Bayesian Student Modelling and Decision-Theoretic Selection of Tutorial Actions in Intelligent Tutoring Systems

A THESIS
SUBMITTED IN PARTIAL FULFILMENT
OF THE REQUIREMENTS FOR THE DEGREE
OF
DOCTOR OF PHILOSOPHY IN COMPUTER SCIENCE
IN THE
UNIVERSITY OF CANTERBURY
by
Michael John Mayo

University of Canterbury
2001

This thesis is dedicated to my family.

Acknowledgements

First of all, I want to thank my loving and ever patient wife Nhung. Without her support, none of this would have been possible.

I would also like to thank my supervisor, Tanja Mitrovic, for her advice and guidance over the years. It was invaluable. Thanks also to both Tanja and Jane McKenzie for reading drafts of this thesis.

During my time as a student, I wrote a number of papers for publication based on the research and ideas presented here. I would like to thank the anonymous reviewers for their helpful comments and criticism.

Financial assistance was provided from the Department of Computer Science. For that, I am highly grateful.

Kurt Hausler was employed to do some of the programming for the SQL-Tutor evaluation study described in Chapter 4. Thanks, Kurt.

Microsoft provided the Bayesian networks API used in the implementation of CAPIT.

Finally, thanks to the numerous participants of the various evaluation studies. These include the numerous undergraduate computer science students who used my version of SQL-Tutor during 1999, and the staff and students of Westburn and Ilam School in Christchurch, New Zealand, who used CAPIT in April and June 2000.

MICHAEL MAYO

University of Canterbury
January 2001

Abstract

This thesis proposes, demonstrates, and evaluates, the concept of the normative Intelligent Tutoring System (ITS). Normative theories are ideal, optimal theories of rational behaviour. Two normative theories suitable for reasoning under conditions of uncertainty are Bayesian probability theory, which allows one to update one's beliefs about the world given previous beliefs and new observations, and decision theory, which shows how to fuse one's preferences with one's beliefs in order to rationally decide how to behave. A normative ITS is a tutoring system in which beliefs about the student (the student model) are represented with a Bayesian network, and teaching actions are selected using decision-theoretic principles. The main advantage of a normative ITS is that the normative theories provide an optimal framework for implementing learning theories. In other words, the particular learning theory underlying the ITS is guaranteed to be optimally applied to the student if it is defined as a set of normative representations (probability distributions and utility functions). In contrast, the more traditional type of ITS with an ad-hoc implementation of a learning theory is not guaranteed to be optimal.

A general methodology for building normative ITSs is proposed and demonstrated. The methodology advocates building an adaptive, generalised Bayesian network student model using machine learning techniques from student performance data collected in the classroom. The Bayesian network is then used as the basis for the decision-theoretic selection of tutorial actions.

The methodology is demonstrated with two implementations. Both implementations were evaluated in a classroom, rather than a lab, setting. The first implementation is an extension to an existing ITS called SQL-Tutor. A Bayesian network-based student model was added to SQL-Tutor, and this was

applied to select the next problem for students. Although this system only partly implemented the normative methodology, the evaluation results were promising enough to continue in this direction.

The second evaluation was more comprehensive. An entirely new ITS called CAPIT was implemented by application of the methodology. CAPIT teaches the basics of English capitalisation and punctuation to 8-10 year old school children, and it uses constraint-based modelling to represent domain knowledge. The system models the child's long-term mastery of the domain constraints using an adaptive Bayesian network, and it selects the next problem and best error message (when a student makes more than one error following a solution attempt) using the decision-theoretic principle of expected utility maximisation. Learning theories define both the semantics of the Bayesian network and the form of the utility functions.

The evaluation of CAPIT was a success. Three groups of children, A, B, and C, were enlisted and given a pre-test. Group B then used a randomised (non-normative) version of CAPIT for a four week period, while Group C used the full normative version of the tutor. All groups were then administered a post-test. The results show that while both Groups B and C gradually mastered the domain constraints, Group C mastered the constraints at a faster rate than group B. Group A, who did not have access to an ITS in the domain, actually regressed on the post-test.

Contents

Notation

The following notation will be used in this thesis to distinguish between *variables*, *values*, *sets of variables*, *sets of values*, and *constants*:

Object	**Formatting**	**Example(s)**
Variable	Italic, Uppercase	*X*
Value	Italic, Lowercase	*Y*
Set of variables	Bold, Uppercase	**X**, **PA**(*X*)
Set of values	Bold, Lowercase	**e**, **pa**(*X*)
Constant	Courier font	`Correct`

Chapter 1

Introduction

Over the past thirty years, considerable research has been invested in the development of computer programs for effectively teaching students. These programs are called Intelligent Tutoring Systems (ITSs). The desire to build ITSs results from observations of the effectiveness of one-to-one tutoring. Compared to traditional classroom models in which one teacher tutors many students, one-to-one tutoring is highly effective. However, the approach is typically not feasible because the number of students greatly outweighs the number of teachers. With the advent of computers, however, the dream took one step towards reality. If a computer could be programmed to *emulate* the teacher, then one-to-one tutoring for all students would become a possibility.

This philosophy of completely replacing the teacher with a computer is not surprising, given the field's roots in Artificial Intelligence (AI). All one had to do, in theory, was program the computer with subject (domain) knowledge and pedagogy. This would require input from both cognitive and instructional science. Ideally, natural language processing advances would allow the student to converse with the computer in the same way she/he would converse with a human tutor.

This aspiring initial approach rapidly proved to be an overly ambitious challenge. Firstly, natural language processing, while it has made great advances in recent years, is not yet powerful enough to support natural language conversations between the tutor and the student. As a result, most ITSs have been developed with alternate interfaces. Secondly, it was realised that an ITS can never completely replace a teacher. There will always be students for whom computer-based instruction is not suitable. There may also be particular topics for which computer-based instruction is not appropriate. The goal, therefore, has been redefined to the much more realistic aspiration of *supporting* the teacher, be it in the classroom, the workplace, or at home. By having ITSs as supporting tools for the teacher, students are able to learn at their own pace and more time becomes available for the teacher to focus on one-to-one situations with students who may not be responding as well to the ITS approach.

This introductory chapter forms a high-level overview of the rest of the thesis. The traditional architecture of an ITS is described in Section 1.1, and in that context, Section 1.2 discusses the central problem; specifically, the types and effects of the uncertainty inherent in an ITS and its consequences. Section 1.3 proposes a solution: normative theories. Normative theories are general theories defining rational behaviour, and they are logically complete and consistent. By implementing normative theories in an ITS, the rationality of the system can be improved. Section 1.4 introduces two fully functional ITSs that were employed to demonstrate these ideas, SQL-Tutor and CAPIT (Capitalisation And Punctuation Intelligent Tutor), and Section 1.5 is a guide to the rest of the thesis.

1.1 Architecture of the ITS

The traditional architecture of an ITS is depicted in Figure 1.1. There are four significant components of the system. The *domain expert* is a representation of the domain knowledge. The quality of the domain knowledge is entirely variable; it can range from expert system-quality in which enough knowledge is represented to enable the ITS to solve problems itself, through to a subset of the knowledge that is just enough for the purposes of teaching.

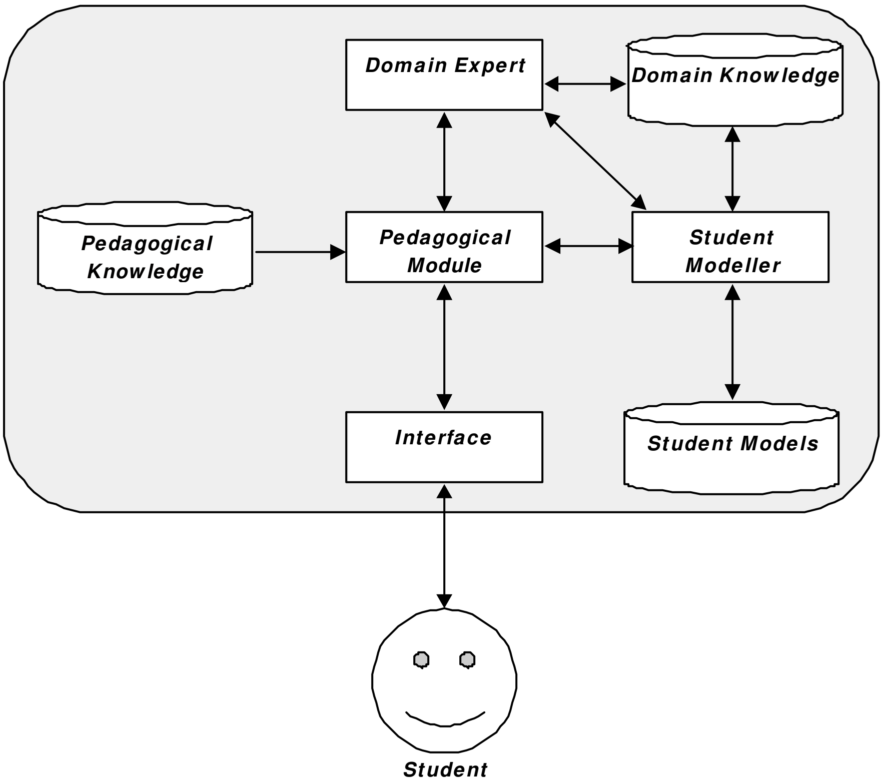


Fig. 1.1. Traditional architecture of an ITS.

The *student modeller* develops a representation of the current student using the ITS. This includes long-term knowledge, such as an estimate of the student's domain mastery, as well as short-time knowledge, such as whether or not the student just violated a rule. Other factors such as motivation can also be modelled, though this is done less frequently. The student modeller will typically save the current student model to the database when the student logs off, and restore it when the student logs back in.

The heart of the ITS is the *pedagogical module*, the subsystem that makes decisions about the teaching of the domain. Pedagogical decisions include, but are certainly not limited to, next problem selection, next topic selection, adaptive presentation of error messages, and selective highlighting or hiding of text. Typically, the pedagogical module will take into account both the current student model and the domain knowledge when making a decision. To illustrate, consider a hypothetical next topic selection. The job of deciding whether to select an entirely new topic or revisit an old one for revision falls to the pedagogical module. On the one hand, the ITS must ultimately cover the entire domain (if that is one of its teaching goals), but on the other hand, the more time spent revising existing topics the better. The particular method by which this decision is made can be referred to as a pedagogical action selection (PAS) strategy.

Finally, the fourth component of an ITS is the *user interface*. The user interface is of importance in an ITS. Firstly, it must provide the motivation for the student to continue. If the student lacks the desire to use the system, the ITS simply will not be effective. Secondly, the interface can improve learning significantly by reducing the cognitive load. If only one component of a problem is the focus of teaching, then the rest of the problem can be "externally stored" within the user interface. This means the student does not need to remember extraneous details and can instead target the component of problem-solving that is of importance.

Although all of the components of an ITS are important, it should be clearly obvious that the student model is the most critically important. If the student model is "bad" in that it does not even approximately describe the traits of the current student, then the quality of decisions made by the pedagogical module will be correspondingly bad. This is regardless of the quality of the pedagogical module itself. Considerable research, therefore, has been invested specifically in student modelling.

1.2 The Problem: Uncertainty

One significant difficulty for any system that models users or students is uncertainty. The ITS must build a student model from minimal amounts (relative to the human tutor) of highly uncertain information. The "keyhole" of the computer keyboard and mouse is a severe limitation. Furthermore, because the ITS bases its decisions (like a human tutor) on the student model, the uncertainty in the student model is carried over and may be realised as poorly-adapted teaching actions. It is this uncertainty that has led to the development of other, "simpler" ITS metaphors, such as that of the discovery environment, which require less computational effort and inference for (its proponents claim) the same degree of benefit.

It is relevant to examine the types of uncertainty inherent in student modelling. A student model is constructed from observations that the ITS makes about the student. These can come in the form of responses to questions, answers to problems, traces of the student's problem-solving behaviour, etc. The student model can in fact be thought of as a *compression* of these observations: the raw data is combined, some of it may be discarded, and the end result is a summarisation in the form of a set of beliefs about the student. The process of compression is usually defined by a set of inference rules mapping observations to beliefs. However, there are two potential sources of error in this process.

Firstly, the amount of raw data and observations may be insufficient to draw strong conclusions. In the extreme case, one could argue that any data is insufficient unless it can be shown to lead to statistically significant hypotheses. Of course, an ITS rarely has sufficient time to acquire enough data to hypothesise about the student's state to this degree, especially given that the state of the student can be expected to change rapidly.

Secondly, the inference rules for building the student model may themselves be sub-optimal. If the inference rules are inconsistent, incomplete or semantically inexplicable, then the quality of the data will have a reduced bearing on the quality of the resulting student model. In other words, poor inference rules will lead to a poor student model regardless of how reliable or unreliable the data acquired from the student is. But reliable data combined with quality inference mechanisms will lead to a quality student model.

Of course, determining the quality of data and inference rules is not a simple task. One can never guarantee that the data is truly representative of the current state of the student; it may just be random noise. Thus, building the student model is a highly uncertain activity.

There is another class of uncertainty in this process. Whereas the first type of uncertainty arose from the construction of the student model from data, the second type arises from the fact that some teaching actions, such as next problem, topic or hint selection, are selected on the basis of the student model. These actions, therefore, can be thought of as *functions* of the student model. As discussed above, if the student model is uncertain then clearly this uncertainty will transfer to the action selection functions. It will manifest itself in the form of actions being selected that are not optimally adapted to the student. However, if the student model is of a relatively high quality and the action selection functions are insensitive to small amounts of uncertainty, it may well be the case that the selected actions remain optimally near-optimally rational. Unfortunately, it is difficult to determine the degree of rationality of the action selection function, especially when the rules defining the function are not based on theory. In a hypothetical worst case, the action selection functions could make random decisions. Typically, however, the action selection function will be some heuristic that considers the values in the student model (e.g. one such heuristic is to give a hint on the rule that the student has most frequently violated in the past), though such an approach cannot guarantee that this is the best possible rule. This type of uncertainty, as well as the uncertainty inherent in the student model, can contribute to overall sub-optimal behaviour in the ITS.

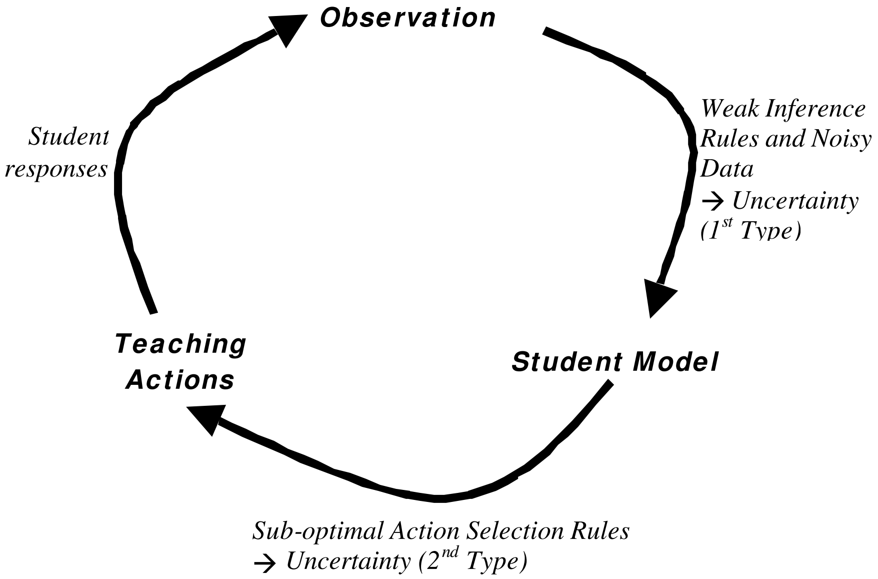

Observation
Weak Inference Rules and Noisy Data → Uncertainty (1st Type)
Student Model
Sub-optimal Action Selection Rules → Uncertainty (2nd Type)
Teaching Actions
Student responses

Fig. 1.2. Sources of uncertainty in an ITS.

Figure 1.2 depicts the two types of uncertainty and where they fit into the cycle of interaction between a student and the ITS. The ITS builds its student model from observations of the student, but noise and weak inference rules introduce uncertainty. The student model is then used to select teaching actions (e.g. problems or instruction), but again poor action selection rules increase the uncertainty. The student responds to the teaching actions, generating more data.

1.3 Towards A Solution: Normative Theories

The question that arises is how to minimise uncertainty. It is very difficult, if not impossible, to eliminate uncertainty of the first type. While the inference rules for building the student model may be optimised (and there is much research in this area), there is no way to guarantee the quality of the data. Uncertainty of the second type (which occurs because of sub-optimal action selection), however, presents a different challenge. Powerful general theories of decision-making, designed specifically for situations involving uncertainty, have been developed. One of them is Bayesian probability theory (Bayes, 1763), which deals with uncertain reasoning, and the other is statistical decision theory (Savage, 1954) that extends Bayesian probability to making decisions and incorporates a measure of preference for the outcomes of actions called utility. If the tenets of these theories are accepted (and to date there have been no substantial reasons why they should not be), then the theories define rationality. It is a challenge to ITS researchers, then, to define action selection functions based solely on these theories for incorporation in their systems. Doing so would eliminate uncertainty of the second type and make their systems more acceptable (i.e. more rational) in the process. A consequence of this is that when an ITS does exhibit irrational behaviour, the cause can be traced uncertainty of the first type rather than of the second.

To describe in more detail exactly how to produce a normative ITS, a general methodology is introduced in this thesis. The methodology uses machine learning and statistical significance tests to construct a Bayesian network student model from student performance data. It describes how to integrate this model with decision-theoretic procedures for PAS. A critical component of the methodology is evaluation in a real classroom. All too frequently, ITSs are evaluated only in the lab and never make it to the classroom. However, classroom evaluation is essential in order to obtain valuable data which can guide the further development of the system.

It is important to point out that normative theories like Bayesian probability and decision theory do not *replace* existing learning theories. Rather, they *complement* these theories. A normative theory is a general mechanism for reasoning under uncertainty. When normative theories are referred to as "optimal" in this thesis, it is meant that given a scenario defined as a probability distribution and a utility function, then a normative system can reason and behave optimally within the confines of the scenario. Normative theories themselves are independent of the scenario semantics. For example, although Bayesian probability theory provides all the mechanisms for representing and reasoning about uncertain knowledge, it does not specify what the knowledge that is being reasoned about actually is. That is the domain of psychological learning theories. To further clarify this, consider that a particular learning theory may be implemented in two different ways: with normative methods, or using an ad-hoc representation such as heuristic rules. If the learning theory is precisely defined, the normative implementation will guarantee that the learning theory is optimally applied to the student in the sense that it is rational, but the ad-hoc implementation may be sub-optimal.

1.4 Demonstrations of Normative ITSs

The effectiveness of normative theories was evaluated in the classroom using two different ITSs. The first system, SQL-Tutor (Mitrovic & Ohlsson, 1999), is an existing ITS for teaching the SQL database language to computer science undergraduate students. Domain knowledge is represented in the form of constraints (Ohlsson, 1994) that specify the form of consistent or correct solutions. For its long-term student model, SQL-Tutor originally had a simple frequency-based overlay model. A Bayesian overlay model replaced this (Mayo & Mitrovic, 2000), and a new method of next problem selection was the implemented. This latter version of SQL-Tutor was evaluated in October 1999.

The main demonstration of the effectiveness of normative theories, however, is the ITS CAPIT (Capitalisation And Punctuation Intelligent Tutor) (Mayo et al., 2000; Mayo & Mitrovic, 2001). CAPIT teaches basic literacy skills to school children in the 8-10 year old age group, and was fully implemented by the author. Like SQL-Tutor, CAPIT uses constraints to represent knowledge. However, CAPIT's student model is far more sophisticated than the simple Bayesian overlay in SQL-Tutor. CAPIT's student model was constructed by application of the methodology described in this thesis. As a consequence, it has a sophisticated, adaptive Bayesian network that can model complex interdependencies between student performance and the domain constraints. The output of the Bayesian network (which is a prediction about how the student will be behave in a given context) is the input to a decision-theoretic process for tutorial action selection. Effectively, CAPIT's entire mechanism for representing and reasoning about the student, and determining its own behaviour, is normative.

CAPIT was fully evaluated in the classroom over a period of four weeks during June 2000. Three classes participated in the evaluation. One of the classes did not have access to CAPIT and use used as a baseline for comparing the pre- and post-test results; the second used a "randomised" non-normative version of CAPIT, and the third class used the normative version of CAPIT. Both classes using CAPIT improved from pre-test to post-test. The pre- and post-tests, and the log file analysis, indicate that the class using the normative version of the system learned the constraints at a faster rate than the class using the randomised version.

1.5 Guide to the Thesis

As a guide to the remainder of this thesis, Chapter 2 introduces the technicalities of Bayesian and decision-theoretic reasoning. After the foundations of probabilistic reasoning using Bayesian networks are introduced, the specific algorithms used in the construction of CAPIT are described. Chapter 3 provides the context for this thesis within existing student modelling and ITS research. Firstly, existing student modelling approaches are surveyed. It was found that student modelling philosophies tend to focus on either long-term or short-term knowledge about the student. Most existing ITSs combine approaches of both types. Secondly, Bayesian and decision-theoretic approaches to student modelling and PAS are considered. It is only recently with the development of advanced algorithms for probabilistic reasoning that Bayesian methods have become feasible on-line. Hence, it is only in the last few years that Bayesian probability and decision theory have found their way into educational technology. In Chapter 4, the extensions to, and the evaluation results of, SQL-Tutor are described. This evaluation produced valuable lessons that led to the development of the general methodology and CAPIT. Chapter 5 describes this general methodology and justifies it. Chapter 6 shows how the methodology was used to construct CAPIT and details the extensive evaluation results. Finally, Chapter 7 presents the conclusions and discusses future directions for research in this area.

Chapter 2

Bayesian Networks and Decision Theory

Both Bayesian probability theory (Bayes, 1763) and statistical decision theory (Savage, 1954) are instances of *normative systems.* A normative system is a set of rules as well as the logical consequences of those rules (Gardenförs, 1989). Therefore, if logical reasoning and behaviour are assumed to define rationality, a normative system can be considered a model of rational behaviour. This implies that, given a specific normative model, the output from a normative system will be optimal for that model. Heuristic or ad-hoc systems cannot guarantee this because the inferences do not have to be logical consequences, and therefore they can be incomplete or inconsistent.

Bayesian probability theory is a normative model that deals with the problem of how to reason under uncertainty. The specific issues it deals with are how to represent uncertain beliefs, and given those uncertain beliefs, how to update them when evidence arrives or other beliefs change. Bayesian reasoning has attracted the interest of researchers in the ITS field recently (e.g. Gertner & VanLehn, 2000; Mayo & Mitrovic, 2000; Millán et al., 2000; Mislevy & Gitomer, 1996) because the theory is well established and rigorous, and therefore increases the prospect of mainstream acceptance of ITSs in general (Everson, 1995). Decision theory extends Bayesian probability by showing how to make decisions when not only the beliefs are uncertain, but also the outcomes of actions. It provides a method of “fusing” preferences with beliefs to give a measure on possible actions called the expected utility.

One aspect of normative systems is that because they are prescriptive, they specify *what* to compute for logical/rational behaviour (the prescriptive requirement) rather than *how* to compute it (the implementation). As a result, considerable research in recent times has been directed at the development of sophisticated algorithms for implementing normative reasoning (e.g. Lauritzen & Spiegelhalter, 1988). All the algorithms compute the same thing – they adhere to the tenets of the normative system – but their efficiencies vary considerably and depend to greater and lesser extents on the particular domain being modelled.

This chapter aims to give the reader a firm grounding in Bayesian and decision-theoretic technology. Detailed descriptions are given of the algorithms used both in the construction of CAPIT (specifically, the Bayesian network induction algorithms) and during the execution of CAPIT (the Bayesian network reasoning algorithm and the decision-theoretic expected utility formula). Although complete understanding of the algorithms and mathematics is not necessary to appreciate the main results of the thesis, the details are included because they provide the foundation of the CAPIT tutoring system described later. Taking the time to absorb these algorithms will give the reader an appreciation of how normative approaches to knowledge representation and reasoning differ from other approaches. The most significant mathematical results will be reiterated at the end of this chapter.

The prescriptive requirements of Bayesian probability theory are introduced in Section 2.1. A particular implementation of Bayesian probability – the Bayesian network (Pearl, 1988; D'Ambrosio, 1999) – is then introduced in Section 2.2. Bayesian networks streamline Bayesian reasoning by optimising both the storage requirements and the computation required for inference. This is shown in Section 2.3. They can also be induced from data, as described in Section 2.4. Finally, the prescriptive requirement of decision theory – the principle of maximising expected utility – is introduced in Section 2.5 before conclusions are drawn in Section 2.6.

2.1 Probability Basics

Bayesian probability theory deals with events and the probabilities of those events. If A is an event, then the probability of A is denoted by a real-valued number, $P(A)$. The basic axioms of probability theory (Bayes, 1763; Cowell, 1999) are:

1. $P(A) = 1$ if and only if A is certainly true.
2. $P(A) = 0$ if an only if A is certainly false.
3. $0 = P(A) = 1$.
4. If A and B are mutually exclusive, then $P(A \cup B) = P(A) + P(B)$.

It is pertinent to define a particular class of events, that of a variable X being with certainty in one and only one of the discrete states $x_1..x_n$. We denote the probability of this event by $P(X{=}x_i)$, and it follows from the axioms that:

$$\sum_{i=1}^{n} P(X = x_i) = 1 \tag{2.1}$$

The sequence of probabilities $P(X{=}x_1)$, $P(X{=}x_2)$, …, $P(X{=}x_n)$ define a *probability vector*. A useful shorthand way of referring to this vector is simply $P(X)$.

An important concept is that of the *conditional probability*, $P(X{=}x|Y{=}y) = r$. This represents the statement "If $Y{=}y$ is true, and no other information to hand is relevant to X, then the probability of $X{=}x$ is r." A table defining conditional probabilities for every possible combination of values that X and Y can take is called a *conditional probability distribution* and is denoted by $P(X|Y)$.

Conditional probabilities are essential to a fundamental rule of probability calculus, the *product rule*. The product rule defines the probability of a conjunction of events:

$$P(A \cap B) = P(A|B)P(B) = P(B|A)P(A) \tag{2.2}$$

Frequently, the literature shortens $P(A \cap B)$ to $P(A,B)$, and that convention will be followed here. $P(A,B)$ is also called a *joint probability distribution*, and like the conditional probability distribution, it is a table of values, one entry for each possible combination of values that its variables can jointly take. In the general case, a joint probability distribution over n variables can be defined recursively using the product rule (Equation 2.3):

$$\begin{aligned} P(X_1, X_2,\ldots, X_n) &= P(X_1| X_2,\ldots, X_n)P(X_2,\ldots, X_n) \\ &= P(X_1| X_2,\ldots, X_n)P(X_2| X_3,\ldots, X_n)P(X_3,\ldots, X_n) \\ &= P(X_1| X_2,\ldots, X_n)P(X_2| X_3,\ldots, X_n)\ldots.P(X_{n-1}|X_n)P(X_n) \end{aligned} \tag{2.3}$$

This property of joint probability distributions is called the *general factorisation property*. Note that the product rule allows any ordering of variables in the factorisation.

Rearranging the product rule leads to Bayes' famous theorem:

$$P(A \mid B) = \frac{P(B \mid A)P(A)}{P(B)} \tag{2.4}$$

Bayes' Theorem is frequently used for reasoning about an uncertain hypothesis A given evidence B, and in that context $P(A|B)$ is called the *posterior probability* of A, $P(A)$ is called the *prior probability* of A, and $P(B|A)$ is the *likelihood* of A. The factor ${}^{1}\!/_{P(B)}$ is a normalisation constant, and if ignored, Bayes' Theorem simplifies to:

$$P(A|B) \propto P(B|A)P(A) \tag{2.5}$$

This form of Bayes' Theorem is important because frequently the normalisation step can be left until the very end of a chain of calculations, making the computations more efficient.

While the product rule is used to construct joint probability distributions, *marginalisation* reduces a joint probability distribution to a distribution over a subset of its variables. More specifically, if we have the set of events $\mathbf{X} = \{X_1,$

X_2, ..., X_z} and a joint probability distribution $P(\mathbf{X})$ (equivalent to $P(X_1, X_2, ..., X_z)$), then we can find the *sub-joint probability distribution* $P(\mathbf{X_q})$ for any $\mathbf{X_q} \subseteq \mathbf{X}$ by applying the marginalisation rule:

$$P(\mathbf{X_q}) = \sum_{\mathbf{X}-\mathbf{X_q}} P(\mathbf{X}) \qquad (2.6)$$

This rule is effectively a summation over the variables that are not of interest (those being $\mathbf{X}$-$\mathbf{X_q}$), so the joint probability distribution collapses to a distribution over only those variables of interest ($\mathbf{X_q}$). For example, suppose we have the joint probability distribution $P(A,B,C)$ and we want to marginalise A. If B and C are binary (taking values `Y` or `N`), then using Equation 2.6 to calculate $P(A)$ leads to the following calculation:

$$\begin{aligned} P(A) &= P(A,B{=}\mathtt{Y}, C{=}\mathtt{Y}) + P(A,B{=}\mathtt{Y}, C{=}\mathtt{N}) \\ &\quad + P(A,B{=}\mathtt{N}, C{=}\mathtt{Y}) + P(A,B{=}\mathtt{N}, C{=}\mathtt{N}) \end{aligned} \qquad (2.7)$$

This completes the definition of the basic concepts of probability theory. Conditional and joint probability distributions have been defined, and it has been shown how joint probability distributions can be both constructed using the product rule and reduced via marginalisation. Bayes' Theorem, a rule crucial for Bayesian reasoning under uncertainty, has been introduced.

2.2 Bayesian Network Basics

A significant efficiency issue that any implementation of Bayesian probability theory must deal with is storage requirements. To illustrate, an explicit table representation of $P(X_1, X_2, \ldots, X_n)$ will, if n=16 and each X_i is binary, require 2^{16} = 65,536 different entries. Thus, a naïve direct application of the theory to an implementation would result in gross inefficiency in terms of storage requirements. A more efficient alternative is to define some economical function f that can be evaluated to yield specific entries from the joint probability distribution or a subjoint distribution thereof. This way, the entire joint probability distribution table would not need to be explicitly represented. For example, if we wished to determine an entry from a subjoint distribution, say $P(X_2=x_2,X_3=x_3)$, we could simply evaluate the function $f(X_2=x_2,X_3=x_3)$, instead of marginalising $P(X_2,X_3)$ from $P(X_1, X_2,\ldots ,X_n)$ and then looking up the appropriate entry in $P(X_2,X_3)$.

The representation of $P(X_1, X_2,\ldots ,X_n)$ as a factorisation of n-1 conditional probabilities and one probability vector (Equation 2.3) is a possible definition of f. Instead of representing the entire joint probability table explicitly, we represent each factor separately and multiply appropriate entries from each factor's table every time f is evaluated. This would be very efficient for some queries, such as $P(X_n=x_n)$, which is represented explicitly, and $P(X_{n-1}= x_{n-1}, X_n=x_n)$, which can be computed with a single multiplication using the product rule, but for other queries like $P(X_1=x_1)$, the new representation will be no more efficient. Ideally, the factors should be compressed even further.

One approach to achieving this is to look for a certain property of the conditional probability tables $P(X_1|X_2\ldots X_n)$, etc, called *conditional independence*. To illustrate conditional independence, consider the smaller joint distribution $P(A,B,C)$. The product rule and the general factorisation property mean that we can express this as:

$$P(A,B,C) = P(A|B,C)P(B|C)P(C) \tag{2.8}$$

Now, suppose that the factor $P(A|B,C)$ has the property that it is always equal to $P(A|C)$. That is, for every pair (a,c), $P(A=a|B,C=c)$ remains constant as B varies. We therefore say that A is conditionally independent of B given C. In standard notation, this is expressed as:

$$A \amalg B \mid C \tag{2.9}$$

We can therefore drop B from the conditional probability $P(A|B,C)$ altogether and rewrite the representation as:

$$P(A,B,C) = P(A|C)P(B|C)P(C) \tag{2.10}$$

Storage-wise, this new representation is more efficient than Equation 2.8. That is, if A, B and C are binary, then the factor $P(A|B,C)$ which requires 2^3 entries has been replaced by the factor $P(A|C)$ requiring only 2^2 entries.

Because of the efficiency gains that conditional independence gives us, Bayesian networks were developed to make the conditional independencies explicit. In a Bayesian network, a variable that conditions another variable in the factorisation (e.g. C conditions A and B in Equations 2.8 and 2.10, and B conditions A in Equation 2.8), becomes the parent of that variable in the network. It makes no sense to have directed loops in the network because this would represent a factorisation impossible to derive from the product rule. A Bayesian network, therefore, is a directed acyclic graph. Figure 2.1 depicts a Bayesian network for the distribution defined in Equation 2.8, and Figure 2.2 depicts a network for the more efficient representation defined by Equation 2.9.

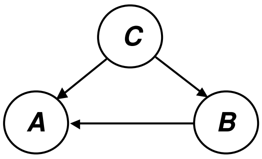


Fig. 2.1. A Bayesian network for $P(A,B,C) = P(A|B,C)\ P(B|C)P(C)$

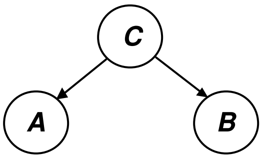


Fig. 2.2. A Bayesian network for $P(A,B,C) = P(A|C)\ P(B|C)P(C)$

The key advantage Bayesian networks give us is the ability to define the conditional independencies first, before specifying numerically the actual conditional probability distributions.

A general conditional independence property of Bayesian networks is that any variable X in the network is conditionally independent of its non-descendents **ND**(X) given its parents **PA**(X) (Pearl, 1988):

$$X \perp\!\!\!\perp \mathbf{ND}(X) \,|\, \mathbf{PA}(X) \tag{2.11}$$

That is, if a variable’s parents become known, then any more information about nodes that are not on a directed path from X will be irrelevant. This is the so-called *directed Markov property* of Bayesian networks. It effectively sets bounds on the influence of new evidence, an important consideration for efficient inference that will be discussed in more detail later.

The question arises as to how to use the directed Markov property to reduce the size of a general representation such as the factorisation in Equation 2.3. Now, the product rule does not limit the order in which variables are factorised, so therefore if an ordering of nodes can be found such that each variable in the factorisation is conditioned on only its parents and its non-descendents, then the non-descendents will “drop out” of the equation and each variable will be conditioned only on its parents. That is, we want to use conditional independence to simplify Equation 2.3 to:

$$\begin{aligned} P(X_1, X_2,\ldots, X_n) &= P(X_1|\,\mathbf{PA}(X_1))P(X_2|\,\mathbf{PA}(X_2))\ldots P(X_n|\,\mathbf{PA}(X_n)) \\ &= \prod_{i=1}^{n} P(X_i \,|\, \mathbf{PA}(X_i)) \end{aligned} \tag{2.12}$$

It happens that a suitable ordering of nodes $X_1, X_2,\ldots, X_n$ will always exist in a Bayesian network, and that ordering is called a *topological ordering*. A relatively simple algorithm can find the topological ordering (Cowell, 1999): initialise an empty list, then iteratively delete from the network any variable with no parents, and append it to the end of the list until all the variables have been appended. The list will then be a topological ordering of the nodes from which Equation 2.12 can be defined. Equation 2.12 is also known as a *recursive factorisation*, and is a standard method of numerically representing a Bayesian network. Note that a recursive factorisation corresponds to the topology of the Bayesian network, as there is one factor for each node and its parent set.

Changing the numeric representation of a Bayesian network from Equation 2.3 to Equation 2.12 is a major efficiency gain. To illustrate, consider the example joint probability distribution from the start of this section that had n=16 binary variables. Recall that 65,536 entries were required to explicitly represent $P(X_1,...,X_{16})$. Now however, if each variable is on average dependent on only two parents, then an average of $2^3 = 8$ conditional probabilities per factor need to be stored. This means that the entire joint probability distribution (specified by Equation 2.12) can be specified by approximately $8n = 128$ entries, a considerable saving. Of course, the size of the specification will increase as the number of dependencies increases, but, in practice, real-world Bayesian networks are sparsely connected and therefore they benefit from this representation.

While conditional independence is highly advantageous for specifying a Bayesian network compactly, Equation 2.12 does not capture all the conditions under which variables within a network are independent. More generally, two variables are *d-separated* if evidence about one cannot influence the other. To determine whether or not two nodes are d-separated, one must consider all the undirected paths between the two nodes. Any node on any of the paths may "block" the dependence along that path, and therefore if all the paths between the two variables are blocked at least once, the two nodes will be independent (i.e., d-separated). The smallest set of nodes that d-separates two nodes X and Y is called the *cut-set* of X and Y (Pearl, 1988).

Consider a node on a path in the Bayesian network. There are three classes of connection along that path: *serial*, *diverging*, and *converging*, and they are depicted in Figure 2.3.

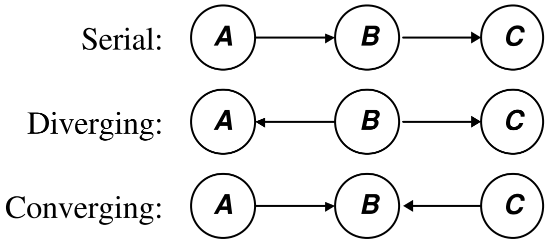


Fig. 2.3. Serial, diverging and converging connections to a node *B* on a path.

Serial and diverging connections block the path if they are instantiated. That is, if *B* becomes known and *B* is a serial or diverging connection, then *B* effectively d-separates *A* and *C*. Note that a diverging connection represents the special case of *A* being conditionally independent of its non-descendents given its parents, and is equivalent to Figure 2.2.

The interesting case is that of the converging connection. If *A* and *C* converge to *B*, it transpires that they will be d-separated as long as *B* and none of *B*'s descendents are observed; if *B* or one of its descendents are observed, *A* and *C* become dependent. This interesting property of converging connections arises because *A* and *C* represent multiple explanations or causes of the converging node *B*. For example, suppose *A* is the proposition that *Student X has mastered the topic* and *C* is the proposition that *Student X performs poorly on exams*. If *B* is the proposition that *Student X failed the exam*, then *A* and *C* are two possible explanations for *B* and would therefore converge to *B* in a Bayesian network. If we subsequently observe *B* (say, to find that the proposition is true and the student did fail the exam), then *B*'s causes become dependent because if *A* is subsequently observed to be true as well, then it (intuitively) has some bearing on *C* (we might revise the probability of *C* downwards) and vice versa.

Thus, d-separation characterises independence arising from lack of evidence as well as evidence. Note that any system for reasoning under uncertainty must capture these properties, as they are basic attributes of human reasoning (Jensen & Lauritzen, 2000).

Finally, Bayesian network semantics will be mentioned. Consider the arcs in Figure 2.1. By applying Bayes' Theorem, the direction of any arc can be reversed as long as a directed cycle is not induced. While changing the arc directionality may change the d-separation properties of the network, the overall joint probability distribution will be invariant. Therefore, technically, networks differing only in arc directionality can be considered equivalent. However, semantics are conventionally used to make particular configurations of arc directions unique. While not entailed by the underlying theories, the addition of semantics is convenient. The most common interpretation of an arc is causality: if *A* is a parent of *B*, then *A* is said to exert a causal influence on *B*, or precede *B* temporally, and not the other way around. Other semantics are certainly possible. For example, Collins et al. (1996) define arcs as pointing from skill to sub-skill, and Pearl (1988) has proposed a more general taxonomic hierarchy semantics for arcs.

2.3 Inference in Multiply-Connected Bayesian Networks

Inference is the general problem of computing the posterior probability $P(\mathbf{Q}|\mathbf{E}=\mathbf{e})$, for some evidence **E=e** and query **Q** (where $\mathbf{Q} \subseteq \mathbf{X}$ and $\mathbf{E} \subseteq \mathbf{X}$). We defined the posterior probability using Bayes' Theorem (Equation 2.4) in the previous section, but the problem is how to calculate this quantity efficiently. To illustrate how the naïve approach is inefficient, consider the Bayesian network depicted in Figure 2.4 and its recursive factorisation, Equation 2.13. The Bayesian network is a simple and contrived student model in which a student's mastery of the domain topics (T_1 and T_2) implies her mastery of the various concepts ($C_1..C_3$), which in turn influences her performance on test questions ($Q_1..Q_4$). The questions may require the student to have mastered more than one concept. With this Bayesian network, we can perform *inferential* queries, such as $P(Q_1|T_1=\texttt{Mastered})$; *diagnostic* queries such as $P(T_2|Q_3=\texttt{Failed})$; or a combination of both, such as $P(C_2|Q_4=\texttt{Correct}, T_1=\texttt{Not-Mastered})$.

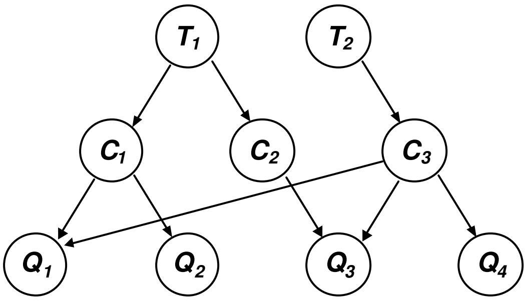


Fig. 2.4. Graphical structure of a Bayesian network.

$$P(\mathbf{X}) = P(Q_1|C_1,C_3)P(Q_2|C_1)....P(C_1|T_1)...P(T_1)P(T_2) \qquad (2.13)$$

Naïvely, we could compute $P(\mathbf{Q},\mathbf{E})$ and $P(\mathbf{E})$ by marginalising $P(\mathbf{X})$, and then use the product rule to calculate $P(\mathbf{Q}|\mathbf{E})$ from which the probability table corresponding to our specific evidence, $P(\mathbf{Q}|\mathbf{E}=\mathbf{e})$, can be read. However, this approach is in ge neral, intractable because it is a *global* computation. For even the simplest query, the marginalisation steps require summations and multiplications over all the variables in the network.

A more efficient approach utilises conditional independence and the topology of the Bayesian network to perform a series of more efficient local, rather than global, computations. *Local computation* in a Bayesian network is the process of computing a variable's posterior probability distribution from the posterior distributions of its neighbours – and only its neighbours. Thus, when evidence arrives at a node, its neighbours update themselves, then their neighbours update themselves, and so on, until the entire network "absorbs" the evidence. This process is analogous to propagation in a neural network, except that it is probabilistic consistency, as opposed to "activation", that spreads across the network.

Inference via local computation is highly efficient for singly-connected Bayesian networks (Pearl, 1988). Various algorithms with propagation time proportional to the number of variables in the network are described by Pearl (1988) and, specifically for implementation in an ITS, Murray (1999). However, the situation is more complex when the network is multiply-connected, because there are loops in the underlying undirected graph. To illustrate, consider Figure 2.5.

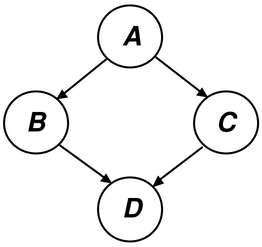


Fig. 2.5. A Bayesian network for $P(A,B,C,D) = P(D|B,C)P(B|A)P(C|A)P(A)$

Figure 2.5 depicts a Bayesian network that is perfectly legal because it has no directed cycles, but it does contain the undirected loop *A-B-C-D*. As a result, evidence arriving at, for example, *D*, can propagate via two different paths (*D-B-A* and *D-C-A*) to *A*. The problem with local computation is that *A* will be unable to detect that it is receiving the same evidence twice, as it updates only from its immediate neighbours *B* and *C*. This is not a problem for singly-connected networks because there is always at most one path between any two variables, but singly-connected networks are highly restrictive and not suitable for many domains. For example, even the simple Bayesian network depicted in Figure 2.4 is multiply-connected, as can be seen from an isomorphic rearrangement of the nodes depicted in Figure 2.6.

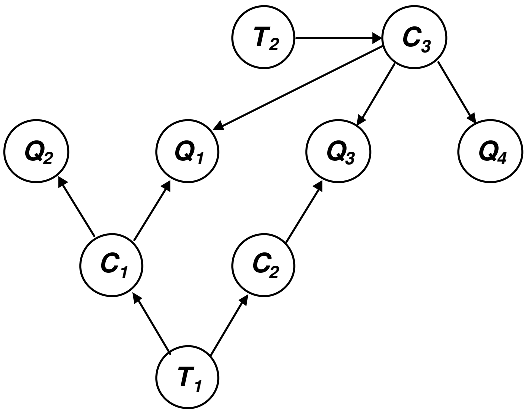
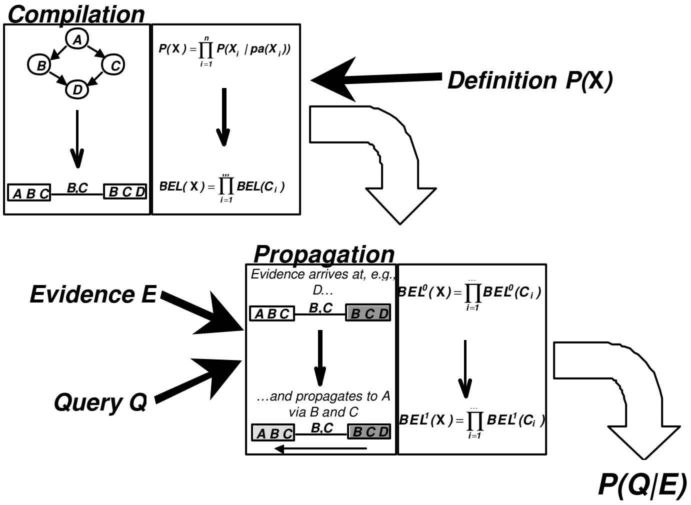


Fig. 2.6. The same Bayesian network as depicted in Figure 2.4, but with its nodes rearranged to make the undirected cycle T_1-C_2-Q_3-C_3-Q_1-C_1 explicit.

We therefore present a general algorithm for inference in multiply-connected Bayesian networks. The algorithm was introduced by Lauritzen & Spiegelhalter (1988) and further clarified by Jensen, Lauritzen et al. (1990) and Jensen, Oleson et al. (1990). It is most recently described in Jensen & Lauritzen (2000). Cowell et al. (1999) provides a tutorial introduction and overview of the algorithm for Bayesian network novices. The algorithm is a cornerstone for exact Bayesian network inference. Numerous variations (e.g. Kjaerulff, 1999) have since been proposed since in the literature, and the algorithm has been implemented in most of the common Bayesian network shells, e.g. HUGIN (Andersen, Oleson, et al., 1989), SMILE/GENIE (on World Wide Web at http://www2.sis.pitt.edu/~genie/), and MSBN (http://research.microsoft.com/ msbn).

The basic approach of the Lauritzen/Spiegelhalter algorithm (hereafter referred to as the "L & S" algorithm) is to transform the Bayesian network into a singly-connected structure, and then perform local computations on that structure rather than the original network. The algorithm can be thought of as having two stages: a compilation stage, in which the input is the original Bayesian network specification and the output is the singly-connected structure, and a propagation stage, in which evidence is absorbed and queries are performed on the new structure. If the network specification changes, compilation must occur again to produce an updated singly-connected structure before further queries can be performed.

Another way of considering the algorithm is to think of two parallel processes; one graphical, in which the network structure is manipulated, and one numerical, in which the probabilities are manipulated. The numerical calculations always correspond to the graphical operations. Figure 2.7 depicts the architecture and functionality of the algorithm from these viewpoints. The details of this diagram will be explained in the following sub-sections.



Compilation
A
B
C
D
A B C
B,C
B C D
$P(X) = \prod_{i=1}^{n} P(X_i \mid pa(X_i))$
$BEL(X) = \prod_{i=1}^{\dots} BEL(C_i)$
Definition P(X)
Propagation
Evidence E
Query Q
Evidence arrives at, e.g., D...
...and propagates to A via B and C
$BEL^0(X) = \prod_{i=1}^{\dots} BEL^0(C_i)$
$BEL^1(X) = \prod_{i=1}^{\dots} BEL^1(C_i)$
P(Q|E)

Fig. 2.7. Overview of the L & S algorithm.

2.3.1 Graphical Compilation

The graphical compilation procedure involves taking the original Bayesian network and transforming it into a *junction tree*. The junction tree representation is equivalent to the original Bayesian network, except that it is singly-connected even if the original network was multiply-connected. The generation of a junction tree requires five steps, which are listed in Table 2.1.

Step	
1	Marry co-parents.
2	Moralise network.
3	Triangulate network.
4	Form junction graph of cliques.
5	Form junction tree.

Table 2.1. Steps required to generate a junction tree.

The first two steps are relatively straightforward. The *marrying* of co-parents is the simple addition to the directed network of an edge between any two nodes that are parents of the same child, but not already neighbours. The *moralisation* of the network is the dropping of all directionality. That is, the directed network is turned into an undirected graph. The output of these two steps when applied to the example Bayesian network in Figures 2.4 and 2.6 is shown in Figure 2.8. The only non-adjacent co-parents in the original network are (C_1, C_3), and (C_2, C_3), and so edges between these pairs are added to the network. (The new edges are dashed in Figure 2.8.)

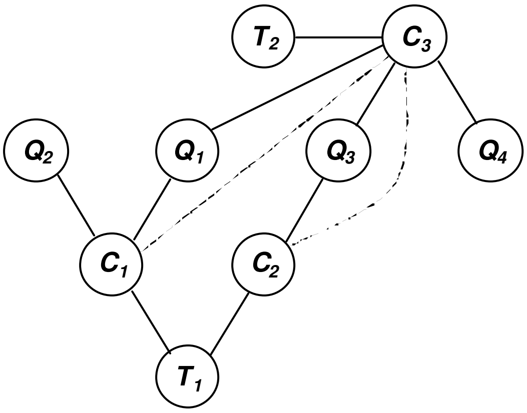


Fig. 2.8. The married, moralised graph.

The next step is *triangulation*: "short cuts" (or, in graph-theoretic terminology, *chords*) are successively added to every cycle of length 4 or more that does not already have a chord, until no such cycles exist. To illustrate, consider Figure 2.8. A number of cycles exist in this graph, such as Q_1-C_3-C_1 and C_3-C_1-T_1-C_2-Q_3. However, neither of these are candidates for shortening because the former is a cycle of length 3 (not 4 or more), and the latter already has a chord, C_2-C_3. In fact, there is only one cycle in Figure 2.8 appropriate for shortening, and that is the cycle C_3-C_1-T_1-C_2. There are two possible chords that could shorten this cycle: C_1-C_2 or T_1-C_3. In both cases, the new edge renders the graph fully triangulated. We have chosen to add the edge C_1-C_2, and the result is depicted in Figure 2.9. (An exact algorithm for triangulating the network will be discussed later.)

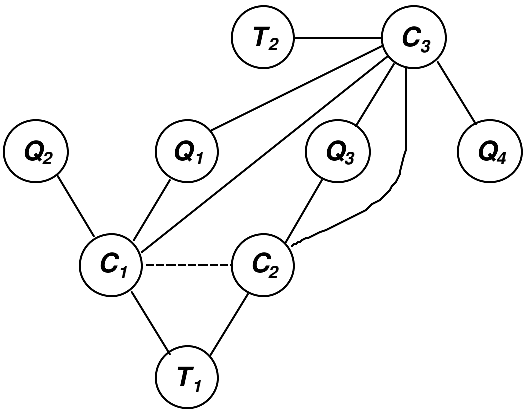


Fig. 2.9. The triangulated graph.

The fourth step involves firstly identifying the *cliques* in the triangulated graph, and secondly forming a new graph called a *junction graph*. A clique is a graph-theoretic concept defined as a "maximal, complete" subgraph. A subgraph is *complete* if every node in the subgraph is adjacent to every other node. For example, $\{C_1,C_2,T_1\}$ is a complete subgraph in Figure 2.9, but $\{C_3,Q_3,Q_4\}$ is not because there is no edge Q_3-Q_4. The *maximal* property adds the criterion that one cannot find another node in the network to add to the subgraph such that the subgraph will still be complete. In other words, $\{C_1,C_2,T_1\}$ is maximally complete because there is no other node that can be included in this subgraph whilst maintaining the completeness property. However, $\{C_1,C_3\}$ is complete but not maximally so because $\{C_1,C_3, Q_1\}$, the subgraph formed by adding Q_1, is complete. The cliques of Figure 2.9, therefore, are: $\{T_2, C_3\}$, $\{C_3, Q_4\}$, $\{C_1, C_2, C_3\}$, $\{C_2, C_3, Q_3\}$, $\{C_1, Q_2\}$, $\{C_3, Q_1, C_1\}$, and $\{C_1, C_2, T_1\}$.

