaluated include:

- SQL database commands (Mitrovic & Ohlsson 1999; Mitrovic, 2003).
- Punctuation (Mayo, Mitrovic & McKenzie, 2000).
- Database modelling (Suraweera & Mitrovic, 2002).

Mitrovic and Ohlsson (1999) include an evaluation of the effectiveness of the SQL tutor. The post-study exam performance of students in the tutor group outperformed those not using the tutor by 0.75 standard deviations. These results are problematic however as the tutor group was comprised of self-selected volunteers and no figures are given regarding the pre-study relative performances of the two groups. A later study (Mitrovic, 2003) indicates an approximately 0.50 standard deviation difference between post-test performance of tutor-users and non-users, but this study is also characterized by self-selection and no pre-test comparison of the groups. Mayo and Mitrovic (2001) used three groups in their evaluation of two versions of the CAPIT punctuation tutor: a control group not using any tutor, a group using an initial version of the tutor, and a group using a version of the tutor that employs "a Bayesian network for long-term student modelling and decision theory to select the next tutorial action." While the pre and post-test scores of both the second and third groups increased, students in the control group "regressed." The authors provide an analysis concluding that groups B and C "improved significantly," and the improvement is "much more significant" for group C. An evaluation conducted by Suraweera and Mitrovic (2002) indicates that the difference in pre-test and post-test scores of students who used the KERMIT database modelling tutor versus those of a control group using "a cut-down version of the system" was statistically significant.

The central construct in CBM is that of a *constraint*. A constraint specifies certain conditions that must be satisfied by all correct solutions. When a student's work violates a constraint, we gain specific information about the student's mental model. The paradigm does not

consider it important to know how the student arrived at a specific problem state; what is important is simply the set of violated constraints.

A CBMT has numerous constraints pertaining to the domain in question. Not all constraints are relevant to all problems and to all problem states that a student arrives at. For example, in the domain of hypothesis testing, there are some constraints that are relevant only to problem instances in which the population standard deviation is known. Further, a constraint that checks the correctness of the sample mean calculated by the student is relevant only after the student has calculated the sample mean. For each constraint there is a *relevance condition* that specifies when the constraint is relevant, and a *satisfaction condition* that specifies a condition that should hold for any correct solution satisfying the relevance condition.

More formally, a constraint is an ordered pair <Cr, Cs> where Cr is the relevance condition and Cs is the satisfaction condition. A constraint can be interpreted as a logical statement of the form:

IF The relevance condition is satisfied by the problem state.

THEN The satisfaction condition should also be satisfied by the problem state, otherwise the student has committed an error.

Consider the following example from Ohlsson and Rees (1991) in the domain of algebra:

C_r $\frac{(x+y)}{d}$ is given as the answer to $x/d_1 + y/d_2$

C_s $d = d_1 = d_2$

If the solution provided by the student matches the relevance condition (C_r), it must be true that $d = d_1 = d_2$. If C_r is true, but C_s is violated (i.e., false), then the student has made an error. A constraint is violated if and only if its relevance condition is true and the satisfaction condition is false.

An example from our domain of statistical hypothesis testing is:

C_r The population standard deviation is unknown, and
The sample size ≥ 30, and
The student has input the test statistic to be used ("z" or "t").

C_s The value entered for the test statistic to be used is "z."

Using a CBMT, a student submits an input (which might be a complete or partial solution to the problem). The input, and some state variables that are part of the *a priori* problem definition, are passed on to the constraint checker, which identifies all violated constraints. If any constraints are violated the tutor provides remediation based on messages associated with each violated constraint. When a problem state does not violate any constraint, the tutor simply lets the student proceed. Whether or not the system provides feedback on each and every student action (like entering a value in a field) or only when the student explicitly submits a partial/full solution for evaluation, is a matter of design.

Because they lack planning capabilities, CBMT typically do not include a "solver" – a component that is able to solve the given problem by itself. The individual constraints encode pieces of domain knowledge, but there is no integrative mechanism to tie these together in a problem solving sequence.

While there is nothing in the CBM paradigm to indicate that a CBMT should always be supplied *a priori* with an "ideal" solution, this tends to be a characteristic of CBMT (Mitrovic et al., 2003; Martin, 2001, page 37).

THE HYPOTHESIS TESTING PROBLEM

Statistical hypothesis testing is widely used in decision making and in research, and is a fundamental component of undergraduate and graduate social science courses in statistics. A basic text covering the fundamentals is Levine, Stephan, Krehbiel and Berenson (2001). There are many types of hypothesis testing problems; the one for which we have built the two tutors is the case of hypothesis testing for the mean of a single population. Here the problem is to determine, based only on sample data, if the mean of some attribute of the members of a population is greater than, less than or equal to a given value. For example, suppose that a company is deciding on whether to open a retail outlet at a given location. The company's management has decided that it would be profitable to open an outlet only if the mean household income of the area is greater than $78,000. Determining the true – that is, population – mean is too expensive, time consuming and challenging. Therefore, a decision must be made on the basis of a sample.

To briefly illustrate the major issues in hypothesis testing, let's say that a sample of households is polled and the mean household income of the sample is $78,560. Is this enough to establish that the actual (population) mean household income in the town is greater than $78,000? After all, it is possible that the actual mean household income is less than or equal to $78,000, but due to the vagaries of sampling our mean value is greater. Generally, confidence in a decision will be increased if the sample size is large and if the standard deviation of the sample is low. Of concern as well is tolerance for error; here error means concluding that the population mean really is greater than $78,000 when it isn't, or concluding it isn't greater than $78,000 when it actually is.

It is possible to break down the goal of solving the problem into sub-goals as shown in Figure 1.

Each node in Figure 1 represents a goal, and the nodes emanating from a node represent its subgoals. The numbers attached to the subgoals indicate the sequence in which they need to be satisfied. In some instances, the ordering simply reflects pedagogically desirable sequencing. Ultimately, the decision to be made in a hypothesis testing problem is to either reject or not reject one hypothesis (for example, the population mean is less than or equal to $78,000) in favour of another (the population mean is greater than $78,000).

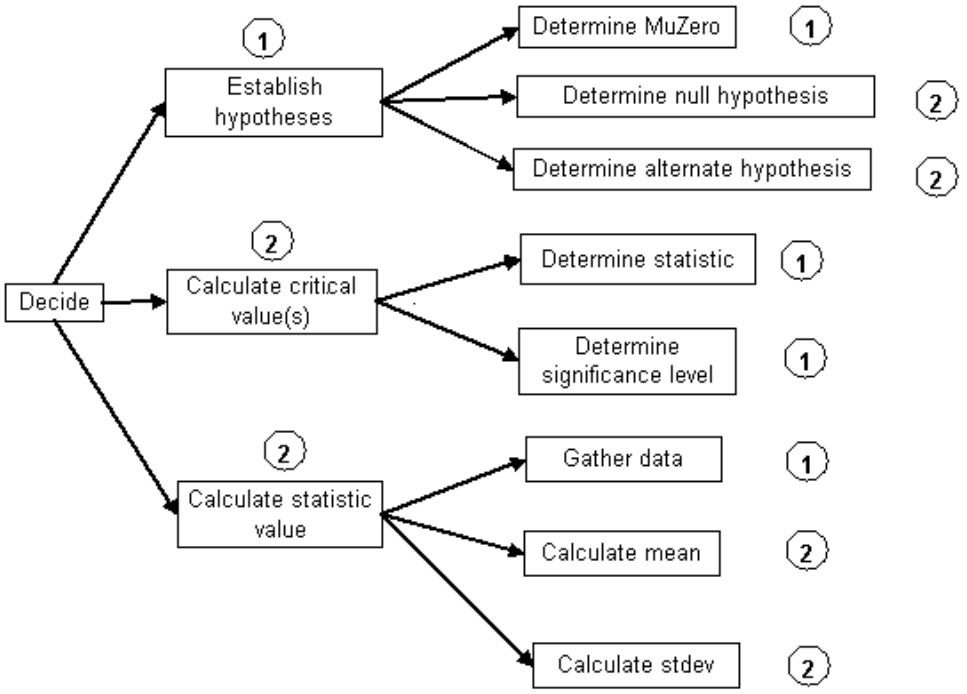


Fig. 1. Goal structure for hypothesis testing problems for mean of a single population.