In the junction graph, each node corresponds to a clique. Since there are seven cliques in Figure 2.9, there will be seven nodes in the junction graph.

Furthermore, variables from the original graph are likely to appear in more than one clique; to capture this in the junction graph, an edge is added between two cliques if their intersection is non-empty. Figure 2.10 is the junction graph derived from Figure 2.9.

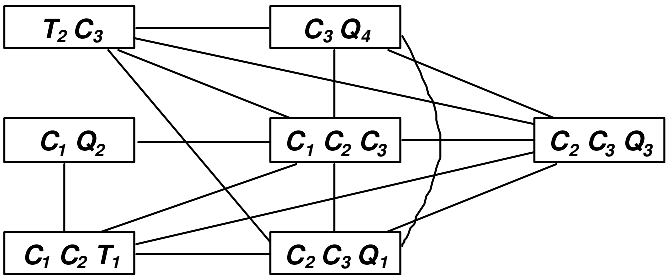


Fig. 2.10. The junction graph.

Recall that the motivation for the compilation stage is to produce a singly-connected structure on which inference via local computation is possible. This structure, the junction tree, is formed by simply "pruning" the junction graph until only a tree remains (see Section 2.3.5 for an algorithm that does this). However, the junction tree has an additional property not present in the junction graph; namely, the *running intersection property*: if any two cliques in the junction tree contain a mutual variable *X* from the original network, then every clique on the path between those two cliques must also contain *X*. This ensures that the junction tree does not have two or more disconnected "representations" of the same variable. The running intersection property thus restricts the way in which a junction graph can be "pruned" to a junction tree. A junction tree for Figure 2.10, with the clique intersections labelling the edges, is depicted in Figure 2.11.

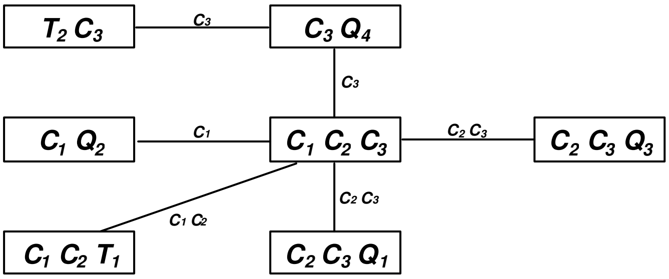


Fig. 2.11. A junction tree with the running intersection property.

2.3.2 Numerical Compilation

A logical mapping exists between the original form of a Bayesian network and its recursive factorisation. For each variable *X*, there is one and only one factor *P(X*| **PA**(*X*)*)* in the recursive factorisation. Now that the original network has been converted into a junction tree, a new numeric representation with a logical mapping between cliques and factors can be derived. More specifically, if **C** is the set of cliques in the junction tree, a suitable new representation is:

$$P(X_1, X_2,\ldots, X_n) = \prod_{i=1}^{n} P(X_i \mid \mathbf{PA}(X_i)) = \prod_{\mathbf{C}_i \in \mathbf{C}} P(\mathbf{C}_\mathbf{i}) \tag{2.14}$$

where *P(*$\mathbf{C}_\mathrm{i}$*)* is called a *clique marginal*, which is simply a joint probability distribution over the variables in the clique.

One important change in the processing from now on is that *beliefs* rather than probabilities will be computed. A belief is equivalent to a probability, with the exception that individual beliefs are allowed to exceed 1, and there is no ensuing requirement that the entries in a belief table or vector must sum to unity. When beliefs are utilised, the relative rather than absolute differences between values becomes important. Other than that, beliefs and probabilities are identical. Beliefs allow for slightly more efficiency in the processing because the tables do not need to be normalised after each calculation. A belief table, denoted by *BEL*, can always be transformed into a probability table by dividing every entry in the belief table by the normalisation constant *Z*, i.e.:

$$P(\mathbf{X}) = Z^{1} BEL(\mathbf{X}) \tag{2.15}$$

where *Z* is the sum of the entries in *BEL(*$\mathbf{X}$*)*. We therefore aim to transform the recursive factorisation of the Bayesian network into a product of beliefs (rather than probabilities) on the clique marginals. Thus, our new representation will be:

$$P(X_1, X_2,\ldots, X_n) \propto BEL(X_1, X_2,\ldots, X_n) = \prod_{\mathbf{C}_i \in \mathbf{C}} BEL(\mathbf{C}_i) \qquad (2.16)$$

The process by which the clique marginals $BEL(\mathbf{C}_1)$ etc., are derived from the original conditional probabilities is straightforward and involves rearranging the terms of the recursive factorisation. Cowell et al. (1999, pp. 34) define an algorithm to achieve this, which is given in Listing 2.1. The algorithm transforms the recursive factorisation into a *potential representation*, from which the clique marginals can easily be derived via marginalisation.

```
• For each clique $\mathbf{C_i} \in \mathbf{C}$, define a function
  $a_i(\mathbf{C_i})$.
• Initialise $a_i(\mathbf{C_i})$ := 1 for each $\mathbf{C_i}$.
• For each factor $P(X|\mathbf{PA}(X))$ in the recursive
  factorisation:
    o Find one clique $\mathbf{C_i}$ containing both $X$
      and $\mathbf{PA}(X)$ and redefine $a_i(\mathbf{C_i})$ :=
      $a_i(\mathbf{C_i}) P(X|\mathbf{PA}(X))$.
```

List. 2.1. Defining a potential representation from a recursive factorisation (Cowell et al., 1999, pp. 34)

More mathematical details of this process are available in Lauritzen & Spiegelhalter (1988) and Jensen et al. (1990).

2.3.3 Graphical Propagation

Having constructed a junction tree and transformed the representation of the Bayesian network into a product of clique marginals (Equation 2.16), evidence can now be propagated and queries performed. Propagation on a junction tree starts with a single clique receiving evidence, and its neighbours successively *calibrate* themselves to absorb the evidence. The evidence “flows” via the variables that are the intersection of the neighbouring cliques. Figure 2.12 depicts the propagation of a single piece of evidence on a junction tree consisting of two cliques. The evidence arrives at **CL$_i$** and propagates to **CL$_j$** in one time step.

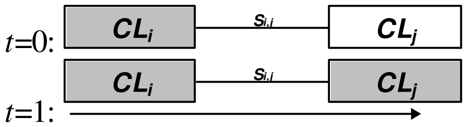


Fig. 2.12. Evidence arrives at **CL$_i$** at time *t*=0, and **CL$_j$** calibrates itself to **CL$_i$** at time *t*=1.

Propagating single pieces of evidence is relatively simple and this process could be used to sequentially propagate multiple pieces of evidence. However, a major advantage of the L & S algorithm is that multiple evidence items can be propagated on a junction tree *simultaneously*, and therefore much more efficiently (especially for parallel-processor implementations of the algorithm, e.g. Kozlov & Pal Singh (1994)). Figure 2.13 depicts a case where evidence is propagating to a single clique **CL$_j$** from two of its neighbouring cliques, **CL$_i$** and **CL$_k$**.

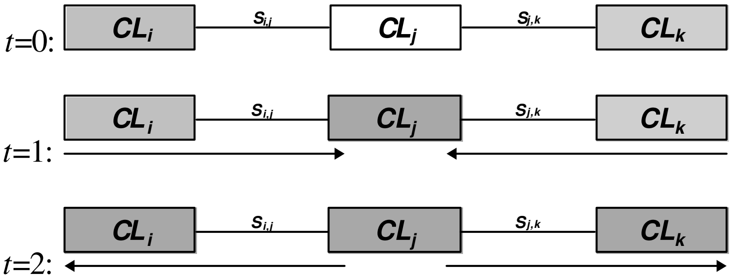


Fig. 2.13. Evidence arrives at **CL$_i$** and **CL$_k$** at time *t*=0. **CL$_j$** calibrates itself to both **CL$_i$** and **CL$_k$** at time *t*=1, then **CL$_i$** and **CL$_k$** calibrate themselves to **CL$_j$** at *t*=2.

An object-oriented scheme for propagation in junction trees is provided by Jensen et al. (1990). In this formulation, each clique in a junction tree is an object with two simple recursive methods: *DistributeEvidence* and *CollectEvidence*. The *DistributeEvidence* method has the clique asking each of its neighbouring cliques to calibrate themselves to it, and then it recursively calls *DistributeEvidence* in all of them. In this way, evidence is propagated around all the cliques in the tree. *CollectEvidence* is the inverse: the clique calls

CollectEvidence in all of its neighbours, then calibrates itself to them. Using these simple methods, this approach to Bayesian reasoning can be easily conceptualised.

2.3.4 Numerical Propagation

The numerical formulation of propagation consists of two rules for updating clique marginals: a simple rule for propagating evidence from one clique to another, and a more generalised rule for absorbing multiple evidence. Both rules are derived from the product rule.

Let $BEL^t(\mathbf{X})$ denote the belief distribution over variables in set **X** at time *t*. Consider two adjacent cliques in the junction tree, $\mathbf{CL_i}$ and $\mathbf{CL_j}$. Now, $\mathbf{CL_i}$ and $\mathbf{CL_j}$ must have a non-empty intersection in order to be adjacent, and that intersection can be defined as another set:

$$\mathbf{S_{i,j}} = \mathbf{CL_i} \cap \mathbf{CL_j} \qquad (2.17)$$

$\mathbf{S_{i,j}}$ is called the *separator* of $\mathbf{CL_i}$ and $\mathbf{CL_j}$. Now consider the remainder of both cliques when the separator is subtracted:

$$\mathbf{R_i} = \mathbf{CL_i} - \mathbf{S_{i,j}}, \mathbf{R_j} = \mathbf{CL_j} - \mathbf{S_{i,j}} \qquad (2.18)$$

$\mathbf{R_i}$ and $\mathbf{R_j}$ are called the *residuals*.

Because a clique can be viewed as the union of the residuals and separators, a clique marginal can be defined by the product rule as:

$$BEL^t(\mathbf{CL_i}) = BEL^t(\mathbf{R_i}, \mathbf{S_{i,j}}) = BEL^t(\mathbf{R_i}|\mathbf{S_{i,j}})BEL^t(\mathbf{S_{i,j}}) \qquad (2.19)$$

and by marginalisation, we can compute the subjoint belief over the separators from the joint belief over the clique:

$$BEL^t(\mathbf{S_{i,j}}) = \sum_{\mathbf{CL_i}-\mathbf{S_{i,j}}} BEL^t(\mathbf{CL_i}) \qquad (2.20)$$

Note that we can apply Equation 2.20 to both of the cliques $\mathbf{CL_i}$ and $\mathbf{CL_j}$ to get $BEL^t(\mathbf{S_{i,j}})$. Therefore, to indicate from which clique the distribution over separators was computed, a subscript *i* is used. For example, $BEL_i^t(\mathbf{S_{i,j}})$ indicates a distribution computed by applying Equation 2.20 to the belief over the *i*th clique, $BEL^t(\mathbf{CL_i})$, and not the *j*th clique, $BEL^t(\mathbf{CL_j})$.

Given that two neighbouring cliques $\mathbf{CL_i}$ and $\mathbf{CL_j}$ can both compute belief distributions over their common separators, we say they are *consistent* if $BEL_i^t(\mathbf{S_{i,j}}) = BEL_j^t(\mathbf{S_{i,j}})$, and *inconsistent* if $BEL_i^t(\mathbf{S_{i,j}})\ ?\ BEL_j^t(\mathbf{S_{i,j}})$. Ideally, they should always be consistant. However, if evidence arrives at one clique but not its neighbouring cliques, then that clique will become inconsistent with its neighbours. To absorb the evidence, the neighbouring cliques make themselves consistent via calibration. Thus, “propagation” or “evidence absorption” in a junction tree is the successive calibration of cliques to ensure consistency.

Calibration of a single item of evidence is a relatively simple process. Suppose evidence arrives at clique $\mathbf{CL_i}$ at t=0, as in Figure 2.12. $BEL^0(\mathbf{CL_i})$ is the posterior belief distribution over the clique given the evidence, but so far its neighbour has not been updated. Therefore, $\mathbf{CL_j}$ will be inconsistent with $\mathbf{CL_i}$ and must calibrate.

By Equation 2.19, $BEL_j^0(\mathbf{S_{i,j}})$ is a factor of $BEL^0(\mathbf{CL_j})$. To calibrate, we essentially replace $BEL_j^0(\mathbf{S_{i,j}})$ in the factorisation with $BEL_i^0(\mathbf{S_{i,j}})$, and compute the new belief distribution over the clique, $BEL^1(\mathbf{CL_j})$. That is, we delete the factor reflecting that state-of-affairs prior to the evidence ($BEL_j^0(\mathbf{S_{i,j}})$), and insert an equivalent factor that does incorporate the evidence ($BEL_i^0(\mathbf{S_{i,j}})$). This new factor also makes the cliques consistent. The entire operation can be captured by a single equation:

$$BEL^1(\mathbf{CL_j}) = \frac{BEL^0(\mathbf{CL_j})}{BEL_j^0(\mathbf{S_{i,j}})} BEL_i^0(\mathbf{S_{i,j}}) \qquad (2.21)$$

By successive application of Equation 2.21, evidence propagates from clique to clique throughout the junction tree.

A generalisation of Equation 2.21 is used to update a clique (e.g. **CLj**) when evidence arrives at more than one of its neighbours (e.g. **CLi** and **CLk**), as in Figure 2.13. To do this, the separators between the clique and each of its neighbours must be considered. (In this example, the separators are **Si,j** and **Sj,k**.) The update rule allowing **CLj** to absorb evidence from both sources simultaneously is a generalisation of Equation 2.21:

$$BEL^1(\mathbf{CL_j}) = \frac{BEL^0(\mathbf{CL_j})}{BEL_j^0(\mathbf{S_{i,j}})BEL_j^0(\mathbf{S_{j,k}})} BEL_i^0(\mathbf{S_{i,j}})BEL_k^0(\mathbf{S_{j,k}}) \qquad (2.22)$$

Now that **CLj** has been updated, it will still be inconsistent with its neighbours because the evidence from **CLi** has not propagated to **CLk** and vice versa; it has only got as far as **CLj**. The final step is to apply Equation 2.21 to calibrate **CLi** and **CLk** to **CLj**:

$$BEL^2(\mathbf{CL_i}) = \frac{BEL^0(\mathbf{CL_i})}{BEL_i^0(\mathbf{S_{i,j}})} BEL_j^1(\mathbf{S_{i,j}}),\ BEL^2(\mathbf{CL_k}) = \frac{BEL^0(\mathbf{CL_k})}{BEL_k^0(\mathbf{S_{j,k}})} BEL_j^1(\mathbf{S_{j,k}}) \qquad (2.23)$$

To summarise, the numeric propagation part of the L & S algorithm is basically the propagation of consistency from a clique to its neighbours in a junction tree (a process called calibration). Consistency is a property belonging to pairs of neighbouring cliques, and is achieved when marginalising on the variables shared by both neighbours yields the same belief distribution.

Finally, the last step by which a general query *P(*$\mathbf{Q}|\mathbf{E=e}$*)* is calculated will be mentioned. We have already shown how the junction tree absorbs the evidence **E=e**. The final step is to find one or more cliques containing the variable(s) **Q**, and marginalise *BEL(***Q***)* from them. *BEL(***Q***)* can then be normalised to yield the posterior probability distribution over **Q**.

2.3.5 Efficiency and Implementation Issues

The compilation stage of the L & S algorithm contains two steps that admit many possible solutions: graph triangulation (Step 3), and junction tree construction (Step 5). In this section, some algorithms for graph triangulation and junction tree construction are described.

Graph triangulation has long been known to be an NP-Hard problem (Yannakakis, 1981). As a result, considerable research has been undertaken to develop heuristic, near-optimal solutions. One common approach is called *one-step look ahead triangulation* (Cowell et al., 1999) and is shown in Listing 2.2. Basically, this approach takes each node and its neighbours, and "fills in" edges between them to form a completely connected subgraph in the network.

Note that the order in which nodes are numbered depends on a user-defined criterion, *c(V).* Nodes can be selected to maximise or minimise this quantity. While the algorithm is relatively straightforward, its efficiency and the quality of the resulting triangulation will depend on how *c(V)* is defined. Cowell et al. (1999, pp. 58) suggest that *c(V)* be set to the size of the subjoint distribution over *V* and its neighbours, and therefore the function should be minimised.

```
• Start with all vertices unnumbered.
• Set $i := n$, where $n$ is the number of nodes in the graph.
• Do until there are no more unnumbered vertices:
    o Select an unnumbered node $V$ that optimises the criterion $c(V)$.
    o Number it with $i$ (i.e. set $V_i := V$).
    o Form the set $\mathbf{C}_i$ consisting of $V_i$ and its neighbours in the graph.
    o Add edges between the nodes in $\mathbf{C}_i$ to make $\mathbf{C}_i$ a complete subgraph.
    o Decrement $i$.
```

List. 2.2. One-Step Look Ahead Triangulation (Cowell et al., 1999, pp. 58).

Graph triangulation is not always necessary. It is quite possible that following the marrying and moralisation steps of the compilation, the network will already be triangulated and that therefore the execution of one step look ahead is unnecessary. *Maximum cardinality search*, given in Listing 2.3, is an algorithm for determining whether or not a graph is triangulated. It is a highly efficient algorithm that operates in O(*n*+*e*) time, where *n* is the number of nodes and *e* is the number of edges in the network. Note that **ne**(*V*) is a function returning *V*'s neighbours in the network, but excludes *V* itself.

```
• Set Output := "Graph is Triangulated".
• Set $i$ := 1.
• Set $\mathbf{L}$ := {}
• Set $\mathbf{V}$ to all the nodes in the network.
• For each node $V \in \mathbf{V}$, set $c(V)$ := 0.
    o Do while $\mathbf{L}$ ? $\mathbf{V}$:
    o Set $\mathbf{U}:=\mathbf{V}-\mathbf{L}$
    o Select any $V$ from $\mathbf{U}$ maximising $c(V)$ and
      number it $i$ (i.e. set $V_i := V$)
    o Set $\mathbf{P}_i$ := $\mathbf{ne}(V_i)$ n $\mathbf{L}$.
    o If $\mathbf{P}_i$ is not a complete subgraph, Then:
              Set Output = "Graph is not
              triangulated".
      Else
              Set $c(W)=c(W)+1$ for each $W \in$
              $\mathbf{ne}(V_i)$ n $\mathbf{U}$.
    o Set $\mathbf{L} := \mathbf{L} \cup \{V_i\}$
    o Increment $i$.
• Report Output.
```

List. 2.3. Maximum Cardinality Search (Cowell et al., 1999, pp. 55).

Besides determining the triangulatedness of a graph, maximum cardinality search also provides a useful numbering of the nodes $V_1 \ldots V_k$. This ordering is special because it enables the efficient construction of a junction tree with the running intersection property. Thus, even if it is known that the graph is triangulated, maximum cardinality search is necessary to obtain the node ordering. Listing 2.4 is an algorithm that takes the numbered ordering of nodes produced by maximum cardinality search and outputs not only the cliques

CL$_1$... **CL**$_p$ for the junction graph, but also a particular numbering of those cliques that is useful for building the junction tree from the junction graph..

- Start with the node numbering $V_1...V_k$ and the sets $\mathbf{P}_1...\mathbf{P}_k$ obtained by maximum cardinality search.
- Denote the cardinality of $\mathbf{P}_i$ by p_i.
- Call V_i a "ladder node" if $i=k$ or if $i<k$ and $p_{i+1}<1 + p_i$.
- Denote the jth ladder node, in ascending order, by λ_j.
- Define the clique $\mathbf{CL}_j = \{\lambda_j\} \cup \mathbf{P}(\lambda_j)$.

List. 2.4. Finding the Cliques of a Triangulated Graph (Cowell et al., 1999, pp. 56).

The ordering of the cliques produced by Listing 2.4 is crucial for the final algorithm presented here: junction tree construction. Listing 2.5 shows how to construct such a junction tree from the cliques and the clique ordering. Other algorithms for finding the optimal junction tree have been proposed by Jensen & Jensen (1994) and Kjaerulff (1992).

- Associate a node of the tree with each clique $\mathbf{CL}_i$.
- For $i=2..p$, add an edge between $\mathbf{CL}_i$ and $\mathbf{CL}_j$ where j is any one value in $\{1,…,i-1\}$ such that $\mathbf{CL}_i \cap (\mathbf{CL}_1 \cup \mathbf{CL}_{i-1}) \subseteq \mathbf{CL}_j$.

List. 2.5. Junction Tree Construction (Cowell et al., 1999, pp. 55).

To summarise, maximum cardinality search is an algorithm for determining if a network needs to be triangulated or if it is already in the triangulated state. If it is not triangulated, one-step look ahead triangulation is an effective near-optimal algorithm to triangulate the graph. Maximum cardinality search also orders the nodes in the network in a special way, such that it becomes easier to find the cliques and construct a junction tree with the running intersection property, as demonstrated by the relatively simple algorithms shown in Listings 2.4 and 2.5.

2.3.6 Summary

Various other exact inference algorithms besides L & S have been proposed, but it has been shown that many of them contain a hidden triangulation step (Shachter et al., 1991). However, a key advantage of the L & S algorithm is that the computationally difficult triangulation step is shifted to the compilation stage, and only needs to be invoked once following the specification of the Bayesian network, rather than once per query. Propagation on the junction tree is therefore mostly highly efficient, as long as the specification of the Bayesian network does not change between queries.

Propagation will not be efficient, however, when the networks is highly connected or nearly complete. In this case, the junction tree will clearly have a high ratio of cliques to variables, and therefore propagation time will be high. Fortunately for most real-world problems, the number of cliques compared to the number of variables is much lower, and therefore the L & S algorithm is an effective choice of inference algorithm.

2.4 Bayesian Network Induction

Bayesian networks can be learned from data. In this section, two classes of Bayesian network induction algorithm are introduced and described. The first class deals with the induction of the network's conditional probability tables (the *parameters*) from data when the structure of the Bayesian network is already known. The second class deals with the induction of the Bayesian network structure itself.

2.4.1 Parameter Learning

An approach to parameter induction when no data is missing is described by Heckerman (1999) and Krause (1998). Let $X{=}x_k|\ \mathbf{PA}(X) = \mathbf{pa}(X)$ (which can be shortened to $x_k|\mathbf{pa}(X)$), represent an observation of variable X in state x_k when its parents $\mathbf{PA}(X)$ are in state $\mathbf{pa}(X)$. A standard Bayesian network maintains, for each possible $x_k|\mathbf{pa}(X)$, a single conditional probability $P(x_k|\mathbf{pa}(X))$. An approach to learning conditional probabilities is to treat $P(x_k|\mathbf{pa}(X))$ itself as an uncertain variable, and to calculate its expected (average) value from the data. By assuming that the probability distribution over $P(x_k|\mathbf{pa}(X))$ is a special type of distribution called a Dirichlet distribution, the expected value then corresponds to the frequency. Suppose $x_k|\mathbf{pa}(X)$ has been observed α_k times while $\mathbf{PA}(X)=\mathbf{pa}(X)$ has been observed α times. Obviously $\alpha_k=\alpha$. Then, it is proven by Heckerman (1999) that the expected value of $P(x_k|\mathbf{pa}(X))$ is:

$$E[P(x_k|\mathbf{pa}(X))] = \frac{\alpha_k}{\alpha} \tag{2.24}$$

If $\mathbf{PA}(X)=\mathbf{pa}(X)$ is observed a further N times, while N_k further observations of $x_k|\mathbf{pa}(X)$ are made, the expected value of the conditional probability updates simply to:

$$E[P(x_k|\mathbf{pa}(X))|\text{observations}] = \frac{\alpha_k + N_k}{\alpha + N} \tag{2.25}$$

This is a useful rule for on-line learning. The parameters α and α_k are known as *sufficient statistics*, because they are adequate to define the conditional probabilities of a Bayesian network. Once they have been calculated, the training data from which they came can be discarded.

Other more complex approaches to parameter induction deal with cases of incomplete and noisy data (Bauer et al., 1997; Neal & Hinton, 1999).

2.4.2 Structural Learning

An algorithm for learning the structure of a Bayesian network from data is described in Cheng et al. (1997, 1998). The algorithm makes the assumption that the higher the *mutual information* between two variables in the data, the more likely it is that an arc should connect them in a Bayesian network. The mutual information between *X* and *Y* is defined as:

$$I(X,Y) = \sum_{X,Y} P(X,Y) \log \frac{P(X,Y)}{P(X)P(Y)} \qquad (2.26)$$

The algorithm has three phases: *drafting*, *thickening* and *thinning*. In the first phase, *I(X,Y)* is computed for every pair of variables (*X*,*Y*). The pairs are then sorted into the list *L* according to their mutual information, from highest to lowest. Pairs for which *I(X,Y)* is less than some small threshold *e* are excluded from *L*. Starting with the pair (*X*, *Y*) in *L* that has the highest mutual information, if there is no undirected path between *X* and *Y* in the Bayesian network so far, an undirected edge is added between *X* and *Y*, and (*X*,*Y*) is deleted from *L*. This is repeated successively until *L* contains only pairs of variables that are not directly adjacent, but are connected via a longer path. The output of this phase is either one singly-connected network spanning the entire network, or multiple disconnected singly-connected networks.

To illustrate the algorithm in action, let us consider the induction of a Bayesian network with five variables *A*,..., *E*. Suppose eight pairs of nodes have a mutual information greater than *e*, and that they are ordered from highest mutual information to lowest yielding *L* = {*A*-*B*, *B*-*E*, *E*-*C*, *A*-*C*, *B*-*C*, *A*-*D*, *D*-*C*, *D*-*E*}. Now, *A*-*B*, *B*-*E*, and *E*-*C* are added directly to the network since there is no path already between them. *A*-*C* and *B*-*C* cannot be added because the addition of the first three edges resulted in a paths connecting *A* and *C*, and *B* and *C*. *A*-*D* is next on the list to be added, and after this is done, the network has become singly-connected. The remaining pairs in *L* cannot be connected with an edge because a path already exists between them. The edges remaining in *L* are {*A*-*C*,*B*-*C*,*D*-*C*,*D*-*E*}, and the state of the network after drafting is depicted in Figure 2.14.

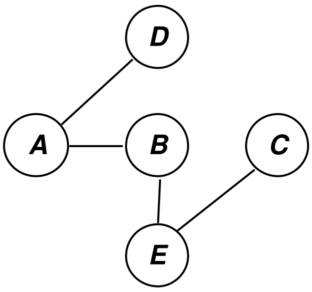


Fig. 2.14. The network after the drafting stage.

The next step, thickening, uses d-separation to add additional edges to the network. The algorithm considers each of the remaining (*X, Y*) pairs in *L*. It then invokes a search procedure to find the cut-set that d-separates *X* and *Y*. If a cut-set cannot be found, then *X* and *Y* must be dependent and so an edge is added between them. After this stage, the algorithm is guaranteed to have found all the edges in the final Bayesian network. Figure 2.15 depicts the thickened network. Edges *A-C*, *B-C* and *D-C* have been added from *L* because no cut-sets can be found d-separating these pairs of nodes.

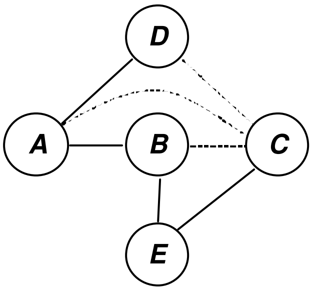


Fig. 2.15. The network after the thickening stage. New edges added from *L* are dashed.

However, unwanted surplus edges may have been added as a result of the linear order in which edges are added to the network from *L*. For example, suppose an edge is added between X_1 and Y_1 because no cut-set can be found that d-separates them, and assume that later in the sequence the edge X_2-Y_2 is also added. It may be the case that X_2-Y_2 is sufficient to create a cut-set d-separating X_1 and Y_1, thus obviating the need for the edge X_1-Y_1 in the first place. X_1-Y_1 is thus a surplus edge.

The third step, thinning, compensates for this problem. It considers every pair of adjacent nodes (*X*,*Y*) and temporarily removes the edge *X*-*Y* from the network. It then attempts, using a procedure similar to that used in the previous step, to find a cut-set between *X* and *Y*. If such a cut-set is found, edge *X*-*Y* is permanently removed from the network; otherwise, it is returned. The output of this step is the final (undirected) structure of the Bayesian network. Consider our example. Note that edge *A*-*C* was added to the network before edges *B*-*C* and *D*-*C*. If it is the case that the addition of these latter two edges results in a cutset (e.g. {*B*,*D*}) d-separating *A* and *C*, then the thinning step would remove *A*-*C* permanently. The result of this is depicted in Figure 2.16.

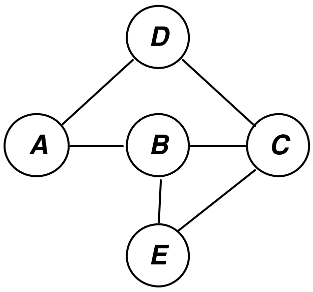


Fig. 2.16. The network after the thinning stage. Note that the edge *A*-*C* has been dropped.

All that remains is to orientate the edges. Consider the node *B* on a path *A*-*B*-*C* in Figure 2.16. Recall that if *B* is a converging connection, then *B*'s neighbours *A* and *C* on the path will be independent until *B* or one of its descendents is instantiated, at which point they become dependent. Therefore we can analyse the data to determine all the triplets of variables along a path in the network having this property, and thereby identify all the converging connections in the network. The remaining nodes must be either serial or diverging connections. In theory, this approach may not orient any edges at all (e.g. consider a Bayesian network with no converging connections at all), but in practice, a very high proportion of edges are oriented. When an edge cannot be oriented, the orientation task is left to the domain expert. In the case of our example, nodes *D* and *B* are found to be converging connections on all their paths, which allows every edge except *E*-*C* to be orientated.

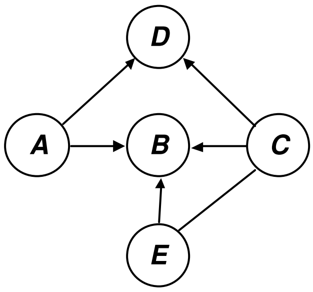


Fig. 2.17. The network after its edges have been oriented. Note that edge *E-C* cannot be oriented.

2.5 Decision Theory Basics

In this section, decision theory (Horvitz et al., 1988; Savage, 1954) will be briefly introduced. Whereas Bayesian networks are used to update beliefs from initial beliefs and observations, decision theory is a rational means of optimising behaviour by "fusing" uncertain beliefs with preferences. Suppose a rational agent is faced with the problem of selecting a single action from a set of possible actions $D = \{d_1, d_2\ldots, d_q\}$. (It is possible that one of the actions is the decision to perform no action.) If $X = \{x_1, x_2, \ldots, x_n\}$ represents the possible outcomes of *D*, then decision theory requires the agent to have a real-valued preference *U(X, D)* defined for each combination of values that *X* and *D* can take where $P(X|D)$ is non-zero. *U(X,D)* is known as the *utility function*, and it is assumed that the agent's preferences can in fact be translated into such a numeric form. Sometimes it is more convenient to encode the agent's preferences along more than one dimension – this is permitted and is known as the *multi-attribute utility*, but it will not be discussed here.

The agent must also be able to estimate the conditional probability distribution of *X* given *D*, a value that we have already shown can be efficiently calculated with a Bayesian network. One can then characterise a decision problem as a tree in which each path from root to leaf is composed of a sequence of edges alternately labelled with a decision and an outcome. The simplest case of a single decision *D* followed by a single outcome *X* is depicted in Figure 2.18.

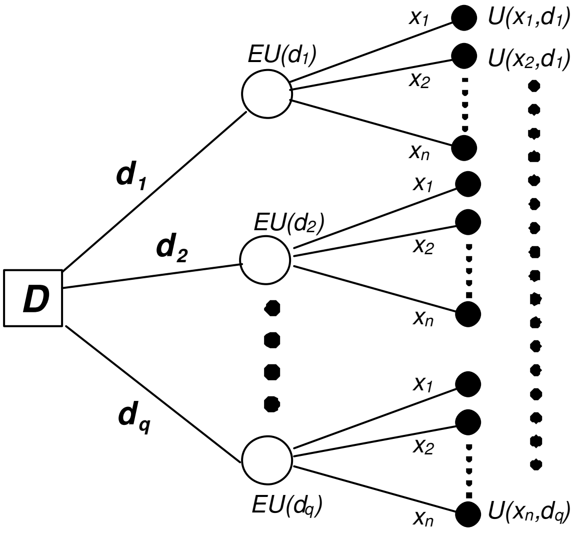


Fig. 2.18. A decision tree for a single decision.

The interpretation of the decision tree is as follows. The square nodes represent decision points where the decision-maker chooses an action $D=d_i$ and thereby selects a branch of the tree to follow. The circular nodes are chance nodes where "nature" selects an outcome/branch $X=x_j$ with probability $P(X=x_j| D=d_i)$. When a leaf node has been reached, the process is over and the final state has some utility $U(X=x_i, D=d_j)$ as defined by the utility function. Working backwards from leaf to root, one can calculate the expected utility (see below) of each alternative $\{d_1, d_2..., d_q\}$ from the individual utilities on the leaves. The chance nodes are also labelled with the expected utilities. Note that this scheme can be extended to modelling multiple decisions rather than just one. For example, if one is going to make two decisions D_1 followed by D_2 with outcomes X_1 and X_2 respectively, one could define a decision tree in which the nodes representing D_2 and X_2 succeed the chance nodes representing the first set of outcomes, X_1.

A more compact, though less clear and explicit, representation is the *influence diagram* (Howard & Matheson, 1984). An influence diagram is the extension of the Bayesian network to incorporate nodes representing decisions (squares) and utility functions (diamonds). Figure 2.19 illustrates an influence diagram equivalent to the decision tree in Figure 2.18. As in a Bayesian network, the arcs indicate causality, so the outcome X is interpreted as being caused by the decision D that is made, and both D and X lead to the utility U. D is treated as if it is a normal node in a Bayesian network, with the exception that it must always be instantiated to one of its values. The utility node U specifies a utility value for each combination that its parents can take. Clearly, then, one can see that the diagram is specified by exactly the same information as the decision tree; namely, a utility function $U(X,D)$ and a conditional probability distribution $P(X|D)$.

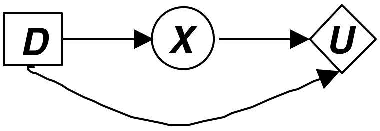


Fig. 2.19. A simple influence diagram.

The expected utility of an action is a measure defined as the probability-weighted sum of the utilities of each possible outcome of the action:

$$E[U(X,D)] = \left(\sum_{X} P(X \mid D) U(X,D) \right) \qquad (2.27)$$

The principle of maximising expected utility states that the agent should select the action $D=d_i$ that maximises this quantity.

Expected utility is best explained by considering the decision tree in Figure 2.18. Each leaf in the tree has a utility. However, prior to making the decision D, the outcomes of D are uncertain. That is, if a decision such as $D=d_3$ is selected, then it is possible to follow any path through the tree, as long as it contains the edge d_3 and the probability of the outcome is positive. The expected utility accounts for this uncertainty by averaging the utilities of the potential outcomes.

Another explanation can be made in terms of the influence diagram in Figure 2.19. Because one of the parents of U, namely X, can be uncertain, one must allow for this uncertainty. In other words, the uncertainty in X propagates to U in a similar fashion to the way uncertainty propagates between nodes in Bayesian networks. The utility function U can therefore be thought of as having a probability distribution over different possible utility values, and the expected utility is simply the average value of this distribution.

Upon reflection, it is apparent that considering only the next *single* decision is not truly optimal. A decision $D_1=d_1$ may have a low expected utility, but it may well be the case that when D_1 is considered in conjunction with the decision that will follow it, e.g. D_2, then one of the decision *sequences* starting with $D_1=d_1$ may in fact have the highest expected utility. In other words, one should ideally consider all possible future action sequences, not just the next action. In practice, however, an agent is limited computationally and must make a decision within a finite amount of time. This is certainly the case with decision-theoretic ITSs, which must make the decisions on-line. However, some systems do perform look-ahead beyond the next single action. This is most important in planning in robotics (e.g. Dean, 1990).

It is also possible to take into account the cost of a decision. This is a factor independent of the decision's outcomes, but it may be important. For example, in medical diagnosis a test such as a biopsy may be highly informative (and therefore have high expected utility) compared to a simpler test (with lower expected utility), but the higher cost of the biopsy may make the simpler test more appealing initially. The expected utility function can therefore be redefined to account for cost:

$$E[U(X,D)] = \left(\sum_{X} P(X \mid D) U(X,D) \right) - cost(D) \qquad (2.28)$$

The practice of limiting the look-ahead of an agent and incorporating cost into the decision-theoretic calculations is an extension to decision theory known as *bounded rationality* (Simon, 1976).

Computationally, Equation 2.28 is straightforward enough to apply without requiring complex algorithms for optimisation, provided the $P(X|D)$ factors are calculated efficiently. It has already been shown in Section 2.3 that efficient algorithms exists for this.

2.6 Summary

Two normative theories, Bayesian probability theory and decision theory, have been introduced. The former is a system for reasoning under uncertainty, and the latter a rule for acting under uncertainty. Both are highly relevant to ITS design, because they provide a rational means for the specification and application of learning theories. The heart of Bayesian reasoning is Bayes' Theorem (Equation 2.4) which shows one how to compute the posterior probability of a hypothesis from its prior probability and an observation. A Bayesian network and its associated algorithms are an efficient means of representing and evaluating Bayes' Theorem for multiple hypotheses and observations. Decision theory combines the posterior probabilities with preferences, and by a process of weighted averaging, assigns an expected utility to each alternative. The alternative with maximum expected utility is always selected.

A number of mathematical results were introduced in this chapter. Conceptually speaking, the most significant are Equation 2.3, which shows how to construct a joint probability distribution, and the definition of marginalisation (Equation 2.6) that defines the mechanism for extracting individual probabilities from joint distributions. Bayes Theorem (Equation 2.4) is essential for a conceptual understanding of how reasoning is implemented. The fundamental mathematical representation of Bayesian networks is given by Equation 2.12. Regarding decision theory, the main result is Equation 2.28 that defines the expected utility of an action. Technically, Equations 2.22 and 2.23 define the focal point of Bayesian inference using the Lauritzen-Speigelhalter algorithm. Equations 2.24 and 2.25 are the key rules for induction of Bayesian network conditional probabilities when learning from data, and Equation 2.26 (the definition of mutual information) is the central measure used to determine the structure of induced Bayesian networks. These key results are applied later in the thesis.

Other models of uncertain reasoning such as fuzzy sets (Zadeh, 1983) and Dempster-Shafer theory (Shafer, 1986) are not as general as probability theory (Horvitz et al., 1988). One reason for the proliferation of alternate schemes for uncertain reasoning in AI was the intractability of the direct application of the tenets of probability theory (Jensen, Lauritzen et al., 1990). However, since then recent algorithms for implementing normative theories such as the one described in Section 2.3 have overcome this problem. It has been demonstrated that Bayesian networks optimise the spatial storage requirements of a joint probability distribution's representation, and that the compilation of a junction tree on which inference is performed optimises the time per query. The induction of Bayesian networks from data has also been discussed and illustrated. In Chapter 6, an ITS with a Bayesian network student model learned from data (both off- and on-line), and pedagogical strategies based on expected utility maximisation, is introduced.

Chapter 3

The Student Model and Its Applications

A *student model* is defined by Holt et al. (1994) as "...*a representation of the computer system's beliefs about the learner and... therefore, an abstract representation of the learner...*" Student modelling therefore encompasses the entire spectrum of possible attributes that a student can have. However, it is important to distinguish between student modelling in practice and the more general task of user modelling. A student model is a core component of many intelligent computer-based instructional systems, and research on student modelling has mainly focused on representing traits of the student directly related to the desired pedagogical outcomes of such systems, such as the student's mastery of the domain and their domain-specific behaviour. User modelling, in comparison, is much more general and research can be focused on traits other than mastery. User modelling domains are typically non-teaching domains.

It is useful to consider student modelling in the light of instructional design and learning theory. There are three main learning theories: *behaviourism*, *cognitivism*, and *constructivism*. The first two theories are certainly compatible with the notion of modelling the student, but constructivism presents some difficulties. However, student modelling is not entirely at odds with constructivist theories.

The oldest learning theory is behaviourism, which treats the learner as if he/she is a black box. The behaviourist learner is similar to a machine that produces responses when exposed to stimulus. The task of learning is to teach the student a particular response to be made when a particular stimulus is presented, a process known as conditioning. Repeatedly presenting the stimulus and reinforcing correct responses while punishing incorrect responses is one means of achieving this. Instructional goals based on behaviourism are characterised by being "specified, quantified, terminal behaviours" (Mergel, 1998). For example, a student is said to have mastered the domain when he or she scores more than 90% on a test.

Cognitivism builds on behaviourism by postulating that learning is the "*…acquisition or reorganisation of the cognitive structures through which humans store and process information.*" (Good & Brophy, 1990, pp. 187). In other words, cognitivism explains learning as the formation and reformation of mental representations of the domain knowledge. So the cognitivist learner is definitely not a black box, and cognitivist models describe this internal representation. The knowledge structures are called *schema*, and schema can be combined, extended or altered to accommodate new information. Memory in this model is divided into sensory (which lasts up to four seconds), short-term (which can be retained for up to 20 seconds, and has a maximum capacity of 7 plus or minus 2 items), and long-term (in which "deeper" processing such as the generation of linkages occurs). The theory also proposes a number of "effects" (Mergel, 1998), which can increase the efficiency and effectiveness of learning, such as the "organisational effect" (categorised information is easier to remember than uncategorised information) and the "meaningful effect" (new information related to existing schema is easier to learn).

The third and final theory is constructivism. Whereas behaviourism and cognitivism are *objective* theories of learning, in which predetermined behaviours and/or cognitive structures are transferred to the student, constructivism is a *subjective* theory in which learners are said to construct their own reality based on experience. New knowledge is formed, rather than transferred, from the learner's previous experiences. Their existing mental structures and beliefs are used to interpret objects and events. Thus, every student is expected to construct his or her own unique reality. Furthermore, constructivism says that domain knowledge should not be decomposed and presented to the student in parts; instead, learning should take place in realistic settings with all the ambiguities and extraneous details that entails. As a result, the student will be able to construct knowledge better adapted to the context in which they will apply it. There are a number of other important constructivist principles following from this subjective view of learning.

Where does the traditional ITS with a student model fit within this psychological framework? The notion of using a test to assess the mastery level of student is decidedly behaviourist. ITSs that also attempt to model the internal state of the student are clearly cognitivist in origin. However, constructivism is at odds with the traditional ITS. If every student constructs a reality unique to himself or herself, based on his or her prior knowledge and experiences, then it becomes futile to assume that a pre-specified student model can be a reasonable description of the student. Furthermore, the traditional ITS decomposes domain knowledge into parts and teaches students part-by-part. This is the basis for traditional student modelling: if the student has mastered 50% of the parts, for example, then the ITS considers that she has mastered 50% of the domain knowledge. This is completely at odds with constructivism, which advocates realistic settings in which problem-solving is learned in its full and glorious detail without decomposition. Because of these disparities, learning software based on constructivist principles has in the past been based on the exploratory or simulation environment. There is no direct intervention or instruction, and students are left to explore the environment at their own pace.

However, as pointed out by Akhras & Self (2000), constructivism and student modelling can be compatible if the “intelligence” in the ITS is directed at supporting constructivist principles. For example, an important constructivist principle is reflection, and the student needs to reflect on problem-solving in order to construct their knowledge. If the student demonstrates an inability to reflect, for example, then this information could be stored in a student model. The student model, in turn, is used to tailor the environment such that reflection is encouraged. In this way, student modelling can be compatible with constructivism.

Another issue is that recent research has shown that some students may lack the meta-cognitive skills required to make efficient use of constructivist environments. For example, Aleven & Koedinger (2000) provide evidence that children lack the ability to seek help (or even recognise when they need help) in the domain of Geometry. It may well be the case, therefore, that student model-directed intervention is suitable for certain types of student in certain domains. This view is supported by Mergel (1998), who states that novices can often become "lost" in constructivist environments because they have no prior knowledge on which to build. A number of theorists have advocated a transitory model of learning, where novices are taught initially in an objective behaviourist or cognitivist fashion, before progressing to a subjective constructivist environment when they become intermediate-level students (Mergel, 1998). However, there is no currently no universal agreement and further research should be undertaken in this area from an ITS perspective.

Having shown where student modelling fits into instructional science, the remainder of this chapter reviews approaches to student modelling from two different perspectives. The first perspective (Section 3.1) considers the various approaches to student modelling that have arisen over the field's history, discussing the advantages and disadvantages of each. Student models are classified according to their content, and the persistence of their representation (whether it is long- or short-term). Over the past twenty years, the student modelling field appears to have become more cognitivist as increasingly detailed cognitive models have been advanced. However, the drawback of this is a decrease in the tractability of these models. The second perspective (Section 3.2) has a much more recent and constrained focus, considering only Bayesian student modelling. Bayesian network-based student models, e.g. Gertner & VanLehn (2000) and Millán et al. (2000), have become popular recently and it is fitting, therefore, to examine the various different approaches in order to place the work of this thesis in context. Section 3.3 continues in this vein, examining how Bayesian network student models have been used to solve particular pedagogical decision tasks such as problem selection. Section 3.4 summarises the chapter.

3.1 Student Models: A Survey

Student models can be classified by a number of factors, ranging from how they are generated, to their content, and then to their application. In this section, two factors are used as the basis for classification. Firstly, the *persistence* of the representation is considered important. Do the beliefs about the student typically last for a short duration, such as subsequent to a single problem attempt, or are they retained to build up a more long-term model of the student? Most ITSs implement both types of representation, having a short-term representation that is used to update a long-term student model. For example, SQL-Tutor (Mitrovic & Olhsson, 1999) combines overlays (the long-term representation) with Constraint-Based Modelling (CBM, the short-term representation). A particular approach may be more specific to one type of representation (long-term or short-term) than the other. For example, CBM is basically a definition of a short-term representation that has little to say about how the student should be modelled long-term.

The second factor is *content*. What beliefs about the student does the ITS actually model? The answer to this question generally depends on the persistence. Short-term beliefs must, by their nature, be very specific, e.g. "the student violated rule *X* on problem *Y*". These beliefs can be inferred or they might be observed matters of fact. The long-term model, on the other hand, typically contains a much greater proportion of inferred beliefs. These beliefs can also be much more abstract, such as beliefs about the student's domain mastery, misconceptions, and behaviour. Figure 3.1 depicts the categorisation of student models that will be considered in this section.

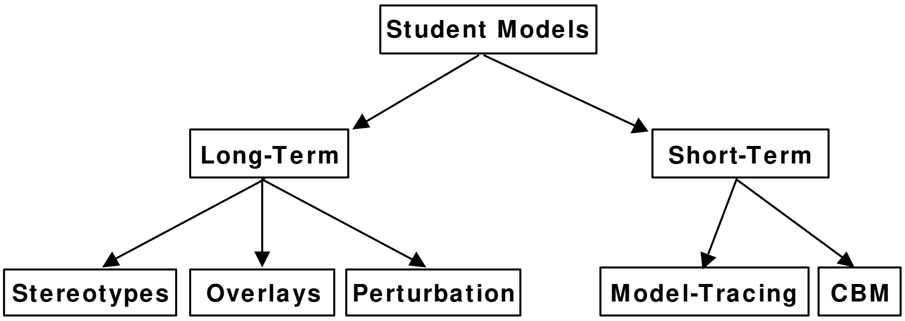


Fig. 3.1. Approaches to student modelling.

The main distinction between the four long-term student modelling approaches is as follows. *Stereotypes* are the simplest student model. They are simple abstract classifications of students into groups that are fixed, either permanently or initially. *Overlays* are more detailed representations of the student. They focus on modelling the student's domain knowledge as a subset of a domain expert's knowledge. The domain must therefore be decomposed into a set of items, and the overlay is simply a set of masteries over the items. *Perturbation* models add common misconceptions and other "bugs" to the overlay model, so the student's knowledge becomes an overlay on the expert's knowledge plus domain mal-knowledge. These three approaches are discussed in more detail in Section 3.1.1 to 3.1.3.

Two approaches which have different stances on short-term student modelling are *model tracing* and *constraint-based modelling* (CBM). The main distinction between the two is that model-tracing tutors represent both the procedural and declarative knowledge possessed by the student, whereas constraint-based tutors represent only the declarative knowledge. The student modelling philosophy of model tracing is discussed in Section 3.1.5, and CBM is described in Section 3.1.6.

3.1.1 Stereotypes

Two types of stereotype exist, the *fixed* stereotype and the *default* stereotype.

Fixed Stereotypes

Fixed stereotyping is the simplest approach to student modelling whereby the student's responses cast the student into a predefined stereotype. A basic example is to assign to each student a level. For example, WPS-Tutor (Wheeler & Regian, 1999) is an ITS for teaching algebra and geometry word problem solving skills to children. Problems are divided into levels, and each level is only slightly more difficult than the previous level. When the student solves two or more problems without help on the current level, the level increases. In WPS-Tutor, the level is the main description of the student in the student model. Fixed stereotyping makes the broad assumption that all students with the same stereotype will have the same domain-specific behaviour.

Furthermore, although the students may change their stereotype from session to session, the stereotypes themselves do not change or adapt.

Another recent example of fixed stereotyping is from the domain of software engineering ethics, in which protocol analysis identified five different stereotypes (Winter & McCalla, 1999). Interestingly, in this domain, personality types rather than mastery was modelled. Students were presented with hypothetical scenarios and then asked questions about the course of action they would take. Each stereotype represented a particular ethical mindset, such as the "straight and narrow" student who is very concerned about ethical issues but deficient about her/his responsibilities, and the "opportunistic" student who would copy source code from a rival company but not breach the privacy of co-workers and so forth. Out of the 82 students who completed the scenarios, a total of 15 did not fit any of the stereotypes. Students were also able to "role play" by responding in different ways on subsequent runs through the scenarios.

Milne et al. (1996) also describe fixed stereotyping. ATULA is a system for teaching mathematical network theory. Empirical data was collected from 134 students prior to their first use of the system. The data consisted of personality and background information acquired by questionnaire (which included psychological personality tests). Post-tests were also administered. The personality and background information was the primary input to the statistical cluster analysis; subsequent "fine tuning" of the clusters was performed using the post-test results. In the end, six male and six female student stereotypes were derived. The aim was to use the stereotypes as a guide for the teaching actions of the system. When a new student uses the system, the probability of the student being in each stereotype is calculated after they complete the same initial questionnaires. The student is then assigned to the most probable stereotype. However, the student's stereotype may also change during the session. For example, if the stereotype contains an assumption about the student's ability on a particular task, and this is observed not to be the case, then it may be necessary to switch the student to a more appropriate stereotype.

In general, fixed stereotyping is a very coarse-grained representation of a student and the approach is not useful for more complex analysis. It is also questionable whether fixed stereotyping is even valid because it may be the case that the stereotypes vary from one group of students to another. However, clearly for open domains where the knowledge cannot be easily decomposed into atomic units (such as software engineering ethics), this approach may be the only realistic student-modelling alternative.

Default Stereotypes

A more flexible approach to stereotyping is to consider the settings of each initial stereotype to be "default" values only. Thus, the students are only stereotyped when they start using the system, and as observations are made and evidence arrives about the student, the settings of the initial stereotype are gradually replaced by more individualised settings. This approach can be used in conjunction with an overlay model, where the default stereotype provides the initial values for the overlay. This approach is used in a number of systems surveyed by Kay (2000a).

An example of an ITS with a default stereotype student model is STyLE-OLM (Dimitrova et al., 1999), a learning environment for scientific terminology. The student engages in a superficial form of natural language dialogue with the system about the domain concepts. The system has a set of rules for building up its own default beliefs about the student's beliefs from this dialogue. For example, if the student misuses a term or demonstrates an incorrect belief, then inferences drawn from these can be added to the student model. However, these default inferences are open to scrutiny by the student. By the process of dialogue, the student is able to examine and revise the beliefs in the student model.

In the extreme case of default stereotyping, there may be only one default stereotype representing the "average" student, but that stereotype can be quickly adapted. This is the approach of MANIC (Stern et al., 1999), which initialises its student models to a default called the "population student model" that is derived from empirical analysis. MANIC is discussed in more detail in Section 3.2.

3.1.2 Overlay Models

The classic student model is the overlay model. This model projects a simple measure of mastery onto the elements of the domain that an expert would be expected to have mastered. The student's domain knowledge is therefore represented as a subset of the expert's knowledge. This is a highly tractable approach to student modelling because the specification is very abstract. As long as the expert's knowledge can be broken down into generic items (e.g. rules, facts, etc.), then an overlay model can be constructed. Figure 3.2 depicts an overlay model for a simple domain that can be decomposed into ten skills or items.

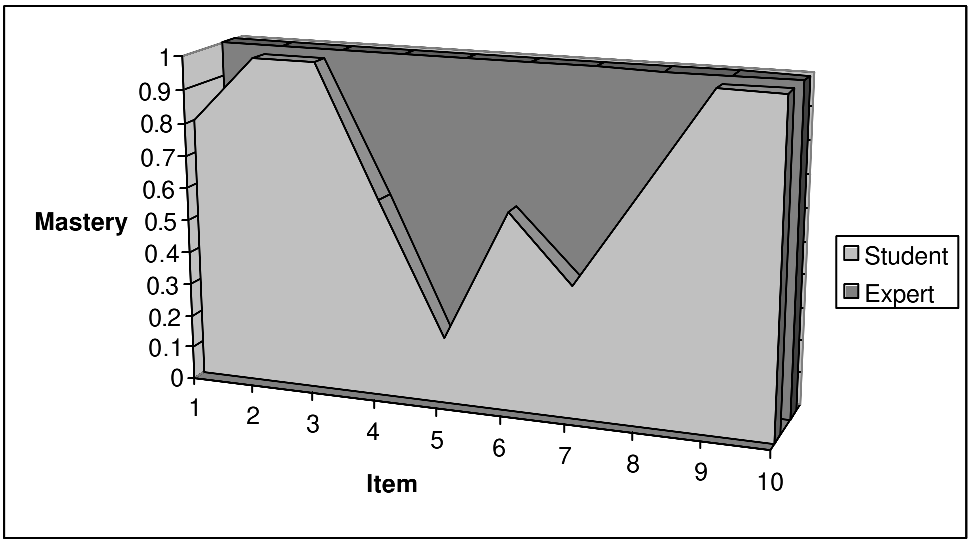


Fig. 3.2. An overlay student model.

The mastery of each item in the example ranges from 0 (complete novice) to 1 (expert mastery). An expert is thus represented by an overlay with 1 for each item. A student is represented by an overlay with *at most* 1 for each item.

Interestingly, there are at least two different interpretations of the measure of mastery in the literature. In some systems, mastery is considered a binary variable that can be `mastered` or `not-mastered`, and the overlay on the item is the system's belief that the item is in the mastered state. This could be termed the probabilistic interpretation, and is representative of ITSs that use Bayesian probability for student modelling. On the other hand, the measure could also be interpreted as an actual state of the student. So, for example, if the mastery of an item is 0.5, it means that the system believes that the student has only half- or semi-mastered the skill and needs more practice, but is at least not a complete novice at the skill. This is representative of the more traditional style of the overlay model. In practice there should be little difference in behaviour between systems with differing interpretations, as both will act to maximise the measure. However, it is interesting to note that a value of 0.5 actually represents the state of maximum uncertainty when the probabilistic interpretation is used, and so diagnostic strategies may produce different results depending on whether the measure is considered probabilistic or absolute.

The overlay model can also be specified at any level of granularity, and if it is specified at a low level of granularity, then rules can be defined to compute the overlay at a higher level. This is the purpose of OLAE (VanLehn & Martin, 1997). The measure of mastery is in many systems a simple function of the frequency with which the item has been used correctly or incorrectly (e.g. Clancey, 1983,1987, Kay, 2000b), or some function of the frequencies of different observations of student behaviour, e.g. Bloom et al. (1997). “Bounded” representations where the measure is uncertain but between a lower and upper limit have also been proposed (Elsom-Cook, 1990).

More recently, Bayesian probability theory has attracted the interest of ITS researchers who have implemented the theory in their overlay models (e.g. Millan, 2000; Reye, 1998). The probabilistic overlay is a set of uncertain, probabilistic variables representing the student’s mastery of a domain. The main advantage this approach is that the overlay model can be updated in a non-ad-hoc way from observations. That is, Bayes’ theorem (Equation 2.4) is available compute the posterior probability of domain mastery from its prior probability of mastery and observations made about the student. Bayesian networks also provide a graphical and therefore intuitive means of defining the dependence relationships between domain items.

Differential model

A more structured variant on the overlay model is the differential model (Holt et al., 1994). The differential model is essentially an overlay on *expected knowledge*, which in turn is an overlay on the expert’s domain knowledge. Expected knowledge is domain knowledge that a student should have at a particular point in time. Figure 3.3 depicts a differential model. In this example, only items 5 and 7 have masteries lower than the expected mastery. Instruction should therefore focus on these two items.

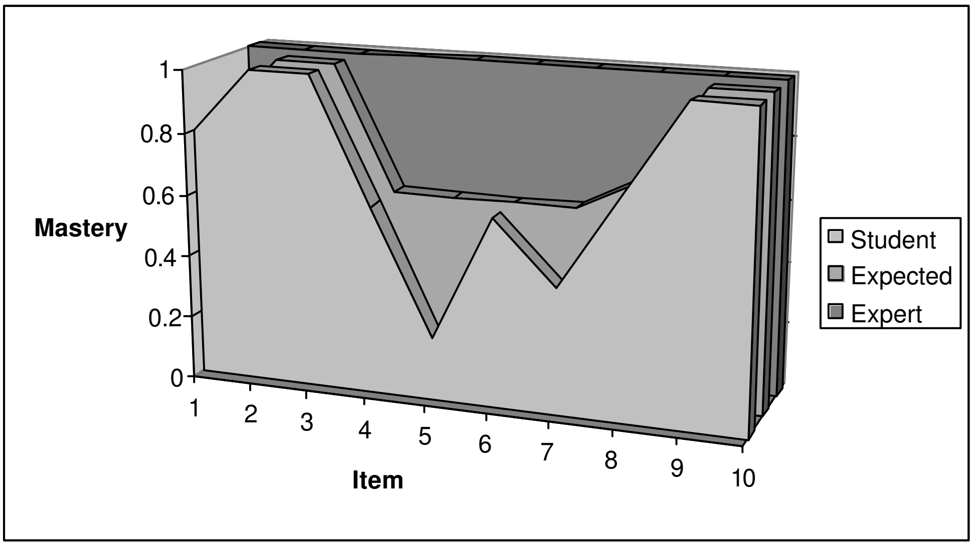


Fig. 3.3. A differential overlay student model.

In a sense, the differential model is less strict than the overlay model because it considers only gaps in the expected knowledge to be significant; no inferences are made about the student's mastery outside of the expected knowledge. For example, if it is expected that the student knows fact *A*, but it is not expected that the student knows *B*, then if the student fails to demonstrate both *A* and *B*, the differential modeller can infer that the student does not know *A*, but cannot infer anything about *B*. A system with a differential model is WEST (Burton & Brown, 1978).

3.1.3 Perturbation Models

A significant problem with the overlay and differential models is their presumption that the student's knowledge is a subset of the expert's knowledge. This assumption completely disregards the fact that real students can infer facts and rules of the domain that are totally false, as a result of mis- or preconceptions about the domain, or faulty reasoning processes. These "bugs" are collectively referred to as the student's *mal-knowledge*. The philosophy of the perturbation model, then, is to represent the student as a subset of the expert's knowledge *and* the possible mal-knowledge. By modelling the student's misconceptions, the system will be better able to provide remediation. This scenario is depicted in Figure 3.4.

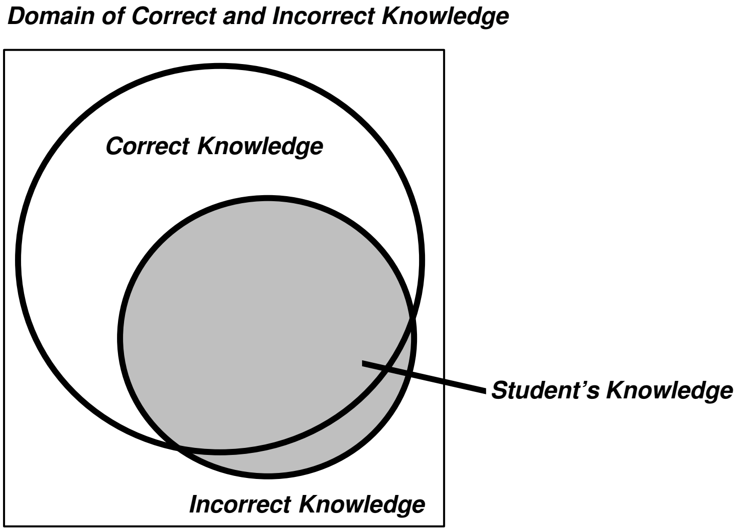


Fig. 3.4. The perturbation model.

Like the overlay model, this approach gives no guidelines on the way in which domain and mal-knowledge should be represented, as long as the knowledge can be decomposed into items. There are three approaches to perturbation modelling: enumerative, generative, and reconstructive. In many tutors, two or more of these types are combined.

Enumerative Modelling

In this scenario, the ITS knowledge engineer uses protocol analysis to determine the possible bugs and misconceptions that students can have. The mal-knowledge then becomes part of the domain model, along with the expert's knowledge.

DEBUGGY (Burton, 1982) is an early ITS with an enumerative perturbation model of the student. It is implemented in the domain of elementary subtraction, and can be used to diagnose student's knowledge and mal-knowledge from test results. Figure 3.5 is an example of an incorrect solution to a typical subtraction problem.

```
  307
 -135
 ----
  232
```

Fig. 3.5. A student's incorrect solution to a subtraction problem.

The buggy answer can be explained by the mal-rule **0-n=n**, which is a modification to the subtraction procedure stating that when the top digit of a column is 0, then the answer for the column is the bottom digit. In this case, the bug results in the digit 3 from 135 being copied directly to the second column of the solution. Another example of a bug is **when borrowing from a column with zero on top, leave that column alone and borrow from the next column instead**. In total, DEBUGGY contains 110 different "primitive" bugs.

However, primitive bugs are not enough in an enumerative modeller. It is possible for a student to combine bugs to form new "bug compositions", which explain more of the student's behaviour than the constituent bugs do separately. The main problem with searching the space of bug compositions, however, is tractability. DEBUGGY attempts to minimise this problem by executing the following heuristic diagnosis algorithm. Given a student's test results, DEBUGGY matches 110 primitives as well as 20 common compositions to each of the student's incorrect answers. If any bug explains at least one incorrect answer, it is added to the hypothesis set. The hypothesis set is then trimmed according to a set of trimming rules. One such rule, for example, is to remove any bug that explains the same behaviours as another bug – exactly which of the bugs is removed depends on the rule. Pairs of bugs in the hypothesis set are then iteratively composed and compared to the student's behaviour. If any composition explains more of the student's behaviour than its constituents, then the composition is also added to the hypothesis set. There is a limit in DEBUGGY of four primitives per composition. The next step is coercion – DEBUGGY attempts to explain any remaining inconsistencies between the student's behaviour and the bugs in the hypothesis set as noise. The output of this process is a classification of bugs as consistent (meaning that they are predictable from the problem state) or inconsistent (implying that they are unpredictable random slips). Ideally, there should be many more consistant than inconsistent bugs.

Enumerative perturbation modelling is still a popular approach to student modelling. Two recent perturbation models are described by Webb et al. (1997) and Virvou & Tsiriga (2000).

Webb et al. (1997) describe the application of Feature-Based Modelling (FBM) to the prediction of student problem-solving actions, again in the subtraction domain. The goal of FBM is to use a machine learning algorithm to associate context and actions. The context is the set of features describing the current state of the problem, e.g. "subtrahend is zero", and the actions range from the general (e.g. "result is correct") to the specific (e.g., "subtrahend is subtracted from minuend in the second column"). Many of the actions are essentially subtraction bugs, but this approach extends perturbation modelling by asserting that cognitive assumptions are not necessary and that machine learning techniques are all that are required to predict errors from problem states. Two machine learning algorithms, FBM-IS and C4.5 (Quinlan, 1993) were tested after being trained on more than 30,000 data items acquired from the results of tests administered to primary school students. A basic strategy of assuming that every problem state is correct yields 90% accuracy, and this strategy was used as a baseline for comparison. FBM-IS and C4.5 increase that accuracy, statistically significantly, to 93% and 92% respectively. The prediction of errors is much more difficult, however. In the most general case, FBM-IS and C4.5 can predict the incorrectness of a state with 68% and 67% accuracy, but the accuracy for specific predictions about *which* error was made drops to 37% and 44%, respectively. This approach is somewhat weak because composite bugs cannot be predicted by the system, which restricts its expressiveness.

A similar approach was taken in the design of EasyMath (Virvou & Tsiriga, 2000), an ITS for algebraic powers (i.e. addition, multiplication, etc. of integer powers). EasyMath was constructed following an empirical study in which 240 students were given a test covering the entire domain, and a library of common mistakes was determined. Unlike DEBUGGY, but similarly to Webb et al.'s (1997) approach, no attempt is made to match composite bugs against student solutions; only single bugs are matched. However, this was a hindrance to diagnosis because the domain is simple and constrained.

Other systems with enumerative perturbation models include LMS (Sleeman & Smith, 1981; Sleeman, 1982) from the domain of algebra, and PROUST (Soloway & Johnson, 1981; Bonar & Soloway, 1985), an ITS for teaching Pascal. PROUST has a substantial bug library, which took a considerable effort to formulate.

A problem with the enumerative approach is three-fold. Firstly, the computational tractability of searching the space of bug compositions is a severe problem. The more recent tutors discussed above (Webb et al., 1997; Virvou & Tsiriga, 2000) circumvent the problem by not trying to match composite bugs at all, but the fact that composite bugs appear in domains as simple as subtraction (Burton, 1982) suggests that they are common to many other domains. In a domain more complex than subtraction, it may be completely intractable to match composite bugs. Secondly, even within a narrow domain, it has been shown that bugs vary greatly from school to school, and even class to class (Holt et al., 1994). This greatly limits the generality of any bug library. Thirdly, and partly as a result of this variability, the bug libraries are enormously expensive to elicit in terms of time. Therefore, the ITS community considered machine learning techniques for building bug libraries.

Generative Modelling

Generative modelling is an approach where the ITS uses a cognitive model to explain the student's erroneous behaviour. No bug library is required because it is assumed that the system will be able to deduce the underlying misconceptions that leading to the bug from the cognitive model.

An early exploration of generative modelling is described by Matz (1982). The cognitive basis of Matz's approach is that problem solving consists of two components: base rules and extrapolation techniques. Base rules encompass the knowledge extracted from examples or read directly from a textbook. Extrapolation techniques are rules for applying base rules to unfamiliar situations. Matz's theory is that errors can be explained as the result of failed extrapolation. Specifically, the student may use a base rule inappropriately in a new situation, or he/she may modify a base rule inappropriately to solve a new problem. If the general forms of the extrapolation errors in a domain can be determined, then the majority of buggy student behaviour can be explained.

Matz analysed errors made by children learning algebra, and found a number of domain-specific extrapolation errors capable of explaining large numbers of errors. For example, one extrapolation error is to decompose the topmost operator of an algebraic expression over its parts. This strategy is sometimes correct, e.g.:

$$A(B+C) = AB + BC \tag{3.1}$$

but when applied inappropriately, it becomes plainly incorrect, e.g.:

$$\sqrt{(A+B)} = \sqrt{A} + \sqrt{B} \tag{3.2}$$

Algebra rules can also be falsely revised to cope with new situations. For example, suppose the student solves the problem

$$(X\text{-}3)(X\text{-}4) = 0 \tag{3.3}$$

to obtain the solution $X = 3$ or $X = 4$. The student may learn from this that the correct rule to solve problems such as

$$(X\text{-}A)(X\text{-}B) = 0 \tag{3.4}$$

is $X = \text{Solve}[X\text{-}A = 0]$ or $X = \text{Solve}[X\text{-}B = 0]$. However, when faced with a new, but slightly similar problem of the form

$$(X\text{-}A)(X\text{-}B) = K \tag{3.5}$$

the student incorrectly revises the solution rule to $X = \text{Solve}[X\text{-}A = K]$ or $X = \text{Solve}[X\text{-}B = K]$. Yet a third example of extrapolation error in the algebra domain arises from conceptual change. Because students are taught arithmetic first, they falsely extrapolate the rules of arithmetic to algebra. For example, in arithmetic, concatenation means addition:

$$1\tfrac{1}{2} = 1 + \tfrac{1}{2} \qquad (3.6)$$

However, in algebra, concatenation sometimes means addition, but more frequently means multiplication, e.g.:

$$1\tfrac{1}{2}\, XY = (1 + \tfrac{1}{2}) * X * Y \qquad (3.7)$$

Thus, conceptual changes can lead to extrapolation errors.

VanLehn's (1982, 1990) REPAIR theory is in a similar vein to this and was constructed after an extensive study of children solving arithmetic problems, where many types of bugs (some often occurring in combination) were found. Like Matz's work, the theory focuses on consistent (i.e. predictable) bugs. However, REPAIR theory also explains the fact that bugs are not always totally predictable and consistent. Specifically, there is a phenomenon called *bug migration* where students shift back and forth between different bugs in the same situation. According to repair theory, this is because when students arrive at an impasse during problem solving (i.e. a point where they do not know the next correct step in the procedure), they can select *any* incorrect step in an attempt to "repair" the procedure so the problem can be solved. Different repairs lead to different bugs. Repair theory was implemented in a system called SIERRA (VanLehn, 1990).

Reconstructive Modelling

Reconstructive modelling is, like generative modelling, an approach to reconstructing erroneous reasoning. However, the assumptions are stricter. It is assumed that the domain is composed of a set of operators. The operators are atomic in the sense that the students, whatever their mastery level, can apply the operator correctly every time to a problem state. A procedure in the domain is composed of a sequence of operators, and mal-knowledge forms when the student learns an incorrect sequence of operators. The aim of the machine learning algorithm is to induce the operator sequence that best fits a student's or group of students' observed errors. The reconstructive model therefore becomes an explanation of the errors.

ACM (Automated Cognitive Modeller) (Langley et al., 1990) is an example of an ITS with a reconstructive student model. The tutor is implemented in the domain of elementary subtraction. The atomic operators are basic transformations of the solution state, such as `Add-Ten` to increment a value by ten, `Decrement` to reduce a value, and various focus-of-attention shifters. To reconstruct a student's buggy solution, the system performs a depth-first search of the space of operator sequences. The size of the search tree is kept in check by a number of heuristics. For example, "psychologically implausible" branches of the search do not have to be explored, nor do branches that are incompatible with the current solution state. Furthermore, as a result of the observation that student's buggy solutions are often shorter than the length of the correct solution, the search depth is limited to the length of the correct solution plus one. The machine learning side of ACM allows the system to characterise the behaviour of the student without needing to perform a complete depth-first search every time an erroneous answer is submitted.

Another reconstructive tutor is INSTRUCT (Mitrovic et al., 1996). One problem with ACM and other, earlier reconstructive tutors is that inferences are made about the student from very small amounts of information. For example, in ACM (Langley et al., 1990), buggy solutions are reconstructed solely from the initial problem state and the buggy student's solution. This leads to expensive search procedures and less reliable results. INSTRUCT resolves the problem by tracking the student's problem solving actions so that additional information (namely, the relative ordering in which the operators were applied) can be used to better diagnose the student's knowledge. Sequences of operators thus observed are called *traces*. The machine learning task in INSTRUCT is to induce the procedures that the student has knowledge of by example. INSTRUCT also takes account of the fact that students, once they learn a particular trace, can "chunk" the operators within the trace into a single macro-operator.

Combinations

An ITS combining both enumerative and generative techniques is SPENGELS (Bos & van de Plassche, 1994). SPENGELS teaches the correct conjugation and spelling of English verb forms. It has an underlying cognitive model of the spelling process as an algorithm, which was derived from pen-and-paper protocol analysis. The expert algorithm consists of two stages. Firstly, a decision tree is used to determine the correct suffix of the verb. Non-leaf nodes in the decision tree represent questions about the grammatical class of the verb (e.g. "Is it present tense?", "Is the form finite?", etc.). The second stage is to take the verb and the suffix and combine them using a set of morphological and spelling rules. The student's ability to classify verb forms correctly using the decision tree, and their mastery of the second stage rules, is represented by a frequency-based overlay. A considerable amount of mal-knowledge is represented and used to diagnose students' errors. The mal-knowledge is enumerative in that it contains explicit buggy rules for, e.g., over-generalisation, morphological errors and spelling errors. However, it is also generative in two ways. Firstly, the system can combine buggy rules to find the sequence that best explains the student's incorrect answer. Secondly, it can traverse the decision tree of the expert spelling algorithm itself in order to attempt to find the point where an incorrect branch of the decision tree was followed. Therefore, bug diagnosis in SPENGELS uses a combination of both explicit mal-knowledge and inference from domain knowledge to explain bugs.

Drawbacks

Unfortunately, the main drawback of both generative and reconstructive modelling is that when existing approaches are generalised to achieve good domain coverage, they tend to generate many more implausible than plausible bugs. This is primarily because of the huge search spaces involved. Perturbation modelling was further weakened by reports such as that of Sleeman et al. (1989) who showed that a simple re-teaching strategy in the domain of algebra was just as effective as bug remediation. The validity of perturbation modelling has therefore become questionable and further research needs to be done (Holt et al., 1994).

3.1.4 Model-Tracing

Model-tracing is an entire framework for building ITSs (Singley, 1995). However, its principles have implications for student modelling. According to Singley (1995), the "*...defining feature of a model-tracing tutor is the individualization of feedback and practice based on an articulate model of proficient performance.*" In other words, the model-tracing tutor must give feedback on the student's problem solving performance, in addition to the more traditional instruction on the knowledge underlying the problem solving. Another principle of model-tracing is making explicit the goal-structure of a problem, so students master the abstract as well as the concrete skills of the domain. These principles effectively require the tutor to model the student's skill in problem solving as well as their conceptualisations of the domain, so specific remediation on performance can be given. This implies that the tutor must know how to solve the problems. More precisely, model tracing entails the representation of not only the declarative, factual knowledge that the student uses (e.g. knowing Pythagoras' theorem), but also the procedural, goal-oriented knowledge (e.g. being able to apply Pythagoras' theorem). This is effectively defines a short-term student model, because beliefs about how the student is solving the current problem is only relevant in the short-term. After the problem is solved or abandoned, this specific information is no longer important and it can be integrated into a longer term, more abstract model (such as an overlay).

The exemplary model tracing tutors are the family of tutors called *cognitive tutors* that are based on the ACT-R theory developed by Anderson and co-workers (Anderson et al., 1996; Anderson & Lebiere, 1998; Anderson, 1993). ACT-R is a theory of cognition that, amongst other things, assumes that all knowledge is either declarative or procedural. Procedural knowledge is represented as a production set. The cognitive tutors represent an expert's knowledge as a large production set called the ideal student model. There are eight principles that the cognitive tutors adhere to, each of which is derived from ACT-R (Anderson, 1996). The principle assumption of cognitive tutoring is that the representation of the domain knowledge as a production set essentially *is* the expert's representation. Because of this assumption, tutoring can be reduced to the process of transferring production rules from system to student, and students are tutored specifically on productions because that is how the knowledge is represented cognitively.

Cognitive tutors have been developed for LISP (Farrell et al., 1984), geometry (Aleven et al., 1998; Aleven & Koedinger, 2000), introductory statistics (Gluck et al., 1998), and (with a very large scale evaluation) algebra (Koedinger et al., 1997) amongst others. Some of these tutors have been coupled with a long-term student modelling technique based on Bayesian probability theory called knowledge tracing, which is discussed in Section 3.2.

Another example of a model-tracing tutor is ANDES (Gertner & VanLehn, 2000). ANDES is an ITS implemented in the domain of physics. It uses much more sophisticated Bayesian techniques than knowledge tracing for reasoning about the student's mastery and behaviour, and the Bayesian aspects of it are discussed in more detail in Section 3.2.1. Like the ACT-R tutors, ANDES solves physics problems using a production set. It also attempts to capture both the declarative and procedural knowledge used by the student during problem-solving in order to diagnose the student's problem solving strategy.

Unfortunately, the price of such complex cognitive modelling in model tracing systems is a substantial decrease in their tractability. The problem arises because of the massive amounts of uncertainty inherent in reasoning about a student, especially from the "keyhole" of the computer's input devices. In general, the number of influences on the student that the computer does not know about (e.g. the textbook the student is reading while using the tutor; the knowledgeable friend sitting at the terminal next to the student; etc.) may be so great that sophisticated reasoning about the student becomes, as Self (1988) argued, intractable. It is a major combinatorial problem to match an incomplete sequence of student actions against a set of possible problem-solving strategies, in order to determine the most likely strategy that the student is following.

To tackle this problem, cognitive tutors can attempt to capture as many as possible of the problem-solving steps that the student takes. For example, ANDES' user interface tries to be "*…as much like a piece of paper as possible…*" (Gertner & VanLehn, 2000) in that the student can perform every task from drawing diagrams, to defining variables, to entering equations within it. While this "capture every step" approach may work in highly procedural, well-defined domains such as mathematics and physics, it may not be a realistic solution in other domains where the ordering of problem-solving steps are either ill-defined (e.g. legal reasoning) or irrelevant (e.g. punctuation). It is thus not clear how the cognitive tutoring approach will generalise.

Another tractability issue is the elicitation of knowledge for cognitive tutors. It has been estimated that the ACT-R tutors required, on average, ten hours of elicitation time per production rule (Anderson et al., 1996). ANDES also has a production set of considerable size and complexity, as well as knowledge contained outside the production set, such as lists of possible equations that the student might use. Clearly, there are domains larger and more complex than physics, and this approach may be uneconomical and infeasible in such domains.

3.1.5 Constraint-Based Modelling

There is empirical evidence that one of the cornerstones of model tracing, that a student follows a single path or strategy to solve a problem (which the system therefore tries to infer), is false. A number of studies surveyed by Ohlsson (1994) suggested that students, in fact, rapidly switch between several different strategies when problem solving, even after instruction. Some of the strategies are correct while others are incorrect. Ohlsson (1994) defines this as the *radical strategy variability* phenomenon, and if it turns out to be the normal case, then it invalidates approaches that assume the student follows only a single path to a problem's solution.

Another curious phenomenon about students is their ability to "catch" and correct errors before or after making them (Mitrovic & Ohlsson, 1999). This phenomenon appears to arise because students acquire the ability to discriminate correct and incorrect solutions *before* they acquire the actual skills to solve the problem correctly. Olhsson calls this initially-acquired type of knowledge *evaluative*, and the latter *generative*. He suggests that once a student acquires the evaluative knowledge, they have the cognitive skills to learn (perhaps by trial-and-error in the least efficient case) the generative knowledge. Furthermore, Ohlsson claims that diagnostic information about a student is most readily available in the problem states that the student arrives at. While the diagnostic information can be selected from the problem solving steps (as in model tracing), radical strategy variability and tractability problems make this a much more difficult task. Therefore, an ITS may only need to model evaluative knowledge. The student can effectively be left to his or her own devices to induce the generative domain knowledge.

Thus, Constraint-Based Modelling (CBM) is an attempt to define a domain-independent representation for evaluative knowledge. In its most general form, a constraint is a pair of patterns <Cr,Cs> where Cr is known as the *relevance condition* and Cs is called the *satisfaction condition*. The patterns match states of the student's solutions. To illustrate, suppose the Cr of a hypothetical constraint is defined to match any string of the form `n+n=*`, where `n` is a non-negative integer and `*` stands for any other string. Then the Cr would match strings like "4+7=11", "1+1=3" and "102+232=ABCDEFG", but not "123-56" and "ABCDEFG". The Cr is said to define the class of the student solution. Now, the Cs is also a pattern, so suppose it is defined in our example as `n1+n2=`*`sum`*`(n1,n2)`. This is a more specific pattern than the Cr. If *`sum`* is a basic function that takes two numbers and returns the sum of its inputs, then this string will match only valid mathematical statements such as "4+7=11", but not mathematically incorrect statements such as "1+1=3", "ABCDEFG", etc. Thus, the Cs defines the consistency (correctness) of the solution. If the Cr matches the solution and the Cs does not, the constraint is *violated* and constraint-specific tutoring can begin. In an implementation, the code fragment in Listing 3.1 would represent the matching process.

```
If Matches(Student-Solution, Cr) Then
 If Not Matches(Student-Solution, Cs) Then
 Constraint-Is-Violated;
Else
 Constraint-Is-Satisfied;
```

List. 3.1. Code for determining constraint violation or satisfaction.

Note that it makes no sense to match the Cs without first matching the Cr: the system *must* know that a constraint is relevant initially, as the satisfaction condition can be very general and match many solutions. Consider, for example, a constraint from the punctuation domain where Cr = "The word is a singular possessive" and the Cs = "The word ends with an apostrophe followed by the letter *s*". Obviously, this Cs could fail to match many possible words, but it only makes sense to match the Cs after determining firstly that the word is a singular possessive, by matching the Cr.

The main advantage of CBM is that it is highly tractable. Little computational effort is required for constraint matching. Furthermore, an algorithm exists for "merging" constraints into a unified structure called a RETE network that can increase the efficiency of constraint matching even more (Mitrovic, 1998; Forgy, 1982). To illustrate its effectiveness, SQL-Tutor is a CBM tutor (Mitrovic & Ohlsson, 1999) containing over 500 constraints, and runs efficiently on PCs and the Web.

Like cognitive tutors, an overlay model can be projected onto the constraints to measure the student's mastery of each constraint. SQL-Tutor initially had a simple overlay on constraints determined by frequencies (Mitrovic & Ohlsson, 1999) but later supplanted this with an overlay based on probabilities (Mayo & Mitrovic, 2000). The overlay model is not even necessary for a CBM tutor. The initial version of CAPIT had no long-term model such as an overlay, and the analysis of evaluation study results show that students still learned constraints (Mayo et al., 2000).

It is also worthwhile mentioning the elicitation time for constraints as opposed to production rules, which was slightly more than one hour (Mitrovic & Ohlsson, 1999). This represents a substantial improvement.

Finally, constraints offer a solution to another of the ITS knowledge acquisition bottlenecks: problems with multiple solutions. Traditionally, the ITS designer would have to enumerate each correct solution to a problem. In a CBM tutor, however, the ambiguity is naturally relegated to individual ambiguous constraints. These constraints, when they are relevant to a solution, can be used to generate all the possible correct solutions. For example, in a punctuation tutor, an ambiguous constraint may state that direct speech can be enclosed in double speech marks (""), which is standard to New Zealand English, *or* single quotes (''), as it appears in many American publications. This is an advantageous property of CBM that may make building tutors in less well-defined domains easier.

3.1.6 Summary

The content and persistence of traditional student models have been examined. Approaches to student modelling were divided into two classes: *short-term*, which models information relevant immediately to the current situation, such as the constraints that were violated on the last problem attempt or the student's current goal structure, and *long-term*, which is a more comprehensive and abstract representation of the student's mastery and other traits are built up over time. Examples of long-term modelling are stereotyping, overlays, and perturbation models. Examples of short-term representations are the student modelling aspect of model-tracing and CBM.

Perhaps the most pertinent issue in student modelling is the quality of the representation versus the tractability of the inferences versus the inherent uncertainty. In other words, what is the most appropriate level of detail that a student model should have before the inference becomes computationally too complex, and the uncertainty becomes too great? Clearly, model-tracing tutors that try to determine the exact solution path a student is following, and perturbation tutors that try to match composite bugs, are reaching the limits of tractability and certainty, and in more complex domains they may become infeasible. On the other hand, it is largely agreed that approaches such as fixed stereotyping are too limiting for ITSs. Approaches such as CBM represent a compromise. However, no-one to date appears to have conducted a comparative study of the effects of different student modelling techniques on the teaching effectiveness of ITSs. This is therefore a promising area for future research.

3.2 Bayesian Student Modelling

Recently, student modelling with Bayesian networks has attracted the interest of ITS researchers. Unlike the previous section where general approaches to student modelling were described, this section focuses specifically on issues related to Bayesian student modelling and how they are solved. Specific Bayesian network-related issues include how conditional probabilities can be elicited, and how the structure of the network can be defined (and if it should be defined *a priori* or not).

Three classifications of Bayesian network student model are considered: *expert centric*, *efficiency-centric*, and *data-centric*. Expert-centric student models are unrestricted products of domain analysis. That is, an expert specifies either directly or indirectly the complete structure and conditional probabilities of the Bayesian student model, in a manner similar to that with which expert systems are produced. This is the general approach of ANDES (Gertner & VanLehn 2000; Gertner et al., 1998; Gertner, 1998; Conati et al., 1997), HYDRIVE (Miselvy & Gitomer, 1996), DT-Tutor (Murray & VanLehn, 2000), and the Bayesian domain model of ADELE (Ganeshan et al., 2000). One possible disadvantage of this approach is that the resulting models may include so many variables that it becomes infeasible to evaluate the network efficiently on-line. For example, tractability was considered an important issue in the initial evaluation of DT-Tutor. Efficiency-centric models, on the other hand, work the other way: the model is partially specified or restricted in some way, and domain knowledge is "fitted" to the model. The restrictions are generally chosen to maximise some aspect of efficiency, such as the amount of numeric specification required and/or the evaluation time. This is the methodology of Reye (1998), Murray (1998), Collins et al. (1996), Mayo & Mitrovic (2000), and to a degree, Millán et al. (2000). In general, restrictions to increase efficiency can introduce incorrect simplifying assumptions about the domain. Finally, the data-centric model is a new class of Bayesian student model, introduced and represented by CAPIT in this thesis, in which the structure and conditional probabilities of the network are learned primarily from data. CAPIT's student model dispenses with attempting to model unobserved student states such as domain mastery and instead concentrates on modelling the relationships between observed variables to predict student performance. MANIC (Stern et al., 1999) is the closest existing Bayesian system the authors could find to the data-centric approach, but it learns only the probabilities and not the structure of the network, and is therefore closer to the efficiency-centric specification than the data-centric one. Figure 3.6 shows how existing Bayesian network student models fit this classification.

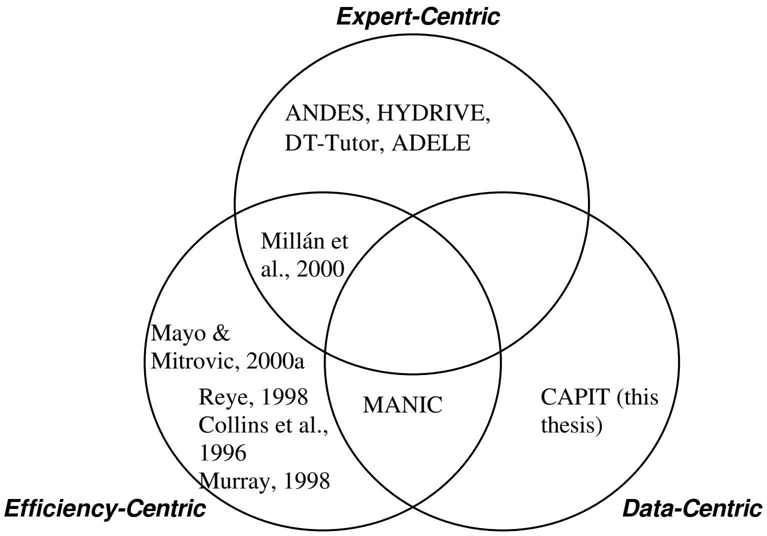


Fig. 3.6. A classification of Bayesian network student models.

3.2.1 Expert-Centric Models

ANDES (Gertner & VanLehn 2000; Gertner et al., 1998; Gertner, 1998; Conati et al., 1997), HYDRIVE (Miselvy & Gitomer, 1996), DT-Tutor (Murray & VanLehn, 2000) and ADELE (Ganeshan et al., 2000), are examples of tutors with large Bayesian networks with structures mostly engineered from complex domain analysis. To match the domains as closely as possible, their networks are not structurally restricted in any way. However, both networks do have a high proportion of variables representing unobserved, internal student states. A major hurdle for these systems, then, is how conditional probabilities can be elicited or defined for these variables in the absence of data.

ANDES' solution is to use "coarse-grained" conditional probabilities definitions such as noisy/leaky-ORs and noisy/leaky-ANDs. A noisy/leaky-OR variable is a binary variable with a high probability of being true only if at least one of its parents is true, and is very probably false otherwise; a noisy-AND variable has a high probability of being true if all of its parents are true, and is very likely false otherwise. In practice, restricting conditional probabilities to noisy/leaky-ORs and noisy/leaky-ANDs significantly reduces the number of required probabilities and makes the modelling of unobserved variables much simpler because only the structure and node type (noisy/leaky-AND or noisy/leaky-OR) needs to be specified. However, it does reduce the specificity of the system.

The Assessor Bayesian network is ANDES' main student modelling component (Conati et al., 1997). It has two parts: a static component that persists from problem to problem, and a dynamic component that is constructed for every new problem and discarded after the problem is either solved or abandoned. The static component models the student's abilities to apply rules in specific contexts as well as generally; the dynamic component models the facts, goal, strategies, and rule-applications relevant to modelling the student's problem-solving steps on the current problem. The dynamic part of the Assessor network is constructed on-line from the problem's solution graph, which is a representation of all possible solution paths for a particular problem. The solution graphs in turn are constructed off-line from the physics problem solver.

HYDRIVE is an ITS for teaching troubleshooting skills to aircraft hydraulics system technicians. The student model is a Bayesian network that reasons from observations of specific troubleshooting actions, to characterisations of more general constructs such as systems, strategies and procedures. Unlike cognitive tutors which attempt to wholly model the expert, the goals behind the design of HYDRIVE's student model was only to capture the factors important to discriminating between proficient and less proficient students. To determine these factors, a cognitive knowledge elicitation protocol called PARI was utilised. The PARI analysis determined the structure and semantics of the Bayesian network.

Unlike ANDES and a number of other schemes that define mastery variables as having the basic values `true` and `false` only, the mastery variables in HYDRIVE take linguistic "fuzzy" values. For example, a student's *Strategic Knowledge* can be `expert`, `good`, `okay` and `weak`. The conditional probabilities relating these variables were determined by qualitative estimates from expert instructors, patterns observed in PARI traces, and other "reasonable" modifications arising from simulation studies.

The designers of HYDRIVE suggest testing their student model with actual data to assess its predictive performance. However, to date no such assessment appears to have been reported in the literature. An assessment would be somewhat difficult because of the multiple values (expert, good, etc.) used to describe the student's mastery of each item. Each student used to test HYDRIVE would have to be assigned a set of such values beforehand, but this assignment would be highly subjective due to the granularity and fuzziness of the terms. If HYDRIVE had used simple binary variables to represent their mastery, an evaluation might have been easier to perform because test students would only need to be assigned (more realistically) values of true and false indicating their mastery, rather than expert, good, okay or weak.

On a more general front, this issue illustrates one difficulty of expert-centric and cognitive modelling approaches: there are no clear, specific and detailed guidelines for cognitive modelling. How should the model be defined? How should details such as the specific representation of the student's mastery be resolved? Model tracing is an extreme example of the stance that a student model should be as isomorphic as possible to the expert's cognitive representation of the domain. The opposite stance taken in this thesis is that the function of the student model is to predict student behaviour, and that therefore the student model design should not be driven primarily by extreme cognitive fidelity, but instead by actual student performance data. Of course, some cognitive modelling will be required, but only to the extent that it helps predict student performance. The philosophy of HYDRIVE, where the student model was designed only to capture those elements discriminating between expert and non-expert behaviour, is one example of this approach.

DT-Tutor is a generalised domain-independent architecture for student modelling and pedagogical action selection (PAS). Like ANDES and HYDRIVE, it models the student's knowledge, but it goes much further and attempts to model other hidden states such as the student's morale, independence, and focus of attention. A preliminary version of this system has been constructed but no details have been given as yet as to how the conditional probabilities are obtained, although it appears to follow ANDES' lead by simplifying the probabilities to noisy/leaky-ORs and -ANDs in the domain knowledge representation network.

Models that largely represent unobserved, internal student states suffer a major disadvantage: the model structure and/or parameters cannot be adapted on-line to the student. To illustrate, consider a very simple Bayesian student model with two variables, *Observations* (observed) and *Student State* (unobserved), and a conditional probability *P(Student State|Observations)* relating them. When *Observations* becomes known, the posterior probability of *Student State* is updated. However, the conditional probability *P(Student State|Observations)* itself can never be updated. As a result, two students with the same *Observations* will be considered equivalent by the system even though the observations may only encompass the most recent interactions. This is essentially the case in ANDES and HYDRIVE. As an alternative, another system might comprise two observable variables, *Before* and *After*, related by a conditional probability *P(After|Before).* Now, given observations of *Before* and *After*, *P(After|Before)* can be updated. Such a system therefore adapts to the student, and its conditional probabilities will be a summary of all, not just the most recent, previous interactions. This is the approach implemented in CAPIT in Chapter 6.

3.2.2 Efficiency-Centric Models

The student modelling approach in the ACT-R tutors is called *knowledge tracing* (Corbett & Anderson, 1995; Corbett & Anderson, 1992). Knowledge tracing is a simple Bayesian overlay on production rules. It is assumed that each production rule can be in one of two states, `learned` or `unlearned`, with a certain probability. The mastery state of a production rule can only shift from `unlearned` to `learned`, and not the other way around (so "forgetting" is assumed impossible). There is a probability of a transition after each attempt at, or instruction on, a production rule. The current probability of mastery is simply the sum of the probability of mastery prior to an action and the conditional probability of learning the rule given the action. A rule is assumed to be mastered when the probability of it being `learned` exceeds 0.95. This approach has been shown to reasonably accurately predict student post-test performance (Corbett & Anderson, 1995). More recently, a third state has been added to the mastery representation of each rule (Corbett & Bhatnagar, 1997).

This new state represents a rule that can be successfully applied within the tutoring system, but not during a test outside of the ITS environment. In the words of the authors, it is "*…indistinguishable from an ideal [i.e. a learned] rule in modelling the student's tutor performance and … indistinguishable from no [i.e. an unlearned] rule in predicting test performance.*" The addition of this state yields increases in post-test predictive accuracy.

A similar approach using dynamic Bayesian networks (Russell & Norvig, 1995, Ch. 17) has been proposed by Reye (1998). In fact, Reye's model generalises knowledge tracing, and a similar approach was used in the student model of SQL-Tutor (Mayo & Mitrovic, 2000; Chapter 4 of this thesis). The idea is to model the student's mastery of a knowledge item over time. The tutor's current belief that the student has mastered the item (M_t) depends on its previous belief (M_{t-1}), the outcome of the student's last attempt at the item (O_{t-1}), and the pedagogical response of the tutor to the last attempt (A_{t-1}). Using dynamic Bayesian networks, not only can the tutor's current beliefs be determined, but also its future beliefs at time *t+1* or beyond, although this is likely to be much more uncertain. This model is depicted in Figure 3.7 for a single knowledge item. In Figure 3.7, the student failed an attempt at the item at time *t-1* and so the tutor provided remedial help.

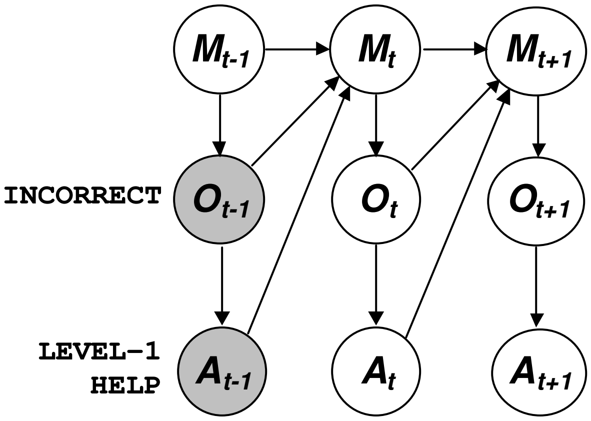


Fig. 3.7. A DBN modelling the mastery of the student on a single knowledge item equivalent to Reye's approach.

One problem with this approach is that the complexity-reducing assumption that mastery of a knowledge item is probabilistically independent of the mastery of any other items is unrealistic. Suppose, for example, that the knowledge items are “high level”, such as concepts or topics. Then we would expect the mastery of some items to be dependent on the mastery of the items that are pre- and co-requisites. This is a basic assumption of many systems and the rationale behind many approaches to course sequencing, e.g. Brusilovsky (2000). Alternatively, the knowledge items could be “low level”, such as constraints (Ohlsson, 1994; Mitrovic & Ohlsson, 1999; Mayo et al., 2000). Clearly, we would expect many dependencies between constraint mastery based on factors such as syntactic and/or semantic relatedness. We demonstrate later that in the punctuation domain, a model with dependencies between items makes better predictions of student performance than a simpler model similar to Figure 3.8.

There are Bayesian student models that allow some dependencies to be expressed whilst remaining efficiency-centric. They are the singly-connected hierarchical structures described by Murray (1998), Collins (1996), and Stern et al. (1999). A singly-connected network has the property that for every pair of nodes in the network, there is one and only one path between the nodes. Bayesian networks with a singly-connected topology have the advantage of evaluating in linear time (Pearl, 1988; Murray, 1999), and while they can express dependence between knowledge items, the singly-connected assumption means that certain types of dependence (namely, undirected loops) cannot be represented. This is clearly a strong restriction, because all the expert-centric models described above contain undirected loops in their Bayesian networks.

The problems of single-connectedness are illustrated by MANIC (Stern et al., 1999), which attempts to learn the probabilities (but not the structure) of its network from observations of the student. MANIC’s hierarchical structure leads to the simplifying assumption that its variables are conditionally independent given their single mutual parent. Unfortunately, the data acquired from students directly contradicted this, and so Stern et al. were forced to compensate by introducing several ad-hoc “fixes” to the network, such as “merging” dependent nodes and deleting irrelevant nodes. The introduction of ad-hoc fixes like this essentially jeopardised the system’s normative status. A clear solution to this problem would have been to drop the restriction that the network was a hierarchy, although this would have led to a more complex model and the necessity for more complex learning and inference algorithms.

Interestingly, Millán et al. (2000) recently proposed an architecture that is to a degree both expert- and efficiency-centric. Their Bayesian network is selected to optimise the amount of numeric specification required, and to achieve this, the directionality of the arcs between groups of variables is fixed. The variables are also limited to binary states with specific semantics. However, the topology of the network does not have to be singly-connected which makes it quite flexible.

3.2.3 Data-Centric

Both the structure and conditional probabilities of the network are learned from data collected from real-world evaluations of the tutor in the data-centric approach. There are a number of benefits of this approach. Firstly, because the model is induced from actual data, its predictive performance can easily be evaluated by testing the network on data that was not used to train it. Secondly, data-centric models can be expected to be much smaller than the typical expert-centric model because the latter represents both observed and hidden variables, while the former models only observable variables. We describe a general methodology for building data-centric models in Chapter 5, and introduce a specific tutor with a data-centric model (CAPIT) in Chapter 6.

3.3 Applying The Bayesian Student Model

Student modelling is a futile activity if it is without application. Perhaps the most straightforward and obvious objective of student modelling is *assessment*: measuring the student's overall competency within a domain. However, a much more important application of student modelling is adapting the behaviour of the ITS to optimise the learning of the domain by the student. Tasks that can be adapted include curriculum sequencing, feedback and hint generation, and instructional grain size adjustment. While both domain-specific and domain–independent pedagogic knowledge are important for solving these problems, the most critical ingredient is obviously the student model. The class of decision problem where the student model is critical will be referred to as Pedagogical

Action Selection (PAS). In this section, we focus on applications implemented in ITSs whose student model is a Bayesian network.

3.3.1 Assessment

A system designed specifically for assessment is OLAE (VanLehn & Martin, 1997), which integrates with the ANDES physics tutor (Gertner & VanLehn, 2000). Because ANDES' student model contains approximately 290 probabilities of mastery, it can be very difficult for a human tutor to assess the student's overall competence. OLAE is a graphical tool that simplifies this task. The human tutor can select groups of rules (e.g. all the rules comprised within a single chapter of a textbook) and OLAE automatically computes the probability that the student has mastered the group (i.e. the probability that the student has mastered the chapter). This overall probability is defined straightforwardly as the product of the individual probabilities. This is, in fact, a strange way of assessing the student because it computes only the probability that the student has mastered all (say, *n*) of the rules. A student that mastered *n-1* rules with a high probability but has a very low probability of mastery of the *n*th constraint will consequently have a very low overall probability of mastery. A better approach would have been to assess the mean probability of mastery, and then calculate the standard deviation to determine the proportion of rules likely to have a low probability of mastery.

Millán et al.'s (2000) diagnostic Bayesian network described in Section 3.3.2 could also fit into this category.

There is little in the way of other assessment tasks in the literature. Corbett et al. (1998) use their Bayesian student model to predict student performance on a post-test, which therefore validates their student model, but they do not report on any other more sophisticated assessment techniques.

3.3.2 Pedagogical Action Selection

Unfortunately, only a handful of papers describe how Bayesian student models have been applied to a task other than assessment. Of those that do, there seem to be three general approaches: *alternate* strategies, *diagnostic* strategies, and *decision-theoretic pedagogical* strategies. The three classes and the systems that fall into them are given in Table 3.1.

Alternate	**Diagnostic**	**Decision-Theoretic**
LISP Tutor ANDES ADELE SQL-TUTOR Milne et al., 1996	Millán et al., 2000 Collins et al., 1996	DT-TUTOR CAPIT

Table 3.1. Decision-making with the student model.

Alternate strategies

Alternate strategies optionally take the posterior probabilities of the Bayesian network and use them as the input to some heuristic decision rule. To illustrate, the LISP Tutor uses a simple heuristic to decide whether or not to advance the student to the next section: if the probability of the student's mastery exceeds some threshold, the student advances (Anderson et al., 1996).

ANDES selects hints for the student based on the solution path that the student is following to solve the current problem (Gertner et al., 1998). However, the student's solution path is by no means certain (e.g. the student could be on paths *A*, *B* or *C* with posterior probabilities *P(A)*, *P(B)*, and *P(C)*), and therefore the system uses the heuristic of assuming that the most probable solution path (e.g. *A*, assuming *P(A)>P(B)* and *P(A)>P(C)*) *is* the student's solution path. However, this is a sub-optimal heuristic as demonstrated by a simple counter-example. Suppose the optimal hint for solution path *A* as defined by the learning theory is H_1, but the optimal hint for both paths *B* and *C* is H_2. Then if it is the case that $P(B) + P(C) > P(A)$, hint H_2 will be optimal, but the rule will incorrectly select hint H_1.

ANDES also has heuristic decision procedures disconnected entirely from the student model. For example, a simple matching heuristic is used to generate feedback on incorrect equation entries (Gertner, 1998).

ATULA (Milne et al., 1996) also uses the same "most probable explanation" strategy as ANDES for stereotyping students. While it does not have a Bayesian student model, it does calculate a probability that the student is in each of the six possible different stereotypes. It assigns the student to the most probable stereotype only, and this stereotype is used to guide the presentation of the subject matter. Again, like ANDES' hint selection, it is possible to demonstrate sub-optimal behaviour arising from this strategy.

Another system using heuristic decision procedures is ADELE (Ganeshan et al., 2000). ADELE has a Bayesian network model of the domain knowledge, and it uses a heuristic based on focus-of-attention to select the node in the network about which to provide a hint. Decision-theoretic processes were considered but abandoned because they were considered too inefficient.

SQL-Tutor uses a heuristic for problem selection (Mayo & Mitrovic, 2000). The main rationale for this was that, like ADELE, the computation required for exact decision-theoretic computation (which would have involved more than 500 constraints) made direct application of decision theory intractable. The heuristic used was based on Vigotsky's Zone of Proximal Development (Vigotsky, 1978), and did tend to select problems of an appropriate complexity level efficiently. However, this approach is not guaranteed to select the most appropriate problem.