OUR IMPLEMENTATIONS

We have implemented a CBMT and an MTT for the above problem domain. Both implementations use a relational database to store problem data. The CBMT also uses a relational database to store the constraints; all other aspects have been implemented in Java. The model-tracing tutor uses Java for the user interface and relies on the Java Expert System Shell (JESS) for its rule engine (Friedman-Hill, 2003).

User Interface

As our aim was to compare the two paradigms, we kept the user interface as similar as possible for the two tutors. The only differences are that the MTT has a "Guidance" tab and a "Guide" button (discussed below), which are absent in the CBMT.

When the tutor is started, a window appears showing the available problems (in a pane on the left). Our current implementations work at the level of individual problems; there is no sequencing of problems based on the student's performance. When a student selects a problem to work on, the tutor displays a problem pane (Figure 2). The problem pane has several tabs (the "Description" tab is selected in Figure 2) and the student is free to move among the tabs as needed. To start working on the selected problem, the student selects the "Work Area" tab.

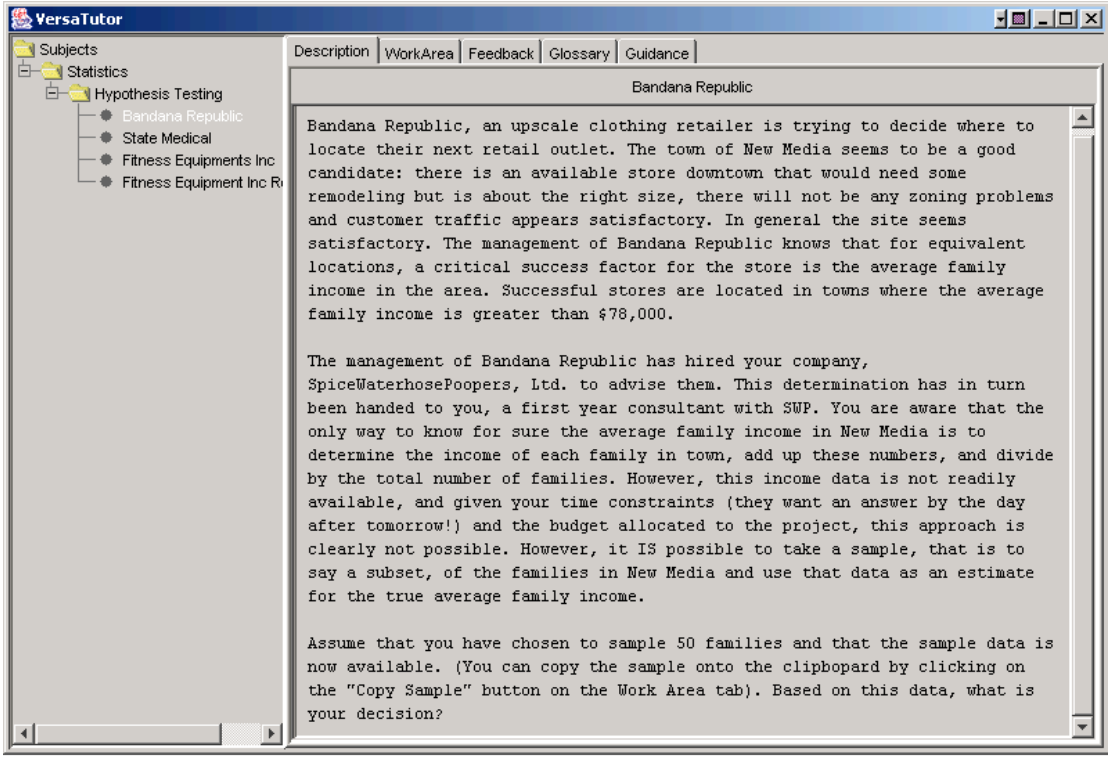


Fig. 2. Problem pane.

Solving a problem consists of selecting (from a master list) the variables relevant for that problem's solution, and supplying values for those variables. Students type in values for the selected variables and submit the (possibly partial) solution for evaluation. The tutor evaluates the solution and provides updates in the answer status column. (See Figure 3.)
In the case of errors, the feedback pane is shown (Figure 4). The tutor provides progressive hints, or the correct solution, on the student's request. If there are multiple errors, then the "Errors" list has multiple rows.

When the student has supplied correct values for all the appropriate variables, the system treats the problem as having been completed.

In our implementation, the sample data based on which the student will test the hypotheses are not given *a priori*, but are generated randomly each time. Thus students can return to problems they have already solved and work on them again with new random data sets. The students use the "Generate Data" button to actually generate the data and the "Copy data" button to copy the data to the clipboard for further analysis using statistical analysis or spreadsheet software.

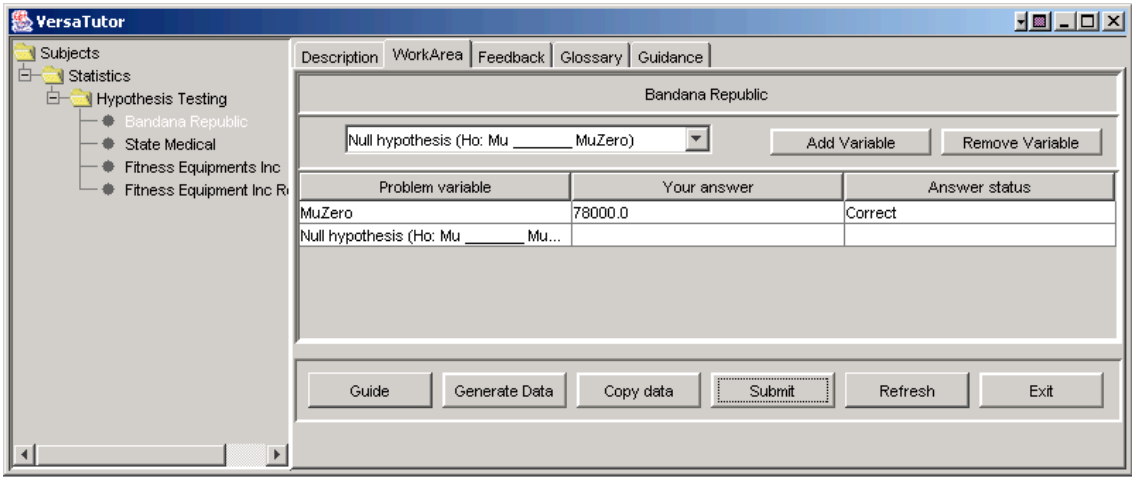


Fig. 3. Work area showing selected variables and answer status.