Finally, it is worth summarising the reasons why ad-hoc decision procedures seem so prevalent (this includes systems with ad-hoc student models as well as systems with ad-hoc or alternative PAS). Firstly, there is the obvious convenience, simplicity and efficiency of ad-hoc strategies. They are straightforward to implement and understand, and they can be invoked on-line with little or no noticeable delays during execution. Indeed, in some domains (such as SQL), ad-hoc strategies may be the only efficient strategies in the absence of highly specialised algorithms. Secondly, there is the perception that precision in intelligent tutoring is not required, which therefore makes ad-hoc strategies all the more appealing. In the words of Katz et al. (1992), "*…in low risk decision-making situations such as tutoring…imprecise student modelling is adequate…*" This attitude implies that student modelling is not an endeavour worthy of such sophisticated approaches to uncertain reasoning. In a sense, this is true because any inferences a system can make about a student are likely be highly uncertain and therefore one could argue that rigorous theoretical reasoning may be no improvement on ad-hoc reasoning. On the other hand, the very fact that student modelling is a highly uncertain activity implies that we should be using the most rigorous and theoretically sound tools because otherwise we will be introducing even more uncertainty into the student model. That is, under equivalent conditions, ad-hoc methods will at best perform as well as normative methods; but normative methods may out-perform ad-hoc methods.

Diagnostic Strategies

Millán et al. (2000) describe an approach for optimising test question selection. The approach expands on a strategy suggested by Collins et al. (1996). The Bayesian network has nodes representing questions, concepts, topics, and one node (say, A) for the overall proficiency of the student. Answers to questions are observed, and the probability distributions over the other (hidden) nodes are inferred via Bayesian updating. Interestingly, this approach uses theory rather than direct expert opinion to initialise the conditional probabilities. The theory is Item Response Theory, which states that there is a logistic relationship between the student's mastery of a concept and the probability of a correct response. The logistic function is formed from the estimates of the probabilities of a slip (when an expert student slips and making an error) and a guess (when a novice student guesses and gets the item correct) for each item.

Because the purpose of the network is adaptive testing, Millán et al. describe a procedure for question selection. The question selection criterion is informativeness. That is, questions are selected to maximise the precision of the variable A representing that student's overall proficiency. When $P(A=\texttt{Proficient})$ or $P(A=\texttt{Not-Proficient})$ exceeds some maximum threshold t, then the system stops selecting questions because the probability of mastery or lack of mastery is sufficiently precise. The claim is that this is an optimal method of question selection for diagnosing the student's state. It is also a highly tractable procedure.

Is the approach decision-theoretic? In one sense it is, if the utility function is thought of as a function of the probabilities of the network rather than a disconnected function representing subjective preference. That is, because the adaptive testing procedure aims to maximise informativeness, then the utility to be maximised is essentially the precision of the variable A. On the other hand, the use of a threshold t to determine when question selection should terminate is not decision-theoretic, because the halting criterion should (technically) be reached when the expected utility of every unasked question drops to zero or below, and this is not the case.

Therefore, there is a case for Millán et al.'s approach being consistent, with some modification, to decision theory. However, because utility is defined as informativeness, this approach cannot be used for PAS. It is useful only for the diagnostic testing of the student's state of knowledge, and furthermore, it makes the implicit assumption that the student's knowledge does not change for the duration of the test so remedial actions such as instruction cannot be performed until afterwards. Therefore, outside of testing, the approach has limited applicability.

Decision-Theoretic Pedagogical Strategies

Decision-theoretic strategies are utilised in both DT-Tutor (Murray & VanLehn, 2000) and CAPIT (Mayo & Mitrovic, 2001; Chapter 6 of this thesis). The key difference between this system and that of Millán et al.'s (2000) is that the utility function essentially encodes pedagogical knowledge (i.e. a learning theory) related to the decision being made by assigning utility to outcomes according to their pedagogical, rather than diagnostic, worth. Therefore, while diagnosis is obviously an important component of expected utility maximisation, it is only a secondary component. For example, in CAPIT, as shall be described, the expected utility of an action (e.g. problem selection) depends on the likely outcomes of the action (e.g. how many errors are made). In DT-Tutor, the action's impact on many different factors related to the student (e.g. their morale, etc.) has an influence on expected utility. Diagnosis, therefore, is only required to the extent that it discriminates between alternate actions. The key difference between the DT-Tutor and CAPIT, as Figure 3.1 depicts, is that DT-Tutor has a static, expert-centric student model whereas CAPIT has a data-centric student model that can adapt on-line. This impacts on action selection because the crucial $P(X|D)$ component of the expected utility function (Equation 2.28) is evaluated from the Bayesian model.

3.4 Summary

To summarise, student modelling has been introduced and overviewed. The advantages of CBM, namely tractability and cognitive plausibility, have been described. IOAM calibrated to domain knowledge was then proposed, which is the class of tutor to which CAPIT belongs. Bayesian student modelling was focused on and discussed in detail, as was the application of Bayesian models to tasks such as assessment and curriculum sequencing.

This overview has focused on systems that exhibit normative approaches to both student modelling and PAS. It has been argued by Self (1994), Katz et al. (1992) and others that such specificity is not necessary because complete accuracy is not required for a student model to be useful. This claim, while it may be partly true, misses the point of normative modelling. The main advantage of the normative approach is that it isolates the uncertainty. Bayesian reasoning and decision-theoretic PAS are guaranteed rational. Therefore, if something goes wrong and the system appears to be behaving irrationally, then these procedures cannot be to blame. Rather, one should look at the probabilities and utilities specifying the model in the first place. Heuristic approaches fail to isolate the causes of erroneous behaviour in this way.

Chapter 4

Case Study: A Probabilistic Student Model for SQL-Tutor and Its Application To Problem Selection

This chapter describes an initial attempt at building an intelligent pedagogical decision strategy in an existing ITS. This implementation and the subsequent evaluation led to the development of the general methodology, and the punctuation tutor CAPIT.

SQL-Tutor is an intelligent tutor for the SQL database language (Mitrovic, 1998; Mitrovic & Ohlsson, 1999; Mayo & Mitrovic, 2001). Knowledge is represented in the form of constraints (Olhsson, 1994). A simple frequency overlay forms the long-term student model, and a basic heuristic is used for next problem selection. The system was extended in two ways. Firstly, the frequency overlay was changed to a probabilistic overlay to support more robust reasoning about the student. Secondly, a more intelligent decision procedure for problem selection was implemented. Unfortunately, the size of the SQL domain meant that some compromises had to be made to achieve tractability. Specifically, and unlike CAPIT, the problem selection strategy is not entirely normative. However, the results were positive and they encouraged us to continue in this line of research.

The initial version of SQL-Tutor is introduced in Section 4.1. The probabilistic student model and the new problem selection method are then described in Section 4.2. An evaluation of the new problem selection method was held in October 1999 and the results are given in Section 4.3. Finally, a number of lessons were learned from this implementation that guided our future research. These are discussed in Section 4.4.

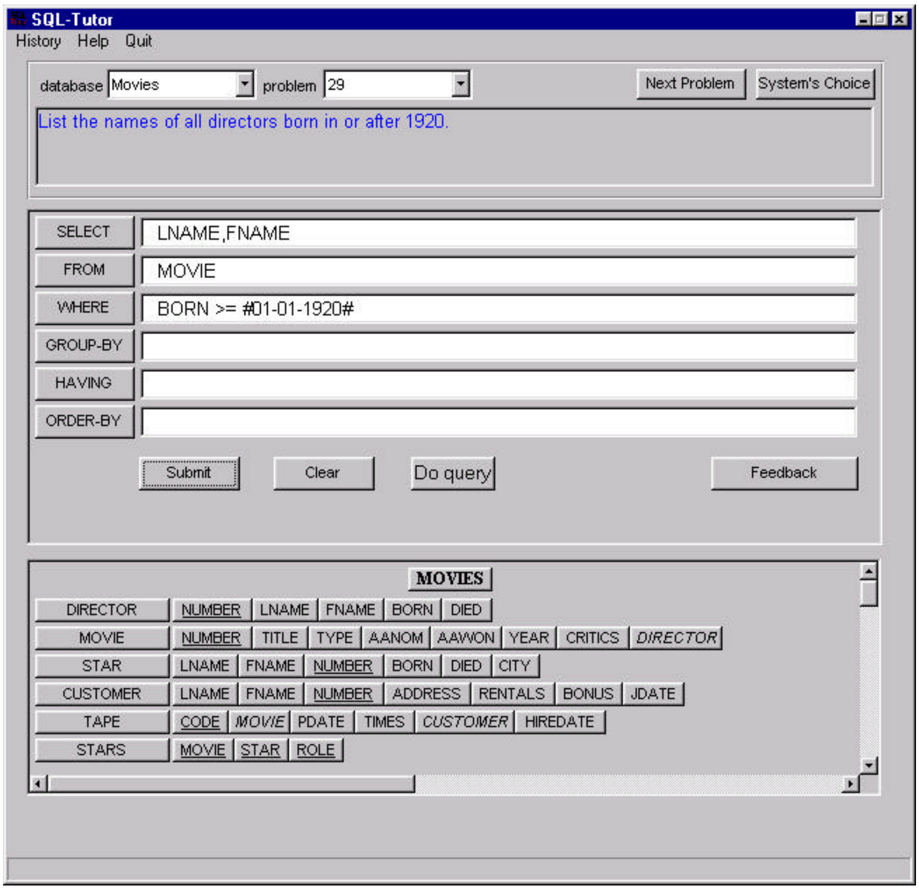
4.1 SQL-Tutor

SQL-Tutor is a practice environment for undergraduate university students enrolled in database courses. There are three functionally identical versions for Solaris, MS Windows and the Web.

Problems are presented in the form of English statements that the student must convert into equivalent SQL. The student learns the concepts and fundamentals of SQL because the system gives feedback on violated constraints in the student's solutions. SQL-Tutor only has one correct solution (the ideal solution) for each problem, even though multiple correct solutions may exist. The ideal solution is used to determine the correctness of the student's solution.

Figure 4.1 depicts the main interface to SQL-Tutor. The problem statement is shown at the top of the window. At the bottom of the window, the database schema is depicted. By showing the database schema, the student's loading on his or her short-term memory is reduced, allowing the student to concentrate on the specifics of the query. The student enters the SQL query for the current problem (the student's solution) into the middle portion of the window.

The interface has a separate text field for each clause in the student's solution. By breaking up the SQL statement in this way, the complexity of the solution is further reduced. When the student feels that the solution is correct, or if the student needs help, he/she can click *Submit* and the system will match the constraints of the domain against the current state of the solution and give feedback. The level of feedback is adjustable by the student and by the system, and ranges from very brief (correct/incorrect), to feedback on every violated constraint, to showing the complete ideal solution.



database Movies
problem 29

List the names of all directors born in or after 1920.

```
SELECT    LNAME,FNAME
FROM      MOVIE
WHERE     BORN >= #01-01-1920#
GROUP-BY
HAVING
ORDER-BY
```

MOVIES

DIRECTOR	NUMBER	LNAME	FNAME	BORN	DIED				
MOVIE	NUMBER	TITLE	TYPE	AANOM	AAWON	YEAR	CRITICS	*DIRECTOR*	
STAR	LNAME	FNAME	NUMBER	BORN	DIED	CITY			
CUSTOMER	LNAME	FNAME	NUMBER	ADDRESS	RENTALS	BONUS	JDATE		
TAPE	CODE	*MOVIE*	PDATE	TIMES	*CUSTOMER*	HIREDATE			
STARS	MOVIE	STAR	ROLE						

Fig. 4.1. SQL-Tutor Interface

Figure 4.2 shows the architecture of SQL-Tutor. The system contains definitions of several databases and a set of problems for each database with their ideal solutions. Each problem is assigned a difficulty level. The difficulty level depends on many features, such as the wording of the problem, the constructs needed for its solution, the number of required tables/attributes, etc., and was determined by a domain expert. Each student is given a level of mastery, which dynamically changes in accordance with their performance. The student's level is incremented if he/she solves two or more problems consecutively at or above his/her current level, each within three attempts.

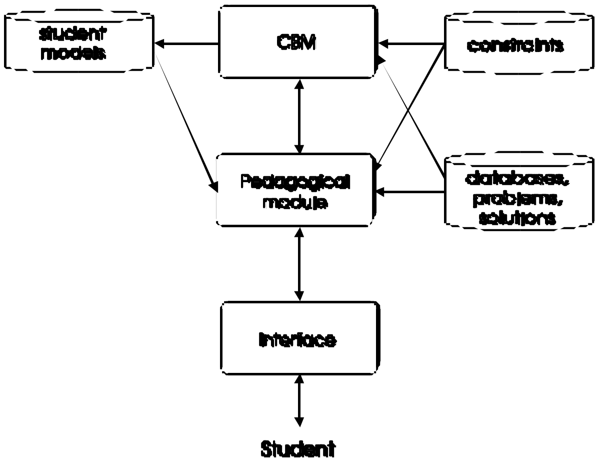


Fig. 4.2. Architecture of SQL-Tutor

The basic components of the system are the interface, the pedagogical module and the CBM student modeller. The pedagogical module (PM) observes every student's action and reacts appropriately. At the beginning of the session, a problem must be selected for the student. When the student enters the solution, the PM sends it to the student modeller, which analyzes the solution in order to identify possible errors. If any errors exist, the PM generates appropriate feedback messages. After the first attempt a student is only told whether his/her solution is correct or not. The level of detail increases if the student is not able to correct the solution.

The conceptual domain knowledge is represented in terms of over 500 constraints. A student’s solution is matched to the constraints to identify any that are violated. Long-term student knowledge is represented as an overlay model that tallies the percentage of times the constraint has been satisfied (i.e. used correctly).

There are three ways to select the next problem in SQL-Tutor. Students can work through a pre-specified sequence of problems by clicking *next problem*, or they can turn problem selection over to the system by clicking *system’s choice*. In the latter case, SQL-Tutor examines the student model and selects the first problem with a level within ±1 of the student’s level, and also relevant to the student’s most frequently violated constraint. The rationale for this rule is that if the student has violated the same constraint several times, it is appropriate to target it for instruction. This problem selection strategy is overly simple. In a real classroom, it was often the case that selected problems were too complex or simple for the student, or they jumped to another part of the domain seemingly not connected to the previous problem. The motivation for the changes to SQL-Tutor described in the next section, therefore, was to improve this situation.

4.2 Extensions to SQL-Tutor

To improve *system's choice* problem selection in SQL-Tutor, a new method based on Bayesian networks was implemented. There were two main differences between the new method and the original. Firstly, the student model was changed from a frequency overlay to a probabilistic overlay. Secondly, a new rule for problem selection that considers all of the constraints in the domain (rather than just the single most violated) was implemented.

4.2.1 Probabilistic Student Model

The new student model consists of a set of binary variables $Mastered_1$, $Mastered_2$,…,$Mastered_n$, 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 considering two frequencies from SQL-Tutor logs from previous evaluation studies. The frequency with which c was both satisfied and relevant in SQL-Tutor logs was divided by the frequency with which the constraint was relevant. That is, the initial probability of mastery is set to the fraction of times in the past that the constraint was satisfied when relevant. The logs were only analysed to the point where the user receives the first constraint-specific feedback about c, ensuring that the effects of learning did not bias the initial probabilities.

This initialisation method could not be performed for all of the constraints, because some were new and did not appear in past SQL-Tutor logs, and others had simply never been relevant to a student's solution before. For these constraints, $P(Mastered_c = \mathtt{YES})$ was initialised to 0.5 to represent the state of maximum uncertainty.

(a) If constraint *c* is satisfied, then $P(Mastered_c = \mathtt{YES})$ increases by 10% of (1-$P(Mastered_c{=}\mathtt{YES})$).
(b) If constraint *c* is violated and no feedback about *c* is given, then $P(Mastered_c = \mathtt{YES})$ decreases by 20%.
(c) If constraint *c* is violated but feedback is given about *c*, then $P(Mastered_c = \mathtt{YES})$ increases by 20% of (1-$P(Mastered_c{=}\mathtt{YES})$).

Table 4.1. Heuristics used for updating the student model

The student model is updated after the student submits his/her solution to a problem and receives feedback. During this evaluation study, the system used the heuristics in Table 4.1 to update the probabilities. This is one of the areas in which normative techniques should have been applied, but were not. In particular, Bayes' theorem (Equation 2.5) should have been applied to update the probability of mastery. However, this implementation and evaluation was exploratory in the sense that it *led* to the ideas presented in this thesis about using normative as opposed to ad-hoc systems. The extensions to SQL-Tutor, therefore, should be read in that light. CAPIT later demonstrated a completely normative solution to this problem.

It is interesting to consider the effects of the heuristic used here as compared to the effect Bayes' theorem would have had if it had been used. Let M denote mastery of a hypothetical constraint (M can take values YES or NO) and let L denote the outcome of the last attempt at the constraint (SATISFIED or VIOLATED). The case of violation with feedback will be ignored for the purposes of comparison. $P(M)$ therefore denotes the system's prior belief that the constraint is mastered, and $P(M|L)$ is the system's posterior belief given an observation of the student attempting the constraint, with outcome L. Table 4.1(a) and (b) are essentially two *ad-hoc* rules for computing $P(M|L)$ from $P(M)$ and L. However, Bayes' theorem is the normative means of performing this calculation. Substituting $P(M)$ and $P(L|M)$ these probabilities directly into Bayes' theorem (Equation 2.5) yields:

$$P(M|L) = Z^{1}P(L|M)P(M) \qquad (4.1)$$

where Z^{-1} is the constant of normalisation. Equation 4.1 can be expanded to show how individual entries in the *P(M|L)* table can be calculated. The expansion is:

$$P(M = m \mid L = l) = \frac{P(L = l \mid M = m)P(M = m)}{P(L = l \mid M = \text{YES})P(? = \text{YES}) + P(L = l \mid ? = \text{NO})P(? = \text{NO})} \qquad (4.2)$$

Both Equation 4.1 and 4.2 require a definition of *P(L|M)*, the probability that the constraint will be satisfied (or violated) given that the constraint is in the state of mastery (or non-mastery). This information is required by Bayes' Theorem but not by the ad-hoc method. Simply defined, *P(L|M)* is a function of the probabilities that the constraint will be violated if mastered (an unlucky slip) or satisfied if not mastered (a lucky guess). This is because the mastery variable is binary; if *M* had more states (e.g. an intermediate level of mastery), then a much more complex definition of *P(L|M)* would be required. For the purposes of this comparison, let the slip and guess parameters be 0.1. The conditional probability *P(L|M)* can therefore be defined by Table 4.2.

	M=YES	**M=NO**
L=SATISFIED	0.9	0.1
L=VIOLATED	0.1	0.9

Table 4.2. Definition of *P(L|M)*.

Now it is possible to compare Bayes' theorem with the heuristic given in Table 4.1. The update rules are depicted for the cases where the constraint *L* is both satisfied (Figure 4.3) and violated (Figure 4.4). The prior probability of mastery (the input to the function) is along the x-axis of both figures, and the posterior probability (the output) is along the y-axes.

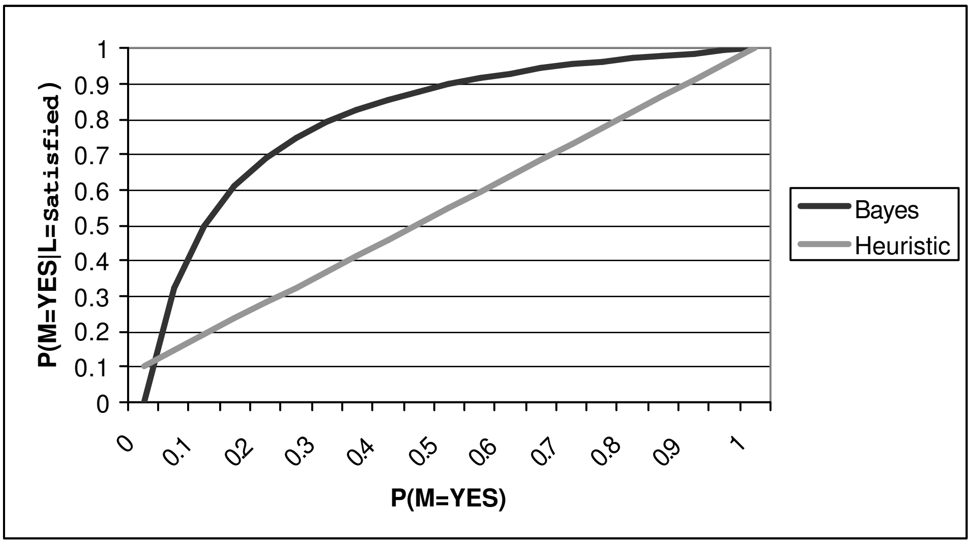


Fig. 4.3. Comparison of functions for updating *P(M)* when the constraint was satisfied.

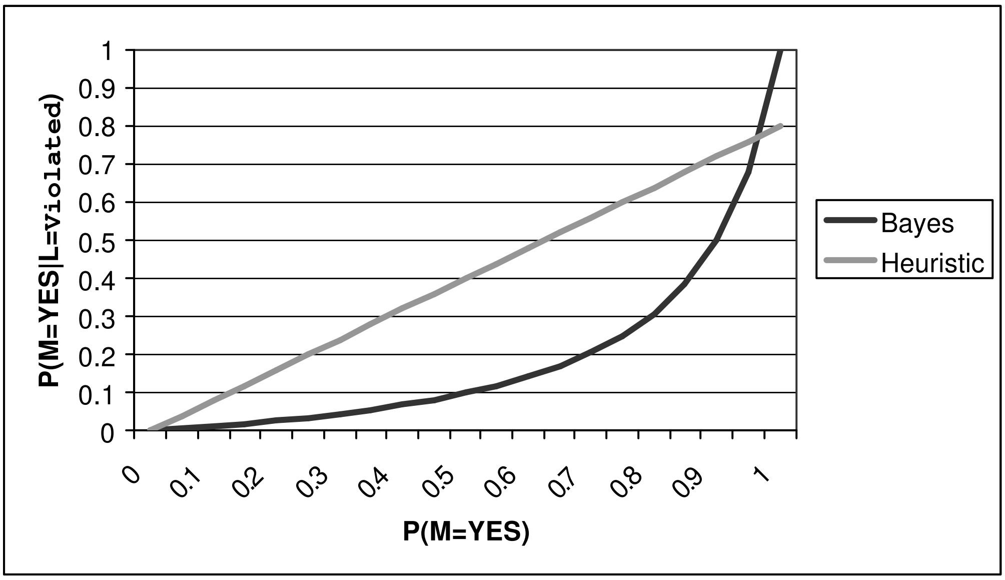


Fig. 4.4. Comparison of functions for updating *P(M)* when the constraint was violated.

Clearly, the heuristic produces a linear relationship between prior and posterior probability, but Bayes' theorem results in a non-linear relationship. In Figure 4.3, Bayes' Theorem rewards constraint satisfaction with a considerably greater increase in posterior probability than the heuristic, especially if the prior probability is lower. Conversely, a violation leads to a considerably greater relative decrease in posterior probability when Bayes' theorem is applied.

This analysis shows that the effect of using the heuristic instead of Bayes' theorem would be, if anything, a reduction in the sensitivity of the student model to individual constraint satisfactions and violations. That is, the current version of SQL-Tutor would require more satisfactions of a constraint to reach the same level of certainty that the constraint was mastered, than an equivalent version of the system implementing Bayes' theorem. On the other hand, Bayes' theorem makes use of more information about the domain (namely, the slip and guess parameters) than does the heuristic, which is independent of constraint-specific parameters. This can be a double-edged sword: if the parameters are accurate, then it should enhance the model; but if the parameters are less accurate due to a lack of prior information about the students (e.g. if they are set to constants, as they were in this study) then Bayes' Theorem could actually introduce more uncertainty.

Note the curvature of the Bayes' Theorem functions in Figures 4.3 and 4.4. The curvature tends to diminish (and the curve approaches linear) the closer the slip and guess parameters get to 0.5 (although the slope of the curve still differs from that of the heuristic). It would be an interesting avenue of future research to determine which function best describes actual student behaviour.

4.2.2 Predicting Student Performance on Single Constraints

We use the simple Bayesian network depicted in Figure 4.5 to predict the performance of a student given a problem *P* on a single constraint *C*. $Mastered_c$ is the mastery variable from the student model. Both $RelevantIS_{c,p}$ and $RelevantSS_{c,p}$ are `YES/NO` variables. $RelevantIS_{c,p}$ is `YES` if constraint *C* is relevant to problem *P*'s ideal solution. Because this can be determined from the problem database, $RelevantIS_{c,p}$ is always known with certainty. $RelevantSS_{c,p}$ is `YES` if constraint *C* is relevant to the student's solution to problem *P*. This is uncertain because the student may either enter an incorrect solution not relevant to *C*, or she/he may enter an alternate correct solution that is also not relevant to the constraint in question. $Performance_{c,p}$ is a three-valued node taking values `SATISFIED`, `VIOLATED` or `NOT-RELEVANT`. The arcs indicate that the relevance of the constraint to the student solution, $RelevantSS_{c,p}$, depends on the relevance of the constraint to the ideal solution, $RelevantIS_{c,p}$. The performance of the student on his/her next attempt at the constraint, *Performance*$_{c,p}$, is dependent on whether or not the student has mastered the constraint *C* and *C*'s relevance to the student solution.

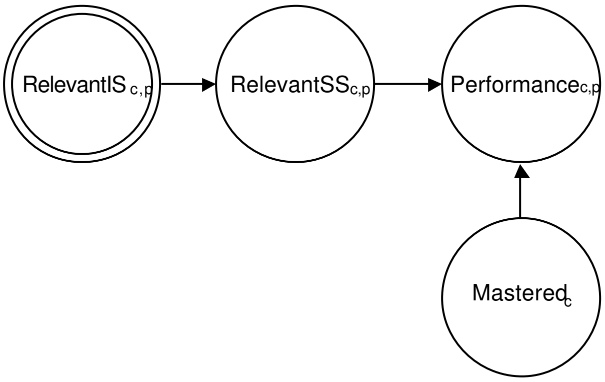


Fig. 4.5. A simple Bayesian network for predicting student performance on a single constraint.

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 4.3 defines the conditional probability *P(RelevantIS*$_{c,p}$*|RelevantSS*$_{c,p}$*)* as a function of two parameters of constraint *C*, α_c and β_c. Parameter α_c (β_c) is defined as the probability of a constraint being relevant to the student's solution if it is (not) relevant to the current problem'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 $\alpha_c = 0.9$ for example, there is a high probability that constraints relevant to the ideal solution will also be relevant to a student solution.

		RelevantIS$_{c,p}$	
		YES	NO
RelSS$_{c,p}$	YES	α_c	β_c
	NO	$1-\alpha_c$	$1-\beta_c$

Table 4.3. Definition of *P(RelevantSS*$_{c,p}$*|RelevantIS*$_{c,p}$*)*

Like the initial probabilities of mastery, values for α_c and β_c were determined from past SQL-Tutor logs. However, these conditional probabilities are not available *directly* from the data. All that can be determined from the logs are the frequencies with which constraints are relevant to the ideal and student solutions, or both. Equation 4.3 shows how α_c was calculated using a rearrangement of Bayes' Theorem (Equation 2.4). A similar calculation was performed for β_c. For new or previously unused constraints, α_c and β_c were initialised to 0.5 to reflect the state of maximum uncertainty.

$$
\begin{aligned}
\alpha_c &= P(RelevantSS_{p,c} = \texttt{YES} \mid RelevantIS_{p,c} = \texttt{YES}) \\
&= \frac{P(RelevantSS_{p,c} = \texttt{YES} \;\&\; RelevantIS_{p,c} = \texttt{YES})}{P(RelevantIS_{p,c} = \texttt{YES})} \\
&= \frac{\text{\# times C is relevant to both the SS and the IS in the logs}}{\text{\# times C is relevant to IS in the logs}}
\end{aligned}
\qquad (4.3)
$$

$RelevantSS_{c,p}$

$Mastered_c$

$Performance_c,$

	YES	YES	NO	NO
	YES	NO	YES	NO
SATISFIED	1-*$Slip_c$*	*$Guess_c$*	0	0
VIOLATED	*$Slip_c$*	1-*$Guess_c$*	0	0
NOT-RELEVANT	0	0	1	1

Table 4.4. Definition of $P(Performance_{c,p}|RelevantSS_{c,p},Mastered_c)$

Table 4.4 is the conditional probability distribution of $Performance_{c,p}$ given its parent variables $RelevantSS_{c,p}$, and $Mastered_c$. $Slip_c$ ($Guess_c$) is 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 4.4, $P(Performance_{c,p}$ = NOT-RELEVANT$)$ = 1.0, because these columns represent 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, these will not be known with certainty because the mastery of a constraint and its predicted relevance to the student solution is uncertain. The probability distribution over $Performance_{c,p}$, therefore, will be uncertain.

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: α_p = 0.9, β_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].

4.2.3 Evaluating 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 200 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 of more capable peers*". This can be interpreted as saying that a student should be given a problem that is slightly above their current level but not so difficult as to discourage the student.

Let the strategy we propose for selecting problems be discussed. 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 suitable complexity will generate an appropriate number of feedback messages. This is the basis of the evaluation function we propose.

The algorithm for evaluating problems is given in Listing 4.1. 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 appropriate number of feedback messages the 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.

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

List. 4.1. The problem evaluation function.

Like the mastery probability update rule, this function is a heuristic. Ideally, decision-theoretic methods should have been used for problem selection. In fact, this was what was originally intended. However, initial tests indicated that an on-line decision-theoretic procedure incorporating the performance probability of all five hundred constraints would have been intractable, and so the heuristic method in Listing 4.1 was used.

The evaluation function assumes that feedback will be generated for every constraint where $P(Performance_{c,p} = VIOLATED) > 0.45$. It does not attempt to calculate the probability of a particular number of feedback messages, again because of tractability concerns. The constant 0.45 was chosen because initial tests showed that it gave the best results. The problem with the highest value is selected from the pool of unsolved problems within 1 level of the student's level.

4.3 Evaluation Study Results

We performed an evaluation study in October 1999, with second year students enrolled in an introductory database course. The students were randomly assigned to versions of SQL-Tutor either with the probabilistic student model/problem selector (the experimental group) or without it (the control group). A total of 18 students were placed in the control group, while 14 were placed in the experimental group. The study consisted of one 2-hour session in which students sat a pre-test, interacted with the system, and then completed a post-test. Timing of the study was a constraint, as students needed to have some overall understanding of databases prior to using SQL-Tutor. The only possible time for the study, therefore, was the last week of the school year, which had a negative effect on the number of participating students.

4.3.1 Appropriateness of Selected Problems

All student actions performed in the study were logged, and later used to analyse the effect of the proposed problem-selection approach on learning. Both groups had access to the two problem selection methods described in Section 4.1: clicking *next problem*, or clicking *system's choice.* In the case of the control group, clicking *system's choice* led to a problem being selected using the original approach, whilst the Bayesian approach was used for the experimental groups.

	Exper. group	**Control group**
Next problem	3.18	2.10
System's choice	2.69	4.55

Table 4.5. Average number of attempts per solved problem.

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 problems selected by *next problem* and 79 problems selected by *system's choice*. The number of attempts it took to solve each problem was counted, and the averages are given in Table 4.5. The problems selected for the control group by the heuristic were the most difficult for the students, requiring 4.55 attempts on average to solve. The students in the experimental group were able to solve problems selected by the Bayesian approach on an average of 2.69 attempts, compared to 3.18 attempts when problems were visited in turn. The proposed problem selection method, therefore, compares favourably with the heuristic approach used by the control group.

The average of 2.10 attempts by the control group when picking the *next problem* compared to 3.18 for the experimental group can be explained as follows. When the student clicks *next problem*, the problem immediately following the last attempted problem on the list of problems is selected. The problems are arranged on the list from easiest to hardest. To illustrate, consider a hypothetical situation where a student selects two problems, one via *system's choice* and the other with *next problem*. The first problem is meant to be the most appropriate problem for the student, and therefore does not depend on the ordering of the problems in the list. The second problem will be the problem on the list following the first problem, and so it may not be well-adapted at all. In fact, if the *system's choice* problem occurs later in the list (where the problems are harder), then the *next problem* may well be very difficult. This is reflected in the averages: the experimental group takes fewer attempts to solve *system's choice* problems because they are well-adapted, but more attempts to solve problems selected via *next problem* because the next problem on the list may not necessarily be as well-adapted. Furthermore, because the experimental strategy progresses to more difficult problems at a faster rate (as shall be explained), this leads to the number of attempts by the experimental group being higher when *next problem* is selected.

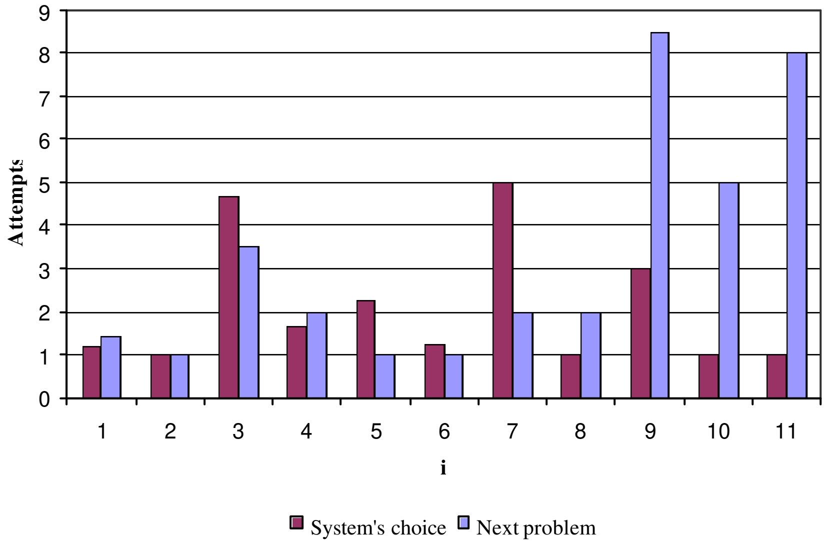


Fig. 4.6. The average number of attempts to solve the *i*th problem by students in the experimental group.

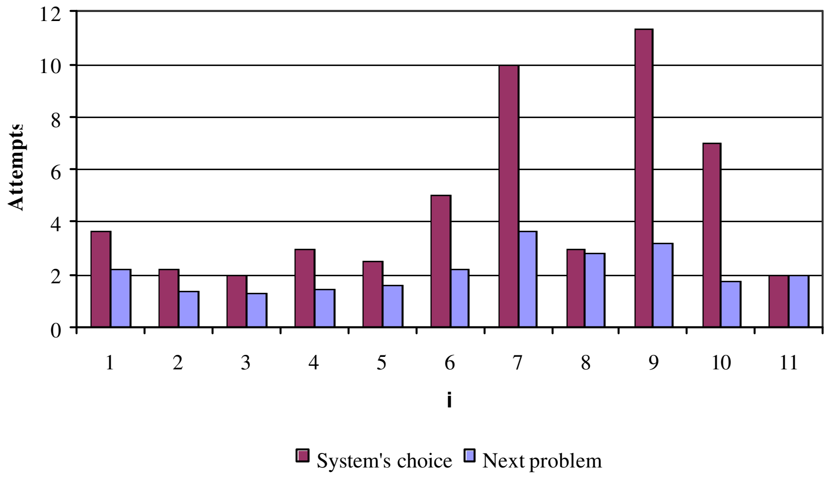


Fig. 4.7. The average number of attempts to solve the *i*th problem by students in the control group.

The new *system's choice* method is only slightly better on average than the *next problem* option for the experimental group, but its advantages are clearer when the problem solving session is analysed temporally. The students begin with simple problems, and progress to more complex ones. Figure 4.6 illustrates the average number of attempts that students in the experimental group took to solve the *i*th problem, and Figure 4.7 depicts the same graph for the control group. It can be seen that the initial problems selected by *next problem* are easier for the experimental students than those selected by the Bayesian approach. This is explained by the fact that the Bayesian approach 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. The opposite trend occurred in the control group (Figure 4.7), in which *system's choice* progressively selected more difficult problems. This is perhaps due to the original problem selection rule that selects problems on the basis of a single constraint (the most violated) as opposed to the new rule that considers all the constraints.

4.3.2 Pre/post tests

Pre- and post-tests 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 4.6. The mean scores in the pre-test for the two groups are very close, 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, conducting 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 a result, we can draw no conclusions from a comparison of the pre- and post-test results.

Question	**Experimental group**	**Control group**
1	0.22	0.25
2	2.67	2.73
3	0.62	0.73
Total	3.44	3.50

Table 4.6. Means for the pre-test

4.4 Lessons Learned

The feasibility of normative techniques were explored. Although the improvements to SQL-Tutor did have some heuristic elements, the evaluation results suggest that this approach leads to an improvement. Of course, an entirely Bayesian and decision-theoretic solution should have replaced SQL-Tutor's student model and problem selection heuristic. This did not happen for a number of reasons, mainly the complexity of SQL-Tutor and the size of the domain. Initial tests showed that the number of constraints (more than 500) made modelling all within one connected Bayesian network intractable, and so a number of compromises were made so the system could tractably evaluate problems. There are similar problems in other domains. For example, ANDES models many different physics rules and because it would be infeasible to consider all of them at once, the Assessor network considers only those rules appearing in the problem's solution graph (Conati et al., 1997) because it is assumed that only those rules can be relevant. This is a strong assumption because it presupposes that all possible correct solutions or problem-solving methods are accounted for by the system. CBM does not make such an assumption because of phenomena such as radical strategy variability that suggest that the modelling of procedural knowledge is near-impossible. As a result, CBM tutors do not require a problem-solving component, but must take into account the fact that any constraint can potentially be relevant to a student's solution. It is therefore necessary to model each constraint during every problem attempt, and this leads to the "multiple smaller networks" approach and the problem selection heuristic of Listing 4.1.

Another source of sub-optimality in this implementation was the existence of levels. The method of considering only problems within one level of the student's level effectively stereotypes the student. Although a domain expert assigned the problem levels, there is no guarantee that the most appropriate problem will actually be within the selectable levels. Ideally, an adaptive system would not require levels at all. However, again for tractability reasons, levels are necessary for SQL-Tutor because they limit the amount of problem evaluation that must be performed.

Finally, SQL-Tutor was also not designed specifically for the evaluation of PAS strategies. It is a large system that allows the student considerable control over the settings, such as the level of feedback (there are five) and the method of problem selection (*system's choice*, *next problem,* or from the list). This flexibility makes it difficult to compare, for example, two problem selection strategies, because there are a number of additional uncontrolled variables.

Therefore, although this evaluation did not demonstrate pure decision-theoretic PAS, it did provide a number of guidelines for the research that followed. The first guideline was to select a domain where the number of constraints could be kept to a manageable quantity. This would ensure that enough data could be collected on each constraint to reliably estimate parameters such as the prior probability of the constraint being satisfied. The tutor still had to be practical, but a smaller number of constraints means that firstly, dependencies between constraints can be modelled, and secondly, pure normative methods would be tractable using standard algorithms. Better hardware and more sophisticated algorithms will lead to gains in tractability. The second guideline for future research was to eliminate all heuristics such as the problem selection heuristic and the mastery probability update rules. Additionally, the system should not contain subjective prior knowledge such as problem levels. The third and final guideline was to eliminate all sources of variability other than the decision strategies being used. That is, the student should not be permitted to modify the tutoring environment (e.g. the level of feedback, etc.), so that carefully controlled experiments can be performed. Although this is overly restrictive for general tutoring systems, it is necessary in order to evaluate and compare subtleties such as the difference between two decision strategies. Note also that the changes in the third guideline could have been implemented in SQL-Tutor, but because of the other problems, an entirely new tutor was later developed. We used these guidelines to formulate a general methodology for building the PAS component of a decision-theoretic tutor.

Chapter 5

A Methodology for Building Decision-Theoretic Pedagogical Action Selection Strategies

A methodology for developing normative tutors is proposed in this chapter. The basis of the methodology is to make explicit and draw together three “merits” or desirable qualities of an ITS design process. This first is that of *evaluation during design*. This is considered critically important by the ITS community. Systems should be evaluated in the classroom, and the feedback should be used to further improve the system. The second desirable quality is *machine learning*. More often than not, simple approaches such as frequency counts are used to construct long-term student models. However, such approaches may overlook properties of the domain and/or the student population that a slightly more powerful solution may detect. Interdependencies between knowledge items (see Chapter 3, Section 3.2.2) are one phenomenon that a frequency-based approach misses. Furthermore, because evaluation during design is also advocated, live student performance data can be used as the input to the machine learning algorithms. The third desirable quality of an ITS design process is to implement *normative methods* as the mechanism for representing and reasoning about the student's and system's behaviour. This is the main thrust of this thesis, and the reasoning behind this has been extensively discussed earlier on.

There are five steps involved in the methodology, and each step is now justified. Briefly, the first step is data acquisition. This involves a partly-complete version of the ITS (with no long-term student model and randomised PAS) being deployed in the classroom for the purpose of collecting live student performance data. This step provides the data source for the next step, which is machine induction of a Bayesian network student model. The reasoning for this is that initialising the student model from actual student data is likely to lead to a more effective initial state than default values would lead to. The third step is decision-theoretic strategy construction. The point of this step is to immediately define the decision problems the ITS must solve and then link these to the student model. A student model in isolation is effectively useless; its value comes from the predictions it can make about the student, which in turn are used to select actions. The fourth step is another machine learning step. Once a new student is using the system and data is being acquired from this student, this new data can gradually replace the older data. Effectively, the Bayesian network is biased towards the current student over time. Finally, the fifth step is a full-scale evaluation. The purpose is essentially to test the completed system in the classroom to ensure that the additional computational effort required to perform the normative calculations actually pays off. The final system must be shown to be better than the initial randomised version.

Table 5.1 illustrates the five steps in the methodology. The order of the steps is certainly not rigid; the designer could iterate from one step to an earlier step, if required. Sections 5.1-5.6 describes each step in more detail, and then section 5.7 compares the methodology to other design processes.

1	Randomised Data Collection
2	Model Generation
3	Decision-Theoretic Strategy Implementation
4	On-line Adaptation
5	Evaluation

Table 5.1. The five-step methodology for designing decision-theoretic PAS strategies.

5.1 Randomised Data Collection

The first step is randomised data collection, in which an almost fully functional version of the tutor is tested in a classroom representative of the intended population of users of the system. The only difference between this and the final version of the tutor is the PAS strategy. In the initial version of the tutor, the PAS strategy is random. In other words, given a set of alternatives (such as unsolved problems), the tutor makes the selection completely randomly. All actions should be logged as records of form <*State, Action, Outcome*>, where *State* is a description of some state prior to the action selection (e.g. the state of the student model, or the recent history of the student, or a combination thereof), *Action* is the pedagogical action that is randomly selected (e.g. the next problem), and *Outcome* is the observed outcome(s) of the action (e.g. correct or incorrect). Because PAS selection is random, the data should be uniformly spread over all the possible actions.

5.2 Model Generation

The next step is model induction, the construction of a Bayesian network for predicting student performance given the current student and an action selected by the tutor. More specifically, using normative methods the task is to build a predictor for the value of *Outcome* given values for *State* and *Action*. The data from Step 1 serves as the source from which the model is induced. At this stage, prior and expert knowledge can be optionally added to the network. This can be achieved either before learning by adding dependencies and probabilities between the variables, or after learning, by fine-tuning the induced network. Cheng et al. (1998) describe a Bayesian network induction algorithm capable of starting from a partial specification. However, the rationale for any decision at this stage should be to enhance predictive performance. Also, because the network is being learned from tutor log data, the variables can only represent observations. Internal hidden states will be implicit in the conditional probabilities relating the observable variables.

5.3 Decision-Theoretic Strategy Implementation

The third step is implementation of the decision-theoretic strategy. This is an encoding of the expected utility function (Equation 2.28) to rank the outcomes of a pedagogical decision. When combined with the estimates of the probabilities of each outcome, the expected utility of each pedagogical action can be calculated.

Recall that the expected utility function has two main components; the utility function *U(X,D)*, and the conditional probabilities of outcomes given decisions, $P(X|D)$. The Bayesian network constructed in the previous step is used to provide the outcome probabilities, $P(X|D)$. However, the utility function is not yet defined. In fact, it is at this point that teaching knowledge is incorporated into the system. The utility function essentially ranks the outcomes of tutorial actions. To illustrate, if *D* represents a possible next problem and *X* is the number of errors the student makes when attempting the problem, then *U(X,D)* can be defined to be maximal for some appropriate number of errors. This incidentally is the strategy used to select problems in CAPIT, and will be discussed in more detail in the next chapter.

5.4 On-Line Adaptation

Step four involves implementing an on-line Bayesian network learning algorithm. In Step 2, the Bayesian network is constructed from population data. Stern et al. (1999) coined the phrase "population student model" to refer to a probabilistic model induced from population data. However, as data is acquired directly from the current student, this population data should be gradually discounted. In other words, the population student model needs to gradually become more individualised to the student. Additionally, as the student state changes over time, even old data acquired from the student will need to be discounted. While there are a number of existing algorithms for Bayesian network induction from data, there is little in the way of *on-line* Bayesian network induction algorithms. Furthermore, the online learning algorithms that do exist (e.g. Heckerman, 1999; Bauer et al., 1998) make the assumption that the data-source is essentially static and unchanging over time, in direct contrast to an actual student whose state changes constantly. Therefore, when the methodology was applied to CAPIT, as will be described in the next chapter, an existing algorithm for on-line conditional probability learning was modified to account for this. The much more difficult problem of updating a network's structure on-line was not considered.

5.5 Evaluation

The fifth step is an evaluation of the decision-theoretic PAS strategy. This is necessary to ensure that the decision-theoretic strategies actually provide a benefit for the extra computational effort they require. One strategy that requires virtually no computational effort is the randomised PAS strategy implemented for Step 1. Therefore, one can compare the PAS strategy developed in Steps 2-4 with the random PAS strategy used in Step 1. In other words, the decision-theoretic and randomised PAS strategies are to be evaluated in a controlled experiment in which one group of students (the control group) use a version of the tutor with the original, randomised strategy, and the second group of students (the experimental group) use the version with the decision-theoretic strategy. Both versions of the tutor should be identical and limit the student's ability to modify the tutoring environment (e.g. feedback levels) so as to make the comparison as valid as possible. We have performed this evaluation with CAPIT.

5.6 Summary

The methodology is a five-step process for constructing a Bayesian network student model and a decision-theoretic PAS strategy for one or more tasks. A summary of the methodology is as follows. Firstly, data is collected by evaluating the initial randomised version of the system in a classroom. This data is used to induce a Bayesian network that predicts a probability distribution over *Outcomes* given a *State* and an *Action*. A utility function over the different outcomes is also defined at design-time. At run-time, for each potential next action being considered, *P(Outcomes|Action)* is computed by the Bayesian network. When combined with the utility function, this information is sufficient to compute the expected utility of an action. The action with the highest expected utility is always selected. When the actual outcomes are observed, this information and other information such as the student's *State* and the *Action* that was finally selected are added to the training data and used to update the Bayesian network on-line.

Figure 5.1 illustrates the process. Some of the components of the figure are labelled to clarify the stage of the methodology in which they are created. For example, the training data is accumulated during Step 1 and updated by the procedure created in Step 4, and the Bayesian student model is constructed in Step 2 and updated during Step 4.

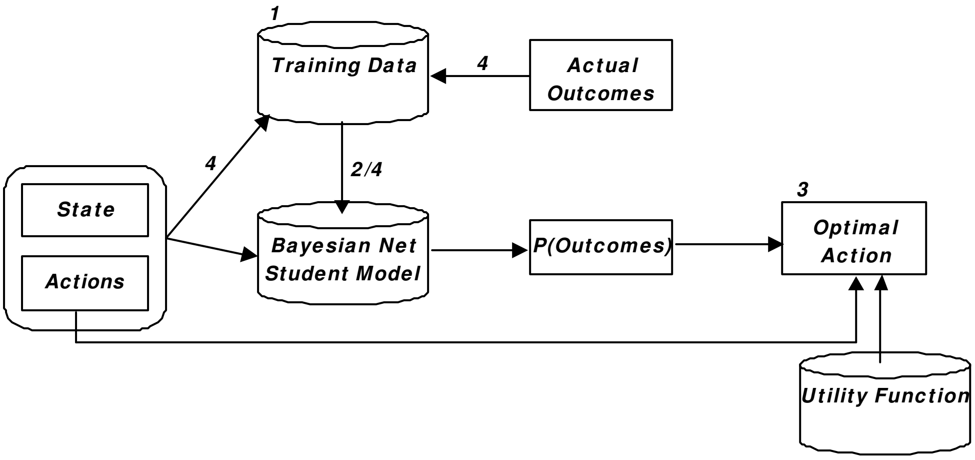


Fig. 5.1. Overview of the methodology.

5.7 Comparison to other ITS Design Methodologies

Bloom et al. (1997, pp. 251) advocate evaluation as a crucial step in the design of intelligent tutoring systems. In their words, "*Having end-users involved from the onset of a development project helps ensure that the system developed is more likely to satisfy the functional requirements and performance criteria, as well as meet their affective requirements.*" Two types of evaluation discussed by Bloom et al. and Mark & Greer (1993) are formative and summative evaluation.

Formative evaluation considers the architecture and behaviour of the ITS. Does it work as expected? Can it be used effectively? This type of evaluation is characterised by experts inspecting all the internal aspects (e.g. the domain knowledge and pedagogy) as well as the external aspects (the behaviour) of the tutor to assess its usefulness and suggest improvements. It may also include pilot studies in which students use the system either individually or in groups in a classroom setting, which can be extremely useful for assessing the user interface. Detailed results are then used to improve the system.

Summative evaluation, on the other hand, considers the teaching effectiveness of a (usually) completed system. It is an extensive external assessment of the tutor. It includes both subjective analyses directed at students and instructors (e.g. questionnaires, ratings) as well as objective analysis from acquired data (e.g. student model and log analysis). The process advocated by Bloom et al. is to iterate the process of design followed by formative evaluation, and to finish with a final summative evaluation.

This procedure was used to develop the intelligent tutoring system LEAP (Bloom et al., 1997). LEAP teaches customer interaction skills to customer contact employees (salespeople) in a corporate environment. The system was modelled on the metaphoric "interactive book" and comprises three instructional components: a multimedia guide book, a "what if?" analysis module in which students can observe expert responses to certain situations, and a conversation rehearsal module in which students can simulate conversations with customers. LEAP went through four design-evaluate cycles. Major changes to the domain models, pedagogy, and user interface were introduced as a result of the first two evaluations; more refined changes were made following the latter two. A summative evaluation of the final version of LEAP is described in Bloom et al. (1997). Interestingly, there were a number of design issues still outstanding in the final version of the system, despite the considerable evaluations. Most significantly, only one of the three instructional methods (namely, conversation rehearsal) was used frequently. Other issues involved the lack of flexibility of some parts of the system, and the detail and type of feedback. Despite these drawbacks, the students' abilities and confidence were shown to improve, and the system was well-liked by both instructors and students. Overall, the LEAP evaluation was described as a "quality learning experience" and could therefore be described as a successful ITS.

The methodology discussed in this chapter is in the same vein as that of the methodology used to design LEAP. However, the focus is much narrower: rather than using evaluation as a component in the design of an entire system, we consider only the development of the student model and intelligent decision strategies within an existing system. This allows more focus on the comparison of different normative techniques (e.g. different Bayesian network structures) for decision making, but makes the implicit assumption that the rest of the system, (e.g. the user interface, feedback messages, etc.), is complete and remains static during the iterative development. This ensures that differences between strategies can actually be isolated.

Another difference between the method proposed here and LEAP's design is that whereas LEAP had a designer to interpret the results of each evaluation and make the subsequent improvements to the system, the philosophy of the methodology proposed here is to make the designer as "transparent" as possible. That is, the improvements to the system should come from objective processes such as machine learning rather than designer decisions. This is, of course, feasible when one is considering only the student model and the selection strategies; other issues such as user interface design can only be dealt with by a human designer. Naturally, the designer cannot be completely removed from any design process (e.g. she/he must at least select the Bayesian network learning algorithms that will be used), but his/her influence can be minimised. Despite these differences, the key philosophy of both processes is the incorporation of evaluation into the design process.

EasyMath also involved extensive evaluation in the design process (Virvou & Tsiriga, 2000). EasyMath is a tutor for algebraic powers, and the designers of the system realised from the outset that involvement of teachers and students would be highly beneficial. They were also concerned that a number of ITSs have been developed and tested in laboratory settings only, and not within real classrooms. In their own words, ITSs "*...have often been criticised that they are mainly research products, which have not been used in real classrooms.*" Evaluation in the classroom was therefore a design goal. The development of EasyMath was a three-step process. Firstly, the *inception* phase was an empirical study of the algebraic powers domain. Teachers were asked to construct a test covering the entire domain and the test was administered to over 200 students. The results were used to determine an enumerative bug library. Teachers were also asked to specify their requirements during this phase. The *elaboration* phase occurred next, which involved testing the first executable version of EasyMath with two teachers and ten students. The system was evaluated by observation and questionnaires. Finally, the third phase, *construction*, extended the initial prototype into the final version of EasyMath using the results of the initial evaluation as a guide.

The approaches of Bloom et al. (1997), Virvou & Tsiriga (2000), and myself during the development of CAPIT are frameworks for incorporating evaluation into the ITS design process. The main difference between the approaches is that machine learning is advocated in this thesis as a suitable method for developing decision strategies, whereas the more traditional software-engineering approaches of Bloom et al. and Virvou & Tsiriga would have a knowledgeable human designer programming the decision strategies from evaluation results. The methodology is most similar to Virvou & Tsiriga's approach in that two evaluations are performed during the design process: one evaluation to collect data (equivalent to the elaboration phase), and a final evaluation of the completed system (equivalent to the construction phase).

To conclude, a five step methodology has been proposed for designing normative student modelling and PAS components for ITSs. In the next chapter, a demonstration of the application of the methodology is described.

Chapter 6

Case Study: Using the Methodology to Construct CAPIT

The application of the general methodology to the development of a Bayesian student model and decision-theoretic strategies for the intelligent tutor CAPIT (Capitalisation And Punctuation Intelligent Tutor) is described in this chapter. CAPIT is an ITS that teaches the basic mechanical rules of English capitalisation and punctuation to children in the 8-10 year old age group. Section 6.1 introduces CAPIT and describes the system's interface and overall architecture. In Sections 6.2 and 6.3, the problems and constraints represention are described in detail. The application of the general methodology to the construction of CAPIT's student model and decision-theoretic strategies is then described. Specifically, Section 6.4 describes the first step, randomised data collection. Section 6.5 shows how the optimal Bayesian network student model was selected used statistical significance tests (Step 2 of the methodology). In Sections 6.6 and 6.7, the problem and feedback selection strategies are described (constituting Step 3 of the methodology). The implementation of Step 4, a mechanism for adapting the conditional probabilities of the Bayesian network to the student, is described in Section 6.8. The last step in the methodology, an extensive evaluation, is described in Section 6.9. Finally, the results are summarised and discussed in Section 6.10.

6.1 CAPIT: Capitalisation And Punctuation Intelligent Tutor

CAPIT (Mayo et al., 2000; Mayo & Mitrovic, 2001) is the second intelligent tutor to implement Ohlsson's Constraint-Based Modelling (CBM) (Ohlsson, 1994), the other being SQL-Tutor that was introduced in Chapter 4. CBM was described in Section 3.1.5. CAPIT is implemented in Visual Basic 6, and it runs on any 32-bit Windows platform. The Bayesian network reasoning module it uses is MSBN which is provided by Microsoft (http://research.microsoft.com/msbn).

Traditional capitalisation and punctuation exercises for children tend to fall into one of two categories (Bouwer, 1998): *completion* (the student must punctuate and capitalise a fully lowercase, unpunctuated piece of text), and *check-and-correct* (the student needs to check for errors, if any, and correct them). CAPIT poses problems of the first class, the completion exercise. If the child makes a mistake, an error message is displayed. For example, Table 6.1 depicts one of the shorter problems in the system, a student's incorrect attempt at punctuating and capitalising it, and the tutor's correct solution.

(a) the driver said it will rain
(b) The driver said, “it will rain”.
(c) The driver said, “It will rain.”

Table 6.1. (a) A problem, (b) a student's incorrect solution, and (c) the correct solution.

There are two errors in the student's solution: the direct speech does not start with a capital letter, and the period is outside the quotation marks. Currently, CAPIT displays only one error message at a time, and the student is expected to correct the error (and any others) and resubmit the problem before any more feedback is displayed. If the student submitted the solution given in Figure 6.1(b), a feedback message such as *The full stop should be within the quotation marks! Hint: look at the word* rain *in your solution* would be displayed. Error messages are typically short and relate to only a single mistake, but if the student wants more detailed information, she/he can click *Why?* to be shown further explanatory material.

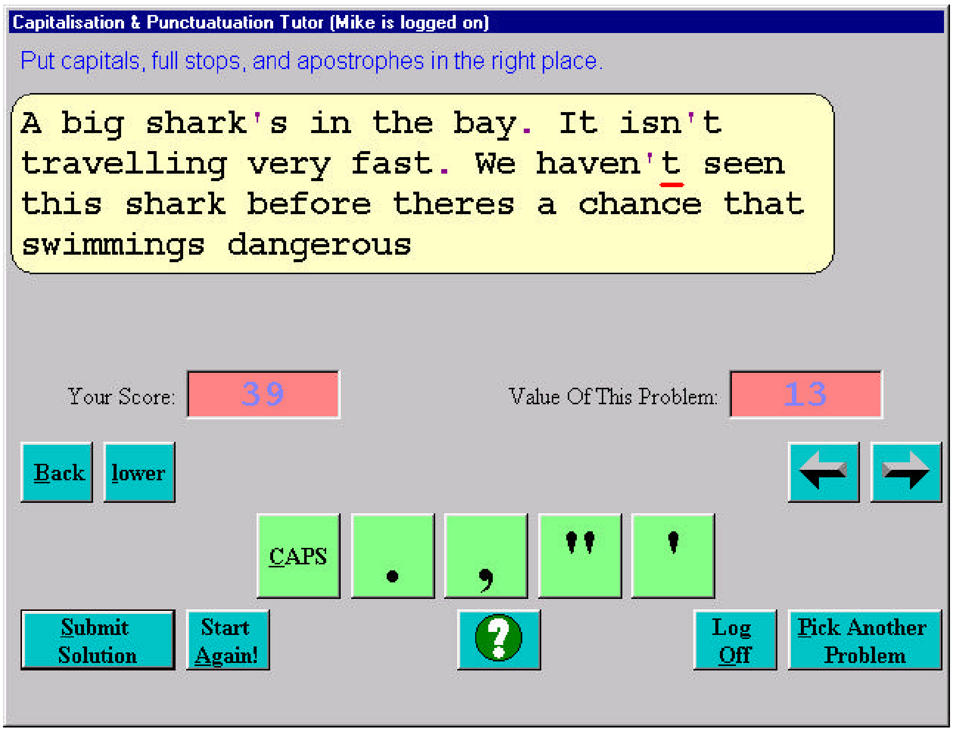
The current version of CAPIT contains 45 problems and 25 constraints. The problems are relevant to the constraints in roughly equal proportions, although a small number of constraints (such as capitalisation of sentences) are relevant to all the problems. The constraints cover the following parts of the domain:

- Capitalisation of sentences.
- Capitalisation of the names of both people and places.
- Ending sentences with periods.
- Contracting *is* and *not* using apostrophes (e.g *haven't*).
- Denoting ownership using apostrophes (e.g. *John's dog*).
- Separating clauses using commas.
- Separating list items using commas (e.g. *apples, oranges, lemons and pears*).
- Denoting direct speech with quotation marks.
- The correct punctuation of the possessive pronoun *its*.

All the constraints are listed in Appendix C.

6.1.1 Interface

CAPIT's main user interface, showing a partially completed problem, is depicted in Figure 6.1. Brief instructions relevant to the current problem are clearly displayed at the top of the main interface. This reduces the cognitive load by enabling the learner to focus on the current goals at any time without needing to remember them. Immediately below the instructions, and clearly highlighted, is the current problem. In this area, the child interacts with the system by moving the cursor using keyboard or mouse, capitalising letters, and inserting punctuation marks. The child can provide input either by pointing and clicking the mouse, or by pressing intuitive key combinations such as *Shift-M* to capitalise the letter *m*. By requiring the cursor to be positioned at the point where the capital letter or punctuation mark is to go, the child's ability to locate errors as well as correct them is tested.



Capitalisation & Punctuatuation Tutor (Mike is logged on)

Put capitals, full stops, and apostrophes in the right place.

A big shark's in the bay. It isn't travelling very fast. We haven't seen this shark before theres a chance that swimmings dangerous

Your Score: 39

Value Of This Problem: 13

Fig. 6.1. CAPIT's main user interface.

Motivation is provided in two ways. Firstly, whenever a correct solution is submitted, some points are added to the child's score. The number of points added is equal to the number of punctuation marks and capital letters in the solution that was just submitted. Secondly, whenever a correct answer is submitted, an animation is displayed. These simple strategies were found to be highly effective motivators for children in the target age group of 8-10 years old.

6.1.2 Architecture

Figure 6.2 shows the architecture of CAPIT. The student model comprises a record of the outcome of the previous attempt at each constraint (the short-term model) and the current configuration of the Bayesian student model (the long-term model). The student modeller is a pattern matcher that takes the student's solution to a problem and determines which constraints are violated. It then passes the violated constraints (if any) to the pedagogical module. The pedagogical module is the core component of the system. It currently performs two significant PAS tasks: firstly, given the violated constraints, it selects the single violated constraint about which feedback should be given; secondly, when *Pick Another Problem* is clicked, or when the student solves the current problem, the pedagogical module selects the most appropriate next problem for the student. The current version of the pedagogical module can perform PAS in two ways: randomly, or using decision theory. More details of these strategies will be given later in the paper.

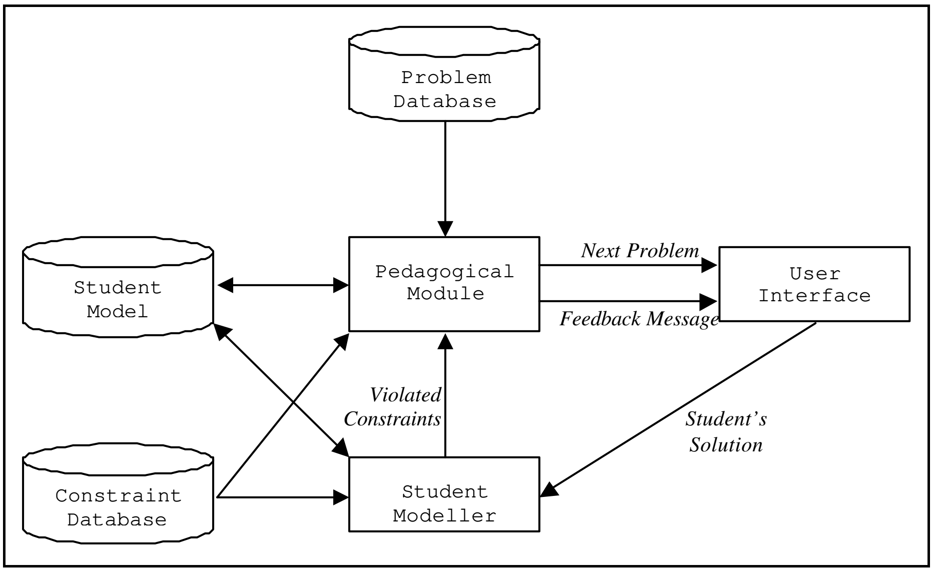


Fig. 6.2. The architecture of CAPIT.

6.2 Problem Representation

Problems in CAPIT are represented as arrays of words. Each word in the representation is properly punctuated and capitalised, and the tutor generates the initial problem text by removing the punctuation marks and capitals, and stringing the words together into a single piece of text that the student can then edit. Each word also has one or more *tags* associated with it. The tags specify the semantic and/or grammatical classes of a word, to the degree that it is relevant for punctuation and capitalisation. For example, Table 6.2 is the tutor's internal representation of a short problem. Each word in this problem has one or more tags. The tags themselves have no explicit meaning in the system; their meaning is implicit, deriving from the fact that they determine which constraints are relevant to the word and the problem.

The tag DEFAULT indicates that a word does not need to be punctuated or capitalised (although the system does not actually prevent the student from doing so), such as *driver* and *will* in the example. As DEFAULT words have their capitalisation and punctuation requirements completely specified (they do not have any), such words do not require any other tags. Other tags such as L-CASE indicate that a word does not need to be capitalised, but says nothing about the punctuation requirements (and vice-versa for the tag NO-PUNC). Other types of words need more specific tags. For example, *The* is the first word in the sentence and therefore carries the tag SENTENCE-START. Similarly, *rain* is the last word in both the sentence and the direct speech. This fact is reflected by one of its tags, DIRECT-QUOTE-ENDING-SENTENCE. A longer example, more representative of the complexity of the problems in the database, is given in Table 6.3. All the problems represented in CAPIT's knowledge base are listed in Appendix A.

The	SENTENCE-START,NO-PUNC
driver	DEFAULT
said,	WORD-PRECEDING-DIRECT-QUOTE, L-CASE,ONE-PUNC
"It	DIRECT-QUOTE-START,ONE-PUNC
will	DEFAULT
rain."	DIRECT-QUOTE-ENDING-SENTENCE, L-CASE,TWO-PUNC

Table 6.2. Problem representation for *The driver said, "It will rain."*

There's	SENTENCE-START,IS-CONTRACTION, ONE-PUNC
a	DEFAULT
bee	DEFAULT
buzzing	DEFAULT
past	DEFAULT
me.	SENTENCE-END,ONE-PUNC,L-CASE
It's	SENTENCE-START,IS-CONTRACTION, ONE-PUNC
taking	DEFAULT
its	ITS-POSSESSIVE-PRONOUN,NO-PUNC, L-CASE
honey	DEFAULT
back	DEFAULT
to	DEFAULT
its	ITS-POSSESSIVE-PRONOUN,NO-PUNC, L-CASE
hive.	SENTENCE-END,ONE-PUNC,L-CASE
I	SENTENCE-START,NO-PUNC
hope	DEFAULT
it	DEFAULT
knows	DEFAULT
its	ITS-POSSESSIVE-PRONOUN,NO-PUNC, L-CASE
way	DEFAULT
home.	SENTENCE-END,ONE-PUNC,L-CASE

Table 6.3. Representation of a more complex problem.

6.3 Constraint Representation

Constraints in CAPIT are the representation used for modelling domain knowledge and also short-term knowledge about the student. Constraints specify the correct and acceptable patterns of capitalisation and punctuation, and are also the basis for determining what errors the student made after each attempt. Longer-term student modelling is achieved using a Bayesian network, into which the short-term knowledge (i.e. which constraints were satisfied or violated on the last attempt) are integrated by the processes of instantiation (see Section 6.5) and on-line learning (see Section 6.8).

Following the model proposed by Ohlsson (1994), each constraint is defined as a tuple <C_r, C_s> where C_r is the *relevance condition* and C_s is the *satisfaction condition*. If a constraint is relevant to a solution, then it must also be satisfied or the constraint is *violated*. CAPIT has additional short explanatory feedback messages for each constraint that may be displayed whenever the constraint is violated. The feedback messages give the student a clue as to how to satisfy the constraint. Some constraints have more detailed explanations that can be accessed by clicking the *Why?* button when the feedback message is being displayed.

CAPIT implements constraints based on regular expressions. In Section 6.3.1, regular expressions are introduced and defined. In Section 6.3.2, a formal definition of the constraints used in CAPIT is given. Finally, in Section 6.3.3, some examples of constraints and their application are provided.

6.3.1 Regular Expressions

A regular expression is a standard, highly expressive, and compact representation for patterns occurring in a string of symbols. They are a common feature of most operating systems and many programming languages. Briefly, a regular expression is said to *match* a string if the pattern it represents occurs in the string.

Regular expressions are composed of both literal characters and special characters and character sequences. For example, in the regular expression `^the`, the character ^ is a special character denoting the start of the string and the characters `the` are literals. Such a regular expression would match strings whose initial three characters are `the`, such as `these` and `therefore`, but not `those`. Another special character is `$`, denoting the end of the string. An expression such as `ch$` would, therefore, match `beach` but not `beaches`.

Logical operations are possible within a regular expression. Disjunctions are denoted by the special character `|`. For example, the regular expression `(^apple$)|(^orange$)` will match either the string `apple` or the string `orange`. Conjunctions are defined simply by concatenating patterns together. For example, the expression `^a.*e$` matches any string that starts with letter `a` *and* ends with letter `e`, such as `apple` or `ahoy there`. The special character `.` denotes any literal character, and `*` denotes zero or more repetitions of the previous character. The special character concatenation `.*` therefore denotes a string containing any or no characters.

Other special characters particularly useful in CAPIT's knowledge base include `+`, which is similar to `*` but denotes at least one repetition of the previous character (rather than zero or more repetitions), and `?`, which denotes only one or zero repetitions. These special characters can be applied to entire patterns as well. For example, `(happy)*` will match `happyhappy`.

A short-hand method of representing a complex disjunction is to use square brackets to define a positive or negative character set. Character sets are denoted by square brackets. For example, the regular expression `[abc]` matches any string containing one of the characters in the set, such as `plain` (matched by the `a`) or `back` (matched by the `b` and the `c`). A negative character set matches any string containing characters *not* in the set. Negative character sets are denoted by the `^` symbol within the square brackets. For example, `[^abc]` matches the `s` in `scab`, but fails to match `aabbcc` at all. Character sets can also be shortened using ranges. A range is denoted by the dash symbol (`-`) within the square brackets of a character set. For example, `[a-g]` is equivalent to the character set `[abcdefg]`.

For the sake of completeness, all the special characters and sequences available in the version of the regular expression language used in CAPIT are defined in Appendix B.

6.3.2 Constraint Definition

A constraint in CAPIT is formally defined as follows. The relevance condition, C_r, is a disjunction of one or more tags. Recall that a tag is a symbol attached to a word in a problem. Any word may have one or more tags. The function of the tags is to define the semantic and/or grammatical class to the word, to the degree required to successfully punctuate and capitalise it. The tags, in turn, determine which constraints are relevant to the word, and therefore the problem. For example, the first word *The* in the sentence depicted in Table 6.2 has the tag `SENTENCE-START` associated with it, which means the problem will be relevant to any constraint whose C_r includes `SENTENCE-START`. Consider constraint C_3 (defined in Appendix C), which determines whether or not there are too many capital letters in a word. This constraint is relevant to words whose tags include SENTENCE-START, NAME-OF-PERSON, NAME-OF-PLACE, or DIRECT-QUOTE-START. The C_r is therefore SENTENCE-START|NAME-OF-PERSON|NAME-OF-PLACE|DIRECT-QUOTE-START, which would match the problem depicted in Table 6.2 twice, via the words *The* and *It*. Note that a constraint can be relevant to a problem more than once, if its C_r matches more than one word in the problem.

The satisfaction condition of a constraint, C_s, is defined as a modified regular expression. The C_s differs from a standard regular expression in that it may contain a function %SYMBOLSET% that is evaluated at run-time. This function simply returns a string containing all and only the punctuation symbols that the system currently lets the student solve problems with. In CAPIT, only four punctuation symbols are permitted: the comma, the period, the apostrophe, and the quotation mark. The modified regular expression [%SYMBOLSET%] is therefore converted at run-time to the standard regular expression [\.,'?] (the escape character \ is necessary because a period is a special character in a regular expression by default, so it must be escaped in order to refer to an actual period).

The satisfaction condition is matched to each word in the problem that the relevance condition matches. Because the student may have added punctuation marks to the solution, the word includes all characters between the white-space characters preceding and succeeding the word. In other words, the correctly punctuated sentence *The driver said, "It will rain."* would be broken up into words by taking each contiguous non-whitespace sequence of characters, as it has been in Table 6.2. The last word of a correct solution would therefore be *rain."* rather than just *rain*. This entire sequence of characters, including letters and punctuation marks, are then matched to the satisfaction condition.

The constraint's C_s will be matched to more than one word if the C_r matched more than one word. In this case, the C_s might match one word but fail to match another word. The rule for resolving this is simple: if the constraint fails to match *any* of the relevant words, then the constraint is violated. If it matches *all* of the relevant words, then it is satisfied.

6.3.3 Examples

This section gives examples of some of the constraints in CAPIT's knowledge base and shows how they are matched to student solutions. All the constraints are listed in Appendix C.

In general, the constraints fall into two categories. *General constraints* are relevant to many different types of word and encapsulate "common sense" knowledge, such as the rule that words starting with a capital letter (such as a person's name) do not need capitals at any other positions in the word (Constraint C_3), the rules that some words do not need to capitalised and punctuated at all (Constraints C_1 and C_2), and the fact the some words require two and only two punctuation marks (Constraint C_{22}).

On the other hand, the bulk of the rules form the set of *specific constraints.* Specific constraints govern the correct punctuation and capitalisation of specific constructs, such as the capitalisation of sentences (Constraint C_4), the correct punctuation of an *is* contraction (Constraint C_9), the fact that *its* when used as a possessive pronoun does not require an apostrophe (Constraint C_{15}), and the correct punctuation of direct speech (Constraints C_{17}-C_{21} and C_{23}-C_{25}).

Table 6.4 depicts four examples of constraints. The first constraint is general, and the remaining three are specific constraints of varying complexities.

	Cr	**Cs**	**Msg**
C_1	`{DEFAULT|L-CASE}`	`^[a-z0-9%SYMBOLSET%]*$`	This word doesn't need any capital letters!
C_5	`{NAME-OF-PERSON}`	`^[%SYMBOLSET%]*[A-Z0-9]`	Each word in a person's name should start with a capital!
C_{15}	`{ITS-POSSESSIVE-PRONOUN}`	`[^']s$`	No apostrophe is required in *its*!
C_{23}	`{DIRECT-QUOTE-ENDING-SENTENCE}`	`[^%SYMBOLSET%]+((\.+"+)|"+|\.+)?$`	The full stop should be within the quotation marks!

Table 6.4. Four constraints from the knowledge base.

Constraint C_1 is a simple constraint defining words that do not need any capital letters at all. The relevance condition of C_1 is `DEFAULT|L-CASE`, which means that the constraint will be relevant to any words in the solution with either the tag `DEFAULT` or the tag `L-CASE`. Recall from Section 6.2 that `DEFAULT` means that the word needs no punctuation or capital letters. `L-CASE` is like `DEFAULT` but simply means that only capital letters are not required; it has no bearing on whether or not punctuation marks are necessary for the word. The satisfaction condition is the regular expression `^[a-z0-9%SYMBOLSET%]*$`, which matches any string consisting of zero or more lower case letters, numbers, or punctuation symbols.

The correct capitalisation of a person's name is defined by constraint C_5. In problems featuring the name of a person, the words in the person's name are identified by the tag `NAME-OF-PERSON`, which matches the relevance condition of C_5. The satisfaction condition for this constraint is `^[%SYMBOLSET%]*[A-Z0-9]`, an expression matched by any word whose first alphanumeric character is either an upper case letter in the range A-Z, or a number in the range 0-9. The constraint ignores any punctuation marks preceding the first letter or number of the word, as specified by the `^[%SYMBOLSET%]*` portion of the expression (which matches zero or more punctuation symbols at the start of the word). It is assumed that if there are any punctuation symbols preceding the first alphanumeric character (such as a quotation mark), then other constraints will handle them.

Constraint C_{15} is specific to a particular word, namely *its* when used as a possessive pronoun. Every occurrence of *its* as a possessive pronoun in any problem must have the tag `ITS-POSSESSIVE-PRONOUN` associated with it, which will make constraint C_{15} relevant. The constraint is satisfied only when the regular expression `[^']s$` matches. That is, if the student incorrectly punctuates *its* to *it's*, then the constraint will be violated. Furthormore, it will only be violated under this condition because the negative character set `[^']` matches any character that is not an apostrophe.

The final example is constraint C_{23}, which partly determines the correct punctuation of direct speech. This constraint is relevant to words in a problem with the tag `DIRECT-QUOTE-ENDING-SENTENCE`. This tag should be attached to any word that is both the last word in a direct quotation, and ends a sentence. In other words, in the example *The driver said, “It will rain.”*, the word *rain* carries the tag DIRECT-QUOTE-ENDING-SENTENCE. There are several other constraints relevant to this tag, and therefore relevant to *rain*, namely constraints C_{20} and C_{21}. Constraint C_{20} matches the word if it is followed by a period, and C_{21} matches if it is followed by a quotation mark. Both are necessary to correctly punctuate the last word of both a direct quote and a sentence. Constraint C_{23}, however, specifies the correct order of the punctuation symbols: the period should precede the quotation mark (and therefore be enclosed in the direct speech). This constraint can only be violated if *rain* is followed by both the period and the quotation mark, in that order. If the order is reversed, the constraint will be violated. In all other circumstances (e.g. where the word is succeeded only by a period and not a quotation mark), the constraint will be satisfied. This is because the other two constraints, C_{20} and C_{21}, are designed to deal with these situations.

In a more exact form, the satisfaction condition of C_{23} is defined as [^%SYMBOLSET%]+((\.+"+)|"+|\.+)?$. Note that this regular expression has no start-of-string marker (^), but it does have the end-of-string marker ($). Therefore, the expression matches patterns occurring only at the end of a string. The [^%SYMBOLSET%]+ portion of the expression matches at least the last character in the word that is not a punctuation mark. So, for example, in *The driver said, “It will rain.”*, the word *rain* is matched by this portion of the expression, but the period and the quotation mark following it are not. The rest of the expression specifies the correct patterns of punctuation that can follow the *n* in *rain*. Recall that the symbol | is a simple logical disjunction, and \. is a special means of referring to a period character in a regular expression. The portion of the expression ((\.+"+)|"+|\.+)? therefore matches one or more periods followed by one or more quotation marks, or one or more quotation marks, or one or more periods. It also matches the situation where no periods or quotation marks succeed *rain*, as indicated by the ? operator (indicating one or zero occurrences of the preceding pattern). In such a case, constraint C_{23} would be satisfied but constraints C_{20} and C_{21} would generate errors. The only case where this constraint fails to be satisfied, however, is when there is one or more quotation marks followed by one or more periods. In other words, the constraint is violated when the order of the punctuation marks is incorrect. Note also that the word *rain* in Table 6.2 is associated with the tag TWO-PUNC, which in turn is relevant to constraint C_{22}. This constraint limits the total number of punctuation marks attached to the word to two. Effectively, therefore, the only punctuation of *rain* that will satisfy all the relevant constraints is to succeed the word with a period followed by a quotation mark. This is the correct way to punctuate the word.

It is worth mentioning some characteristics of this system of constraints. Frequently, the same error will violate many constraints. For example, as previously discussed, incorrectly punctuating *rain* in the example problem depicted in Table 6.2 may violate any combination of the constraints C_{20}, C_{21}, C_{22} or C_{23}. Similarly, when *its* as a possessive pronoun appears in a problem, it has the tags ITS-POSSESSIVE-PRONOUN and NO-PUNC associated with it. This leads to both the specific constraint C_{15}, and the general constraint C_2, being relevant to the word. Incorrectly punctuating the possessive pronoun *its* to *it's* will therefore violate both constraints. Making other errors, e.g. *i'ts*, will violate only the general constraint C_2.

Another characteristic of this system of constraints is that some constraints may be very similar, differing only in their relevance condition and feedback message. For example, constraints C_5 and C_6 are both violated when the first letter of a word is lower case, and therefore they both have the same satisfaction condition. However, C_5 is relevant to words forming the name of a person (with tag NAME-OF-PERSON) whilst C_6 is relevant to words forming the name of a place (with tag NAME-OF-PLACE). This redundancy is necessary for two reasons. Firstly, it allows more targeted instruction on errors. If C_5 and C_6 were combined into a single constraint, then a single more generic error message would have to cover cases where both place names and people names are not capitalised. This may be too abstract for teaching purposes, especially where the target user population comprises children. The second reason for the redundancy is to maintain the semantic richness of the constraint knowledge base. Although it happens that C_5 and C_6 have the same satisfaction conditions, other constraints may be relevant only to words with the NAME-OF-PERSON tag but not the NAME-OF-PLACE tag, and vice-versa.

6.4 Worked Example: Analysing A Solution Attempt

This section briefly demonstrates how CAPIT determines which constraints are satisfied and violated when a problem attempt is submitted. Table 6.5 depicts a problem from CAPIT's problem database. When presented to the student, the initial problem text is:

> *the teachers chalk marker and overheads were stolen the principles filing cabinet telephone and typewriter are also missing*

The student's task is to add apostrophes of possession, commas separating list items, and capitals and periods to signify the start and end of sentences.

The	SENTENCE-START, NO-PUNC
teacher's	POSSESSIVE-NOUN-NOT-ENDING-IN-S, ONE-PUNC, L-CASE
chalk,	INTERMEDIATE-ITEM-IN-LIST, ONE-PUNC, L-CASE
marker	INTERMEDIATE-ITEM-IN-LIST-PRECEDING-CONJUNCTION, NO-PUNC, L-CASE
and	DEFAULT
overheads	FINAL-ITEM-IN-LIST, NO-PUNC, L-CASE
were	DEFAULT
stolen.	SENTENCE-END, ONE-PUNC, L-CASE
The	SENTENCE-START, NO-PUNC
principal's	POSSESSIVE-NOUN-NOT-ENDING-IN-S, ONE-PUNC, L-CASE
filing	DEFAULT
cabinet,	INTERMEDIATE-ITEM-IN-LIST, ONE-PUNC, L-CASE
telephone	INTERMEDIATE-ITEM-IN-LIST-PRECEDING-CONJUNCTION, NO-PUNC, L-CASE
and	DEFAULT
typewriter	FINAL-ITEM-IN-LIST, NO-PUNC, L-CASE
are	DEFAULT
also	DEFAULT
missing.	SENTENCE-END, L-CASE, ONE-PUNC

Table 6.5. An example from CAPIT's problem database.

Let us suppose that the student makes some of the corrections, but fails to make all of the corrections. Furthermore, he or she incorrectly capitalises and punctuates some words that did not need to be changed. Here is just such an attempt:

> *The teacher's chalk marker and Overheads were stolen. the principles filing cabinet, telephone, and typewriter are also missing.*

In this example, the apostrophe is missing from *principles* and the comma separating list items is missing after *chalk*. An extraneous comma has also been added after *telephone*. This comma is unnecessary because the single-word list items *telephone* and *typewriter* are already separated by the conjunction *and*. The word *Overheads* is unnecessarily capitalised, but the first word of the second sentence, *the*, has not been capitalised when it should have been.

The system first determines the relevant constraints. This is achieved by matching the tags of each word against the relevance condition of each constraint. In this case, specific constraints C_8, governing the punctuation of possessive singular nouns, and C_{11}, C_{12} and C_{16}, specifying the correct punctuation of items in a noun list, are relevant. Constraints C_4 and C_7, dealing with the punctuation of sentences, are also relevant. There are some general constraints that are relevant as well; namely, C_1, C_2, C_3, and C_{14}.

The next step is to match the satisfaction condition of each relevant constraint against each of the relevant words in the problem to determine if the constraint has been violated or satisfied. Constraint C_{11} is relevant to *chalk*, because *chalk* is the second item is a list of nouns, and therefore has the tag `INTERMEDIATE-ITEM-IN-LIST`. However, the constraint is violated because the satisfaction condition of C_{11} requires the word to be followed immediately by a comma. General constraint C_1 is also violated, because the incorrectly capitalised word *Overheads* has tag `L-CASE`, which is relevant to C_1 and specifies that the word must contain no uppercase letters. Failing to capitalise the first word in the second sentence violates constraint C_4, because C_4 requires words with tag `SENTENCE-START` to begin with a capital letter. The missing apostrophe in *principles* violates constraint C_8, because *principles* has the tag `POSSESSIVE-NOUN-NOT-ENDING-IN-S` which makes the constraint relevant and therefore requires the word to have an apostrophe. Constraint C_{16} is violated by the addition of a comma following *telephone*, because *telephone* is an `INTERMEDIATE-ITEM-IN-LIST-PRECEDING-CONJUNCTION`, and therefore does not require a comma. The incorrect punctuation of *telephone* also violates constraint C_2 because *telephone* has tag `NO-PUNC` which, being relevant to C_2, means that no punctuation marks at all are required for the word.

In total, six of the ten relevant constraints are violated. The current version of the system displays only one feedback message per attempt, so the system now has to choose which of the six feedback messages to display. In this thesis, that problem is solved using the Bayesian network student model and decision theory (see subsequent sections of this chapter for more information on this). After the feedback message is displayed, the student can attempt to correct the solution.

Note that some of the constraints are relevant more than once, and may match some words but not others. For example, *teacher's* was correctly punctuated and therefore matches the satisfaction condition C_8. However, *principles* is also relevant to C_8 but fails to match the constraint's satisfaction condition. The rule in this case is to consider the constraint violated if it fails to match at least once.

6.5 Data Collection

Initial data for Step 1 of the decision-theoretic PAS strategy development was acquired from a preliminary study of CAPIT at Westburn School, Christchurch, New Zealand in March 2000 (Mayo et al., 2000). A version of CAPIT was used in which problems and error messages were selected randomly. The problems came from the pool of all unsolved problems, and the error message was selected from the set of constraints violated on the current attempt (recall that there is one error message per constraint). The preliminary study consisted of four 30-45 minute sessions. Details of each problem attempt and error message displayed were logged. Subsequent analysis revealed the following averages.

Each student made 89 attempts at 28 different problems, on average. 21 of the problems were eventually solved, and 7 abandoned. Students violated an average of 181 constraints during the sessions, of which feedback was given on 68.

A total of 3300 records of the form <*State, Action, Outcome*> were acquired during this step. In this particular implementation, *State* is defined as a record of the outcome of the last attempt at each constraint (which may include attempts at previous problems, if for example, a constraint was relevant to the last problem but is not relevant to the current problem), *Action* is the problem that was selected randomly, and *Outcome* is a record of the constraints that were violated and satisfied following the problem attempt.

An example of a record is given in Figure 6.3. Both the *State* and the *Outcome* consist of 25 values, one for each constraint. Prior to the problem being presented, the student had satisfied constraints 1 and 25 on the last attempt, violated constraint 2, violated constraint 24 but received feedback on it, and had not even attempted constraint 3 before. The system then selected problem 80, and after the student clicked *Submit*, the satisfied constraints included 1,3, 24 and 25, but constraint 2 was violated.

	$\mathbf{C_1}$	$\mathbf{C_2}$	$\mathbf{C_3}$	**...........**	$\mathbf{C_{24}}$	$\mathbf{C_{25}}$
State	`S`	`V`	`NR`	....	`VFB`	`S`
Action	`Problem_80` (*The teacher said, "The crab lives in its shell."*)					
Outcome	`S`	`V`	`S`	....	`S`	`S`

Fig. 6.3. Example of a single record in the dataset collected during the first evaluation study.

6.6 Model Selection

The data acquired from the preliminary study was used to generate the best Bayesian network for long-term student modelling. The selection criterion was the ability of the network to predict student performance on constraints. An issue at this point was whether to use a model in which the constraints were independent of each other, as in Reye's model, or whether to allow (more realistically) any dependencies between constraints to be learned from the data. This decision is quite significant because a model in which constraints are assumed to be independent can be formulated with only four variables, whereas a model in which any dependencies between constraints are allowed is much more complex and in this system must consist of at least twice the number of variables as there are constraints. Figures 6.4, 6.5 and 6.6 illustrate the competing "small" and "large" specifications. In each diagram, L_i represents the outcome of the last attempt at the *i*th constraint, and can take values `S` (satisfied), `V` (violated), `VFB` (violated with feedback), or `NR` (has not been relevant before). N_i is the predicted outcome of the next attempt, whose values can take one of the values {`S, V, NR`}. Note that neither network explicitly models unobserved student states. In both cases, the output from the network is a set of posterior probability distributions over the N_i variables given the values for the L_i variables. The large networks (Figures 6.5 and 6.6) only need to be evaluated once per problem, whereas the small network (Figure 6.4) needs to be evaluated once per relevant constraint per problem.

Note that in all the large networks, the *L* layer nodes are always root (parentless) nodes. This was by design rather than as a result of the learning algorithm. The reason for this is two-fold. Firstly, the *L* nodes precede the *N* nodes temporally; therefore the arcs should always be directed from the *L* nodes to the *N* nodes. Secondly, because the *L* nodes are always known with certainty (since they represent the student's interaction history to date), any arcs between *L* nodes can be effectively ignored. Arcs are only used for uncertain reasoning, but if both cause proposition and effect proposition at either end of the arc is certain, then the arc becomes redundant in the network.

In each diagram, a bold arc indicates that the arc was added to the network prior to the induction of the rest of the network from data, and is fixed during the learning process. Figure 6.4 depicts the structure of the *Small* network. In this example, the network is predicting the outcome of the next attempt at previously violated constraint 11, which is relevant to current problem 35. All the arcs in Figure 6.4 are fixed apriori. Figure 6.5, on the pther hand, depicts the *Large* Bayesian network. As depicted, the student has previously satisfied constraints 1 and 2, violated constraints 3 and 25 (receiving feedback on 3), and has not yet attempted constraint 4. The network is currently configured to predict this student's performance on a problem whose relevant constraints include 2, 3 and 25. In Figure 6.5, none of the arcs are determined beforehand, but in Figure 6.6, arcs from L_i to N_i are initially added.

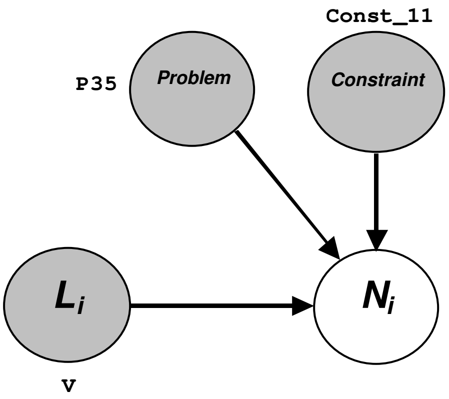


Fig. 6.4. The structure of the small Bayesian network for predicting the outcome of the next attempt at the *i*th constraint.

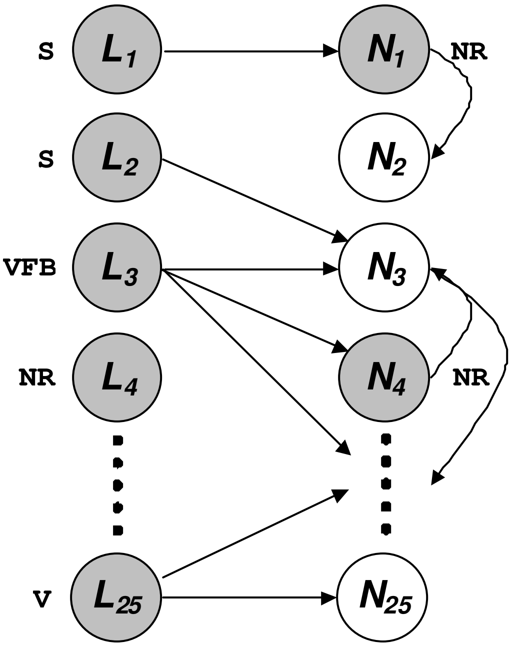


Fig. 6.5. The structure of the large Bayesian network specification after learning.

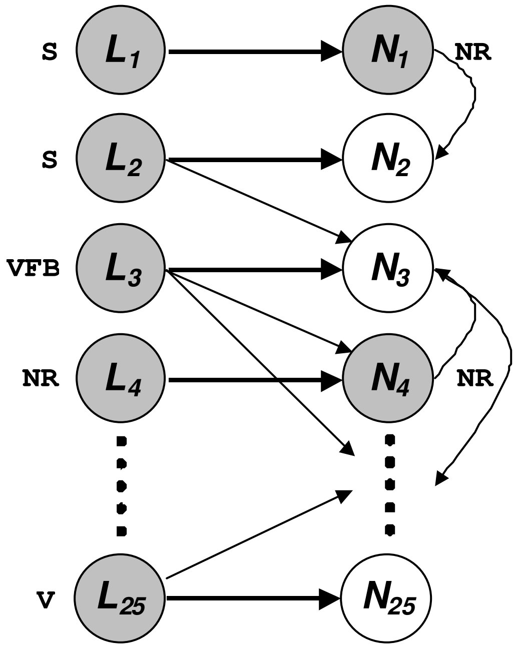


Fig. 6.6. The same network as depicted in Figure 6.5, but with fixed arcs from each node L_i to N_i added to the network prior to learning.

A number of variants of the large network were considered. In each case, the algorithm proposed by Cheng et al. (1998) for structural learning by mutual information maximisation (described in Section 2.4.2) was utilised to learn a Bayesian network structure from the data collected in Step 1. The conditional probabilities were estimated using the Dirichlet priors approach (Heckerman, 1999) which was outlined Section 2.4.1. An important component of the structural learning algorithm is the minimum threshold ε. This essentially determines the minimum amount of mutual information required between two variables before an arc can connect them. For these experiments, ε values of 4, 6 and 10 were selected (initial experiments showed that a threshold below 4 resulted in a network far too complex for on-line evaluation).

Another parameter that we wanted to investigate was the addition of prior knowledge: does it enhance predictive performance? The "obvious" prior knowledge to add is an arc from L_i to N_i, for each constraint i, indicating that at the very least, the outcome of the next attempt at a constraint is partly dependent on the outcome of the previous attempt. We thus formulated six specifications for large networks: *Large(4)*, *Large(6)* and *Large(10)* being the specifications without prior knowledge (see Figure 6.5), and *PLarge(4)*, *PLarge(6)* and *PLarge(10)* being specifications with prior knowledge (see Figure 6.6). For each of the large specifications, the L_i nodes were fixed as root nodes, to reflect the fact that they come before the N_i nodes in temporal order. Thus, there are two types of arc that can be learnt from the data for the large networks: arcs from the *L* layer to the *N* layer, and arcs within the *N* layer.

The data was then divided into training and test datasets. Approximately 20% of the records were selected randomly into the test dataset. The remaining 80% were kept in the training set, and used to train one large network for each of the six specifications. A simpler network equivalent to Figure 6.4 (*Small*) was also trained from this data, although in this case the structure was specified and only the conditional probability $P(N_i|L_i,Problem,Constraint)$ had to be learned. This entire process of training was repeated three times for three different random training/test dataset divisions. The total number of different networks that were generated, therefore, was 21.

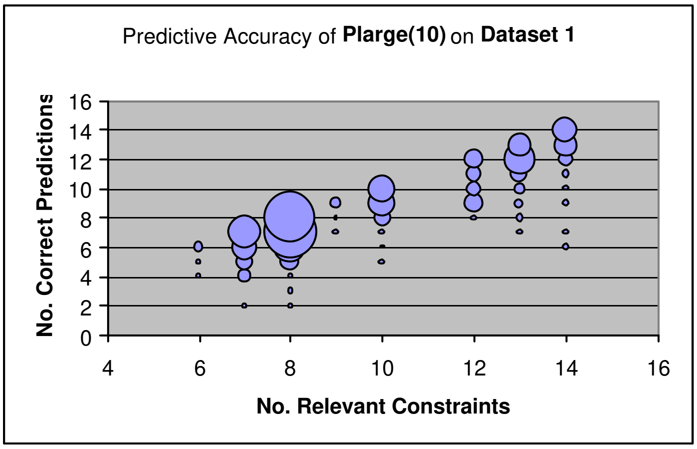
The first question to answer was whether or not the larger networks were better predictors of student performance than the small ones on the test data. Each of the six large networks generated from the *i*th training set was compared to the *Small* network generated from the *i*th training set in the following way. For each problem attempt in the *i*th test set, the large and small networks were given the values for $L_1..L_{25}$. The large networks had their $P_1..P_{25}$ nodes set to NR for each constraint not relevant to the attempt's problem. The values of the remaining node from $P_1..P_{25}$ were then predicted. For the large networks, this required one evaluation of the network, and for *Small*, one evaluation per relevant constraint was necessary. The standard junction tree algorithm for Bayesian network inference was used (Lauritzen & Spiegelhalter, 1988). Then, for each P_i representing a relevant constraint, the predicted value of P_i was compared to the actual value of P_i. The total number of correct predictions was counted. A correct prediction was deemed to occur if the predicted outcome with maximum probability matched the actual outcome. To clarify further, the output of each comparison was essentially a table of tuples of the form <*i*, L(*i*), S(*i*), T(*i*)>, where *i*=1..*n* is the attempt (*n* is the number of attempts in the test set), L(*i*) is the number of correct predictions given by the large network, S(*i*) is the number of correct predictions given by the small network, and T(*i*) is the maximum number of correct predictions (simply the total number of relevant constraints on that attempt).

The coefficient of determination (r^2) was calculated for each network by taking the number of correctly predicted constraints as a function of the number of relevant constraints, for training/testing dataset and each specification. The results are summarised in Table 6.6.

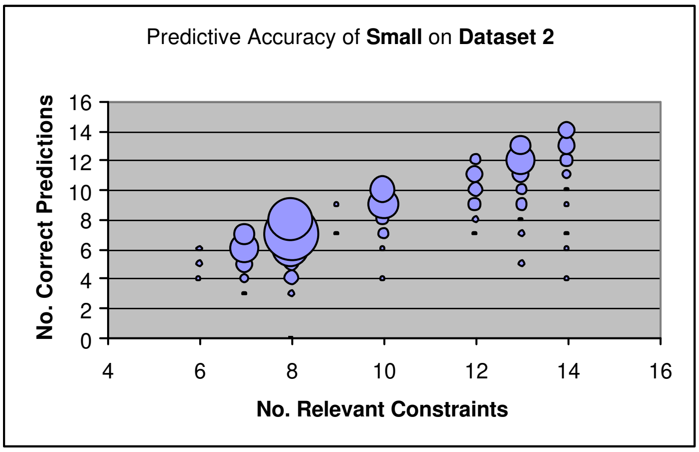
	TestData$_1$ (n=639)	**TestData$_2$ (n=608)**	**TestData$_3$ (n=686)**
PLarge(10)	0.7649	0.7434	0.7638
PLarge(6)	0.7562	0.7385	0.7615
PLarge(4)	0.7101	0.7208	0.7417
Large(10)	0.7446	0.7464	0.7495
Large(6)	0.749	0.7347	0.7511
Large(4)	0.7117	0.7236	0.7387
Small	0.7312	0.7086	0.7314

Table 6.6. Coefficients of determination.

The r^2 values are all within a narrow margin, with the large networks performing slightly better than *Small* most of the time. Visual inspection of the results reinforces this view. Figure 6.7 compares bubble graphs of the network/dataset combination with the highest r^2 against the network/dataset combination with the lowest r^2. The size of the bubble is proportional to the frequency with which *y* correct predictions were made when there were *x* relevant constraints.



(a) $r^2=0.7649$



(b) r^2=*0.7086*

Fig. 6.7. Bubble graphs comparing the (a) accuracy of the network and dataset resulting in the highest r^2 and (b) the network and dataset resulting in the lowest r^2.

Next, we tested to see if the large networks were statistically significantly better predictors of student performance than the small networks. Note that for each of the 18 comparisons, the number of correct predictions made by the small and large networks are paired. That is, for each attempt i=1..n in each of the 18 tests, both an S(i) and a L(i) were generated by the small and large networks respectively, both of which can be considered stochastic functions of i. Therefore, the samples are pair-wise dependent. A paired-difference experiment (McClave & Benson, 1991, pp. 421-7) was used to test for significant differences.

The results of the statistical significance tests are given in Table 6.7. In this table, H_0 the hypothesis that there is no difference in the mean number of correct predictions made by both networks. A positive t value indicates that the large network is better than *Small*. The rejection region for all the datasets is approximately ±2.58 for 99% confidence, and ±1.96 for 95% confidence.

Table 6.7 shows that the *PLarge(10)*, *PLarge(6)* and *Large(6)* specifications all produce Bayesian networks that are statistically significantly better predictors of student performance than the networks produced by the *Small* specification. The other specifications each had at least one comparison where no statistically significant difference was found (indicated by "Accept H_0"). For these tests, a high t value indicates greater significance. For all the networks, the outcomes in the second test set were much more difficult to predict than those of the first and third sets, resulting in lower t values. This is obviously due to the random nature by which the datasets were generated. The exception to this is *Large(10)*, which performed barely acceptably on the second test set but not the first and third. From these results, we were able to eliminate *Small* as a worthwhile specification.

	TestData$_1$ (n=639)	**TestData$_2$ (n=608)**	**TestData$_3$ (n=686)**
	Small	**Small**	**Small**
PLarge(10)	t=5.75, α=0.01	t=3.95, α=0.01	t=5.46, α=0.01
PLarge(6)	t=5.17, α=0.01	t=3.30, α=0.01	t=5.06, α=0.01
PLarge(4)	Accept H_0	Accept H_0	t=2.1, α=0.05
Large(10)	Accept H_0	t=1.98, α=0.05	Accept H_0
Large(6)	t=4.01, α=0.01	t=2.78, α=0.01	t=3.93, α=0.01
Large(4)	Accept H_0	Accept H_0	t=2.05, α=0.05

Table 6.7. Results of two-tailed paired difference experiments comparing each large network against *Small*.

The next task was to determine the most accurate large network. In particular, does the selection of the minimal threshold or the addition of prior knowledge result in improved performance? For this analysis, the three large networks with the highest average t values (*PLarge(10)*, *PLarge(6)* and *Large(6)*) were compared. H_0 was once again the hypothesis tha t there is no difference in the mean number of correct predictions made by both networks, with rejection region for all datasets of approximately ±2.58 for 99% confidence and ±1.96 for 95% confidence. No statistically significant difference was found between *PLarge(10)* and *PLarge(6)* on any of the training/testing dataset divisions. However, significant differences were found between *PLarge(10)* and *Large(6)* as Table 6.8 shows. This implies that prior knowledge does enhance the predictive performance.

	TestData$_1$ (n=639)	**TestData$_2$ (n=608)**	**TestData$_3$ (n=686)**
	Large(6)	**Large(6)**	**Large(6)**
Plarge(10)	t=3.13, α=0.01	Accept H_0	t=2.08, α=0.05

Table 6.8. Results of two-tailed paired difference experiments comparing *PLarge(10)* against *Large(6)*.

To conclude Step 2, *PLarge(10)* was selected as the best specification with which to proceed, because the average *t* value in Table 6.7 for this specification was greater than the average *t* value of *PLarge(6)*. The training and testing datasets were combined into a single dataset and a Bayesian network with the *PLarge(10)* specification was learned. One of the desirable features discussed earlier was to take advantage of the unique ability of a Bayesian network to integrate prior knowledge and data; this has been shown to improve predictive accuracy.

6.7 Next Problem Selection

Step 3 is the implementation of decision-theoretic PAS strategies. The key task is to define a utility function *U(X,D)* specific to the task that can be substituted into the expected utility function (Equation 2.28) to yield the task-specific expected utility function. For CAPIT, we were interested in two specific tasks: next problem selection, and error message selection following an attempt in which multiple constraints are violated. Recall that the utility function essentially encodes the system's pedagogical preference for different outcomes.

The value of the next problem $D \in$ {`Problem_1`, ..., `Problem_45`} can be determined by predicting the student's performance on the problem with the Bayesian network. An appropriate problem is one that falls into the *zone of proximal development*, defined by Vigotsky (1978).[1] This principle implies that utility should be maximised for problems where one or two errors are likely (reflecting a challenging problem), but minimised for problems whose outcome

[1] The zone of proximal development is defined in Chapter 4.

is no errors (being too easy) or several errors (being too hard). This utility function is defined in Table 6.9.

X	***U(X,D)***
`No-Errors`	0.0
`1-Error`	1.0
`2-Errors`	1.0
`3+Errors`	0.0

Table 6.9. The utility function for problem selection. Utility is maximised for problems resulting in one or two errors only.