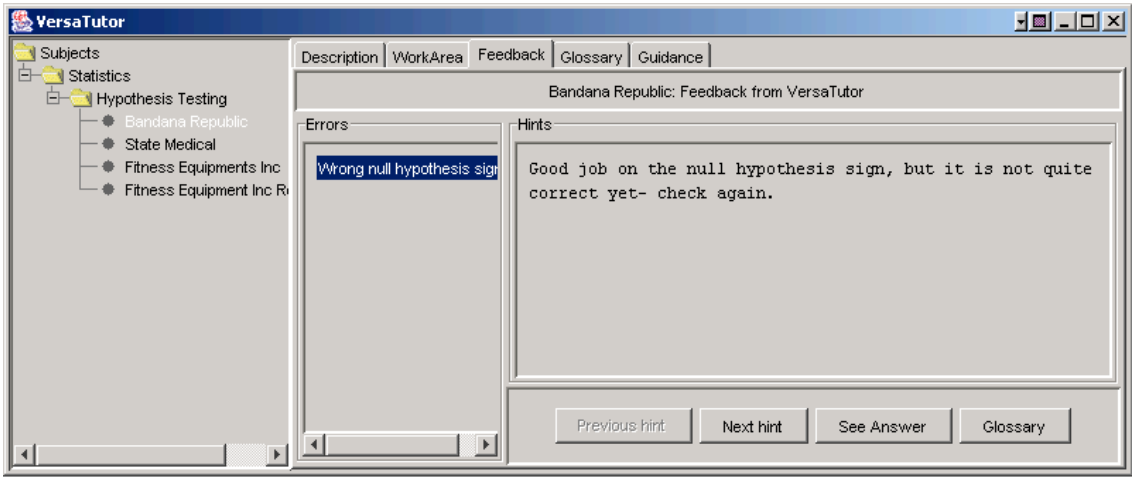


Fig. 4. Feedback in case of an error.

Implementation of the Constraint-based Model Tutor

In our CBMT when the student submits a (possibly partial) solution for evaluation, the user interface invokes a constraint-checking engine. The problem state is represented by a set of variable-value pairs. Some of these variables are specified *a priori* with the problem instance definition, and are simply data in the problem text. Others are "student" variables, that is, variables for which the student has to supply values. The tables below give some examples of *a priori* and student variables.

We developed a constraint checking engine that is generic and can be used for other tutors as well (see Kodaganallur, Weitz & Rosenthal, 2004). Since relevance and satisfaction conditions are both arbitrary Boolean expressions residing in a database, our constraint checking engine has

been written to evaluate expressions at runtime. We devised a language in which the author of the tutor can express the constraint expressions; relevance and satisfaction conditions are Boolean expressions, written as in a standard programming language. Our language provides a number of additional built-in functions. In this work, we used JavaCC, a freely available parser generator (https://javacc.dev.java.net/).

Table 1
A Priori variable examples

***A Priori* Variable Name**	**Description**
problemType	The kind of hypothesis testing problem (testing for mean less than a given value, greater than a given value or equal to a given value).
muZero	The hypothesized value of the population mean.
popStdKnown	Whether the population standard deviation is known.

Table 2
Student variable examples

Variable Name	**Description**
nullHypSign	The comparison operator for the null hypothesis (<, > or =).
studentMuZero	Student's value for mu zero.
Mean	The sample mean.
testStatistic	The test statistic to be used.
zValue	The value for the z statistic (this is applicable only if the proper statistic to be used in a problem instance is the z statistic).

Table 3 provides an example of a constraint. The fields "Description" and "Constraint number" serve as documentation and are not used by the tutor.

There are several points to note about constraints in our system.

- Problem type – In our system, every constraint applies to a certain class of problems. Since we built our system to be able to store problems from several domains, we use this field to divide the entire set of constraints into families. Achieving this through relevance conditions would be too inefficient.
- Relevance condition – This constraint is relevant only to a subclass of hypothesis testing problems and only if the student has specified a value for the "null hypothesis sign" variable. The constraint shows the use of two of the built-in functions of our constraint language.
- Satisfaction condition – This checks if the value entered by the student is the appropriate one for the problem type.
- Hints – The three hints are progressively more explicit hints to be displayed to the student on request.

- Answer – Provides the correct answer. In our system, expressions embedded within "|" symbols are dynamically evaluated and substituted before being passed on to the user interface.
- Variable(s) – Associated with each constraint are zero or more variables. If the constraint is satisfied then the student's value for the associated variable(s) is correct and wrong otherwise.

Currently there are 43 constraints in the system.

Table 3
A constraint

Problem type	HypothesisTesting
Constraint number	30
Description	If the problem type is "one population, testing for Mu <= MuZero then the correct sign for the null hypothesis is "<="
Relevance condition	Equals(problemType, "1PM<=") && exists("nullHypSign")
Satisfaction condition	Equals(nullHypSign, "<=")
Error title	Incorrect null hypothesis
Hint 1	Check your null hypothesis. You want to formulate your null hypothesis in such a way that you do not move away from the status quo unless there is strong evidence indicating otherwise. The null hypothesis is always that the observed mean favours retaining the status quo.
Hint 2	Think about whether a larger value or a smaller value for the mean than MuZero would favour moving away from the status quo, and then formulate the null hypothesis accordingly. In general, one would tend to stick with the status quo unless the data gathered convincingly indicates that there is a case to make changes. In this problem would a small sample mean or a large one convince you to maintain status quo? The null hypothesis is the case for maintaining the status quo.
Hint 3	In this problem a small sample mean would convince you that it makes sense to stick with the status quo and not to take any action whereas a large mean would provide a rationale to do something.
Answer	The null hypothesis should be Mu <= |MuZero|
Variable(s)	nullHypSign

In SQL-Tutor (Mitrovic & Ohlsson, 1999) the ideal solution to each problem is given *a priori* and the relevance and satisfaction conditions are formulated in terms of the elements in the ideal solution and those in the student's solution. In contrast, our implementation in the domain of statistical hypothesis testing does not need to be primed with the correct solution. Indeed, it cannot be, since in our tutor, sample data is generated randomly for each problem. However, the constraints have been written in such a way that they are able to calculate elements of the correct solution dynamically from the problem data provided *a priori* to the tutor. This does not however mean that our tutor has a solver for the problem – the constraints are able to calculate only individual elements of the solution, but lack any mechanism to tie them together.

Thus, whether or not the ideal solution is to be explicitly supplied to the tutor is a function of the architecture of the constraints. Given sufficient *a priori* information about a problem, the constraints could be written in such a manner that the elements of an ideal solution are computed by the constraints. In domains such as SQL this could be a very complex task.

Implementation of the Model-Tracing Tutor

We were able to reuse most of the code for the user interface and problem representation from our CBMT implementation. We used JESS for model tracing and chose to keep the actual remediation text in a relational database. The expert planning rules in our implementation conform to the goal structure shown in Figure 1. JESS is a forward chaining rule engine that does not explicitly support goal seeking. It provides special constructs and mechanisms to enable it to exhibit goal-seeking behaviour within the forward chaining context. We have used these features to implement the planning capabilities.

Model tracing can be a computationally intensive process because of the potentially very large number of possible rule combinations that might need to be tried. In our implementation, model tracing is not a complex process. A partial or complete solution to a problem is a set of variable-value pairs where the values are supplied by the student. In this problem, it turns out that tracing a solution breaks down to tracing each variable separately. Further, the number of pedagogically significant values for each variable is very small (usually two, sometimes three). Our JESS rules are written in such a way that the process of rule execution itself suffices for model tracing.

Generally when a student submits a solution for evaluation, the tutor indicates which of the variables are correct and which are wrong. In the case of portions which remain untraced, the system provides no explicit feedback; there is, of course, the implicit feedback that something was wrong.

Guidance

As with typical model-tracing tutors, our MTT allows the student to ask for guidance at any point in time. The system can suggest the next possible logical steps. The student can also request guidance on receiving an implicit error message when the system is unable to completely trace the student's solution.

Figure 5 provides an example of the system providing guidance.