We assume that the cost of all problems is zero. Let us also assume that ξ comprises the student history (i.e. the instantiations of $L_1..L_{25}$) and the instantiations of N_i to `NR` for those constraints not relevant to D. Substituting into the general expected utility function (Equation 2.28) yields the expected utility of problem d:

$$
\begin{aligned}
E[U(X,D)\,|\,\xi] \quad &= P(\texttt{No-Errors}|D,\xi)\ \mathrm{U}(\texttt{No-Errors},D) \\
&\quad + P(\texttt{1-Error}|\,D,\xi)\ \mathrm{U}(\texttt{1-Error},D) \\
&\quad + P(\texttt{2-Errors}|\,D,\xi)\ \mathrm{U}(\texttt{2-Errors},D) \\
&\quad + P(\texttt{3+Errors}|\,D,\xi)\ \mathrm{U}(\texttt{3+Errors},D) - 0 \\
&= P(\texttt{1-Error}|\,D,\xi) + P(\texttt{2-Errors}|\,D,\xi) \qquad (6.1)
\end{aligned}
$$

The equation basically says that the expected utility of a problem is simply the sum of the probabilities that the student will make one or two errors.

Now we need to calculate $P(\texttt{1-Error}|D,\xi)$ and $P(\texttt{2-Errors}|D,\xi)$ from the Bayesian network. This is not straightforward because the predicted outcomes $N_1..N_{25}$ are not necessarily mutually or conditionally independent. In fact, the best way to deal with this computation is to extend the Bayesian network itself at runtime by adding a deterministic function *NumErrors* to the network, whose inputs are the predicted values of the relevant constraints only, and whose possible states are `{No-Errors, 1-Error, 2-Errors, 3+Errors}`. The function simply counts the number of its parents that are violated, but because the parents of *NumErrors* are likely to be uncertain, the uncertainty is transferred to *NumErrors* by the Bayesian network inference algorithm (Lauritzen & Speigelhalter, 1988) in the correct way. For example, suppose there are only two constraints relevant to the problem being considered and, given the student's history, the posterior probabilities of the constraints being violated are 0.5 and 1. Then the value of *NumErrors* will be `1-Error` or `2-Errors`, with a probability of 0.5 in each case.

The addition of *NumErrors* to the example large network is illustrated in Figure 6.8. The probabilities of Equation 2.28 can now be determined by querying the posterior distribution over the *NumErrors* variable.

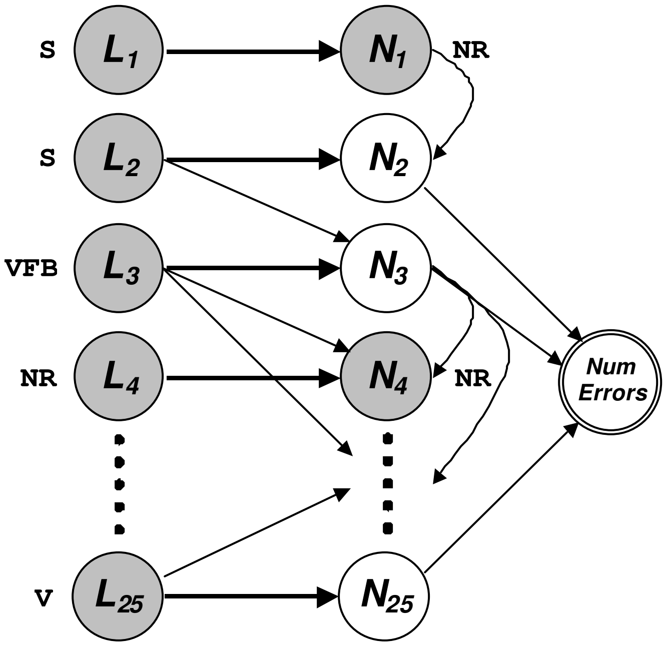


Fig. 6.8. An example of a *Plarge(X)* network with *NumErrors* added as a child of all the N_i nodes representing relevant constraints on the current problem.

6.8 Feedback Message Selection

The strategy for decision-theoretic error message selection is slightly different. In this case, $D \in$ {`FBi` | Constraint *i* was relevant and violated on the last attempt}, where `FBi` is the decision to give feedback on the *i*th constraint. It is assumed that an error message about a constraint can influence the outcome of the next attempt at the constraint, resulting in a satisfaction (desired) or a violation (not desired). Table 6.10 characterises this as a utility function

X	***U(X,D)***
N_i=`V`	0.0
N_i=`S`	1.0

Table 6.10. The utility function for feedback selection.

Because the system gives feedback on only one violated constraint per attempt, the probabilities of these outcomes can be read directly from the network by "pretending" that feedback was given on the *i*th constraint. That is, we instantiate L_i to `VFB` instead of `V` and query N_i to obtain $P(N_i=\mathtt{V}|D, \xi)$ and $P(N_i=\mathtt{S}|D, \xi)$. However, the cost of each feedback message cannot be assumed to be zero, because each constraint will have a different probability of being satisfied without feedback anyway. This probability of satisfaction can be considered the "opportunity cost" of giving feedback on the *i*th constraint, which is therefore defined as:

$$cost(D) = P(N_i=\mathtt{S}|\xi) \tag{6.2}$$

Substituting these values into Equation 2.28 yields the expected utility for feedback:

$$\begin{aligned}&E[U(X,D)|\xi]\\&= \big(P(N_i=\mathtt{S}|D,\xi)U(N_i=\mathtt{S})+P(N_i=\mathtt{V}|D,\xi)U(N_i=\mathtt{V})\big)-P(N_i=\mathtt{S}|\xi)\\&= P(N_i=\mathtt{S}|D,\xi)-P(N_i=\mathtt{S}|\xi)\end{aligned} \tag{6.3}$$

The expected utility of an error message is therefore simply the posterior gain in probability that the constraint will be satisfied, given the feedback message.

6.9 On-line Adaptation of the Student Model

The next step was implementing an on-line learning algorithm so that the Bayesian student model would adapt to the student. Note that online learning is one of the two ways in which the student model is adapted to the student. The first, standard, method of adaptation is to simply instantiate the variables in the network. In other words, the variables $L_1..L_{25}$ are set to known values and the posteriors over $N_1..N_{25}$ are calculated. However, this approach takes account of *only* the outcomes of the most recent attempt at each constraint. On-line learning, on the other-hand, lets us take account of *all* the previous outcomes. This is because the conditional probabilities of the network are incrementally changed after each attempt, and this forms an implicit history of the student's behaviour. Furthermore, the incremental changes can be biased towards the most recent attempts at each constraint so they more accurately reflect the current state of the student. In this section, the particular method of on-line adaptation implemented in CAPIT is described.

It is shown in Section 2.4.1 that that the expected value of $P(X=x_k|\mathbf{PA}(X)=\mathbf{pa}(X))$ is $\frac{\alpha_k}{\alpha}$, where α_k is the frequency with which $X=x_k$ is observed in the data when $\mathbf{PA}(X) = \mathbf{pa}(X)$, and α is the frequency with which $\mathbf{PA}(X) = \mathbf{pa}(X)$ appears.

The problem with this algorithm is that it does not take into account the temporal ordering of the cases, and therefore there is no way to "bias" the conditional probabilities towards most recent cases. This simplification makes it straightforward to convert batch learning algorithms into on-line learning algorithms, as is done by Bauer (1997). However, this is a poor approach for an intelligent tutor to take because the student's state cannot be assumed to be static and unchanging. Dynamically changing the student's state is in fact the very objective of the intelligent tutor. The system therefore needs a way of gradually discarding old data, such as the population student model and old data acquired from the student. However, the effect of the standard approach would be that as α becomes increasingly large, the influence of new cases on the conditional probabilities decreases. To illustrate, CAPIT's population student model was learned from records of approximately 3300 problem attempts. The average student is likely to make only 50-100 problem attempts. Therefore, the standard Dirichlet priors approach would not be expected to adapt the network's parameters to the desired extent. In other words, the initial population data would not be discarded fast enough.

Fortunately, the standard approach can be modified to prefer more recent observations. Our solution is to reduce α to a value such that the effect of new cases becomes significant. Let that value be a_{MAX}. The sufficient statistics can now be replaced by two new statistics, α' and α_k', defined as:

$$\alpha' = a_{MAX},\ \alpha_k' = a_{MAX}(a_k/a) \tag{6.4}$$

The lower the constant a_{MAX}, the more significance new cases will have on the conditional probabilities. In CAPIT, the conditional probabilities are updated after every attempt (so N=1). The update rule (Equation 2.25), therefore, simplifies to:

$$E[P(X=x_k|\mathbf{pa}(X))|\text{One observation of } X=x_k \text{ when } \mathbf{PA}(X)=\mathbf{pa}(X)] = \frac{\alpha_k'+1}{\alpha'+1} \tag{6.5}$$

and

$$E[P(X=x_k|\mathbf{pa}(X))|\,j\ ?\,k, \text{ One observation of } X=x_j \text{ when } \mathbf{PA}(X)=\mathbf{pa}(X)] = \frac{\alpha_k'}{\alpha'+1} \tag{6.6}$$

A value for a_{MAX} was chosen by trials with simulated students. Two students were simulated: a "good" student who got every problem correct, and a "poor" student who made numerous mistakes and frequently abandoned problems. A domain expert analysed the sequence of problems that was selected for each student. It was found that when $a_{MAX} > 5$, the system was not quick enough to present challenging problems to the good student even after several problems were solved correctly in single attempts. This occurred simply because the conditional probabilities did not update fast enough. For the bad student, simple problems were repeatedly selected regardless of the value of a_{MAX}. A value of $a_{MAX} = 5$ was therefore selected.

6.10 Evaluation

Two evaluations were performed. Firstly, an informal evaluation of the system by simulating the behaviour of students were carried out. This was primarily to detect any obvious inconsistencies in the behaviour of the system. The second evaluation was an extensive trial of the system in three classrooms in a New Zealand primary school.

6.10.1 Simulated Student Evaluation

The first step in the evaluation was an informal observation of the behaviour of the decision-theoretic version of CAPIT. The observations were noted during trial runs with both simulated good and bad students.

The system always starts with the easiest problem, which involves merely dividing the text into sentences and inserting capital letters at the start, and periods at the end, of each sentence. For the good student, further problems typically introduced new constraints one at a time until a certain point was reached (probably when the posterior probability of the student satisfying the most common constraints was sufficiently high), after which more difficult problems introducing more than one new constraint (e.g. direct speech problems) were selected. This is similar to a human tutor assessing a good student's capabilities initially with easier problems, before moving more directly to challenging problems.

For the bad student who repeatedly made mistakes and abandoned problems, the tutor appeared to repeatedly select problems from a pool of three-four easier problems; again, a similar strategy to that of a human tutor repeatedly returning to previously abandoned problems that must be mastered before progression to more difficult problems. Note that problems in this system do not have explicit levels – all unsolved problems are available to be selected at any one time. Feedback selection was also observed. In the extreme case of the bad student who repeatedly submitted the same (incorrect) solution with multiple violated constraints, the selection of feedback messages seemed to cycle from the most to the least specific constraints, and back again, with each attempt. Again, there is no explicit rule programmed into the tutor to make it do this. The behaviour is entirely a result of the configuration of the network that was learned from the student population data, and the subsequent adaptation of the model to the student.

6.10.2 Classroom Evaluation

Three classes of 9-10 year olds at Ilam School, Christchurch, New Zealand, participated in a four-week evaluation of CAPIT. The first class (Group A) did not use the tutor at all. The second class (Group B) used the initial version of the tutor with randomised problem and error message selection, and the third class (Group C) used the full version of the tutor with decision-theoretic PAS and the adaptive Bayesian student model. The groups that used the tutor, B and C, had one 45-minute session per week for the duration of the study, and they worked in the same pairs each week. Every interaction between the students and the system was logged. All groups were administered the same pre- and post-tests, and the tests were completed by students in the same pairs as they were put into to use the tutor.

Some significant attributes of the performance of Groups B and C during the evaluation are summarised in Table 6.11. Pairs of students in Group C used CAPIT for approximately 34 minutes more on average than those in Group B, a difference attributable to the teachers. However, Group B actually solved more problems than Group C. This can be explained by the fact that after the evaluation study, it transpired that Group C was actually a less able class than Group B. Initially it had been thought that both classes were of approximately equivalent ability. Additionally, Group C made many more attempts, and asked for more *Why?* explanations, than Group B. The average time per attempt for both groups was approximately 43 seconds. However, despite the additional interaction time, Group C attempted and solved fewer problems than Group B, and abandoned more problems. An interesting discrepancy is the mean number of attempts per solved problem. Group C performed better in this area, perhaps suggesting that the feedback messages were better adapted in their case.

	Group B	Group C
Number of pairs	16	14
Ave. interaction time per pair (mins)	80.9	115
Ave. # attempts	109.7	167.3
Ave. # solved problems per pair	29	22
Ave. # attempted problems per pair	34	30
Ave. # attempts per solved problem	5.8	5.5
Ave. # expl. asked for per pair	10.3	18

Table 6.11. Various averages describing Group B and C's performances on CAPIT.

Further analysis was performed at the level of the individual constraints. Figure 6.9 gives the average number of times each constraint was relevant per user. This reflects the higher number of attempts made by Group C, and makes explicit the constraints that are common to most of the problems, for example, constraint C_4 (a sentence must start with a capital letter) and constraint C_7 (a sentence must end with a period). This table can be compared to Figure 6.10, the frequency with which constraints were violated when relevant. It shows that Group C violated proportionately more constraints that Group B, which corresponds with the averages in Table 6.11. However, it is interesting to note that the constraints defining the correct punctuation of direct speech (constraints C_{17}-C_{25}) were violated proportionately less by Group C. This suggests that the decision-theoretic problem sequencing placed problems involving direct speech (which are more difficult) later in the sequence, after the student had solved some of the easier problems. This strategy allowed the students to focus on learning direct speech punctuation after showing that they had mastered the other constraints.

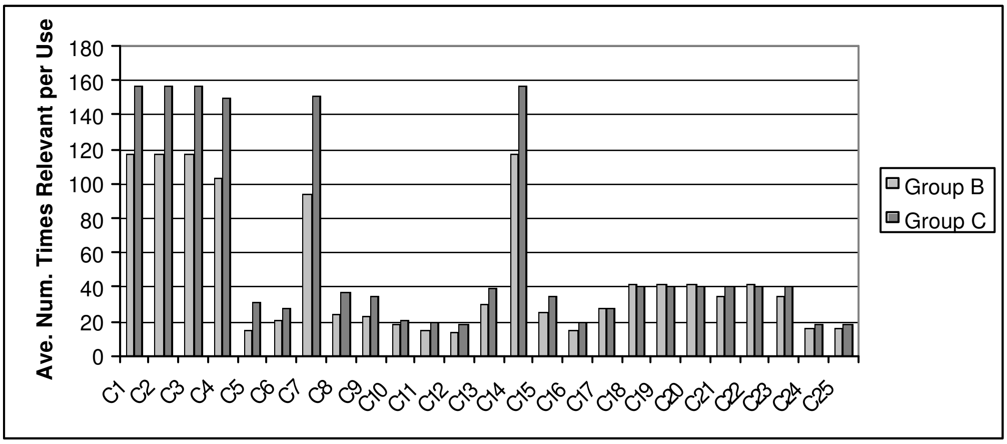


Fig. 6.9. The average frequency per user of constraint relevance to problems.

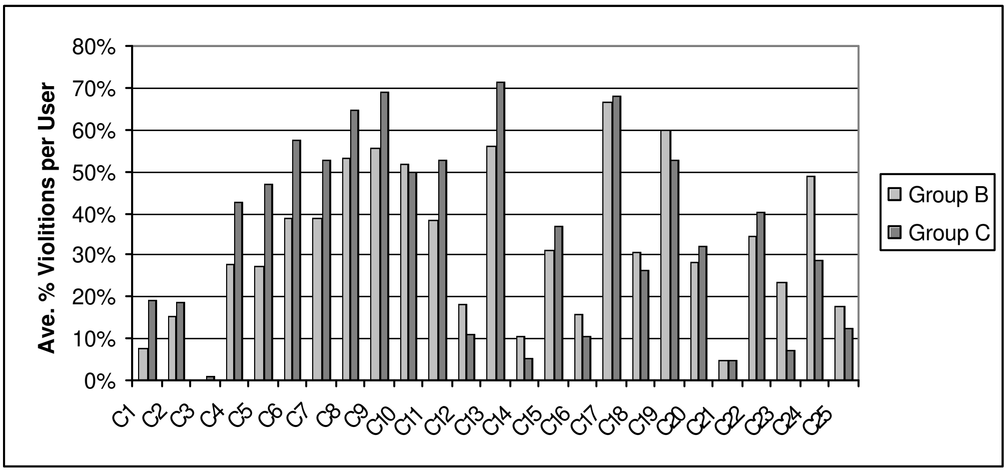


Fig. 6.10. The average proportion per user of violations per relevant constraint.

The pre- and post-tests were comparable (and challenging) and consisted of eight completion exercises similar to those presented by CAPIT, but done manually with pencil-and-paper. Students completed the tests in their assigned pairs. The score for each test was calculated by subtracting the number of punctuation and capitalisation errors from the number of punctuation marks and capital letters required for a perfectly correct solution; it was thus possible for a pre- or post-test to have a negative score. The mean scores and standard deviations (the Y error bars) are shown in Figure 6.11. Both Group B and C show an improvement in mean test scores, although the improvement is more marked for Group C. Group A, the class that did not use the tutor, actually regressed.

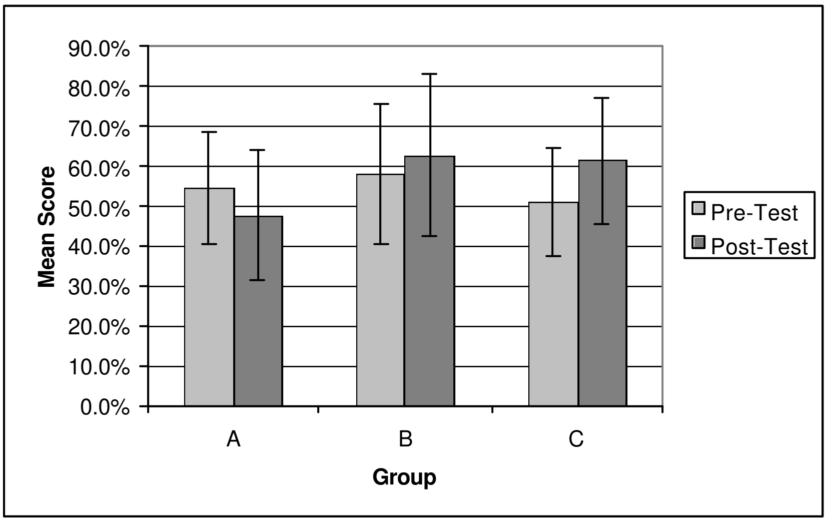


Fig. 6.11. Mean pre- and post-test scores.

Statistical significance tests were performed to compare the individually matched improvements of Groups B and C from pre-test to post-test. Because the same pair of students in each group completed both a pre- and a post-test, a one-tailed paired difference experiment (McClave & Benson, 1991, pp. 421-7) was performed to gauge the significance of the improvement. With H_0 being the proposition that a group did not improve, it was found that Group B improved with 95% confidence (a = 0.05, *t* = 1.86, rejection region ± 1.75) while Group C improved with 99% confidence (a = 0.01, *t* = 3.4, rejection region ± 2.6). The improvement is thus much more significant for Group C, which used the decision-theoretic strategies.

The effect size, which is defined as the difference in the mean gains of the control (Group B) and experimental groups (Group C) divided by the standard deviation of the mean gain of the control group, was also calculated. This measure gives the magnitude of the gain attributable to the normative PAS strategies as opposed to the randomised strategies. The effect size is 0.557, a value that is comparable to the effect size of 0.63 found by Albacete & VanLehn (2000) after a two-hour session with their tutor. (The average total interaction time in our case was less than two hours for both groups.)

The pre- and post-tests analysis, and the frequencies in Figure 6.10, suggest that although Group C was initially less able than Group B, they learned the constraints at a faster rate. The constraint violation frequencies were investigated further to see if this was indeed the case. Each attempt at a problem was analysed, and the total proportion of violated constraints was calculated for each attempt. This was averaged over all students in each group, and the result is depicted in Figure 6.12. This scatter plot shows that Group C initially made more errors than Group B, but that the rate of constraint violation decreased much faster for that group, supporting the hypothesis that Group C learned the rules of the domain more quickly. Figure 6.13 shows the results of the same analysis, as an example, for constraints C_4 and C_7 only, which both depend on the child's cognitive ability to separate the problem text into sentences. The difference is much more marked for these constraints than for the average of all the constraints, but the trend is the same. For both scatter diagrams, a cut-off point of 125 attempts was selected because approximately half of the pairs of students reached this number of attempts, and beyond this number, statistical effects arising from the smaller number of pairs tends to corrupt the trend.

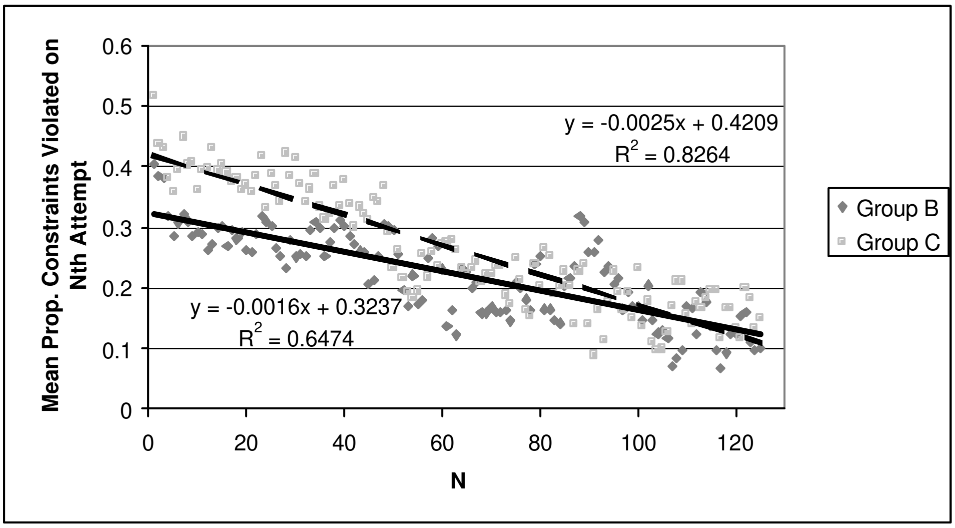


Fig. 6.12. Rate of constraint violation by attempt, for all constraints.

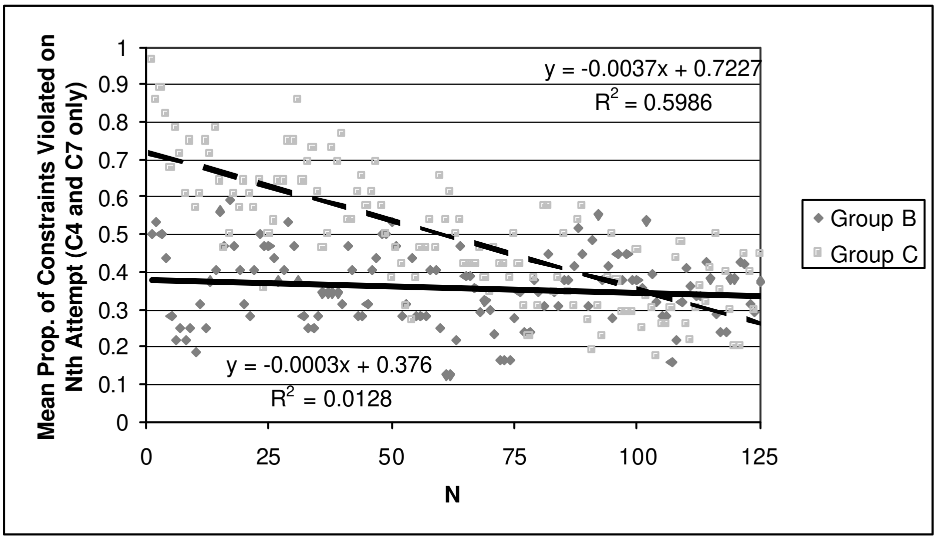


Fig. 6.13. Rate of constraint violation by attempt, for constraints 4 and 7.

Further analysis investigated the mean number of attempts, and the mean time required, to solve the *n*th problem. Figures 6.14 and 6.15 show the results of this analysis. Both line graphs show the same basic trend; Group C was less able initially, but improved at a faster rate than Group B.

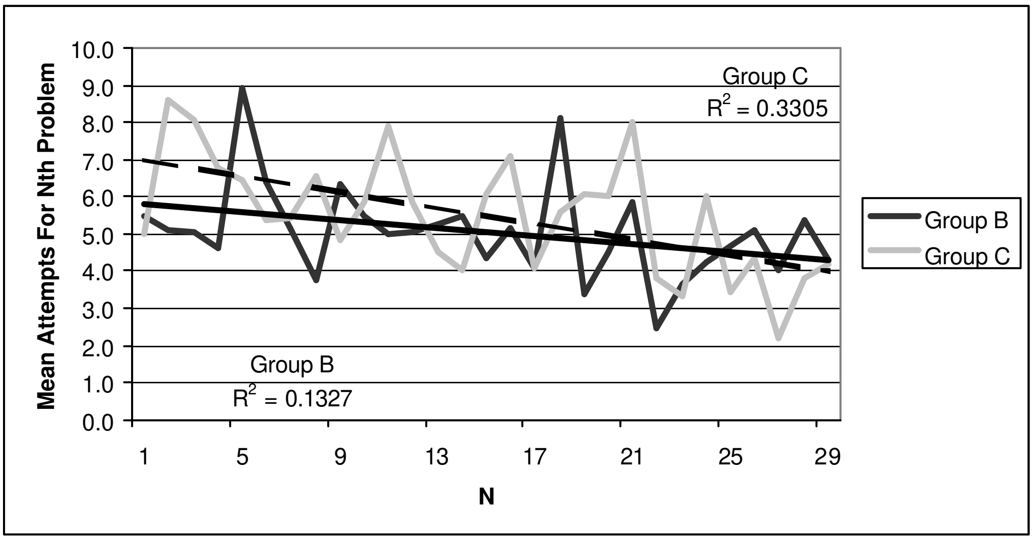


Fig. 6.14. Number of attempts solving the *n*th problem.

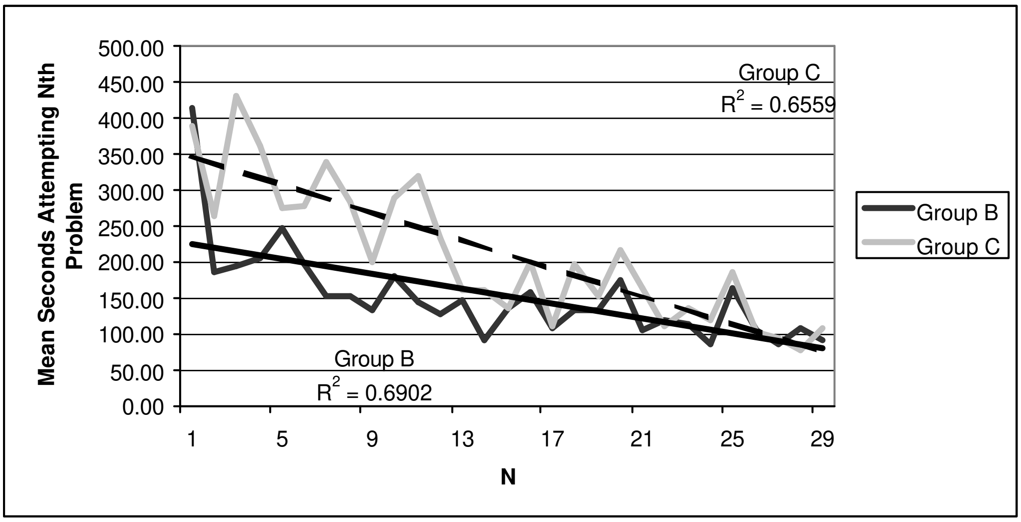


Fig. 6.15. Time required solving the *n*th problem.

To summarise, we have demonstrated that the version of CAPIT with decision-theoretic problem and error message selection, and an adaptive Bayesian network student model, has led to a faster rate of learning than the same system with decisions made randomly. Evidence for this is provided by the pre- and post-tests results, and the analysis of constraint violations over time. This completes the fifth and final step of the application of the general methodology we have proposed for CAPIT.

6.11 Summary

This chapter has demonstrated a general methodology for the design and implementation of decision-theoretic intelligent tutors. CAPIT, a tutor for capitalisation and punctuation, is both a working illustration that decision-theoretic computations in intelligent tutors can be tractable, and proof that the methodology works. It is therefore possible to build intelligent tutors with decision strategies that are guaranteed to be rational, given the specific utility function and posterior probabilities.

One area of concern with this approach is scalability, both to larger domains and different domains. In a larger domain, the space of <*State*, *Action*, *Outcome*> triples may be so large as to effectively render network induction impossible. This may be because the size of the *State* variable is very large (much larger than the 25 constraints modelled CAPIT), or there may be a high number of values that *Action* can possibly take (the limit in CAPIT was 45). A possible solution is to manipulate the learning algorithm to compensate for this additional complexity. That is, the higher the number of variables in the network, the lower the number of edges that should be added to the network. The algorithm used to learn CAPIT's Bayesian network in Section 6.6 could be used with a higher ε value to prevent relationships between variables with lower mutual information from appearing in the network. The effect of this measure would depend on how the constraints are interrelated. For example, if the domain constraints are highly interrelated but with only a low average mutual information, raising ε will actually eliminate most the relationships which would have a negative effect on the student model's predictions. On the other hand, if there are a handful of strong interrelationships amongst the large number of weak ones, raising ε will effectively eliminate only the weaker relationships.

With respect to actions, a possible solution is to break the actions up into groups and then apply decision-theoretic action selection twice: firstly to select the group; secondly to select the action within that group. This way, the total number of actions being considered will be equal to the number of groups plus the number of items in the selected group. The effect of using this strategy on the performance of the system will depend on how the groups of actions are determined. For example, a class of problem in an algebra tutor might be **problems of the form ax+b=c**. This is a precisely defined class in the sense that the skills required to solve the problems in the class are uniform. Similarly, a precisely defined problem grouping for CAPIT would be to group problems according to relevant constraints and problem metrics. For example, **problems that are relevant to only constraints 1, 2, 3, 4, 7, and 12 and consist of two sentences**. On the other hand, a problem class such as **problems in which singular possession must be denoted by an apostrophe** is not precisely defined because it specifies only one of the skills required to solve the problems in the class. The approach of grouping problems and applying decision theory to the groups, therefore, would be expected to remain effective for precisely defined action groupings but degrade for less well defined groupings.

The exact volume of data required for the construction of the Bayesian network student model is also a scalability concern. Bayesian networks have the property that they can be induced from minimal amounts of data. The primary question is how a minimal amount of student performance data affects both the behaviour of the system initially, before the model has had a chance to adapt to the student, and the length of time required to adapt the model to the student. Bayesian networks induced from minimal amounts of data may also be overly sensitive to small changes in the conditional probabilities and structure of the network. Fortunately, one possible work-around to this problem may be to compensate for lack of data by adding more prior knowledge (i.e. arcs and probabilities defined by protocol analysis or an expert) to the network. This is one of the advantages of using Bayesian networks as opposed to, for example, neural networks to which prior knowledge cannot be easily added. However, an in-depth investigation would be needed in this area, especially with respect to the sensitivity of the action selection procedures to the amount and quality of the data acquired during the first step of the methodology.

A limitation of the problem representation in CAPIT is that while lower-level ambiguity can be encoded into the constraints (e.g. in CAPIT, commas separating short clauses can easily be made optional by setting the C_s to the appropriate regular expression), the problem representation used in CAPIT is unable to deal with ambiguity arising from word and sentence semantics. To illustrate, the possessive pronoun *teachers* could be punctuated to either *teacher's* or *teachers'*. The latter is less plausible (unless there is specific contextual evidence that there is more than one teacher), but both are technically correct. CAPIT resolves this problem by accepting only the single most plausible solution as the correct solution (*teacher's* in this example), and treating other solutions as incorrect. This is acceptable for a system designed for children, because the system can control what its users are exposed to. That is, it is not ideal for a child to continually punctuate possessive nouns such as *teachers* to *teachers'* when the goal of the problem is to teach the correct punctuation of singular possessive nouns. However, in other domains, it may be that the system needs to know when a solution is technically correct. This suggests the need for either a problem solving module, or more advanced natural language processing capabilities which in turn would entail a more sophisticated problem representation for CAPIT.

An interesting conclusion of the statistical comparison between different Bayesian network architectures in Section 6.6 is that the *Small* specification produced networks that predicted student performance almost as well as the various large network specifications (the r^2 values varied by at most 0.06). The question arises as to why this is so. An informal analysis of the data collected during Step 1 revealed that, on average, a constraint previously satisfied will be satisfied on the next attempt 91% of the time. This regularity may well explain *Small*'s relatively good performance. However, other domains may not exhibit this degree of regularity. For example, in a domain where the constraints are highly interdependent, the probability of a constraint being satisfied on the next attempt may depend much more on the previous (and current) outcomes of other constraints. On the other hand, *Small* can be expected to outperform the large networks on domains where constraint performance is wholly or mostly probabilistically independent.

This suggests that there must be some careful justification (e.g. the statistical significance tests performed in Section 6.6) when a larger, complex model is chosen over a much simpler one. This issue is important, and it should not be skirted over when describing the rationale for a particular intelligent tutor architecture. Furthermore, the relatively good performance of a Bayesian network with no explicit model of the student's internal representation at predicting student performance begs the question of which domains in general are suitable for such an approach. Suitable domains may be those where the concepts are ill-defined, or where different students are expected to conceptualise the domain in different ways, (e.g. constructivist environments). Also, as discussed earlier, domains where the conceptualisation is too complex to produce a simple, tractable model might benefit.

Another advantage of this approach is that it bypasses the problem of prior probabilities in Bayesian networks. VanLehn et al. (1998) report that different choices of prior probabilities for root nodes in a network can significantly influence the posterior probabilities of other nodes. The work-around suggested by VanLehn et al. (1998) is to treat only the difference between a variable's prior and posterior probability as significant. Our Bayesian model circumvents this problem entirely. Whenever the network is evaluated, the root nodes $L_1..L_{25}$ are always known with certainty because they represent the observed student's history. That is, the causality is always directed from the known ($L_1..L_{25}$) to the unknown ($N_1..N_{25}$), and not the other way around. Whenever the network is properly used, therefore, the L_i nodes must be instantiated. Therefore priors do no even need to be maintained for these variables. The significant probabilities in our networks are the conditional probabilities.

To reiterate, the results of the evaluation study are positive and show that the application of normative techniques such as decision theory to intelligent tutoring is effective. The log analysis shows that the class using the decision-theoretic version of CAPIT learned the constraints of the domain at a faster rate than another class using the randomised version. The pre- and post-test results support this. Furthermore, on the post-test, both classes outperformed another class that did not have access to the tutor at all.

Chapter 7

Conclusions

This thesis has dealt with the application of normative theories to ITS design, for the express purpose of more rationally modelling the student and implementing pedagogical theories. While having acknowledged limitations in some cases, the methodology is a powerful new approach to ITS architecture design. Section 7.1 discusses the main original contribution, that being the methodology itself and its application to CAPIT and subsequent evaluation. Section 7.2 outlines some of the other original contributions of this thesis. In Section 7.3, future research directions that build on the results described in this thesis are outlined. Finally, closing remarks are made in Section 7.4.

7.1 Summary of Main Original Contribution

This thesis has proposed a methodology for the development of normative reasoning components as a solution to the problem of how to handle the uncertainty and sub-optimality inherent in the behaviour of traditional ITSs. When a learning theory is framed as a set of probability distributions and utility functions, normative mechanisms of reasoning guarantee that the learning theory will be applied rationally to the student. Consequently, the cause of sub-optimal or irrational behaviour in such an ITS can be traced to the specification of the theory itself (i.e. the values of the conditional probabilities and utilities) rather than the reasoning mechanisms underlying it. In a traditional ITS, in contrast, both theory specification and reasoning mechanism may be at fault.

The methodology addresses the practicality of applying normative methods to ITS design. In particular, it emphasises testing a prototypical "unintelligent" version of the tutor with randomised action selection and no long-term student model in the classroom, in order to acquire data about student performance in different situations. That data then becomes the source from which a Bayesian network student model is induced, using statistical significance tests to discriminate between alternative induction techniques. Once the initial Bayesian network is determined, it can be easily adapted to the student on-line (using online induction algorithms) and the network's predictions can then be fed into decision theoretic procedures for action selection. Psychological learning theories play an important role in the whole process; they are used to define both the semantics of the Bayesian network and the values of the utility functions.

CAPIT (Mayo et al, 2000; Mayo & Mitrovic, 2001) illustrates the application of the methodology. In this particular case, the nodes in the Bayesian network student model represent constraints (Ohlsson, 1994), and the model is utilised to predict the student's behaviour with respect to the constraints given different problems and the student's unique interaction history with the ITS. Vitgosky's (1978) Zone of Proximal Development is used to define the values of the utility function for next problem selection. A simpler theory defines the utility function for feedback message selection.

CAPIT and the methodology are also original in that they demonstrate that induction of general, unrestricted Bayesian networks from student performance data is entirely feasible. To date, work on induction of Bayesian network student models has been limited to very simple structurally restricted networks such as naive Bayes. This thesis has demonstrated both off-line (Step 2 of the methodology) and on-line (Step 4) Bayesian network induction.

Evaluation is a major component of this thesis. A total of three classroom studies were performed during 1999 and 2000. The first evaluation, of the extended version of SQL-Tutor (Mayo & Mitrovic, 2000), provided the impetus for the development of the general methodology. In turn, CAPIT was developed by applying the methodology and evaluated in the classroom as well. The successful evaluation of CAPIT is proof that application of entirely normative methods to ITSs can be both practical and effective.

7.2 Other Original Contributions

Besides the general methodology and CAPIT, this thesis makes some interesting secondary – but none-the-less original – contributions to other aspects of ITS research.

7.2.1 A New Perspective on Student Modelling

Chapter 3 highlighted two different perspectives on student modelling. The first perspective concerns the persistence of knowledge in the student model. Knowledge and beliefs about the student were shown to be either short or long-term. Short-term knowledge is typically recent and specific. In the case of CAPIT, the short-term knowledge is the outcome of the last attempt at each constraint. The short-term knowledge is then integrated into the long-term component of the student model by a process of inference. In other words, facts and observations of the student's behaviour are transformed into a set of beliefs about the student. Chapter 3 has shown that most ITS student models contain both short-term knowledge (e.g. constraints, traces), and long-term knowledge (typically overlay models).

The second perspective introduced in Chapter 3 was the analysis of Bayesian network student models. Three different motivations underlie these models. Firstly, there is the expert-centric approach that focuses on the Bayesian network being a cognitive model of the student. These networks are essentially hand-crafted by a domain expert. The second approach is efficiency-centric, in which the nature of the Bayesian network is restricted in some way to maximise the computational efficiency or reduce the knowledge-acquisition bottleneck. The cognitive model is then "fitted" to this restricted representation. The third and final class is data-centric, in which student modelling focuses on the student's behaviour rather than their cognitive state. The cognitive state may still be implicit in the model, but because this approach deals explicitly with observables, it is amenable to machine learning. CAPIT was designed as a data-centric tutor.

7.2.2 An Alternative Approach to Cognitive Student Modelling

CAPIT illustrates an alternative to the predominant cognitive approach to student modelling. In the past, considerable research effort has been expended developing more and more realistic cognitive student models. As was argued in Chapter 3, the price of such complexity is computational tractability. The complexity arises from both the complexity of the cognitive model itself and the need to compensate for high degrees of uncertainty in raw data and inferences. More often than not, the ability of the ITS to efficiently evaluate the student model on-line, which is its very purpose, is compromised. However, the argument for these models (largely from ITS researchers with a background in psychology) is that because they are based on psychological theory, they must be the most accurate models. The flow of information in an ITS with a cognitive student model is summarised by Figure 7.1.

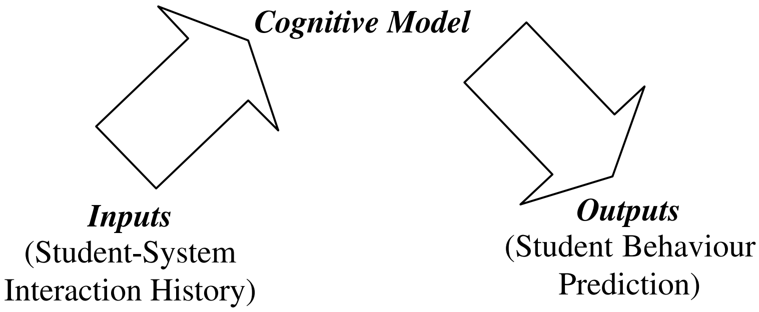

Cognitive Model
Inputs
(Student-System Interaction History)
Outputs
(Student Behaviour Prediction)

Fig. 7.1. Traditional ITS with a cognitive student model.

A concerning aspect of this philosophy is that whilst cognitive student models are many and varied, few of these systems (with the exception of some of the systems produced by the major research efforts, such as the ACT-R tutors) have actually been *validated*. Validation is the simple process of taking a real student history (the *Inputs* in Figure 7.1), feeding them into the student model, and comparing the model's predicted outputs to what the student actually did. The rationale for validation is that a student model, no matter how cognitively compelling it is, will be ineffective if it cannot ultimately account for student behaviour. The capacity for the student model to predict student behaviour is critical for other tasks such as tutorial action selection.

This thesis has proposed an alternative to the traditional approach. This is a system where there is no explicit cognitive model of the student, but instead a direct relationship between the inputs of the model and its outputs. Figure 7.2 depicts such a system. Because there is no need to expend effort developing a complex cognitive model, design and development can be directed towards developing a valid model instead. This approach is conducive to the data-centric method of building Bayesian network student models, described in this thesis.

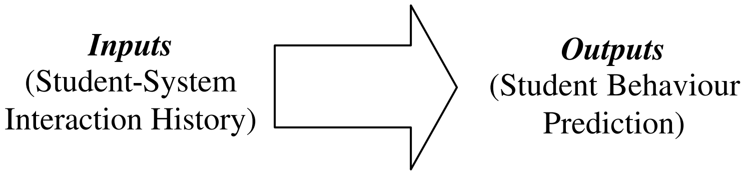


Fig. 7.2. Student model of CAPIT.

7.2.3 Constraint-Based Modelling in a Literacy Domain

Another contribution of this thesis is CAPIT as an extension of Constraint-Based Modelling (Ohlsson, 1994), and more generally an extension of student modelling, to a literacy domain. To date, there has been little work in this area. Only Bouwer (1998) has developed a tutor for punctuation, but that was targeted at university students rather than children. Less recently, Bos & van der Plassche (1994) developed an ITS for English verb-form tutoring. A recent New Zealand study found that "*…40% of employees here are below the minimum level of literacy competence required for everyday life and work.*" (Harmer, 1998). It is surprising, therefore, that more work has not been done in this area by ITS researchers. Literacy skills comprise a challengingly different collection of domains to traditional ITS domains such as mathematics or medical reasoning. For example, in traditional domains, the sequence of problem-solving steps is important and can provide useful diagnostic information. This has been the impetus behind the development of student modelling techniques such as model tracing. On the other hand, in literacy domains, the order of problem solving steps is often irrelevant and it is only the final solution state that is important. Therefore, in CAPIT, the student is free to punctuate and capitalise sentences in any desired order. Ohlsson's (1996) theory of learning from performance errors is relevant here precisely because it is the student solution state rather than the problem-solving sequence of steps that is relevant.

The role of semantics is another key difference between traditional and literacy domains. In traditional domains, the reasoning is formal and symbolic and could, in principle, be implemented as an expert system for solving problems (although not all ITSs do this). The fact that the domain can be decomposed in this way shows that there is a clear distinction between the symbolic reasoning and the semantics. The teaching of the semantics can, effectively, be isolated to canned explanations of the symbols and operators that the student must learn. This is certainly not the case in literacy domains. Here, semantics are highly integrated with student solutions and complex natural language processing problems such as ambiguity arise as a result. It is possible to minimise these issues by restricting problems to certain simple types. For example, the verb form tutor (Bos & van der Plassche, 1994) poses simple single-word completion exercises. However, in the general case, new techniques and methods will have to be introduced to ITSs. This is not a daunting prospect; natural language understanding researchers have struggled with these ideas in the past and natural language processing already has a place in ITS research (see, e.g. Freedman et al., 2000). These techniques will be invaluable in literacy domains. CAPIT therefore represents a first step towards non-trivial ITSs in the literacy domains.

7.3 Future Research Directions

There is considerable work that could be undertaken in this area. Firstly, alternative means of Bayesian student modelling are certainly possible and can fit this framework. A novel example is to model the student as a Bayesian network that is literally a sub-network of a "gold standard" Bayesian network representing the expert's knowledge (Horvitz, 2000). This approach, similar in concept to the overlay model but more sophisticated, is ideal for teaching uncertain reasoning in appropriate domains.

Another alternative is to hybridise Bayesian methods with existing student modelling methods. For example, consider the combination of a Bayesian network and a genetic graph. Genetic graphs are a knowledge representation for ITSs that are based on semantic networks, but with additional richness (Brecht & Jones, 1988). Nodes in a genetic graph represent the facts, rules, skills, or concepts of the domain, and edges define the way in which the domain items relate to each other from a pedagogical perspective. A student can only learn a new item if it is adjacent to an already known item. As a result, the student's progression from novice to expert can be envisaged as the gradual "spreading" of a connected subgraph over the original graph.

Edges are classified according to the type of relationship they represent. Examples are analogy, generalisation/specialisation, refinement, and component; there are several others. The relationship type can indicate how the item should be taught. For example, if a known topic is adjacent to an unknown topic and the relationship is analogy, then that defines a way in which the new topic can be taught and related to what the student already knows. Further structure is possible in the genetic graph by clumping sets of nodes into "islands" which require substantially similar underlying conceptual knowledge.

Genetic graphs are interesting from a Bayesian perspective because they could be represented directly by a Bayesian network. Further uncertain variables could be added to to the model representing the student's conduciveness to the different edge types (e.g. the student may prefer explanation by analogy rather than generalisation/specialisation), and this could be the basis for a decision-theoretic procedure for planning the optimal path through the genetic graph.

A second avenue for future research is to investigate further the application of decision theory to ITSs. The approach taken in this thesis of maximising expected utility is relatively straightforward, but aside from DT-Tutor (Murray & VanLehn, 2000), decision theory has not been widely implemented in existing ITSs. There are considerably more sophisticated decision-theoretic reasoning schemes that could be explored. For example, the decision-theoretic troubleshooting scheme proposed by Breese & Heckerman (1996) could form the reasoning component of a trouble-shooting tutor. More generally, Blythe (1999) gives a highly relevant overview of decision-theoretic planning techniques. Clearly, this is a fruitful avenue for ITS research, which will produce more robust and rational tutors.

Efficiency considerations are a third topic for future research. Bayesian networks and decision theory, despite enormous strides in the development of efficient numeric algorithms, can still be intractable if the domain is large. For example, the initial investigation of how to extend SQL-Tutor that led to the first evaluation study showed early on that it was infeasible to model all 500 constraints as a Bayesian network and have decision-theoretic problem selection. This is why heuristics supplanted normative techniques in that initial evaluation. There are more efficient alternatives to numerical algorithms, and these are *qualitative* algorithms. Qualitative models replace numeric specification with linguistic categories, and inference is performed symbolically on these categories. For example, the qualitative Bayesian networks described by Chao-Lin & Wellman (1998) replace numeric conditional probabilities with qualitative "influences". Qualitative decision-theoretic schemes also exist (e.g., Fargier & Perny (1999)).

A fourth possibility for future research is to continue to test and extend CBM (Olhsson, 1994). Prior to CAPIT, SQL-Tutor was the only ITS using CBM to model student knowledge. CAPIT is implemented in a completely different domain, and CBM has proven useful. The logical next step in the evaluation of CBM is to extend the approach to more complex modelling in a literacy domain. For example, grammar and reading are two such domains that would present significant new challenges to CBM.

Fifth and finally, it would be interesting to investigate how tolerant decision-theoretic methods are to uncertainties in the student model. An implication of considering student modelling from the perspective of decision theory is that the accuracy of the student model may not be as important as it is in diagnostic systems. Decision theory aims to maximise the expected utility of actions, not the accuracy of beliefs. In other words, uncertainty can be tolerated in a student model if the most rational action is unaffected. For example, the next best tutorial action may be invariant regardless of whether or not the student's mastery of a particular item has a probability of 0.8 or 1 attached to it. This is not true in diagnostic systems where accuracy *is* important. The point to bear in mind is that in an ITS, an extremely precise student model is not necessarily the best student model, and the effort required to build in the additional precision may be best spent elsewhere (such as on the construction of a pedagogically-reasonable utility function). It would be fascinating to try and quantify this degree of tolerance for real-world domains.

7.4 Final Remarks

ITS technology is currently in its infancy. Systems developed in the lab have only in recent years found their way into the classroom, but this is usually only for short-term evaluations. Despite this, there is a large potential commercial market for quality ITSs, especially now that large-scale low-cost electronic distribution of software is possible via the Internet. The main differentiating factor between ITSs and other educational software is the built-in "intelligence"; in addition to exercises and multi-media tuition, the ITS has a pedagogy which it applies to make learning more efficient. Research should therefore be directed towards making the pedagogy more effective. Indeed, the community of student modelling researchers are going a long way towards achieving this. The contribution of this thesis is to show that normative theories are effective tools for computationally representing and implementing pedagogical theories.

Appendix A

Problems In CAPIT

Problem	Relevant Constraints
The morning is beautiful. The sun is rising. The birds are chirping. There is not a cloud in the sky.	C001, C002, C003, C004, C007, C014
Charlie Smith has a dog called Ratbag. Aunty Maude baked a cake for Charlie. Ratbag ate the cake. Aunty Maude was not pleased.	C001, C002, C003, C004, C005, C007, C014
We are going to Nelson for the holidays. We live in Christchurch. It is a long drive from Christchurch to Nelson.	C001, C002, C003, C004, C006, C007, C014
The woman's hat was taken by mistake. She complained to the shop's manager. The manager's apology was not accepted.	C001, C002, C003, C004, C007, C008, C014
There's a cricket game today. It's starting soon. Our teacher's the captain of one team.	C001, C002, C003, C004, C007, C009, C014
You can't play hockey. You haven't got a hockey stick. Your mother doesn't want you to have one.	C001, C002, C003, C004, C007, C010, C014
We used pencils, rulers, books and paper. We also sang, danced and played.	C001, C002, C003, C004, C007, C011, C012, C014, C016

We went to the wedding, then drove to the reception. The best man gave a speech, and we all cheered.	C001, C002, C003, C004, C007, C013, C014
Jenny Porter drove from Dunedin to Picton. The car broke down near Timaru. Jim Baird fixed the car for Jenny.	C001, C002, C003, C004, C005, C006, C007, C014
We are playing games on Max's new computer. The computer's monitor is very large. Max's sister Annabelle wants to use the computer.	C001, C002, C003, C004, C005, C007, C008, C014
Jack's going to the movies. There's a movie starring Eddie Murphy. Eddie Murphy's a funny police officer in the movie.	C001, C002, C003, C004, C005, C007, C009, C014
Maggie Jones can't go to the beach. She hasn't tidied her room. We aren't going to wait for Maggie.	C001, C002, C003, C004, C005, C007, C010, C014
Harvey Smith, Matilda West and John Snodgrass are late for school.	C001, C002, C003, C004, C005, C007, C011, C012, C014, C016
Justine Chambers packed her bags, and she left home. She was going to catch a bus, but Andy convinced Justine to go back home.	C001, C002, C003, C004, C005, C007, C013, C014
We visited Auckland's airport. The airport's tour guide showed us around. We watched a plane arrive from Hong Kong.	C001, C002, C003, C004, C006, C007, C008, C014
Westburn School's a great school. My teacher's very nice. She's taking us to the Botanic Gardens.	C001, C002, C003, C004, C006, C007, C009, C014
He can't climb Mount Everest. He isn't fit enough. He had better come back to New Zealand.	C001, C002, C003, C004, C006, C007, C010, C014
I want to go to America, Canada and Australia. She wants to go to Taiwan, Malaysia and Singapore.	C001, C002, C003, C004, C006, C007, C011, C012, C014, C016
Canterbury is south of Marlborough, but it is north of Otago. There are more people in Canterbury, and fewer in Marlborough.	C001, C002, C003, C004, C006, C007, C013, C014

My friend's wallet's in the car. There's money in it. I will put it in the car's glovebox.	C001, C002, C003, C004, C007, C008, C009, C014
The dog's bone isn't here. He can't find it anywhere. There isn't a spot in my parent's house where he hasn't looked.	C001, C002, C003, C004, C007, C008, C010, C014
The teacher's chalk, marker and overheads were stolen. The principal's filing cabinet, telephone and typewriter are also missing.	C001, C002, C003, C004, C007, C008, C011, C012, C014, C016
The baby's cot is set up, but my brother's bed is still in the baby's room. My brother's things will be shifted out, then the baby's room will be ready.	C001, C002, C003, C004, C007, C008, C013, C014
A big shark's in the bay. It isn't travelling very fast. We haven't seen this shark before. There's a chance that swimming's dangerous.	C001, C002, C003, C004, C007, C009, C010, C014
The gardener's planting cabbages, tomatoes and onions. She's growing them for me, my uncle and my mother. There's plenty for everyone.	C001, C002, C003, C004, C007, C009, C011, C012, C014, C016
The writer's staying in his room, because he's writing a book. He's going to finish it soon, and there's a good chance he will be famous.	C001, C002, C003, C004, C007, C009, C013, C014
Our mother said that we mustn't eat lollies, chocolate and ice cream. We shouldn't eat cakes, biscuits and desserts.	C001, C002, C003, C004, C007, C010, C011, C012, C014, C016
Your friend shouldn't play on the computer, and she can't read my books.	C001, C002, C003, C004, C007, C010, C013, C014
Porridge, cereal and toast are available for breakfast, and you can drink tea, coffee or juice.	C001, C002, C003, C004, C007, C011, C012, C013, C014, C016
Angus Harrison planted a tree. It grew until its trunk was two metres wide. Its branches were twenty metres long. Jack Johnson cut the tree down.	C001, C002, C003, C004, C005, C007, C014, C015
Chile is a country west of Argentina. Its capital is Santiago. The South Pacific Ocean is its neighbour to the west.	C001, C002, C003, C004, C006, C007, C014, C015

My mother's car is not starting. Its battery may be flat. Its radiator may be frozen. We will have to take my father's bike instead.	C001, C002, C003, C004, C007, C008, C014, C015
There's a bee buzzing past me. It's taking its honey back to its hive. I hope it knows its way home.	C001, C002, C003, C004, C007, C009, C014, C015
The monster can't believe its luck. The gold wasn't stolen from its cave during its absence.	C001, C002, C003, C004, C007, C010, C014, C015
The elephant is wearing its boots, helmet and jacket. The mouse is wearing its pajamas, necktie and watch.	C001, C002, C003, C004, C007, C011, C012, C014, C015, C016
The robot tried to walk all the way, but its batteries ran out. Its movements got slower and slower, until its legs would not move anymore.	C001, C002, C003, C004, C007, C013, C014, C015
The teacher said, "Open your books."	C001, C002, C003, C004, C014, C017, C018, C019, C020, C021, C022, C023
"Jack and Matilda went to the market," said Mrs Ashton.	C001, C002, C003, C005, C007, C014, C018, C019, C020, C022, C024, C025
He said, "I will visit both Afghanistan and Korea."	C001, C002, C003, C004, C006, C014, C017, C018, C019, C020, C021, C022, C023
The team's captain said, "The ref's decision is wrong."	C001, C002, C003, C004, C008, C014, C017, C018, C019, C020, C021, C022, C023
"That's a good deal," said the salesman. "Here's a better one."	C001, C002, C003, C007, C009, C014, C018, C019, C020, C021, C022, C023, C024, C025
The doctor said, "You shouldn't eat any more vegetables. Your body can't tolerate them."	C001, C002, C003, C004, C007, C010, C014, C017, C018, C019, C020, C021, C022, C023
"We saw planes, trains and automobiles," said the boy.	C001, C002, C003, C007, C011, C014, C016, C018, C019, C020, C022, C024, C025,

The woman said, "Roses are red, but violets are blue."	C001, C002, C003, C004, C013, C014, C017, C018, C019, C020, C021, C022, C023
"The crab lives in its shell," explained the teacher.	C001, C002, C003, C007, C014, C015, C018, C019, C020, C022, C024, C025

Appendix B

Regular Expression Special Characters and Character Sequences

Definition of the regular expression special characters and symbols used in CAPIT, adapted from Microsoft (2000). Literal characters are denoted by `courier` font, and variables standing for sequences of literals are highlighted by *italicising* the font.

Character(s)	Description
`\`	Marks the next character as either a special character or a literal. For example, `n` matches the character `n` but `\n` matches a newline character. The sequence `\\` matches `\`, and `\(` matches `(`.
`^`	Matches the beginning of input.
`$`	Matches the end of input.
`*`	Matches the preceding character zero or more times. For example, `zo*` matches either `z` or `zoo`.
`+`	Matches the preceding character one or more times. For example, `zo+` matches `zoo` but not `z`.

?	Matches the preceding character zero or one time. For example, `a?ve?` matches the `ve` in `never`.
.	Matches any single character except a newline character.
(*pattern*)	Matches *pattern* and remembers the match. Parentheses characters can be matched using `\(` or `\)`.
x | *y*	Matches either *x* or *y*. For example, `z|food` matches `z` or `food`, and (`z|f`)`oo` matches `zoo` or `foo`.
{*n*}	Matches exactly *n* times, where *n* is a nonnegative integer. For example, `o{2}` does not match the `o` in `Bob`, but matches the first two `o`'s in `foooood`.
{*n*,}	Matches at least *n* times, where *n* is a nonnegative integer. For example, `o{2,}` does not match the `o` in `Bob` and matches all the `o`'s in `foooood`. `o{1,}` is equivalent to `o+`. `o{0,}` is equivalent to `o*`.
{*n*,*m*}	Matches at least *n* and at most *m* times, where *m* and *n* are nonnegative integers. For example, `o{1,3}` matches the first three `o`'s in `fooooood`. `o{0,1}` is equivalent to `o?`.
`[xyz]`	A character set. Matches any one of the enclosed characters. For example, `[abc]` matches the `a` in `plain`.
`[^xyz]`	A negative character set. Matches any character not enclosed. For example, `[^abc]` matches the `p` in `plain`.
`[a-z]`	A range of characters. Matches any character in the specified range. For example, `[a-z]` matches any lowercase alphabetic character in the range `a` through `z`.
`[^m-z]`	A negative range characters. Matches any character not in the specified range. For example, `[m-z]` matches any character not in the range `m` through `z`.
`\b`	Matches a word boundary, that is, the position between a word and a space. For example, `er\b` matches the `er` in `never` but not the `er` in `verb`.

`\B`	Matches a nonword boundary. For example, `e*r\B` matches the `ear` in `never early`.
`\d`	Matches a digit character. Equivalent to `[0-9]`.
`\D`	Matches a nondigit character. Equivalent to `[^0-9]`.
`\f`	Matches a form-feed character.
`\n`	Matches a newline character.
`\r`	Matches a carriage return character.
`\s`	Matches any white space including space, tab, form-feed, etc. Equivalent to `[ \f\n\r\t\v]`.
`\S`	Matches any nonwhite space character. Equivalent to `[^ \f\n\r\t\v]`.
`\t`	Matches a tab character.
`\v`	Matches a vertical tab character.
`\w`	Matches any word character including underscore. Equivalent to `[A-Za-z0-9_]`.
`\W`	Matches any nonword character. Equivalent to `[^A-Za-z0-9_]`.

Appendix C

Constraints in CAPIT

	C_r	**C_s**	**Msg**
C_1	DEFAULT| L-CASE	^[a-z0-9%SYMBOLSET%]*$	This word doesn't need any capital letters!
C_2	DEFAULT| NO-PUNC	^[a-z0-9A-Z]*$	This word doesn't need to be punctuated!
C_3	SENTENCE-START| NAME-OF-PERSON| NAME-OF-PLACE| DIRECT-QUOTE-START	^[%SYMBOLSET%]*[^%SYMBOLSET%][^A-Z]*$	There are too many capital letters in this word!
C_4	SENTENCE-START	^[%SYMBOLSET%]*[A-Z0-9]	A sentence should start with a capital letter!
C_5	NAME-OF-PERSON	^[%SYMBOLSET%]*[A-Z0-9]	Each word in a person's name should start with a capital!
C_6	NAME-OF-PLACE	^[%SYMBOLSET%]*[A-Z0-9]	Each word in a place's name should start with a capital!
C_7	SENTENCE-END	\.$	A sentence should end with a full stop!

$\mathbf{C_{8}}$	POSSESSIVE-NOUN-NOT-ENDING-IN-S	's$	An apostrophe is required to show possession!
$\mathbf{C_{9}}$	IS-CONTRACTION	's$	An apostrophe is required to show the contraction of *is*!
$\mathbf{C_{10}}$	NOT-CONTRACTION	n't$	An apostrophe is required to show the contraction of *not*!
$\mathbf{C_{11}}$	INTERMEDIATE-ITEM-IN-LIST	,$	A comma is required to separate items in a list!
$\mathbf{C_{12}}$	FINAL-ITEM-IN-LIST	[^,]$	A comma is not required after the last item in the list!
$\mathbf{C_{13}}$	END-OF-CLAUSE	,$	A comma should separate the two parts of the sentence!
$\mathbf{C_{14}}$	ONE-PUNC	^[^%SYMBOLSET%]*[%SYMBOLSET%]?[^%SYMBOLSET%]*$	No more than one punctuation mark is required in this word!
$\mathbf{C_{15}}$	ITS-POSSESSIVE-PRONOUN	[^']s$	No apostrophe is required in *its*!
$\mathbf{C_{16}}$	INTERMEDIATE-ITEM-IN-LIST-PRECEDING-CONJUNCTION	[^,]$	Items in a list separated by a conjunction don't need to be separated by a comma as well!
$\mathbf{C_{17}}$	WORD-PRECEDING-DIRECT-QUOTE	,$	A comma is needed before the direct speech starts!
$\mathbf{C_{18}}$	DIRECT-QUOTE-START	^[%SYMBOLSET%]*[A-Z0-9]	Direct speech always starts with a capital letter!
$\mathbf{C_{19}}$	DIRECT-QUOTE-START	^"	A quotation mark is needed to show the start of direct speech!

C_{20}	`DIRECT-QUOTE-END|DIRECT-QUOTE-ENDING-SENTENCE|FINAL-ITEM-IN-LIST-DIRECT-QUOTE-END`	`[%SYMBOLSET%]*"[%SYMBOLSET%]*$`	A quotation mark is needed to show the end of direct speech!
C_{21}	`DIRECT-QUOTE-ENDING-SENTENCE`	`[%SYMBOLSET%]*\.[%SYMBOLSET%]*$`	The direct speech should end with a full stop!
C_{22}	`TWO-PUNC`	`^([^%SYMBOLSET%]*[%SYMBOLSET%]+){2}[^%SYMBOLSET%]*$`	Exactly two punctuation marks are required in this word!
C_{23}	`DIRECT-QUOTE-ENDING-SENTENCE`	`[^%SYMBOLSET%]+((\.+"+)|"+|\.+)?$`	The full stop should be within the quotation marks!
C_{24}	`DIRECT-QUOTE-END|FINAL-ITEM-IN-LIST-DIRECT-QUOTE-END`	`[%SYMBOLSET%]*\,[%SYMBOLSET%]*$`	A comma should follow the direct speech!
C_{25}	`DIRECT-QUOTE-END|FINAL-ITEM-IN-LIST-DIRECT-QUOTE-END`	`[^%SYMBOLSET%]+((,+"+)|"+|,+)?$`	The comma should be within the quotation marks!

Appendix D

AI’99 Paper (Published)

The following paper (Mayo & Mitrovic, 1999) was presented and published as a poster paper at AI’99 in Australia, Nov. 1999. It is the proposal for the work that is described in Chapter 4.

Estimating Problem Value in an Intelligent Tutoring System using Bayesian Networks

Michael Mayo and Antonija Mitrovic

Department of Computer Science, University of Canterbury
Private Bag 4800, Christchurch, New Zealand
mjm185@student.canterbury.ac.nz, tanja@cosc.canterbury.ac.nz

SQL-TUTOR [1,2] is an ITS for teaching the database language SQL to upper-level undergraduate students taking database courses. Students using SQL-TUTOR work through a series of problems where the solution is an SQL statement. Although SQL-TUTOR does not solve problems, it does have an ideal solution (IS) for each one. A correct student solution (SS) to a problem may be the same as the IS although there can be more than one correct solution. Figure 1 is an example of a problem in SQL-TUTOR, its IS, and an incorrect SS.

Problem	Ideal Solution	Student Solution

List the titles of all movies that have a critics rating.	`SELECT title`
	`FROM movie WHERE`
	`NOT(critics=’NR’);`
		`SELECT title`
		`FROM movie WHERE`
		`critics NOT ‘NR’;`

Fig. 1. An SQL problem, its ideal solution, and a student’s incorrect solution.

SQL-TUTOR models students using Ohlsson’s Constraint-Based Modeling (CBM) [3]. CBM proposes the modeling of domains as a set of constraints of the form *(Cr, Cs)*. *Cr* specifies the set of student solutions to which the constraint is relevant, and *Cs* specifies the subset of the relevant student solutions where the constraint is satisfied. Each constraint has an associated feedback message that can be displayed if the constraint is violated. In figure 1, the student has violated constraint 168 and the feedback message is: *Make sure NOT is in the right place in the WHERE clause.*

Until recently, problem selection in SQL-TUTOR was based on one simple rule: the first problem relevant to the single constraint that the student has most frequently violated in the past was selected. In a real classroom, this is an overly simple strategy because it was often the case that selected problems were either too complex or too simple. Our research has been aimed at improving this situation.

We propose a new problem selection module based on Bayesian belief networks (BBNs) [4]. Our approach involves applying the following two steps to each potential next problem *p*. Firstly, the system predicts, for each constraint *c*, the potential teaching effects of *p*. The main calculation is of the posterior probability P($Performance_{c,p}$ = VIOLATED), the probability that *c* will be violated by the student should he/she attempt problem *p*. Constraint violations lead to feedback messages, and constraint-specific feedback is the main way that students learn constraints in SQL-TUTOR. A BBN for this is depicted in figure 2. $RelevantIS_{c,p}$ is the probability of *c* being relevant to *p*’s IS. The value for this node is always known with certainty. $RelevantSS_{c,p}$ is the probability of *c* being relevant to *p*’s SS. $Mastered_c$ is the probability of the student having mastered the constraint *c*. Finally, *Performance*$_{c,p}$ predicts the behaviour of the student on constraint *c*. All the nodes except *Performance*$_{c,p}$ are binary variables. *Performance*$_{c,p}$ is a three-valued node taking values {SATISFIED, VIOLATED, NOT-RELEVANT}.

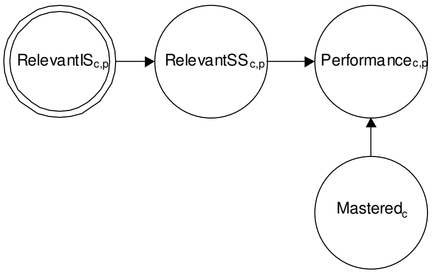


Fig. 2. A Bayesian network for predicting student performance on a single constraint.

The second step is to summarise the predictions for *p* over all the constraints *c*. Currently this is done by counting the number of constraints for which P(*Performance*$_{c,p}$ = VIOLATED) > 0.45. This number, *Feedbacks*, is then compared to the student's *OptimalFeedback*. The value of *p* is (- | *Feedbacks* – *OptimalFeedback* |). That is, *p* has a high value if *Feedbacks* is close to or the same as *OptimalFeedback*. The rationale behind this rule is that if the predicted number of feedback messages exceeds *OptimalFeedback* then the student will be overwhelmed with information and the teaching effects of each message will be discounted. On the other hand, if the number of feedback messages is less than optimal, then student learning will be inefficient and the problems may be too easy. Presently *OptimalFeedbacks* starts at 2 and increases linearly with the competence level of the student.

After the student has submitted his solution, the prior probabilities of mastery for each constraint, P(*Mastered*$_1$), P(*Mastered*$_2$)…etc, are updated if the constraint was relevant to the SS.

We have performed several off-line experiments using student history logs from previous user studies of SQL-TUTOR, comparing problems that were selected by the original system against problems that would have been selected by the proposed system. In the majority of cases, the new system outperforms the old system.

Future research will investigate the acquisition of probabilities for the BBNs both subjectively (by an expert) and from data. We also plan an on-line evaluation of the new system in a user study.

1. Mitrovic A., 1998. Learning SQL with a Computerized Tutor, Proc. 29th SIGCSE Tech. Symposium, 307-311.
2. Mitrovic A., Ohlsson, S., 1999. Evaluation of a constraint-based tutor for a database language, Int. J. Artificial Intelligence in education, 10, 3-4, to appear.
3. Ohlsson S., 1994. Constraint-based Student Modeling. In: Greer, J.E., McCalla, G.I. (eds.): Student Modeling: the Key to Individualized Knowledge-based Instruction. NATO ASI Series, Vol. 125. Springer-Verlag, Berlin, 167-189.
4. Pearl J. 1988. Probabilistic reasoning in intelligent systems: networks of plausible inference (revised 2nd edition). Morgan Kauffman, USA.

Appendix E

ITS'2000 Paper (Published)

The following paper (Mayo & Mitrovic, 2000) was presented and published at the ITS'2000 conference in Montreal, Canada. It covers the bulk of Chapter 4.

Using a probabilistic student model to control problem difficulty

Michael Mayo and Antonija Mitrovic

Intelligent Computer Tutoring Group
Department of Computer Science, University of Canterbury
Private Bag 4800, Christchurch, New Zealand
mmayo@cosc.canterbury.ac.nz, tanja@cosc.canterbury.ac.nz

Abstract. Bayesian networks have been used in Intelligent Tutoring Systems (ITSs) for both short-term diagnosis of students' answers and for longer-term assessment of a student's knowledge. Bayesian networks have the advantage of a firm theoretical foundation, in contrast to many existing, ad-hoc approaches. In this paper we argue that Bayesian nets can offer much more to an ITS, and we give an example of how they can be used for selecting problems. Similar approaches may be taken to automating many kinds of decision in ITSs.

Keywords: instructional design; student modeling; evaluation of instructional systems

1 Introduction

Our research is aimed towards developing a general methodology for using the student model to solve decision problems in an ITS. There has been much research in the field of student modeling, and student models that can reasonably accurately predict student post-test performance have been developed (e.g. the ACT Programming Tutor [5]). However, to be truly adaptive, an ITS must make optimal use of the information contained in the student model to tailor its actions to the specific needs of the student. Actions include selecting an appropriate problem if the student asks for it, next topic selection, feedback selection, and selective highlighting/hiding of information. We have developed an approach to adaptive decision-making based on Bayesian probability theory. For each alternative, simple Bayesian networks are used to make multiple predictions about student performance on atomic domain elements called constraints. These multiple predictions are then combined heuristically to give an overall measure of the value of the alternative. The approach is demonstrated in a problem selection module for SQL-Tutor [8,9], an ITS for teaching the database language SQL.

Section 2 briefly introduces SQL-Tutor. In section 3 we describe an approach to selecting problems of appropriate complexity via Bayesian networks. The results of a preliminary evaluation study are discussed in section 4, followed by a comparison of this approach to others section 5. Section 6 is conclusions and future research.

2 SQL-Tutor

SQL-Tutor is a practice environment for undergraduates enrolled in database courses. There are three functionally identical versions 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 [8,9] and the system's Web page[1] for details.

The architecture of the system is illustrated in Figure 1. The system contains definitions of several databases, a set of problems for each database and the ideal solutions to them. The solutions are necessary because SQL-Tutor evaluates student solutions by comparing them to the correct ones. Each problem is assigned a difficulty level, which depends on many features, such as the wording of the problem, the constructs needed for its solution, the number of required tables/attributes etc. Each student is given a level of mastery, which dynamically changes in accordance with student's performance.

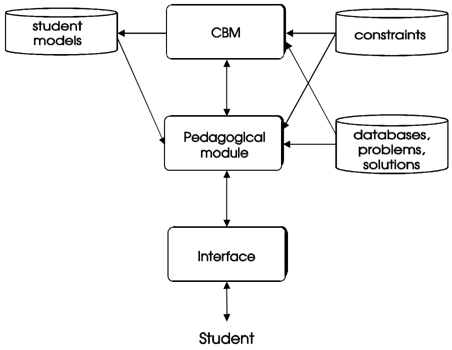


Fig. 1. Architecture of SQL-Tutor

The basic components of the system are the interface, the pedagogical module and the CBM student modeler. The pedagogical module (PM) observes every student's action and reacts appropriately. At the beginning of he session, a problem must be selected for the student. When the student enters the solution, the PM sends it to the student modeler, which analyzes the student's solution in order to identify possible errors. If any errors exist, the PM generates appropriate feedback messages. After the first attempt a student is only told whether his/her solution is correct or not. The level of detail increases if the student is not able to correct the solution.

SQL-Tutor uses Constraint-Based Modeling [10] to diagnose students' solutions. The conceptual domain knowledge is represented in terms of over 500 constraints. A student's solution is matched to the constraints to identify any that are violated. Long-term student knowledge is represented as an overlay model that tallies the percentage of times the constraint has been satisfied (i.e. used correctly). Both students and problems in SQL–Tutor are assigned a level. The student's level is incremented if he/she solves two or more problems consecutively at or above the student's current level, within three attempts each.

[1] See http://www.cosc.canterbury.ac.nz/~tanja/sql-tut.html

There are three ways to select the next problem in SQL-Tutor. Students can work through a pre-specified sequence of problems by clicking *next problem*, they can select a practice problem directly from a menu of problems, or they can turn problem selection over to the system by clicking *system's choice*. In the third case, SQL-Tutor examines the student model and selects the first problem whose level is within +1 or –1 of the student's level, which is also relevant to the constraint that the student has violated most frequently. The rationale for this rule is that if the student has violated the same constraint several times, it is appropriate to target that constraint for instruction. This problem selection strategy is overly simple. In a real classroom, it was often the case that selected problems were too complex or simple for the student, or they jumped to another part of the domain seemingly not connected to the previous problem. We set out here to explore other possibilities for problem selection.

3 Problem Selection using Bayesian Networks

Bayesian networks [2,11] are tools for representing and reasoning with uncertain knowledge using Bayesian probability theory.

3.1 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 $Mastered_1, Mastered_2, \ldots, Mastered_n$, 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 firstly counting the frequencies with which *c* was both satisfied and relevant (i.e. either satisfied or violated) in SQL-Tutor logs from previous evaluation studies, and then by dividing the former frequency by the latter. The logs were only analysed 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 past SQL-Tutor logs either because they were new or they had never been used. For these constraints, P($Mastered_c$ = *YES*) was initialised to 0.5.

If constraint *c* is satisfied, then P($Mastered_c$ = *YES*) increases by 10% of (1-P($Mastered_c$=*YES*)).
If constraint *c* is violated and no feedback about *c* is given, then P($Mastered_c$ = *YES*) decreases by 20%.
If constraint *c* is violated but feedback is given about *c*, then P($Mastered_c$ = *YES*) increases by 20% of (1-P($Mastered_c$=*YES*)).

Table 1. Heuristics used for updating the student model

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 1 to update the probabilities. This heuristic approach was chosen over Bayes' rule, because we do not make the assumption that constraints in the SQL domain are probabilistically independent, whereas many other models (e.g. Reye's model [12]) do. Therefore, applying Bayes' rule would result in a calculation that would be impractical to perform on-line. Dependence between constraints in SQL-Tutor arises at least because each violated constraint generates an error message, and so mastery of a constraint depends to some extent on how many other errors were made at the same time. This point is discussed further in sections 3.3 and 5. Constraints may also be dependent because of semantic overlap, syntactic proximity in problems, and pre- and co-requisite relationships. We believe that models where probabilistic independence between all knowledge items is assumed are unrealistic (e.g. Reye's model [12]).

3.2 Predicting student performance on single constraints

We use a simple Bayesian network (Figure 2) to predict the performance of a student given a problem *p* on a single constraint *c*. *Mastered$_c$* is from the student model. Both *RelevantIS$_{c,p}$* and *RelevantSS$_{c,p}$* are *YES*/*NO* variables. *RelevantIS$_{c,p}$* is *YES* if constraint *c* is relevant to problem *p*'s ideal solution. Because this can be determined from the problem database, *RelevantIS$_{c,p}$* is always known with certainty. *RelevantSS$_{c,p}$* is *YES* if constraint *c* is relevant to the student's solution to problem *p*. *Performance$_{c,p}$* is a three-valued node taking values *SATISFIED*, *VIOLATED* or *NOT-RELEVANT*. The arcs indicate that *RelevantSS$_{c,p}$* is dependent on *RelevantIS$_{c,p}$*. *Performance$_{c,p}$* is dependent on whether or not the student has mastered *c*, and *c*'s relevance to the student solution.

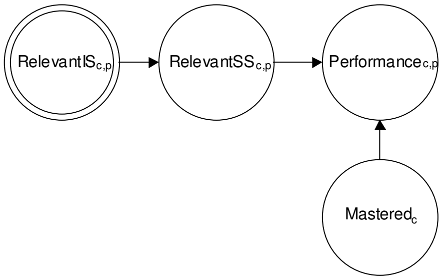


Fig. 2. A Bayesian network for predicting student performance on a single constraint.

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. In Table 2, 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 α_c = 0.9 for example, there is a high probability that constraints relevant to the ideal solution will also be relevant to a student solution.

$RelevantIS_{c,p}$

$RelSS_{c,}$

	YES	**NO**
YES	α_c	β_c
NO	1-α_c	1-β_c

Table 2. P($RelevantSS_{c,p}|RelevantIS_{c,p}$)

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 and student solutions, 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 initialised to 0.5.

$$\begin{aligned}\alpha_c &= \mathrm{P}(RelevantSS_{p,c} = YES \mid RelevantIS_{p,c} = YES) \\ &= \mathrm{P}(RelevantSS_{p,c} = YES \;\&\; RelevantIS_{p,c} = YES) \,/\, \mathrm{P}(RelevantIS_{p,c} = YES) \\ &= \#\text{ times } c \text{ is relevant to both SS and IS in the logs} \,/\, \#\text{ times } c \text{ is relevant} \\ &\quad\;\text{to IS in the logs}\end{aligned} \qquad \textbf{(1)}$$

$RelevantSS_{c,p}$
$Mastered_c$

$Perf_{c,p}$

	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 3. P($Performance_{c,p}|RelevantSS_{c,p},Mastered_c$)

Table 3 is the conditional probability distribution of $Performance_{c,p}$ given its parent variables $RelevantSS_{c,p}$, and $Mastered_c$. $Slip_c$ ($Guess_c$) is 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 3, P($Performance_{c,p}$ = $NOT\text{-}RELEVANT$) = 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.

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: α_p = 0.9, β_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].

3.3 Evaluating 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 [14] 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 of 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 3. 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 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.

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

Fig. 3. The problem evaluation function.

The evaluation function assumes that feedback will be generated for every constraint where P(*Performance*$_{c,p}$ = *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 0.45 value 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.

4 Evaluation

We performed an evaluation study in October 1999, with second year students enrolled in an introductory database course. The students were randomly assigned to a version of the system with and without the probabilistic student model/problem selector (the control and experimental group respectively). The study consisted of one 2-hour session in which students sat a pre-test, interacted with the system, and then completed a post-test. Timing of the study was a constraint, as students needed to get some overall understanding of databases prior to the study. The only possible time for the study was the last week of the school year, which had a negative effect of the number of participating students.

4.1 Appropriateness of selected problems

All student actions performed in the study were logged, and later used to analyse the effect of the proposed problem-selection approach on learning. Both groups had access to the problem selection methods described in section 2: clicking *next problem*, selecting the problem from a menu, or clicking *system's choice*. In the case of the control group, clicking *system's choice* lead to a problem being selected using the simple heuristic discussed in section 2, while the Bayesian approach was used for the experimental groups.

Average attempts	Exper. group	Control group
Next problem	3.18	2.10
System's choice	2.69	4.55

Table 4. Average number of attempts per solved problem.

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. We counted the number of attempts it took to solve each problem, the averages of which are given in Table 4. The problems selected for the control group by the heuristic were most difficult for students, requiring 4.55 attempts on average to solve. The students in the experimental group were able to solve problems selected by the Bayesian approach in 2.69 attempts on average, compared to 3.18 attempts when problems were visited in turn. The proposed problem selection method compares favourably with the heuristic approach used by the control group.

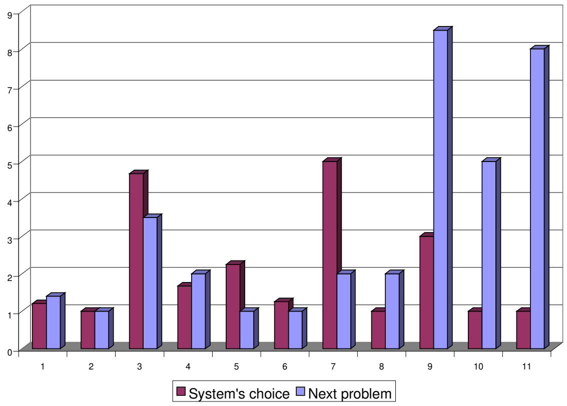


Fig. 4. The average number of attempts to solve *i*th problem by students in the experimental group.

The new *system's choice* method is only slightly better on average than the *next problem* option for the experimental group, but its advantages 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 4 illustrates the average number of attempts students in the experimental group took to solve the *i*th problem. It can be seen that the initial problems selected by *next problem* are easier for students than those selected by the Bayesian approach. This is explained by the fact that the Bayesian approach 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.

4.2 Pre/post tests

Pre- and post-tests 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 5. The mean scores in the pre-test for the two groups are very close, 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 which 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	0.22	0.25
2	2.67	2.73
3	0.62	0.73
Total	3.44	3.50

Table 5. Means for the pre-test

5 Related Work

Other researchers have proposed the use of Bayesian networks in ITSs. ANDES [4,6,7], an ITS for teaching Newtonian physics, uses Bayesian networks for predicting student performance and problem solving behaviour. 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 ideal solution is only one example of a correct solution. The usefulness of the ideal solution in predicting the student's actual solution is determined by the α_c and β_c parameters. Thus, in the SQL domain, we are forced to model the entire domain for each problem.

One approach that does model the entire domain is Collins et al.'s [3] 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 on-line a traditional Bayesian network modeling all the 500 constraints. Furthermore, Collins' example domain of elementary arithmetic divides neatly into 10 categories (e.g. addition theory, subtraction theory etc) that can be easily organised into a hierarchy, whereas in SQL there is no such simple classification of constraints.

Finally, Reye [12] has proposed 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 need to be combined in some way. Reye does not show how this can be done, whereas this is the main emphasis of our paper.

6 Conclusions & Future Work

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 this paper we looked at an existing system for teaching SQL and proposed an improved method for selecting such problems. We use Bayesian networks to predict student performance. Problem value is dependent on the predicted the 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 can, 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 is most likely to generate the optimal number of feedback messages.

Initial evaluations indicate that the proposed solution is promising. However, we have implemented several heuristics due to the inefficiencies of evaluating large Bayesian networks on-line. For example, both Table 1 and Figure 3 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 [1], 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, and other tasks include topic selection, adapting feedback, hint selection, and selective highlighting of text. We are working towards a general framework for solving these type of problems.

Acknowledgments

The work presented here was supported, in part, by the University of Canterbury research grant U6242.

References

1. Chao-Lin Liu & Wellman M. 1998. Incremental Tradeoff Resolution in Qualitative Probability Networks. *Proc UAI-98*, pp. 338-345.
2. Charniak E., 1991. Bayesian networks without tears. *AI Magazine*, Winter 1991, 50-63.
3. Collins J. et al. 1996. Adaptive assessment using granularity hierarchies and Bayesian nets. *Proc. ITS'96,* pp. 569-577.
4. Conati C. et al. 1997. On-Line Student Modeling for Coached Problem Solving Using Bayesian Networks. *Proc. UM97*, pp. 231-242.
5. Corbett A. & Bhatnagar A. 1997. Student Modeling in the ACT Programming Tutor: Adjusting a Procedural Learning Model With Declarative Knowledge. *Proc. UM97*, pp. 243-254.
6. Gertner A. et al. 1998. Procedural help in Andes: Generating hints using a Bayesian network student model. *Proc AAAI-98*, pp. 106-111.
7. Gertner A. 1998. Providing feedback to equation entries in an intelligent tutoring system for Physics. *Proc ITS '98, 254-263.*
8. Mitrovic A. 1998. Experiences in Implementing Constraint-Based Modeling in SQL-Tutor. *Proc. ITS'98,* 414-423.
9. Mitrovic A. & Ohlsson, S., 1999. Evaluation of a constraint-based tutor for a database language, *Int. J. Artificial Intelligence in education*, 10, 3-4, to appear.
10. Ohlsson S. 1994. Constraint-based Student Modeling. In: Greer, J.E., McCalla, G.I. (eds.): *Student Modeling: the Key to Individualized Knowledge-based Instruction*. NATO ASI Series, Vol. 125. Springer-Verlag, Berlin, 167-189.
11. Pearl J. 1988. *Probabilistic reasoning in intelligent systems: networks of plausible inference* (revised 2nd edition). Morgan Kauffman, USA.
12. Reye J. 1998. Two-Phase Updating of Student Models Based on Dynamic Belief Networks. *Proc ITS '98, 274-283.*
13. Schäfer R. & Weyrath T. 1997. Assessing temporally variable user properties with dynamic Bayesian networks. In Jameson A., Paris C. and Tasso C. (Eds.) *Proc. UM'97*, pp. 377-388.
14. Vigotsky L.S. 1978. *The development of higher psychological processes*, Cambridge, MA: Harvard University Press.

Appendix F

IWALT’2000 Paper (Published)

The following paper (Mayo et al, 2000) was published and presented at the IWALT’2000 conference in New Zealand, December 2000. It described the initial version of the CAPIT system and the results of the initial data acquisition study described in parts of Chapter 6

CAPIT: An Intelligent Tutoring System for Capitalisation and Punctuation

Michael Mayo, Antonija Mitrovic and Jane McKenzie

Intelligent Computer Tutoring Group
Department of Computer Science, University of Canterbury
Private Bag 4800, Christchurch, New Zealand
{mmayo, tanja, jane}@cosc.canterbury.ac.nz

Abstract

We describe a new Intelligent Tutoring System (ITS) that teaches the mechanical rules of English capitalisation and punctuation. Students must interactively capitalise and punctuate short pieces of unpunctuated, lower case text (the completion exercise). The system represents the domain as a set of constraints specifying the correct patterns of punctuation and capitalisation, and feedback is given on violated constraints. The ITS was evaluated during several sessions in a classroom of 10-11 year old school children. The results show that the children effectively mastered the 25 rules represented in the system.

1. Introduction

We present CAPIT (Capitalisation And Punctuation Intelligent Tutor), a new Intelligent Tutoring System (ITS) designed for, and evaluated with, school children in the 10-11 year old age group. CAPIT teaches a subset of the basic rules of English capitalisation and punctuation, such as the capitalisation of sentences and the basic uses of commas, periods, apostrophes and quotation marks. Initial indications are that CAPIT motivates children to complete capitalisation and punctuation exercises significantly more so than the traditional approach of using a textbook. We report on results from a user study in which the tutoring system was tested in a real classroom.

2. Punctuation and Capitalisation

Traditional capitalisation and punctuation exercises tend to fall into one of two categories [1]: *completion* (the student must punctuate and capitalise a fully lowercase, unpunctuated piece of text), and *check-and-correct* (the student needs to check for errors, if any, and correct them). Our tutor poses problems of the first class, the completion exercise. If the child makes a mistake, an error message is displayed. For example, Figure 1(a) depicts a short problem in the system.

(a) the teacher said open your books
(b) The teacher said, “open your books”.
(c) The teacher said, “Open your books.”

Figure 1(a). A problem, (b). a student’s incorrect solution, and (c). the correct solution.

Figure 1(b) is an incorrect solution submitted by a student, with two errors: the direct speech does not start with a capital letter, and the period is outside the quotation marks. Currently the system displays only one error message at a time, and the student is expected to correct the error (and possibly any others) and resubmit the problem before any more feedback is displayed. If the student submitted this solution, a feedback message such as “*The full stop should be within the quotation marks! Hint: look at the word* books *in your solution”* would be displayed. Error messages are typically short and relate to only a single mistake, but if the student wants more detailed information, she/he can click *Why?* to be shown further explanatory material. Most of the problems typically are much longer than 1(a) and contain multiple sentences.

Knowledge for the tutor was primarily acquired from capitalisation and punctuation course material used in New Zealand primary schools (e.g. [8]). We also had considerable input from the third author, a professional teacher, who made useful suggestions as to which rules the tutor should cover. For example, a common punctuation error made by children is to insert an apostrophe into the possessive pronoun *its*. Knowledge of common errors like this is more difficult to find in textbooks, and so the input from the teacher was invaluable. The third author also tailored the appropriateness and vocabulary of the system’s feedback, explanatory and introductory messages to the age group.

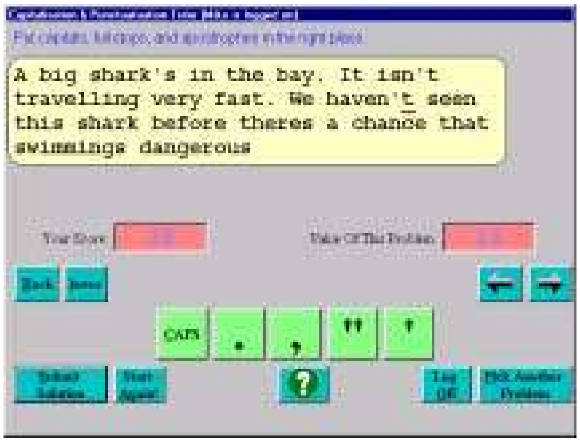
3. Constraint-Based Modelling

CAPIT is the second ITS to implement Ohlsson's Constraint-Based Modelling (CBM) [3], the other being a tutor for the SQL database language [5, 6]. A CBM tutor represents domain knowledge as a set of constraints of the form <C_r, C_s> where C_r is the *relevance condition* and C_s is the *satisfaction condition*. The constraints define which problem states are consistent (or correct), and which are not. A constraint is relevant to a problem if the C_r is true. All relevant constraints must also be satisfied for the problem state to be correct. Otherwise, the problem state is incorrect and feedback can be given depending on which relevant constraints had their satisfaction condition violated.

4. CAPIT

CAPIT's user interface was designed with the target age group of 10-11 year olds in mind. Two issues of importance when designing interfaces for this age group are facileness and motivation.

The main interface is depicted in Figure 2. Brief instructions relevant to the current problem are clearly displayed at the top of the screen. This reduces the cognitive load by enabling the child to focus on the current goals at any time without needing to remember them. Immediately below the instructions, and clearly highlighted, is the current problem. In this area, the child interacts with the system by moving the cursor, capitalising letters, and inserting punctuation marks. The child can provide input either by pointing and clicking the mouse, or by pressing intuitive key combinations such as *Shift-M* to capitalise the letter *m*. By requiring the cursor to be positioned at the point where the capital letter or punctuation mark is to go, the child's ability to locate errors as well as correct them is tested.



Put capitals, full stops, and apostrophes in the right place

A big shark's in the bay. It isn't travelling very fast. We haven't seen this shark before theres a chance that swimmings dangerous

Figure 2. The tutor's main user interface.

Motivation is provided in two forms. Firstly, children accrue points every time they submit a correct solution. The total number of points accrued so far, and the value in points of the current problem, is clearly displayed on the main interface. Secondly, when a correct solution is submitted, a simple animation is displayed as a reward. We have found this to be adequate motivation for 10-11 year olds.

4.1. Architecture

The architecture of CAPIT comprises databases of constraints, problems and student models, the user interface, the student modeller, and the pedagogical module. The interconnections of these components are depicted in Figure 3.

The pedagogical module solves two key decision tasks: it selects the next problem when the child clicks *Pick Another Problem*, and it selects a single error message for display when the child submits an incorrect solution. In the current system, a simple random strategy solves these decision problems.

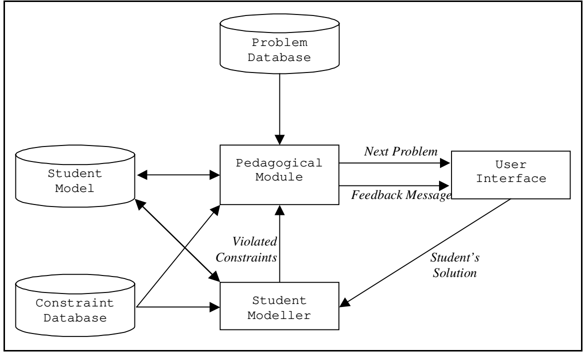


Figure 3. System Architecture

When the user submits a solution, it is passed to the student modeller. The student modeller determines firstly which constraints are relevant to the current solution, and secondly, which constraints are satisfied. The violated constraints are then passed to the pedagogical module so that an error message can be selected.

4.2 Knowledge Representation

In the current version of CAPIT, 45 problems and 25 constraints are represented. The problems are relevant to the constraints in roughly equal proportions, although a small number of constraints (such as capitalisation of sentences) are relevant to all the problems. The constraints cover the following parts of the domain:

- Capitalisation of sentences.
- Capitalisation of the names of both people and places.
- Ending sentences with periods.
- Contracting *is* and *not* using apostrophes.

- Denoting ownership using apostrophes.
- Separating clauses using commas.
- Separating list items using commas.
- Denoting direct speech with quotation marks.
- The correct punctuation of the possessive pronoun *its*.

Problems are represented as arrays of properly punctuated and capitalised words. Each word has one or more *tags* associated with it. The tags specify the semantic and/or grammatical classes of the word, to the degree that it is relevant for punctuation and capitalisation. Figure 4 depicts the tutor's internal representation of a short problem. Each word in this problem has one, two or three tags. The tag `DEFAULT` indicates that a word requires no capitals or punctuation marks. Other tags such as `L-CASE` indicate that a word does not need to be capitalised, but says nothing about the punctuation requirements (and vice-versa for the tag `NO-PUNC`). Other types of word need more specific tags. For example, *The* is the first word in the sentence and therefore carries the tag `SENTENCE-START`. Similarly, *books* is the last word in both the sentence and the direct speech, which is reflected by the tag `DIRECT-QUOTE-ENDING-SENTENCE`.

The	SENTENCE-START,NO-PUNC
teacher	DEFAULT
said,	WORD-PRECEDING-DIRECT-QUOTE, L-CASE,ONE-PUNC
“Open	DIRECT-QUOTE-START,ONE-PUNC
your	DEFAULT
books.”	DIRECT-QUOTE-ENDING-SENTENCE, L-CASE,TWO-PUNC

Figure 4. Problem representation for *The teacher said, “Open your books.”*

<table>
<tr><th></th><th><i>Cr</i></th><th><i>Cs</i></th></tr>
<tr><td>(a)</td><td>{DEFAULT, L-CASE}</td><td>^[a-z0-9%SYMBOLSET%]*$</td></tr>
<tr><td></td><td colspan="2">This word doesn't need any capital letters!</td></tr>
<tr><td>(b)</td><td>{NAME-OF-PERSON}</td><td>^[%SYMBOLSET%]*[A-Z0-9]</td></tr>
<tr><td></td><td colspan="2">Each word in a person's name should start with a capital!</td></tr>
<tr><td>(c)</td><td>{DIRECT-QUOTE-ENDING-SENTENCE}</td><td>[^%SYMBOLSET%]+((\.+"+)|"+|\.+)?$</td></tr>
<tr><td></td><td colspan="2">The full stop should be within the quotation marks!</td></tr>
<tr><td>(d)</td><td>{ITS-POSSESSIVE-PRONOUN}</td><td>[^']s$</td></tr>
<tr><td></td><td colspan="2">No apostrophe is required in <i>its</i>!</td></tr>
</table>

Figure 5. Examples of constraints.

The constraints used in CAPIT comprise three parts: namely, the relevance condition, C_r, which is a set of tags; the satisfaction condition, C_s, which is a regular expression; and a hint that may be shown when the constraint is violated. Figure 5 gives examples of four constraints that range from the very general to the very specific. In addition, most constraints have an associated page of textual explanation that is displayed when the student clicks the *Why?* button.

The tags of each word determine which constraints are relevant to the problem. If at least one word from the problem has a tag that is also in a constraint's C_r, then that constraint is relevant to the problem. For example, constraints 5(a) and (c) are relevant to the problem in Figure 4, but 5(b) and (d) are not.

If a constraint is relevant to a word, then its satisfaction condition, C_s, is evaluated against that word. The main difference between the expressions used in satisfaction conditions and standard regular expressions is the presence of `%SYMBOLSET%`, which refers to a string of all the punctuation marks that the tutor knows about. For example, the current version of CAPIT deals with commas, periods, quotation marks and apostrophes, so the C_s of 5(a) becomes the standard regular expression `^[a-z0-9’”,.]*$` before being matched to a student's solution. More details of the regular expressions used in the system are given in [4].

Constraints in the tutoring system fall into two distinct categories. *General* constraints apply to many different words because they are relevant to general tags such as `DEFAULT` and `L-CASE`. As a result, the feedback is more general and may not address the specific misconceptions that led to the error. Figure 5(a) is one constraint of this class. *Specific* constraints are satisfied only by specific punctuation/capitalisation patterns, and feedback is much more specific in this case. Constraints 5(b), (c) and (d) are examples of these.

5. Evaluation Study

An evaluation of CAPIT was held in March 2000 at Westburn School in Christchurch, New Zealand. A classroom of 28 children in the 10-11 year old age group used the tutor in pairs for four 30-45 minute sessions. In general, the teachers found that CAPIT motivated the children to a high degree.

Details of students' interactions with the tutor (the problems that were attempted, the errors that were made, and the feedback that was displayed) were logged. Subsequent analysis revealed the following averages. Each student made 89 attempts at 28 different problems. The average time for an attempt was 30 seconds, giving a total average interaction time of 45 minutes (equivalent to 90 minutes for each pair of students). 21 of the problems were eventually solved, and 7 abandoned. Students violated an average of 181 constraints during the sessions, of which feedback was given on 68. An interesting observation is that students only asked for more detailed explanations of their errors (by clicking *Why?*) an average of 8 times, or twice per session. The third author noted this during the evaluation study, and suggested that the next version of the tutor should provide some motivation for reading the detailed explanations.

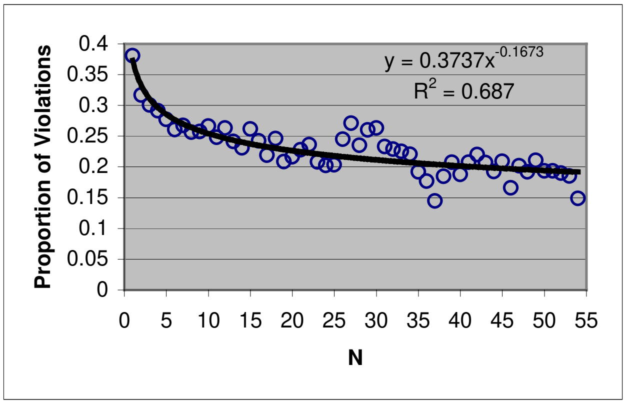


Figure 6. Mean proportion of constraints that were violated on the *n*th attempt.

We analysed how the students learned the constraints by calculating the proportion of violated constraints following the *n*th attempt, averaged across all students and all constraints. The maximum number of attempts was 223, but only one student got this far. Just over half of the students were still using the tutor by the 55th attempt, so the data analysis was concluded at this point. The averages are depicted in Figure 6, and they fit a power trend line with R^2=0.687. A similar result was found in the other CBM tutor [6]. The interpretation of this trend is that the most frequently relevant constraints, such as the constraint that sentences must start with a capital letter, are acquired rapidly initially. More specialised constraints are less frequently relevant and are therefore acquired at a slower rate.

6. Future Work and Conclusions

Bouwer recently described an ITS for Dutch punctuation [1]. There are three significant differences between the two intelligent tutors. Firstly, Bouwer's tutor is targeted at university-level students, and so it focuses on the rhetorical, as well as the grammatical, aspects of punctuation; our tutor is concerned only with the grammar of capitalisation and punctuation. Secondly, his tutor poses check-and-correct as opposed to completion exercises. Thirdly, and most significantly, Bouwer's tutor explicitly represents each possible correct solution for a problem. This means that when new problems are added to the system, even if they use the same rules as existing problems, all their correct solutions solution must also be added. This is a knowledge acquisition bottleneck that is resolved by CAPIT. Constraints also offer a natural representation for multiple correct solutions to the same problem; rather than specifying all possible solutions, the constraint-based tutor localises the ambiguity to specific constraints. For example, a hypothetical constraint specifying the correct separation of hours and minutes when punctuating times could be satisfied by both a colon and a period (e.g. both *11:20* and *11.20* could satisfy the constraint). This ambiguous constraint matches all correct solutions.

The next step in this project will be to implement a pedagogical module with an intelligent, rather than random, decision strategy. We are interested in the application of decision theory to ITSs [2], and plan to learn the structure and probabilities of a Bayesian network from the data collected during this evaluation study. This is in contrast to other proposed and existing architectures that use Bayesian networks (e.g. Reye's model [7]), because the structure of the network will not be fixed a priori. We believe this is an appropriate avenue of future research.

Acknowledgements

The authors would like to thank Misa Mitrovic for evaluating an initial version the tutor, and Westburn School for the use of their computer lab and students.

References

[1] Bouwer A. 1998. An ITS for Dutch Punctuation. In *Proc. ITS-98*, pp. 224-233.

[2] Horvitz E., Breese J. and Henrion M. (1988). Decision Theory in Expert Systems and Artificial Intelligence. *International Journal of Approximate Reasoning* 2:247-302; On WWW at http://www.auai.org/auai-tutes.html

[3] Ohlsson S. 1994. Constraint-based Student Modeling. In: Greer, J.E., McCalla, G.I. (eds.): *Student Modeling: the Key to Individualized Knowledge-based Instruction*, pp. 167-189. NATO.

[4] McMillian M. 1999. *Regular Expressions*. On World Wide Web at http://msdn.microsoft.com/library/periodic/period99/vb99i1.htm

[5] Mayo, M. & Mitrovic, A. 2000. Using a probabilistic student model to control problem difficulty. In *Proc. ITS-2000*, pp. 524-533.

[6] Mitrovic A. & Ohlsson S. 1999. Evaluation of a Constraint-Based Tutor for a Database Language. *Int. J. on A.I. in Education*, 10(3-4), 1999, pp. 238-256.

[7] Reye J. 1998. Two-Phase Updating of Student Models Based on Dynamic Belief Networks. In *Proc ITS '98*, pp. 274-283.

[8] Whinch, G., Blaxell, G. 1994. *Write Well 4.* Horwitz Publications, Australia.

Appendix G

IJAIED Paper (Published)

The following journal paper (Mayo & Mitrovic, 2001) has been published in the *International Journal of Artificial Intelligence and Education*. The article is an extensive description of the methodology and the implementation and evaluation of CAPIT. The bulk of the material in the article is from Chapters 5 and 6, but parts of Chapter 3 also appear.

Optimising ITS Behaviour with Bayesian Networks and Decision Theory

MICHAEL MAYO AND ANTONIJA MITROVIC

Department of Computer Science, University of Canterbury
Private Bag 4800, Christchurch, New Zealand
{mmayo, tanja}@cosc.canterbury.ac.nz

Abstract. We propose and demonstrate a methodology for building tractable normative intelligent tutoring systems (ITSs). A normative ITS uses a Bayesian network for long-term student modelling and decision theory to select the next tutorial action. Because normative theories are a general framework for rational behaviour, they can be used to both define and apply learning theories in a rational, and therefore optimal, way. This contrasts to the more traditional approach of using an ad-hoc scheme to implement the learning theory. A key step of the methodology is the induction and the continual adaptation of the Bayesian network student model from student performance data, a step that is distinct from other recent Bayesian net approaches in which the network structure and probabilities are either chosen beforehand by an expert, or by efficiency considerations. The methodology is demonstrated by a description and evaluation of CAPIT, a normative constraint-based tutor for English capitalisation and punctuation. Our evaluation results show that a class using the full normative version of CAPIT learned the domain rules at a faster rate than the class that used a non-normative version of the same system.