COMPARISON OF APPROACHES

We anchor our comparison of the two approaches around the following four key dimensions:

- Knowledge-base content
- Scope for remediation
- Development effort
- Appropriateness for different types of problem domains

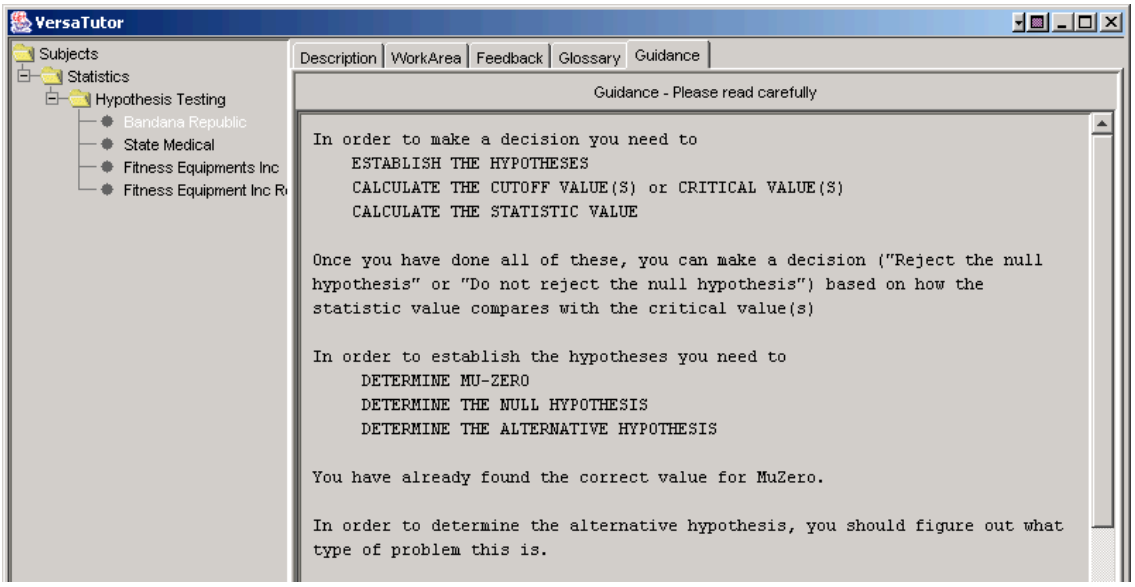


Fig. 5. An example of guidance in our MTT.

We answer these questions primarily based on our experience with our two implementations of the tutor for hypothesis testing, but also on our observations of the characteristics of other tutors reported in the literature.

Knowledge-Base Content

MTTs consist of expert and buggy planning and operator rules; no categorization of the constraints of CBMTs has been reported to date. In this section we identify equivalences between rules and constraints in order to provide a finer grained comparison. Additionally, this enables us to better compare the development efforts required for each.

Operator Rules Correspond To Constraints

There is a clear correspondence between operator rules and constraints. In our implementations, every expert operator rule has a corresponding constraint. Figure 6 shows an example. We call such constraints "operator constraints." In a "pure" CBMT all the constraints are operator constraints. In our system, the bulk of the constraints are operator constraints.

Capturing Planning in CBMTs

Since CBMTs are not concerned about the process by which a student reached a given problem state, in principle they do not have any elements corresponding to the planning rules of MTTs. By defining the problem state accordingly, it is however possible to write constraints in such a way that they capture some planning knowledge. Consider this example:

C_r: The student has not provided the null or alternate hypothesis.
C_s: The student should not have generated a sample.

Expert Operator Rule

IF There is a goal to find the test statistic AND
The population standard deviation is not known AND
The sample size is less than 30
THEN
Set the value of the test statistic to "t"

Corresponding Constraint

C_r: The population standard deviation is not known AND the sample size is less than 30 AND the student has supplied a value for the test statistic
C_s: The test statistic must be "t"

Fig. 6. Expert operator rule and corresponding constraint.

The constraint here ensures that the student first supplies values for both null and alternative hypotheses before proceeding to generate a sample. In our view, this constraint is pedagogically necessary – in hypothesis testing the first step is to state the null and alternative hypotheses. More generally, the presence or absence of values for some variables helps to infer something about the process, and we have a set of constraints in our system that exploits this. We call such constraints "process" constraints. In fact, it is possible to write process constraints in such a way that the complete goal structure inherent in planning rules is captured. However, doing so violates the basic spirit of the CBM paradigm.

CBM is centrally tied to the notion of "solution state." In domains like those considered by SQL-Tutor and the Ms. Lindquist algebra tutor, the solution state is effectively contained in a single string (an SQL statement or an algebraic expression). These strings contain sufficient information to drive rule firings or constraint evaluations. What about domains where this does not hold? For example, in the statistical hypothesis testing domain, the final answer is simply one of two strings "Reject the null hypothesis" or "Do not reject the null hypothesis." If a student simply supplies one of these as the answer to a problem, the tutor can evaluate its correctness, but has no information on which to base remediation. In our CBMT implementation we had to redefine the state to include (and thus force the student to provide) the intermediate computations based on which s/he arrived at the final solution. In effect we were forced to redefine the state to include process information. The same issue would arise in any domain where the student's final solution alone does not have enough information for remediation. In such domains, one would have to deviate significantly from the spirit of CBM in order to implement a CBMT (and doing so is tantamount to mimicking an MTT). We claim that this is a central aspect that defines the domains for which CBM is effectively applicable; we return to this point later in the paper.

Buggy Rules and Buggy Constraints

One of the advantages claimed for CBMTs is that they do not require extensive bug libraries (Ohlsson, 1994; Mitrovic et al., 2001) and are hence less resource intensive to build. In our experience, the equivalent of buggy rules – "buggy constraints" – are required for a CBMT to provide remediation equivalent to that of the corresponding MTT. We provide details in the Remediation section below.

Extraneous Consistency Constraints

We built our CBMT before the MTT. While writing the constraints we stuck to the spirit of CBM and simply looked at various valid configurations of values for variables, eschewing all considerations of process. (The problem state has 20 variables.) This yielded constraints such as:

C_r: The student has selected "z" as the test statistic to be used.
C_s: The student should not have specified any "t" value.

The logic of the above constraint is that if the student believes that the correct test statistic is "z," then it is meaningless to provide a value for the "t" statistic – the two values are inconsistent. There are 12 such constraints, with scope for many more which we did not include. When we later developed the rules for the MTT, we got into a process-centric mind-set: we first developed the goal structure and then went about building the expert and buggy rules. After completing the MTT rules, we found that there were constraints, like the one just above, which had no correspondence in our MTT rule set. Put another way, when viewing the problem with process in mind, we were unable to come up with any likely reasoning that could generate the situations proscribed by these constraints; that is, they are pedagogically insignificant. It appears that, unless writing constraints is influenced by process considerations, it is possible that unnecessary effort might be expended in creating pedagogically insignificant constraints – what we have termed "extraneous consistency constraints."

Remediation

The content of the knowledge base and the information inherent in a student's solution together determine the quality of remediation that a tutor can provide. As Mitrovic et al. (2003) indicate, the remediation provided by CBMTs tends to be "less comprehensive" than that provided by MTTs. In this section we examine this phenomenon in depth.

Buggy Rules and Buggy Constraints

Mitrovic and Ohlsson (1999) claim, "A constraint violation provides the same information as a positive match between a theoretical bug description and a student behavior" and therefore extensive bug libraries are not needed. We found that remediation requirements caused us to create the equivalent of buggy rules within the CBM paradigm. Consider the buggy rule in Figure 7 below:

IF The goal is to calculate the sample standard deviation

THEN

Set the sample standard deviation as $\sqrt{\frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{n}}$

Fig. 7. A bug rule encapsulating a common student error in calculating the sample standard deviation.

This rule confuses the population standard deviation and sample standard deviation formulas, and is pedagogically significant as it represents a common student error. A constraint that only enforces the correct value cannot give the specific remediation that the above buggy rule can. Accordingly, in addition to the constraint that expects the correct value for the sample standard deviation (Figure 8), we also have the constraint in Figure 9, which anticipates the common mistake.

C_r: The student has calculated the sample standard deviation

C_s: The sample standard deviation should be equal to $\sqrt{\frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{n - 1}}$

Fig. 8. The sample standard deviation constraint.

C_r: The student has calculated the sample standard deviation

C_s: The sample standard deviation should not be equal to $\sqrt{\frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{n}}$

Fig. 9. Buggy constraint corresponding to the buggy rule in Figure 7.

Indeed, the same case can be made in viewing the example of a set of rules and their corresponding constraints in Mitrovic et al. (2003). The constraint that the authors state, "catches the same error as the buggy rule" (p. 316) does in fact catch the error, though it cannot provide the equivalent remediation.

The above example demonstrates that incorporating buggy knowledge in a CBMT is indeed required *when identical remediation is necessary*. We refer to such constraints as "buggy constraints." In general then, for identical remediation, buggy operator rules would correspond to buggy constraints.

The following examples, from the domains of two tutors we empirically compare later in this paper, further illustrate this point.

When teaching students to write SQL queries involving the JOIN of two or more tables, it is common for beginning students to specify the tables to be joined, but to leave out the explicit JOIN conditions in the WHERE clause. A tutor that is explicitly written to recognize this common mistake and provide sharply focused remediation would be useful. Incorporating this in a CBMT would essentially mean introducing a buggy constraint.

Suppose the solution to an algebra word problem takes the form "70 * (30-x)." It is a common mistake for students to get everything correct, but omit the parentheses. The tutor that sees the student's input "70 * 30 – x" and is able to provide precise remediation would be very useful. In this sense, a solution like "70 * 40 – x" is qualitatively different from "70 * 30 – x," even though both solutions are wrong. The remediation provided in these two cases should be different, and if this type of functionality were required in a CBMT then the constraint base should be such that these two solutions violate different sets of constraints. Indeed this type of error may indicate an incorrect strategy on the part of the student, and hence provides another example where capturing strategy leads to improved remediation.

Feedback on Errors

A CBMT provides remediation based on violated constraints. This approach raises a number of challenges. What if a single error violates multiple constraints? Should the remediation associated with all violated constraints be displayed? If so, in which order should they be presented? If not, which violated constraint(s) should be used? The challenges become more complicated if a student submits a solution with more than one error. An example from SQL-Tutor is a situation where the student misspells a table name in an otherwise correct answer. This single, straightforward error can lead to a large number of error messages and to potential confusion for the student. In our CBMT implementation of the hypothesis testing tutor, if the student submits a partial or complete solution with values for multiple variables, and some of these values, including the one for the first logical step, are wrong, then the student will get several error messages.

MTTs avoid this problem as they do not try to satisfy downstream goals unless predecessor goals have been satisfied. In our MTT implementation, the same student input would result in only a single error message (corresponding to the first logical step).

It should be noted that CBMT proponents have cited the relative looseness of the paradigm as an advantage as it allows students to use strategies not envisioned by the system designers, switch strategies or indeed be inconsistent, as the tutor only responds to violated constraints (Mitrovic et al., 2001).

Procedural Remediation

Buggy planning rules enable MTTs to provide procedural remediation. In the domain of hypothesis testing, one common error is for the beginning student to calculate the sample mean and arrive at a decision based solely on how it compares with the hypothesized population mean value (MuZero). A buggy planning rule that reflects this faulty goal structure enables targeted procedural remediation. Shorn of procedural considerations, CBMTs cannot provide such remediation.

Guidance

As an MTT has an expert problem solver and can identify where in the problem solving process a student is at any point in time, it has the ability to tutor the student on appropriate follow-on steps. This type of guidance is not a natural feature of a CBMT. Interesting work has been reported in building constraint-based solvers for the domain of SQL, which enable SQL-Tutor to provide planning advice (Martin & Mitrovic, 2000; Martin, 2001). This solver is very domain specific and relies on the tutor being given an ideal solution up front. It is not at all clear from the current state of the art that constraint-based solvers can be constructed for all domains. Further, even in the SQL domain, the solver reported does not solve the problem from scratch; instead it has an algorithm that uses the student solution and the ideal solution and progressively rectifies an incorrect/incomplete student solution. The process is heuristic, and it is not proven to work for all problems within the SQL domain. It has been shown in some cases to be able to work even when the strategy adopted by the student is different from the one inherent in the "ideal" solution. In models (like ours) where the ideal solution need not be given to the tutor, more research is needed to look at the feasibility of building constraint-based solvers to solve problems from scratch. The whole area of constraint-based solvers is in a nascent stage and would have to advance significantly in order to enable CBMT to match the planning remediation available in MTT.

Unexpected but Correct Behaviour

How does a tutor respond when a student follows an unexpected but correct strategy? "Unexpected" in the case of an MTT means that the tutor is unable to trace the student's input. As specified earlier, in this case the tutor knows that it doesn't know what the student is doing. Its reaction in this case is an implicit response that something is wrong, but it should not provide any misleading remediation.

Since CBMTs are not designed to fathom the strategy that a student is adopting to solve a problem, they have been known to give misleading remediation when the student has provided an unexpected, but correct, solution; for example, in our experience with SQL- Tutor, we found correct but unexpected solutions resulted in the system "correcting" non-existent errors (see the next section). Martin's dissertation (2001) reports on this problem in detail and outlines the extensive work needed to be done in order to improve remediation in this scenario.

A Comparison of Remediation Provided by Two Tutors

We were interested in discovering how our observations meshed with the actual performance of fielded tutors from each paradigm. In this analysis we compared the SQL CBMT and Ms. Lindquist (Heffernan & Koedinger, 2002; Heffernan, 2001), an algebra tutor based on model-tracing. (In addition to the usual MTT student model, Ms. Lindquist includes an additional model of tutorial reasoning.) Both were years in development and testing, and each was authored at the institution that championed that approach. SQL-Tutor is available free to students who purchase any one of a selection of database textbooks from Addison-Wesley publishers. (See

http://www.aw-bc.com/databaseplacedemo/ for more details.) The Ms. Lindquist algebra tutor is available at (http://www.algebratutor.org/).

In using SQL-Tutor we, experienced database instructors, found what we consider to be fairly serious drawbacks. We have already mentioned the problem with misspelling a table name in an otherwise correct query. In another instance (problem 176) we used double quotes in place of single quotes to specify a string literal and got six errors. For problem 172, we supplied an answer that was incorrect because we used the OR connective in place of the AND connective, and the tutor accepted our incorrect answer. For problem 175, we used the wrong field to join two tables and the tutor accepted our incorrect answer. For problem 36 we submitted an answer that was correct (and one we expect a student might provide) and the tutor replied that it was incorrect. (It appears that SQL-Tutor still has significant challenges to overcome in terms of accommodating multiple strategies.) To be clear, we did not spend a lot of time interacting with the tutor, nor did we attempt to fool it with devious responses. (We're happy to provide transcripts of our interaction with SQL-Tutor upon request.)

The Ms. Lindquist MTT provides tutoring for algebra. The student is provided with a word problem and must translate the word problem into an algebraic expression, such as 4a + b or (6t +3s)/q. We accessed the Ms. Lindquist tutor and were unable to interact with it in any way such that it gave anything other than correct, targeted feedback.

Certainly this assessment is in no way comprehensive. Further, we are not comparing tutors in the same domain; we suppose it could be argued that algebra is an easier application for a tutor than SQL. However, we do believe it provides confirming evidence for our own observations regarding the difficulties of providing targeted remediation using the CBMT paradigm.

Development Effort

The total effort to develop a tutor comprises the following components:

- Problem modelling
- User interface development
- Constraint engine/model-tracing engine development
- Encoding the constraints/rules

At the outset of this project, we hoped to take advantage of authoring tools that might ease the development task. Generally the focus of MTT and CBMT development tools is to provide the model-tracing/constraint-checking functions and perhaps assist with user interface development. They require the tutor author to encode the rules/constraints in the computer language and format specified by the tool. (Murray (1999) provides a general review of ITS authoring systems.)

For building our MTT, researchers at Carnegie Mellon University very kindly provided us with their "Tutor Development Kit" (Anderson & Pelletier, 1991). Challenges here included the typical issues surrounding university-developed tools that were not designed as commercial products, including limited documentation and functionality, and no support. Additionally, the kit was written in LISP, required a particular version of LISP to run, and was intended for an Apple environment. These were significant drawbacks for us. We then considered the Java

version of the same tool developed at Worcester Polytechnic Institute and found that it too was unsuitable for our task since it was in an early stage of development, and the user interface we needed appeared too complex to be built using that tool. Though more advanced tools are currently under development (Koedinger, Aleven & Heffernan, 2003) our choice at the time, to build the complete tutor ourselves, was clear.

At the start of our work we were unaware of any CBMT development tools available. There is at present (at least) one CBMT authoring tool (Martin & Mitrovic, 2002); it also runs on a platform unsuitable for us. (It is written in Allegro Lisp and runs on an Allegro server. We understand that work continues in an effort to make the tool more accessible to others.)

Problem modelling effort

The effort to model the problem consists of identifying how to represent the problem state, and in the case of MTT, it also consists of the effort to define the goal structure. In our problem domain, the problem state definition turned out to be identical for both the paradigms and took about five person-days. The effort to develop the goal structure was extra for the MTT; we estimate this effort at roughly one person-day.

Constraint/Rule Development Effort

Our CMBT has 43 constraints. We took about 18 person-days to encode the constraints. There was initial effort involved in formulating the manner in which we would express the constraints. We identified some common computations that several constraints might need and devised functions for these. Once we reached the stage of actually encoding the constraints, writing the relevance and satisfaction conditions turned out to be the easy part; writing the remediation messages and hints was the more time consuming portion. On average, encoding each constraint, with all associated remediation, took two person-hours.

As encoded in JESS, our MTT has 76 rules. Of these, one is concerned with initializing some variables, four are pure planning rules, 43 rules have a planning and an operator component, 12 are concerned with guidance, and the rest are pure operator rules. In our style of writing the JESS rules, there are actually two rules corresponding to each (operator) constraint. Also, in order to induce proper goal decompositions, we had to write a number of dummy rules to cause JESS to induce all the subgoals for a given goal. Thus, although the cardinality of the rule set seems to be much larger than that of the constraint set, this did not translate into a proportionately larger effort because the effort incurred in writing a single rule was often shared with that for writing many of the associated rules. Our experience was that getting started with encoding the rules in JESS was a conceptually more difficult task than that needed to initiate writing the constraints. However, once we started the process, progress was then smooth. We estimate that the whole process took about four person-months, including the learning curve involved in learning JESS. If we factor out the learning curve, it seems to us that the time taken to write a constraint and its corresponding rules was about the same since the relevance and satisfaction conditions closely resemble the antecedent and consequent of the corresponding rules. The effort required for writing the planning rules is clearly extra for the MTT.

In domains characterized by complex relationships between problem elements, it is possible that a CBMT developed without consideration of process issues could end up with many extraneous consistency constraints that an MTT would avoid. While the need to create an exhaustive library of buggy rules has been pointed out by the proponents of CBM, our experience indicated that for equivalent remediation CBMT would also need an equivalent set of buggy constraints.

User Interface Development

In reviewing the user interfaces of MTT systems, ranging from the physics tutor Andes to the algebra tutor Ms. Lindquist, and CBMT systems such as the punctuation tutor CAPIT and SQL-Tutor we believe there is no generalizable difference between development requirements in this area. In our implementations, we were able to use a common code base for the user interfaces of the two tutors, except for the added "guidance" functionality of the MTT, which took three person-days to develop. If a tutor development tool is being used to build the user interface, once again there is no reason to believe that one or other paradigm has an advantage here.

Constraint-Checker/Model-Tracer Development

A CBMT requires a constraint checking engine. We were able to build a generic constraint checking engine, in about two person-months.

For the MTT implementation, we used JESS, a freely available rule engine. The process of determining how to write rules in such a manner that rule execution by JESS effectively performed model tracing took about three person-weeks.

As discussed in the MTT implementation section, our model tracing requirements were simple. This might not hold in all domains, in which case a dedicated model-tracing algorithm might be necessary. We do not believe a generic model-tracing algorithm, with cross-domain functionality, can be developed easily as was possible with the CBM case.

A summary comparison of the development efforts (in person-days) is provided in Table 4.

Table 4
Development efforts (in person-days) for the constraint-based and model-tracing tutors

	Constraint-Based	**Model-Tracing**
Problem modelling	5	CBMT + 1
User Interface development	60	CBMT + 5
Constraint Engine/Model-Tracing Engine Development	60	Not applicable
Knowledge encoding	18	120 (including learning JESS)

We developed the CBMT first. Some of the effort expended in developing the CBMT carried over to the MTT. For example, the problem modelling effort was five person-days for the CBMT, and the MTT required only an additional day to develop the goal structure. Similarly, it took an additional five days to modify the CBMT user interface for use in the MTT.

Finally it is worth noting that although we did not use any formal authoring tools for developing either the CBMT or the MTT, we did get a degree of high level tool assistance in using JavaCC to build our generic expression evaluator for the CBMT, and in using JESS for the MTT. Each of these tools contributed significantly to reducing the development time for the corresponding tutor. Whether one paradigm got more tool help than the other is difficult to evaluate and, in our view unimportant, as these tools are readily available and may be used for any new development.

Problem Domains

It is clear that the rule and constraint formalisms allow for the two knowledge bases to contain essentially identical information. However, the essence of MT is to incorporate planning and buggy knowledge and the essence of CBM is to provide remediation based solely on state. We have seen that it is possible to build a CBMT that emulates an MTT by incorporating planning and buggy constraints. However, this violates the CBM paradigm. In addition, as Mitrovic et al. (2003) specify, this compromises cognitive fidelity. Likewise, it's possible to build an MTT that emulates a CBMT by forbidding planning rules, and including "buggy rules" that check for errors but don't capture common student misconceptions, but this would run counter to the foundations of MT.

We have identified two dimensions that enable us to characterize the problem domains for which the two paradigms (in their pure forms) are suited.

As we have stated in the section discussing capturing planning in CBMTs, the most critical dimension in terms of choosing between the two paradigms for a particular domain is the information richness of the solution. Consider for example a physics tutor that focuses on mechanics; a typical problem might ask the student to specify the velocity at which an object is moving when it hits the ground. Clearly the answer alone (say 70 mph) does not contain enough information for a CBMT to provide remediation. Statistical hypothesis testing (as we have noted) falls into this category as well, as the solution to a hypothesis testing problem is simply "Reject the null hypothesis" or "Do not reject the null hypothesis." On the other hand, the solution to an algebra problem, an SQL problem or a punctuation problem does contain sufficient information for a CBMT to provide remediation.

The information content of a solution can be measured as the number of distinct domain-related tokens that it contains. A raw number as the final solution has an information content of 1 and provides very little information on which a CBMT can base remediation. A solution like "3x + 2" to an algebra problem has 4 tokens upon which a CBMT could base its remediation.

The second dimension affecting the selection of paradigm is the complexity of the goal structure that defines the solution process. (Figure 1 provides the goal structure for the hypothesis testing domain.) This complexity is directly proportional to the depth and branching factor of the solution process. As stated earlier, it is generally accepted that MTTs outperform CBMTs with respect to remediation quality. As the problem goal structure complexity increases, it becomes increasingly difficult for a CBMT to fathom where in the problem-solving process the student has made an error and in turn it becomes increasingly difficult for a CBMT to offer targeted remediation.

Consider two domains having the same information richness in the solution, but with the second having a greater complexity of goal structure. Owing to the equality of information

richness in the solution, CBMTs for the two domains would have similar remediation power. MTTs for both domains have greater scope for providing remediation than the corresponding CBMTs. Further, an MTT for the domain with greater goal structure complexity has even greater scope for procedural remediation than an MTT for the other domain. Figure 10 outlines the interaction between these two dimensions.

As the information richness of the solution increases, CBMT becomes increasingly feasible. As goal-structure complexity increases, MTTs become increasingly preferred.

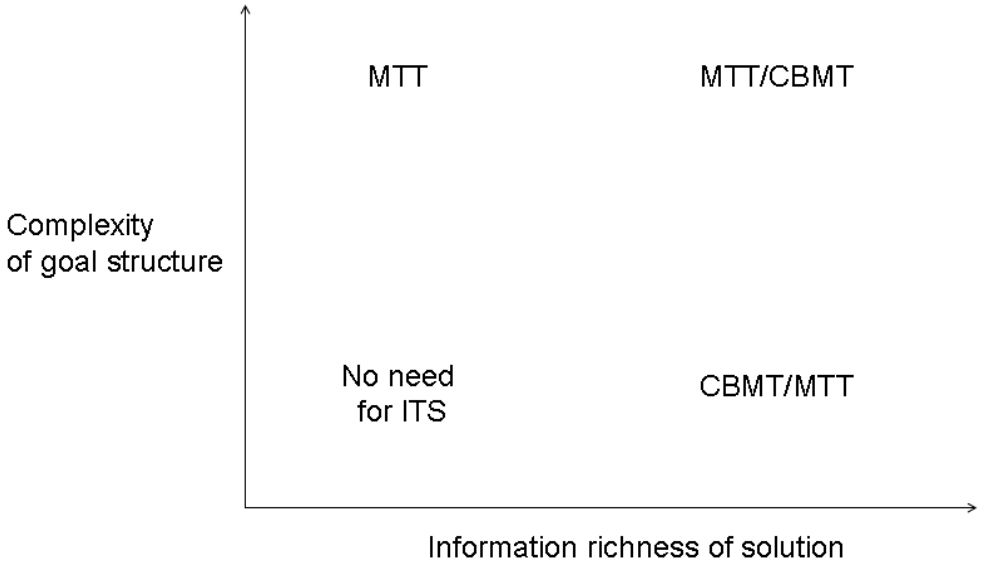


Fig. 10. Feasibility/Preference space for each tutor paradigm.

Given sufficient information richness in the solution, it is conceivable that the incremental cost of building an MTT might surpass the marginal benefit of procedural remediation and tilt the balance in favour of CBMT. An implication of this analysis is that in a domain in which solutions have rich information content, CBM could be used for rapid prototyping (provided a constraint checking engine is available).

Given that the knowledge base of an MTT subsumes that of a CBMT, an MTT with equal remediation can always be built corresponding to any CBMT. The effort required to do so would be comparable to that for the CBMT because the assumption of equal remediation precludes having to build extra planning rules and buggy rules for the MTT. However, building a CBMT with equal remediation to an MTT might not always be possible without deviating from the spirit of CBM, and would be as effort intensive.

CONCLUSIONS

Our experience in implementing an MTT and a CBMT for the same domain has yielded some new insights into how these two paradigms compare.

We have provided a fine-grained comparison of the two knowledge bases. We show that operator rules in MTTs match constraints in CBMTs, and introduce the notion of buggy constraints, planning constraints, and extraneous consistency constraints.

We have demonstrated that buggy constraints, the equivalent of buggy rules, are required for equivalent remediation. So the question of whether CBMT needs to "create extensive bug libraries" is only an issue of the desired quality of remediation.

Our experience and analysis indicate that building a pure CBMT might be easier than building an MTT for the same domain because of the extra effort needed to develop expert and buggy planning rules and buggy operator rules for MTT. We also found that building the MTT rule set was a somewhat more complex exercise because of the need to tie the rules together through a goal structure. On the other hand, it is possible that authors of a CBMT might spend unnecessary time and effort creating extraneous consistency constraints and this cannot be avoided unless the process aspect of problem solving is explicitly considered during the constraint authoring process. In sum, our observations lend support to CBMT proponents who argue that building CBMTs is less resource intensive.

Mitrovic et al. (2003) "believe that CBM is neutral with respect to the domain." We, however, have identified two critical dimensions that characterize the problem domains for which the two paradigms are suited. The greater the information richness of the solution, the more feasible CBM becomes, and the greater the complexity of the goal structure, the greater is the incremental remediation capability of an MTT. It is conceivable that with very high levels of information content in the solution, CBMTs could provide acceptable remediation. In such a situation, the marginal cost of MTTs could outweigh the marginal increase in remediation.

We find that an MTT can be built for every domain in which a CBMT can be built, but the reverse doesn't hold. (Indeed the comparison provided by Mitrovic et al. (2003) was predicated on building an MTT for an existing CBMT.) CBMTs do not provide forward looking hints that are a part of MTTs. More generally, the remediation provided by an MTT will be superior, and the difference in remediation power will increase as the complexity of the goal structure increases.

FUTURE RESEARCH

It should be noted that no *fielded* tutors have been built for a single domain using each paradigm; it seems that doing so would provide an interesting "head to head" comparison of the two approaches. We expect to perform such an evaluation shortly. Additionally, fruitful areas for further research include:

- Exploring constraint prioritization and organization to improve remediation from CBMTs.
- Building generic constraint-based solvers.
- Studying empirically the effects of solution information richness and goal structure complexity on the choice of MT or CBM paradigm.

ACKNOWLEDGEMENTS

The authors wish to acknowledge Seton Hall's Teaching, Learning, and Technology Center for its support of this research in the form of a Curriculum Development Initiative grant. We also wish to thank the three anonymous reviewers for their detailed and extremely helpful comments.

REFERENCES

Anderson, J. R. (1983). *The Architecture of Cognition*. Cambridge, MA: Harvard University Press.

Anderson, J. R. (1993). *Rules of the Mind*. Hillsdale, NJ: Erlbaum.

Anderson, J.R., Boyle, C.F., & Yost, G. (1985). The Geometry Tutor. In A. K. Joshi (Ed.) *Proceedings of the 9th International Joint Conference on Artificial Intelligence* (pp. 1-7). San Francisco: Morgan Kaufmann.

Anderson, J.R., & Pelletier, R. (1991). A Development System for Model-Tracing Tutors. In L. Birnbaum (Ed.) *Proceedings of the 1991 International Conference on the Learning Sciences* (pp. 1-8). Charlottesville, VA: AACE.

Anderson, J. R., Corbett, A. T., Koedinger, K., & Pelletier, R. (1995). Cognitive tutors: Lessons Learned. *The Journal of Learning Sciences*, 4(2), 167-207.

Corbett, A.T., & Anderson, J.R. (1993). Student Modeling In An Intelligent Programming Tutor. In E. Lemut, B. du Boulay & G. Dettori (Eds.) *Cognitive Models and Intelligent Environments for Learning Programming*. Berlin Heidelberg New York: Springer-Verlag.

Corbett, A.T., Anderson, J.R., & O'Brien, A.T. (1995). Student modeling in the ACT Programming Tutor. In P. Nichols, S. Chipman & B. Brennan (Eds.) *Cognitively Diagnostic Assessment* (pp. 19-41). Hillsdale, NJ: Erlbaum.

Corbett, A.T., & Bhatnagar, A. (1997). Student modeling in the ACT Programming Tutor: Adjusting a procedural learning model with declarative knowledge. In A. Jameson, C. Paris & C. Tasso (Eds.) *Proceedings of the Sixth International Conference on User Modeling* (pp. 243-254). Berlin Heidelberg New York: Springer-Verlag.

Friedman-Hill, E. (2003). *Jess in Action: Rule-Based Systems in Java*. Manning Publications, Greenwich, CT. (See also the JESS homepage at: http://herzberg.ca.sandia.gov/jess/index.shtml.)

Gertner, A., & VanLehn, K. (2000). Andes: A Coached Problem Solving Environment for Physics. In G. Gauthier, C. Frasson & K. VanLehn (Eds.) *Intelligent Tutoring Systems: 5th International Conference, ITS 2000* (pp. 131-142). Berlin Heidelberg New York: Springer-Verlag.

Heffernan, N.T. (2001). *Intelligent Tutoring Systems Have Forgotten the Tutor: Adding a Cognitive Model of Human Tutors*. Doctoral Dissertation, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213.

Heffernan, N. T., & Koedinger, K. R. (2002). An Intelligent Tutoring System Incorporating a Model of an Experienced Human Tutor. In S.A. Cerri, G. Gouardères & F. Paraguaçu (Eds.) *Proceedings of the Sixth International Conference on Intelligent Tutoring Systems* (pp. 596-608). Berlin Heidelberg New York: Springer-Verlag.

Kodaganallur, V., Weitz, R., & Rosenthal, D. (2004). VersaTutor – An Architecture for a Constraint-Based Intelligent Tutor Generator. *Proceedings of the Thirteenth Annual World Wide Web Conference (WWW2004)* (pp. 474-475). New York, May 17-22, 2004.

Koedinger, K. R., Aleven, V., & Heffernan, N. T. (2003). *Toward a Rapid Development Environment for Cognitive Tutors.* 12th Annual Conference on Behavior Representation in Modeling and Simulation. Simulation Interoperability Standards Organization.
Available online at http://nth.wpi.edu/pubs_and_grants/03-BRIMS-063.doc

Koedinger, K.R., & Anderson, J.R. (1993). Effective Use of Intelligent Software in High School Math Classrooms. In P. Brna, S. Ohlsson & H. Pain (Eds.) *Proceedings of the World Conference on Artificial Intelligence in Education* (pp. 241-248). Charlottesville, VA: Association for the Advancement of Computing in Education.

Koedinger, K.R., & Anderson, J.R. (1997). Intelligent Tutoring Goes to School in the Big City. *International Journal of Artificial Intelligence in Education*, 8, 30-43.

Levine D.M., Stephan D., Krehbiel, T.C., & Berenson M.L. (2001). *Statistics for Managers Using Microsoft Excel* (3rd edition). Upper Saddle River, New Jersey: Prentice Hall.

Mayo, M., Mitrovic, A., & McKenzie, J. (2000). *CAPIT: An Intelligent Tutoring System for Capitalization and Punctuation. International Workshop on Advanced Learning Technologies,* IWALT 2000 (pp. 151-154). Palmerston North, New Zealand.

Martin, B. (2001) *Intelligent Tutoring Systems: The Practical Implementation of Constraint-Based Modelling*, Ph.D. Thesis, University of Canterbury, New Zealand.

Martin, B., & Mitrovic, A. (2000). Tailoring Feedback by Correcting Student Answers. In G, Gauthier, C Frasson & K. VanLehn (Eds.) *Proceedings of the Fifth International Conference on Intelligent Tutoring Systems* (pp. 383-392). Berlin: Springer.

Martin, B., & Mitrovic, A. (2002). WETAS: A Web-Based Authoring System for Constraint-Based ITS. In P. De Bra, P. Brusilovsky & R. Conejo (Eds.) *Proceedings of the Second International Conference on Adaptive Hypermedia and Adaptive Web-Based Systems,* AH 2002 (pp. 543-546). Berlin Heidelberg New York: Springer-Verlag.

Mayo, M., & Mitrovic, A. (2001). Optimising ITS Behaviour with Bayesian Networks and Decision Theory. *International Journal of Artificial Intelligence in Education* 12, pp. 124-153.

Mitrovic, A. (2003). An Intelligent SQL Tutor on the Web. *International Journal of Artificial Intelligence in Education* 13, pp. 171-195.

Mitrovic, A., Koedinger, K., M., & Martin, B. (2003). A Comparative Analysis of Cognitive Tutoring and Constraint-Based Modeling. In P. Brusilovsky et al. (Eds.) *Proceedings of the 9th International Conference on User Modeling (UM2003)* (pp. 313–322). LNAI 2702. Berlin Heidelberg New York: Springer-Verlag.

Mitrovic, A., Martin, B., & Mayo, M. (2002). Using evaluation to shape ITS design: Results and experiences with SQL-Tutor. *International Journal of User Modelling and User Adapted Interaction,* 12(2-3), 243-279.

Mitrovic, A., Mayo, M., Suraweera, P., & Martin, B. (2001). Constraint-Based Tutors: A Success Story. In L. Monostori, J. Vancza & M. Ali (Eds.) *Proceedings of the 14th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems (IEA/AIE – 2001)*, 931-940, Berlin Heidelberg New York: Spring-Verlag.

Mitrovic, A., & Ohlsson, S. (1999). Evaluation of a Constraint-Based Tutor for a Database Language. *International Journal of Artificial Intelligence in Education*, 10, 238-256.

Murray, T. (1999). Authoring Intelligent Tutoring Systems: An Analysis of the State of the Art. *International Journal of Artificial Intelligence in Education*, 10, 98-129.

Ohlsson, S. (1992). Constraint-Based Student Modeling. *Journal of Artificial Intelligence in Education*, 3, 429-447.

Ohlsson, S. (1994). Constraint-Based Student Modeling. In J. E. Greer, & G. McCalla (Eds.) *Student Modeling: The Key to Individualized Knowledge-Based Instruction* (pp. 167-189).

Ohlsson, S., & Rees, E. (1991). The Function of Conceptual Understanding in the Learning of Arithmetic Procedures. *Cognition and Instruction*, 8, 103-179.

Schofield, J.W., Evans-Rhodes, D., & Huber, B.R. (1990). Artificial Intelligence in the Classroom: The Impact of a Computer-Based Tutor on Teachers and Students. *Social Science Computer Review*, 8(1), 24-41.

Shelby, R., Schulze, K., Treacy, D., Wintersgill, M., VanLehn, K., & Weinstein, A. (2001, July). An Assessment of the Andes Tutor. In S. Franklin, J. Marx & K. Cummings (Eds.) *Proceedings of the 2001 Physics Education Research Conference* (pp. 119-122). Rochester, NY.

Schultze, K.G., Shelby, R.N. Treacy, D.J., Wintersgill, M.C., VanLehn, K., & Gertner, A. (2000). Andes: An Intelligent Tutor for Classical Physics. *Journal of Electronic Publishing*, 6. University of Michigan Press. (Retrieved January 15, 2003 at http://www.press.umich.edu/jep/06-01/schulze.html.)

Shute, V. J., & Psotka, J. (1996). Intelligent tutoring systems: Past, present, and future. In D. Jonassen (Ed.) *Handbook of Research for Educational Communications and Technology* (pp. 570-600). New York, NY: Macmillan.

Suraweera, P., & Mitrovic, A. (2002). Kermit: A Constraint-Based Tutor for Database Modeling. In S.A. Cerri, G. Gouardères & F. Paraguaçu (Eds.) *Proceedings of the Sixth International Conference, ITS 2002*, 377-387, Berlin Heidelberg New York: Spring-Verlag.

Wertheimer, R. (1990). The Geometry Proof Tutor: An "Intelligent" Computer-Based Tutor in the Classroom. *Mathematics Teacher*, 84(4), 308-317.