INTRODUCTION

Intelligent tutors must operate with incomplete and usually highly uncertain information about their students. Knowledge about the student's current state (the student model) is necessary for assessment, and more importantly, adaptive pedagogical action selection (PAS). Frequently however, the student's interaction time will be insufficient for an accurate student model to be inferred. Even if the student was interacting for the requisite time, the student's state is likely to be changing so rapidly (and there are so many other external influences) that the student model is never likely to be complete or totally correct. To compensate for this dearth of valid information, intelligent tutors should be equipped with strong methods for handling uncertainty. Such methods should ideally be provably optimal, i.e. given observations about the student, the method should be guaranteed to perform optimal PAS.

No guarantees of optimality are made by other methods, like those developed by Artificial Intelligence (AI) scientists. In fact, even for perfectly certain student models, these methods are not provably optimal because the very mechanisms of their reasoning (e.g. production rules) are open to incompleteness and inconsistencies.

To overcome this potential for sub-optimality, we have investigated general theories of rationality (known as normative theories) and applied these to the design of an intelligent tutor. This approach has two advantages. Firstly, such theories are not usually the product of AI alone, but the product of a collaboration of scientists from many different fields over many years. Therefore, they are likely to be more widely tested and accepted than the typical AI theory. Secondly, the entrenchment of the intelligent tutor in a theory of rationality means that its behaviour will be guided by general principles of rational behaviour. This implies optimality.

We propose statistical decision theory (Savage, 1954), encompassing Bayesian probability theory (Bayes, 1763), as a particular theory of rationality suitable for application to intelligent tutoring systems. It is only recently that efficient and effective structures and algorithms for Bayesian reasoning, known as Bayesian networks (Pearl, 1988), have become available.

Note that normative theories are not domain specific, so they do not specify *what* is being reasoned about. In an intelligent tutor, this is the task of the learning theory. The main advantage that normative theories confer is that the learning theory can be defined within a rational framework. In turn, this means that the learning theory is guaranteed to be optimally applied to the student, with respect to the chosen normative theories. This compares favourably to the architecture of the traditional intelligent tutor in which the learning theory is defined using a less rigorous scheme (e.g. heuristic rules) that lack optimality guarantees.

In this paper, we review Bayesian and decision-theoretic approaches in existing intelligent tutors, and propose a number of desirable features that a decision-theoretic tutor should have.

We then define a general methodology for the development of decision-theoretic PAS strategies for intelligent tutors incorporating these desirable features. The methodology emphasises the collection of real-world data for evaluating and comparing different Bayesian network specifications. This "data-centric" approach contrasts to existing approaches to Bayesian network design, such as the "expert-centric" approach whereby a domain-expert directly or indirectly specifies the structure and maybe also the probabilities of the network, as in ANDES' Assessor network (Conati et. al., 1997; Gertner & VanLehn, 2000); and the "efficiency-centric" approach where the network is pre-specified to some degree to optimise either the specification size (e.g. Millán, 2000) or the efficiency of evaluation (e.g. Collins et. al., 1996; Reye, 1998), and the domain knowledge is "fitted" to this limited specification.

The proposed methodology is applied to the development of optimal PAS strategies for CAPIT (Capitalisation And Punctuation Intelligent Tutor). CAPIT is a constraint-based tutor that teaches the basic rules of English punctuation and capitalisation to 8-10 year old schoolchildren (Mayo et. al., 2000). The two decision-theoretic PAS strategies we have developed using this methodology are problem selection and error message selection. The strategies have been evaluated in the classroom and compared to randomised versions of the same strategies.

DECISION THEORY AND BAYESIAN NETWORKS

Decision theory and Bayesian probability theory are both instances of normative theories. A normative system encompasses not only a set of rules, but also the set of logical consequences of those rules (Gärdenfors, 1989). Therefore they can be considered logically complete and consistent. Under the assumption that a rational agent will act logically, normative systems can be thought of as prescriptive models for rational behaviour. This is in direct contrast to descriptive models that attempt to describe either the behaviour of an individual (such as an expert or a teacher) or a group of individuals (e.g. a psychological theory derived from observations), which may be subject to logical inconsistencies or incompleteness.

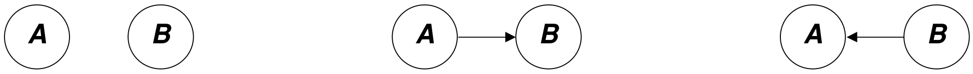
Bayesian probability theory is a set of rules for updating beliefs in an uncertain world. Subjective belief in a proposition such as "variable X is in state x" is represented by the statement $p(X = x) = r$, where $r = 1$ implies that the proposition is certainly true while $r = 0$ implies certain falsehood. A value of r between 1 and 0 indicates the degree of uncertainty between the certain extremes.

To update its beliefs, a Bayesian agent needs to maintain a model of the relationships between its uncertain propositions about the world. Two propositions A and B, for example, are mutually independent if a change in the agent's belief about one proposition does not influence its belief in the other proposition. On the other hand, A and B are dependent if a change in belief about one affects the agent's belief in the other. Dependence is represented by a conditional probability statement such as $p(B|A)$ that defines the agent's "posterior" belief in B given all the possible values of A. It is important to note that this relationship is reversible; given $p(B|A)$, we can always calculate $p(A|B)$ using Bayes' rule (Bayes, 1763):

$$p(A \mid B) = \frac{p(B \mid A)p(A)}{p(B)}$$

Once the relationship between the variables is determined, the joint probability distribution can be calculated. The joint probability distribution defines a table of probabilities, one entry for each different combination of values that the variables can jointly take. For example, if A and B are binary variables, then the joint probability distribution is referred to as $p(A,B)$ and consists of four different probability entries. The sum of all the entries must be 1.

Sometimes it is convenient to represent variables and their dependencies as a directed graph, and this notation is called a Bayesian network. Figure 1 illustrates Bayesian networks for all the possible relationships between two variables.



(a) $p(A,B)=p(A)p(B)$ (b) $p(A,B)=p(A)p(B|A)$ (c) $p(A,B)=p(A|B)p(B)$

Figure 1. Graphical notation depicting (a) two mutually independent variables *A* and *B*, (b) two related variables *A* and *B*, and (c) the same two related variables but with the direction of the arc reversed.

Bayesian networks can be used to model the relationships between observed student actions, student internal states, and outcomes. Figure 2 depicts a simple example of this for illustrative purposes. Note that both *Read Textbook* and *Watched Video* are set with certainty to values `YES` and `NO`. This is called instantiation, and implies that these variables have been observed. When an uninstantiated node is queried, its probability distribution must be updated to incorporate all the currently instantiated nodes in the network. Lauritzen and Spiegelhalter (1988) describe an efficient and effective updating algorithm in common use.

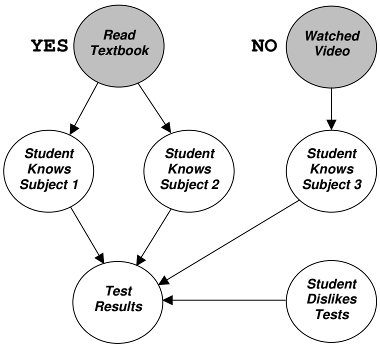


Figure 2. A hypothetical Bayesian network for predicting the performance of a student on a test.

Although the direction of any arc can be reversed, for practical purposes a Bayesian network cannot have any directed cycles. Furthermore, while it is a common convention that the arc directionality corresponds to the direction of causality, it is possible to apply other semantics to arc directionality. For example, Collins et. al. (1996) interpret arcs as emanating from topic to subtopic.

Whereas Bayesian networks are used to update beliefs from initial beliefs and observations, decision theory is a rational means of optimising behaviour by “fusing” uncertain beliefs with preferences. Suppose the agent is faced with the problem of selecting a single action d_i from a set of possible actions $d_1, d_2 \ldots, d_n$. If x is a possible outcome of d_i, then decision theory requires the agent to specify a real-valued preference U(x, d_i) for each possible combination of x and d_i. This is called the utility function. The agent must also be able to estimate the probability of x should it opt for d_i, a value that can be determined from its Bayesian network. The expected utility of d_i is defined, therefore, as the probability-weighted sum of the utilities of each possible outcome less the cost of the action:

$$E[U(x,d_i)] = \left(\sum_x p(x \mid d_i)U(x,d_i)\right) - cost(d_i) \qquad (1)$$

The principle of maximising expected utility says that the agent should select the action d_i that maximises Equation 1.

It is also relevant to mention the foundations of decision theory. Bayesian probability was introduced over 200 years ago, and decision theory was first proposed in 1954 (Savage, 1954). There has been ample time, therefore, for these models to be widely tested. The fact that they are now accepted and applied in a variety of fields is a testament to their rigorous foundations. Intelligent tutors built on these fundamentals, therefore, will be widely accepted, a vision espoused by Everson (1995).

One reason for the neglect of these theories in Artificial Intelligence and related fields, however, was the problem of tractability (Jensen, Lauritzen et al., 1990). In its naïve form, Bayesian probability theory and decision theory are intractable. This led to the development of other schemes, such as fuzzy sets (Zadeh, 1983) and Dempster-Shafer theory (Shafer, 1986), which are not as general as normative theories, but are highly tractable (Horvitz et al., 1988). Since then, however, recent advances in the tractability of the normative reasoning have been made and researchers in these areas are showing renewed interest in normative models.

A noteworthy property of Bayesian networks is that both prior/expert knowledge and data can be seamlessly integrated within a single network. For example, an expert can specify some or all of a Bayesian network, data can be used to learn the rest of it, and then the expert can "fine-tune" the final version. This property is not typical of other representations such as neural networks, and is a significant, natural property of Bayesian networks that will be demonstrated in this paper.

For more technical details, the interested reader is referred to a number of general tutorials on Bayesian networks (D'Ambrosio, 1999; Cowell, 1999). Mislevy & Gitomer (1996) introduce Bayesian networks in the context of intelligent tutors. The learning of Bayesian networks from data is an important component in this paper, and suitable tutorials are provided by Heckerman (1999) and Krause (1998). Decision theory and AI are introduced by Horvitz et. al. (1988) and Russell & Norvig (1995, Ch. 16-17).

NORMATIVE TECHNIQUES IN EXISTING TUTORS

A number of recent intelligent tutors have been proposed with Bayesian network student model. In this section, we review the different approaches to Bayesian student modelling, and then we discuss some of the different applications of the student model to PAS.

Bayesian Network Student Modeling

It is possible to classify Bayesian network student models into three different groups, according to the technique by which they were constructed. *Expert-centric* student models are unrestricted products of domain analysis. That is, an expert specifies either directly or indirectly the complete structure and conditional probabilities of the Bayesian student model, in a manner similar to that with which expert systems are produced. This is the general approach of ANDES (Gertner & VanLehn 2000; Gertner et. al., 1998; Gertner, 1998; Conati et. al., 1997), HYDRIVE (Miselvy & Gitomer, 1996), DT-Tutor (Murry & VanLehn, 2000), and the Bayesian domain model of ADELE (Ganeshan et. al., 2000). One possible disadvantage of this approach is that the resulting models may include so many variables that it becomes infeasible to evaluate the network effectively on-line. For example, tractability testing was an important issue in the initial evaluation of DT-Tutor. *Efficiency-centric* models, on the other hand, work the other way: the model is partially specified or restricted in some way, and domain knowledge is "fitted" to the model. The restrictions are generally chosen to maximise some aspect of efficiency, such as the amount of numeric specification required and/or the evaluation time. This is the methodology of Reye (1998), Murray (1998), Collins et al (1996), Mayo & Mitrovic (2000), and to a degree, Millán et. al. (2000). In general, restrictions to increase efficiency can introduce incorrect simplifying assumptions about the domain. Finally, the *data-centric* model is a new class of Bayesian student model, introduced and implemented in this paper, in which the structure and conditional probabilities of the network are learned primarily from data. This class of student model dispenses with attempting to model unobserved student states, such as their domain mastery, and instead concentrates to modelling the relationships between observed variables to predict student performance. MANIC (Stern et. al., 1999) is the closest existing system the authors could find to the data-centric approach, but it learns only the probabilities and not the structure of the network, and is therefore more efficiency-centric than data-centric. Work in this area is also described by Beck & Woolf (2000), but Bayesian networks are not used. Figure 3 shows how existing Bayesian network student models fit this classification.

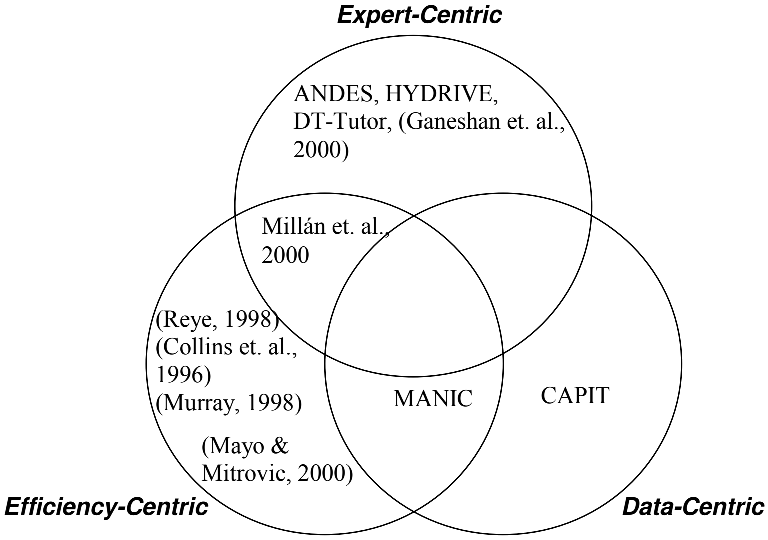


Figure 3. A classification of Bayesian network student models.

Expert-Centric

ANDES (Gertner & VanLehn 2000; Gertner et. al., 1998; Gertner, 1998; Conati et. al., 1997), HYDRIVE (Miselvy & Gitomer, 1996) and DT-Tutor (Murray & VanLehn, 2000), are examples of tutors with large Bayesian networks with structures mostly engineered from complex domain analysis. To match the domains as closely as possible, their networks are not structurally restricted in any way. However, they all have a high proportion of variables representing unobserved, internal student states. A major hurdle for these systems, then, is how conditional probabilities can be elicited or defined for these variables in the absence of data.

ANDES' solution is to use "coarse-grained" conditional probabilities definitions such as noisy-OR and noisy-AND. A noisy-OR variable has a high probability of being true only if at least one of its parents is true, and similarly for noisy-AND variables. In practice, restricting conditional probabilities to noisy-ORs and noisy-ANDs significantly reduces the number of required probabilities and makes the modelling of unobserved variables much simpler because only the structure and node type (noisy-AND or noisy-OR) needs to be specified.

In HYDRIVE, the conditional probabilities are defined subjectively in a "fuzzy-like" fashion. For example, a student's *Strategic Knowledge* takes the vague linguistic values `expert`, `good`, `okay` and `weak`. Tutorial actions and observations of student behaviour modify the probability distribution over these values via conditional probabilities, which were elicited from domain experts.

Finally, DT-Tutor is a generalised domain-independent architecture for student modeling and PAS. Like ANDES and HYDRIVE, it models the student's knowledge, but it goes much further and attempts to model other hidden states such as the student's morale, independence, and focus of attention. A preliminary version of this system has been constructed but no details have been given as yet to how the conditional probabilities will be obtained.

Models that largely represent unobserved, internal student states suffer a major disadvantage: the model structure and/or parameters cannot be adapted on-line to the student. To illustrate, consider a hypothetical very simple Bayesian network with discrete variables, *Observations* and *Student State*. Suppose that the system maintains the conditional probability *p(Student State|Observations)* for computing the posterior probability distribution over the hidden *Student State* from the observable variable *Observations*. This, in a highly abstract form, is how the student models in ANDES and HYDRIVE operate. Now, consider how this model can be adapted to the student. There are two different approaches. The first is to observe the student and instantiate *Observations*, and then update the value of *Student State* from this. This is the standard way in which Bayesian networks are used, but is means that *p(Student State|Observations)* will remain static and that the previous value of *Observations* will be lost. An alternative approach is based on machine learning, and involves modifying *p(Student State|Observations)* itself. If a particular value of *Observations* leads to a particular *Student State*, then *p(Student State|Observations)* is altered to increment slightly the probability that the same *Student State* will be observed again when the same or similar *Observations* are made again in the future.

However, this second approach relies on *Student State* being an observable variable, something that it is not in our simple model. This is an important reason for advocating models that eliminate hidden variables: they are simply more adaptable. Consider an equally simple model defining the relationship between two observable variables, *Observations* and *Next-Observations*. Because the variables are both observable, a conditional probability such as *p(Next-Observations|Observations)* becomes amenable to machine learning, and therefore the system is more adaptable.

Efficiency-Centric

An approach to student modelling using dynamic Bayesian networks (DBNs, Russell & Norvig, 1995, Ch. 17) has been proposed by Reye (1998). Reye's model is a generalisation of the student model used in the ACT Programming Languages Tutor (Corbett & Anderson, 1992; Corbett & Bhatnagar, 1997), and a similar approach was used in the student model of SQL-Tutor (Mayo & Mitrovic, 2000). The idea is to model the student's mastery of a knowledge item over time. The tutor's current belief that the student has mastered the item (M_t) depends on its previous belief (M_{t-1}), the outcome of the student's last attempt at the item (O_{t-1}), and the pedagogical response of the tutor to the last attempt (A_{t-1}). Using dynamic Bayesian networks, not only can the tutor's current beliefs be determined, but also its future beliefs at time *t+1* or beyond, although this is likely to be much more uncertain. This model is depicted in Figure 4 for a single knowledge item.

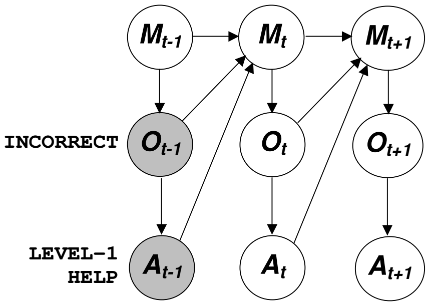


Figure 4. A DBN modelling the mastery of the student on a single knowledge item equivalent Reye's approach. At time *t-1*, the student failed an attempt at the item and so the tutor provided remedial help.

One problem with this approach is that the complexity-reducing assumption that mastery of a knowledge item is probabilistically independent of the mastery of any other items is unrealistic. Suppose, for example, that the knowledge items are "high level" such as concepts or topics. Then we would expect the mastery of some items to be dependent on the mastery of items that are pre- and co-requisites. This is a basic assumption of many systems and the rationale behind many approaches to course sequencing, e.g. Brusilovsky (2000). Alternatively, the knowledge items could be "low level" such as constraints (Ohlsson, 1994; Mitrovic & Ohlsson, 1999; Mayo et. al., 2000). Clearly, we would expect many dependencies between constraint mastery based on factors such as syntactic and/or semantic relatedness. We demonstrate later that in the punctuation domain, a model with dependencies between items makes better predictions of student performance than a simpler model similar to Figure 4.

There are Bayesian student models that allow some dependencies to be expressed whilst remaining efficiency-centric. They are the singly-connected hierarchical structures described by Murray (1998), Collins (1996), and Stern et. al. (1999). A singly-connected network has the property that for every pair of nodes in the network, there is one and only one path between the nodes. Bayesian networks with a singly-connected topology evaluate in linear time (Pearl, 1988; Murray, 1999), and while they can express dependence between knowledge items, the singly-connected assumption means that certain types of dependence (namely, undirected loops) cannot be represented. This is clearly a strong restriction, because all of the expert-centric models described above contain undirected loops in their Bayesian networks.

The problems of single-connectedness are illustrated by MANIC (Stern et. al., 1999), which attempts to learn the probabilities (but not the structure) of its hierarchy from observations of the student. MANIC's hierarchical structure explicitly assumes that its variables are independent of each other given their mutual parent variable. Unfortunately, the data acquired from students directly contradicted this and Stern et. al. were forced to compensate by introducing several ad-hoc "fixes" to the network, such as "merging" dependent nodes and deleting irrelevant nodes. This jeopardised its normative status. A clear solution to this problem would have been to drop the restriction that the network was a hierarchy, although this would have led to a more complex model and the necessity for more complex learning algorithms.

Interestingly, Millán et. al. (2000) recently proposed an architecture that is to a degree both expert- and efficiency-centric. Their Bayesian network is selected to optimise the amount of numeric specification required, and to achieve this, the directionality of the arcs between groups of variables is fixed. The variables are also limited to binary states with specific semantics. However, the topology of the network does not have to be singly-connected which makes it quite flexible.

Data-Centric

This is the approach whereby both the structure and conditional probabilities of the network are learned from data collected from real-world evaluations of the tutor. There are a number of benefits of this approach. Firstly, because the model is induced from actual data, its predictive performance can easily be evaluated by testing the network on data that was not used to train it. Secondly, data-centric models can be expected to be much smaller than the typical expert-centric model because the latter represents both observed and hidden variables, while the former models only observable variables.

This is not to say that data-centric models may not contain hidden variables. If many observable variables are "compressed" in some way into a single hidden variable, then the resulting student model with hidden variables will in fact be smaller than the original model without hidden variables. However, there are some difficult issues to deal with. For example, how is the compression to be performed and more importantly, will the resulting hidden variables be consistent with the original observable variables? In a Bayesian system, it follows that probabilistic inference should be used to deduce the probability distribution over the new hidden variables from the original observable variables. If this is not done, the new compressed model will be probabilistically inconsistent with the original model. However, the additional computation required to maintain this consistency may well offset the space-saving that the compression afforded, rendering the whole process futile. On the other hand, there are theoretically sound methods of compressing Bayesian networks by introducing hidden variables

(Heckerman, 1999). The advantage of adding hidden variables to a Bayesian network is that, if the hidden variables are defined carefully, then the number of arcs in the network can be reduced as a result, without a significant corresponding decrease in the accuracy of the network. If such an algorithm were applied to a Bayesian network student model however, there is no guarantee that the hidden variables will be semantically meaningful (i.e., they might not correspond to states such as concept mastery).

Pedagogical Action Selection

Given a Bayesian student model, the next issue is how to use the model to optimise the pedagogical actions of the intelligent tutor. Unfortunately, only a handful of papers describe how their Bayesian student models are actually applied to a task other than assessment. Of those that do, there seem to be three general approaches: *alternative* strategies, *diagnostic* strategies, and *decision-theoretic pedagogical* strategies. The three classes and the systems that fall into them are given in Table 1.

Table 1. Decision-making with the student model.

Alternative	Diagnostic	Decision-Theoretic
ANDES Ganeshan et. al., 2000 SQL-TUTOR	Millán et. al., 2000 Collins et. al., 1996	DT-TUTOR CAPIT

Alternative strategies

Alternative strategies optionally take the posterior probabilities of the Bayesian network and use them as the input to some heuristic decision rule. To illustrate, ANDES selects hints for the student based on the solution path that the student is following to solve the current problem (Gertner et. al., 1998). However, the student's solution path is by no means certain (e.g. the student could be on paths *A*, *B* or *C* with posterior probabilities *p(A)*, *p(B)*, and *p(C)*), and therefore the system uses the heuristic of assuming that the most probable solution path (e.g. *A*, assuming *p(A)>p(B)* and *p(A)>p(C)*) *is* the student's solution path. However, this is a sub-optimal heuristic as demonstrated by a simple counter-example. Suppose the optimal hint for solution path *A* is H_1, but the optimal hint for both paths *B* and *C* is H_2. Then if it is the case that $p(B) + p(C) > p(A)$, hint H_2 will be optimal, but the heuristic rule will incorrectly select hint H_1. ANDES also has heuristic decision procedures disconnected entirely from the student model. For example, a simple matching heuristic is used to generate feedback on incorrect equation entries (Gertner, 1998).

Another system using heuristic decision procedures is ADELE (Ganeshan et. al., 2000). ADELE has a Bayesian network model of the domain knowledge, but it uses a heuristic based on focus-of-attention to select the node in the network about which to provide a hint. Decision-theoretic processes were considered but abandoned because they were considered too inefficient.

Finally, SQL-Tutor uses a heuristic for problem selection (Mayo & Mitrovic, 2000). The main rationale for this was that, like ADELE, the computation required for exact decision-theoretic computation (which would have involved more than 500 constraints) made direct application of decision theory intractable. The heuristic used was based on Vigotsky's Zone of Proximal Development (Vigotsky, 1978), and did tend to select problems of an appropriate complexity level efficiently. However, this approach is not guaranteed to select the optimal problem.

Diagnostic Strategies

This is the approach of Millán et. al. (2000), which expands on the strategy suggested by Collins et. al. (1996). The basic idea is to select actions whose outcomes are likely to maximise the posterior precision of some node in the network. For example, Millán et. al.'s domain is test question selection, and questions are selected to maximise the system's certainty that the student has mastered the domain concepts. This strategy has limited applicability outside of diagnostic tests.

Decision-Theoretic Pedagogical Strategies

Decision-theoretic strategies are utilised in both DT-Tutor (Murray & VanLehn, 2000) and CAPIT (this paper). Both systems select tutorial actions that maximise expected utility (Equation 1). While diagnosis is obviously an important component of expected utility maximisation, it is only a secondary component. The primary consideration of an expected utility calculation is the likely outcomes of the action, and their pedagogical utility. For example, in CAPIT as shall be described, the expected utility of an action (e.g. problem selection) depends on the likely outcomes of the action (e.g. how many errors are made). In DT-Tutor, the action's impact on many different factors related to the student (e.g. their morale, etc) has an influence on expected utility. Diagnosis, therefore, is only required to the extent that it discriminates between alternate actions. The key difference between the two systems is, as Figure 3 depicts, DT-Tutor has a static, expert-centric student model whereas CAPIT has a data-centric student model that can adapt on-line. This impacts on action selection because the crucial $p(x|d_i)$ component of Equation 1 is evaluated using the Bayesian model.

Summary: Desirable Features

To summarise, there are a number of desirable features of decision-theoretic tutors. The first obvious desirable feature is to select pedagogical actions according to pure decision-theoretic principles rather than heuristics. This, combined with a Bayesian student model, means that the system will be fully normative and therefore its behaviour will be optimal. Secondly, the data-centric approach has two key advantages: the specification size of the network is smaller, and its predictive performance can be readily evaluated. This data-centric approach is therefore an attractive approach. The approach of MANIC (Stern et. al., 1999), in which the probabilities of the Bayesian network are initialised from population data (the "population student model") and subsequently adapted on-line to the current student, was a first step in this direction. A natural extension to MANIC's approach is to abandon the assumption that the student model is a hierarchy, and instead learn its structure as well as its conditional probabilities from data. The methodology presented in the next section shows how to develop just such a system with these desirable features.

A METHODOLOGY FOR DEVELOPING RATIONAL PAS STRATEGIES

In this section, one approach to building a normative intelligent tutoring system is described. While this is the general approach used to build CAPIT, it is by no means the only approach. The approach is described as a five-step methodology. The main point to note is that in the first step, a version of the tutor with no intelligent decision-making capabilities is deployed in a classroom. The point of this is to collect data describing the behaviour of students in the domain. All the student actions and system responses are logged, and then machine learning techniques are then used to induce a Bayesian network model from this data. In turn, the Bayesian network model is the basis for the decision-theoretic PAS strategies. Table 2 illustrates the process, although like any engineering process, the order of the steps is by no means fixed. The steps are now discussed in more detail.

Table 2. The five-step methodology for designing decision-theoretic PAS strategies.

1	Randomised Data Collection
2	Model Generation
3	Decision-Theoretic Strategy Implementation
4	On-line Adaptation
5	Evaluation

The first step is randomised data collection, in which an almost fully-functional version of the tutor is tested in a classroom representative of the intended population of users of the system. The only difference between this and the final version of the tutor is the PAS strategy: this version's PAS strategy is random. That is, given a set of alternatives (such as unsolved problems), the tutor makes the selection completely randomly. All actions should be logged as records of form <*State, Action, Outcome*>, where *State* is a description of some state prior to the action selection (e.g. the state of the student model, or the recent history of the student, or a combination thereof), *Action* is the pedagogical action that is randomly selected (e.g. the next problem), and *Outcome* is the observed outcome(s) of the action (e.g., correct or incorrect). Because PAS selection is random, the data should be uniformly spread over all the possible actions.

The next step is model induction, the construction of a Bayesian network for predicting student performance (the outcomes) given a state and an action. The data from Step 1 serves as the source from which the model is induced. At this stage, prior and expert knowledge can be added to the network. This can be done either before learning by adding dependencies and probabilities between the variables, or after learning, by fine-tuning the induced network. There is at least one Bayesian network learning algorithm that can cope with this type of prior knowledge (Cheng et. al., 1998). However, the rationale for any decision at this stage should be to enhance predictive performance. Also, because the network is being learned from data, the variables can only represent observations.

The third step is implementation of the decision-theoretic strategy. This is an encoding of Equation 1 to design a procedure that selects pedagogical actions that maximise expected utility. Equation 1 has two main components, the utility function $U(x,d)$, and the conditional probabilities, $p(x|d)$, for each possible outcome x of each potential next action d. The Bayesian network constructed in the previous step is used to provide the outcome probabilities, $p(x|d)$. However, the utility function is not yet defined. In fact, it is at this point that learning theories are incorporated into the system. The utility function essentially defines a learning theory for a particular task. To illustrate, if d represents a possible next problem and x is the number of errors the student makes when attempting the problem, then $U(x,d)$ can be defined to be maximal for some optimal number of errors. This is exactly the strategy used in one of the decision strategies in CAPIT, and will be discussed in more detail later.

Step four involves implementing an on-line Bayesian network learning algorithm. In Step 2, the Bayesian network is constructed from population data. Stern et. al. (1999) refer to this as a "population student model". However, as data is acquired directly from the current student, the population data should be gradually discounted. Additionally, as the student state changes over time, older data acquired from the student will need to be discounted as well. While there are a number of existing algorithms for Bayesian network induction from data, there is little in the way of *on-line* Bayesian network induction algorithms. Furthermore, the online learning algorithms that do exist (e.g. Heckerman, 1999; Bauer et. al., 1998) make the assumption that the data-source is essentially static and unchanging over time, in direct contrast to an actual student whose state changes constantly. In our application of the methodology to CAPIT, therefore, we describe a modification to an existing algorithm for on-line conditional probability learning, and avoid the much more difficult problem of updating a network's structure on-line.

Finally, the fifth step is an evaluation of the decision-theoretic PAS strategy. This is necessary to ensure that the decision-theoretic strategies actually provide a pedagogical benefit for the extra computational effort they require. One strategy that requires virtually no computational effort is the randomised PAS strategy implemented for Step 1. We therefore advocate evaluating decision-theoretic and randomised PAS in a controlled experiment in which one group of students (the control group) use a version of the tutor with the original, randomised strategy, and the second group of students (the experimental group) use tutor version with the decision-theoretic strategy. We have performed this evaluation with CAPIT.

CAPIT: AN INTELLIGENT TUTOR FOR CAPITALISATION AND PUNCTUATION

CAPIT (Mayo et. al. 2000) is an intelligent tutor implemented in Visual Basic 6. It runs on any 32-bit Windows platform, and use the MSBN API provided by Microsoft for its Bayesian networks implementation (http://research.microsoft.com/msbn).

It is also the second intelligent tutor to implement Ohlsson's Constraint-Based Modelling (CBM) (Ohlsson, 1994), the other being a tutor for the SQL database language (Mitrovic & Ohlsson, 1999). CBM was proposed in part because of the intractability of modelling approaches that try to infer students' mental processes from problem solving steps, and in part because Ohlsson believes that diagnostic information is most readily available in the problem states that the student arrives at. It is also computationally highly efficient.

A CBM tutor represents domain knowledge as a set of constraints of the form <C_r, C_s> where C_r is the *relevance condition* and C_s is the *satisfaction condition*. The constraints define which problem states are consistent, and which are not. A constraint is relevant to a problem if its C_r is true. All constraints that are relevant to a problem state must also be satisfied for the problem state to be correct. Otherwise, the problem state is incorrect and feedback can be given depending on which relevant constraints had their satisfaction condition violated.

Traditional capitalisation and punctuation exercises for children tend to fall into one of two categories (Bouwer, 1998): *completion* (the student must punctuate and capitalise a fully lowercase, unpunctuated piece of text), and *check-and-correct* (the student needs to check for errors, if any, and correct them). CAPIT poses problems of the first class, the completion exercise. If the child makes a mistake, an error message is displayed. For example, Table 3 depicts one of the shorter problems in the system, a student's incorrect attempt at punctuating and capitalising it, and the tutor's correct solution.

Table 3. (a) A problem, (b) a student's incorrect solution, and (c) the correct solution.

(a) the driver said it will rain
(b) The driver said, “it will rain”.
(c) The driver said, “It will rain.”

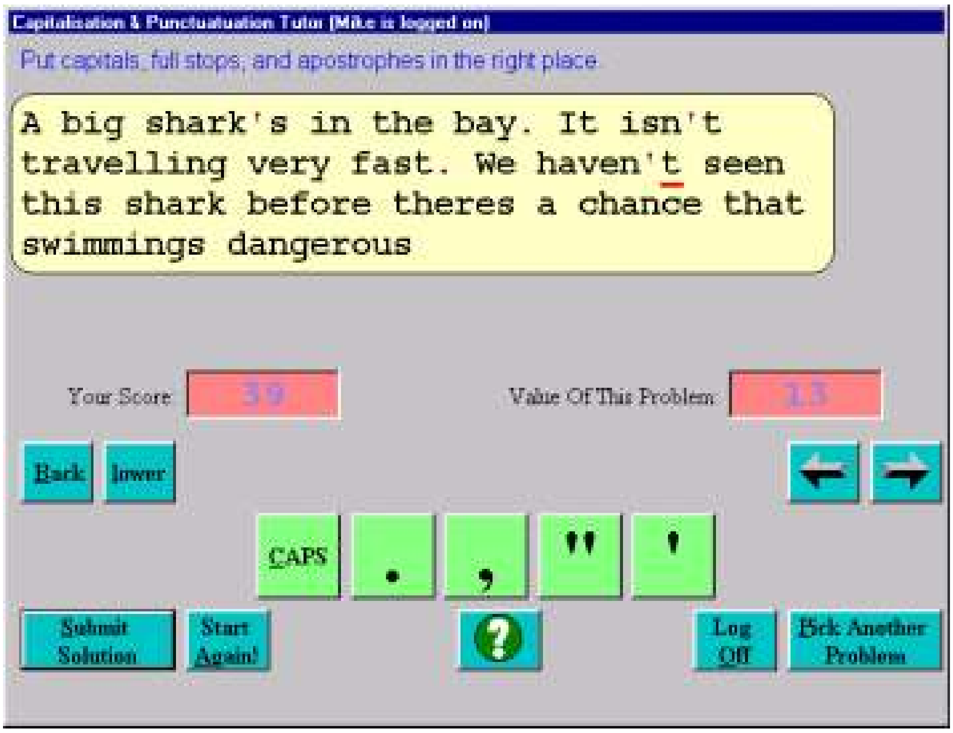
There are two errors in the student's solution in Table 3: the direct speech does not start with a capital letter, and the period is outside the quotation marks. Currently, CAPIT displays only one error message at a time, and the student is expected to correct the error (and any others) and resubmit the problem before any more feedback is displayed. If the student submitted this solution, a feedback message such as *The full stop should be within the quotation marks! Hint: look at the word* rain *in your solution* would be displayed. Error messages are typically short and relate to only a single mistake, but if the student wants more detailed information, she/he can click *Why?* to be shown further explanatory material.

The current version of CAPIT contains 45 problems and 25 constraints. The problems are relevant to the constraints in roughly equal proportions, although a small number of constraints (such as capitalisation of sentences) are relevant to all the problems. The constraints cover the following parts of the domain:

- Capitalisation of sentences.
- Capitalisation of the names of both people and places.
- Ending sentences with periods.
- Contracting *is* and *not* using apostrophes (e.g *haven't* is a contraction of *have not*).

- Denoting ownership using apostrophes (e.g. *John's dog*).
- Separating clauses using commas.
- Separating list items using commas (e.g. *apples, oranges, lemons and pears*).
- Denoting direct speech with quotation marks.
- The correct punctuation of the possessive pronoun *its*.

While the domain coverage is not complete, it is adequate to make CAPIT an intelligent tutoring system with practical application. In our evaluation study, classes used the tutor over a period of one month. While some students quickly mastered all the rules, most of them failed to master all the rules by the end of the evaluation.



Capitalisation & Punctuatuation Tutor [Mike is logged on]

Put capitals, full stops, and apostrophes in the right place

```
A big shark's in the bay. It isn't
travelling very fast. We haven't seen
this shark before theres a chance that
swimmings dangerous
```

Your Score: 39

Value Of This Problem: 13

Figure 5. CAPIT's main user interface.

CAPIT's main user interface, showing a partially completed problem, is depicted in Figure 5. Brief instructions relevant to the current problem are clearly displayed at the top of the main interface. This reduces the cognitive load by enabling the learner to focus on the current goals at any time without needing to remember them. Immediately below the instructions, and clearly highlighted, is the current problem. In this area, the child interacts with the system by moving the cursor using keyboard or mouse, capitalising letters, and inserting punctuation marks. The child can provide input either by pointing and clicking the mouse, or by pressing intuitive key combinations such as *Shift-M* to capitalise the letter *m*. By requiring the cursor to be positioned at the point where the capital letter or punctuation mark is to go, the child's ability to locate errors as well as correct them is tested.

Motivation is provided in two ways. Firstly, whenever a correct solution is submitted, some points are added to the child's score. The number of points added is equal to the number of punctuation marks and capital letters in the solution that was just submitted. Secondly, whenever a correct answer is submitted, an animation is displayed. These simple strategies were found to be highly effective motivators for children in the target age group of 8-10 year olds.

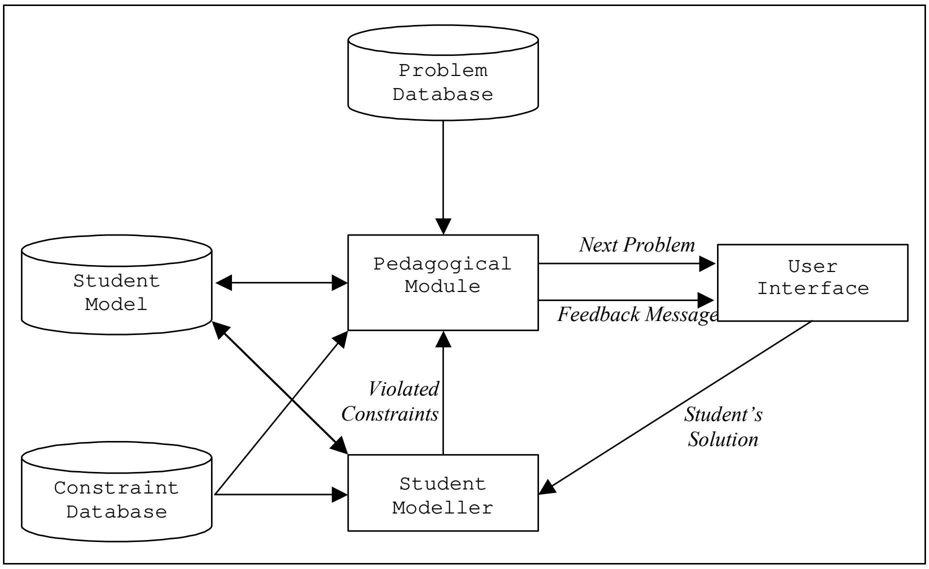


Figure 6. The architecture of CAPIT.

Figure 6 shows the architecture of CAPIT. The student model comprises a record of the outcome of the previous attempt at each constraint[1] (the short-term model) and the current configuration of the Bayesian student model (the long-term model). The student modeller is a pattern matcher that takes the student's solution to a problem and determines which constraints are violated. It then passes the violated constraints (if any) to the pedagogical module. The pedagogical module is the core component of the system. It performs two significant PAS tasks: firstly, given the violated constraints, it selects the single violated constraint about which feedback should be given; secondly, when *Pick Another Problem* is clicked, or when the student solves the current problem, the pedagogical module selects the most appropriate next problem for the student. The current version of the pedagogical module can perform PAS in two ways: randomly, or using decision theory. More details of the decision-theoretic strategies are given later.

Problems in CAPIT are represented as arrays of words. Each word in the problem representation is properly punctuated and capitalised, and the tutor generates the initial problem text by removing the punctuation marks and turning all capital letters into lowercase. Each word also has one or more *tags* associated with it. The tags specify the semantic and/or grammatical classes of a word, to the degree that it is relevant for punctuation and capitalisation. For example, Table 4 is the tutor's internal representation of a short problem. Each word in this problem has one, two or three tags. The tag DEFAULT indicates that a word does not need to be punctuated or capitalised (although the student may incorrectly attempt to do so), such as *driver* and *will* in the example. Because DEFAULT completely specifies the capitalisation and punctuation requirements, DEFAULT words do not require any other tags. Other tags such as L-CASE indicate that a word does not need to be capitalised, but says nothing about the punctuation requirements (and vice-versa for the tag NO-PUNC). Other types of words need more specific tags. For example, *The* is the first word in the sentence and therefore carries the tag SENTENCE-START. Similarly, *rain* is the last word in both the sentence and the direct speech. This fact is reflected by one of its tags, DIRECT-QUOTE-ENDING-SENTENCE. A longer example, which is more representative of the complexity of the 45 problems in the database, is given in Table 5.

[1] An "attempt at a constraint" in this context means an attempt at a problem whose solution is relevant to the constraint.

Table 4. Problem representation for *The driver said, "It will rain."*

The	SENTENCE-START,NO-PUNC
driver	DEFAULT
said,	WORD-PRECEDING-DIRECT-QUOTE,L-CASE, ONE-PUNC
"It	DIRECT-QUOTE-START,ONE-PUNC
will	DEFAULT
rain."	DIRECT-QUOTE-ENDING-SENTENCE,L-CASE, TWO-PUNC

Table 5. Representation of a more complex problem.

There's	SENTENCE-START,IS-CONTRACTION,ONE-PUNC
a	DEFAULT
bee	DEFAULT
buzzing	DEFAULT
past	DEFAULT
me.	SENTENCE-END,ONE-PUNC,L-CASE
It's	SENTENCE-START,IS-CONTRACTION,ONE-PUNC
taking	DEFAULT
its	ITS-POSSESSIVE-PRONOUN,NO-PUNC,L-CASE
honey	DEFAULT
back	DEFAULT
to	DEFAULT
its	ITS-POSSESSIVE-PRONOUN,NO-PUNC,L-CASE
hive.	SENTENCE-END,ONE-PUNC,L-CASE
I	SENTENCE-START,NO-PUNC
hope	DEFAULT
it	DEFAULT
knows	DEFAULT
its	ITS-POSSESSIVE-PRONOUN,NO-PUNC,L-CASE
way	DEFAULT
home.	SENTENCE-END,ONE-PUNC,L-CASE

Table 6. Examples of constraints.

	C_r	C_s	Msg
(a)	{DEFAULT, L-CASE}	^[a-z0-9%SYMBOLSET%]*$	This word doesn't need any capital letters!
(b)	{NAME-OF-PERSON}	^[%SYMBOLSET%]*[A-Z0-9]	Each word in a person's name should start with a capital!
(c)	{DIRECT-QUOTE-ENDING-SENTENCE}	[^%SYMBOLSET%]+((\.+"+)|"+|\.+)?$	The full stop should be within the quotation marks!
(d)	{ITS-POSSESSIVE-PRONOUN}	[^']s$	No apostrophe is required in its!

The constraints used in CAPIT comprise three parts: namely, the relevance condition, C_r, which is a set of tags; the satisfaction condition, C_s, which is a regular expression; and associated explanatory material. Table 6 gives examples of four out of the 25 constraints that range from the very general to the very specific. The error message field of each constraint shows only the hint that is displayed when the constraint is violated; in addition, most constraints have an associated page of textual explanation that is displayed when the student clicks the *Why?* button.

The tags of each word determine which constraints are relevant to the problem. If at least one word from the problem has a tag that is also in a constraint's C_r, then that constraint is relevant to the problem. For example, constraint (a) from Table 6 is relevant to the problem in Table 4 because several words have the tags DEFAULT and L-CASE. Similarly, constraint (c) is relevant to the same problem because the last word has the tag DIRECT-QUOTE-ENDING-SENTENCE. Constraints (b) and (d) are not relevant.

If a constraint is relevant to a word, then its satisfaction condition, C_s, is evaluated against that word. The satisfaction condition is a regular expression, which is a language for pattern matching. The main difference between the expressions used in the tutor and standard regular expressions is the presence of %SYMBOLSET% in the C_s. %SYMBOLSET% stands for a string of all the punctuation marks that the tutor knows about. For example, the current version of CAPIT deals with commas, periods, quotation marks and apostrophes. Therefore the C_s condition of constraint (a), ^[a-z0-9%SYMBOLSET%]*$, becomes the standard regular expression ^[a-z0-9′″,.]*$ before being matched to a student's solution.

Briefly, a regular expression like ^[a-z0-9′″,.]*$ defines a pattern that can be matched to a string. The symbol ^ defines the start of the string and $ defines the end of the string. The square brackets [] define a set of characters, while a negative character set (which matches any characters not in the set) is defined by square brackets with a ^ inside the brackets, e.g. [^a-z]. The symbol * usually follows a character or pattern and means "zero or more repetitions of the previous pattern". Thus, the ^[a-z0-9′″,.]*$ is matched by any string containing zero or more characters that are lower case letters, numbers or punctuation marks. A word containing an upper case letter does not match the expression, and therefore the constraint would be violated. Constraint (d) has a much simpler C_s, [^']s$. This regular expression is simply matched by all strings ending in *s* that do not have an apostrophe in the penultimate position.

The constraints in CAPIT tend to fall somewhere on a continuum between general and specific. *General* constraints apply to many different words because they are relevant to general tags such as DEFAULT and L-CASE. As a result, the feedback is more general and may not address the specific misconceptions that led to the error. Constraint (a) from Table 4 is one example of this class. *Specific* constraints are satisfied only by specific punctuation/capitalisation patterns, and feedback can be more specific in this instance. For example, constraint (b) is satisfied only when a word with the tag NAME-OF-PERSON starts with a capital letter or digit (excluding any punctuation marks at the beginning). Constraint (c) is violated only when the student punctuates a word that ends both a direct quote and a sentence incorrectly, with the quotation mark preceding the period (e.g. see Table 1(b)). In this case, the tutor can tell the student specifically to reverse the order of the punctuation marks. In all other cases, the constraint is satisfied. Similarly, (d) is violated only when the student tries to add an apostrophe before the *s* in the possessive pronoun *its*, and is satisfied otherwise.

DECISION-THEORETIC PAS IN CAPIT

Decision problems conducive to our methodology include next problem selection, error message selection, topic selection, selective highlighting/hiding of text, and timing of interventions to give help. We decided to develop decision-theoretic strategies for the first two of these tasks, problem selection and error message selection. In this section, we describe how the general methodology was applied to develop these strategies for CAPIT.

Step 1: Randomised Data Collection

Initial data for Step 1 of the decision-theoretic PAS strategy development was acquired from a data acquisition deployment of CAPIT at Westburn School, Christchurch, New Zealand in March 2000 (Mayo et. al., 2000). A version of CAPIT was used in which problems and error messages were selected randomly. The problems came from the pool of all unsolved problems, and the error message was selected from the set of constraints violated on the current attempt (recall that there is one error message per constraint). The evaluation study consisted of four 30-45 minute sessions. Details of each problem attempt and error message displayed were logged. Subsequent analysis revealed the following averages. Each student made 89 attempts at 28 different problems. 21 of the problems were eventually solved, and 7 abandoned. Students violated an average of 181 constraints during the sessions, of which feedback was given on 68. A total of 3300 records of the form <*State, Action, Outcome*> were acquired during this step, where *State* is a record of the outcome of the last attempt at each constraint (which may include attempts at previous problems, if for example, a constraint was relevant to the last problem but is not relevant to the current problem), *Action* is the problem that was selected randomly, and *Outcome* is a record of the constraints that were violated and satisfied following the problem attempt.

Step 2: Model Selection

The data acquired from the evaluation study was used to generate the best Bayesian network for long-term student modelling. The selection criterion was the ability of the network to predict student performance on constraints. An issue at this point was whether to use a model in which the constraints were independent of each other, as in Reye's model, or whether to allow (more realistically) any dependencies between constraints to be learned from the data. This decision is quite significant because a model in which constraints are assumed independent can be formulated with only four variables, whereas a model in which any dependencies between constraints are allowed is much more complex and must consist of at least twice the number of variables as there are constraints. Figures 7 and 8 illustrate the competing "small" and the "large" specifications. In both diagrams, L_i represents the outcome of the last attempt at the *i*th constraint, and can take values `S` (satisfied), `V` (violated), `VFB` (violated with feedback), or `NR` (has not been relevant before). N_i is the predicted outcome of the next attempt, whose values are `{S, V, NR}`. Note that in accordance with the "desirable features" discussed earlier, neither networks explicitly model unobserved student states.

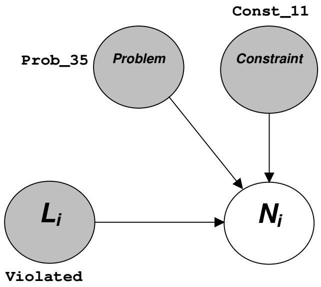


Figure 7. The structure of the small Bayesian network for predicting the outcome of the next attempt at the *i*th constraint. In this example, the network is predicting the outcome of the next attempt at constraint 11 which is relevant to the current problem, 35, and was previously violated.

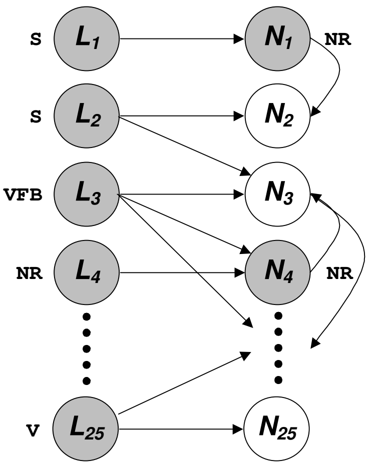


Figure 8. The structure of the large Bayesian network specification after learning. The student has previously satisfied constraints 1 and 2, violated constraints 3 and 25 (receiving feedback on 3), and has not yet attempted constraint 4. The network is currently configured to predict this student's performance on a problem whose relevant constraints include 2, 3 and 25.

A number of variants of the large network were considered. In each case, an algorithm proposed by Cheng et. al. (1998) for structural learning by mutual entropy maximisation was utilised to learn a Bayesian network from the data collected in Step 1. The conditional probabilities were estimated using the standard Dirichlet priors approach (Heckerman, 1999) which is described in more detail later. An important component of the structural learning algorithm is the minimum threshold. This essentially determines the minimum amount of mutual information required between two variables before an arc can connect them. For these experiments, minimum thresholds of 4, 6 and 10 were selected (initial experiments showed that a threshold below 4 resulted in a network far too large for on-line evaluation). Another parameter that we wanted to investigate was the addition of prior knowledge: does it enhance predictive performance? The "obvious" prior knowledge to add is an arc from L_i to N_i, for each constraint i, indicating that at the very least, the outcome of the next attempt at a constraint is partly dependent on the outcome of the previous attempt. We thus formulated six specifications for large networks: *Large(4)*, *Large(6)* and *Large(10)* being the specifications without prior knowledge, and *PLarge(4)*, *PLarge(6)* and *PLarge(10)* being specifications with prior knowledge. For each of the large specifications, the L_i nodes were fixed as root nodes, to reflect the fact that they come before the N_i nodes in temporal order. Thus, there are two types of arc that can be learnt from the data for the large networks: arcs from *L* layer to the *N* layer, and arcs within the *N* layer.

The data was then divided into training and test datasets. Approximately 20% of the records were selected randomly into the test dataset. The remaining 80% were kept in the training set, and used to train one large network for each of the 6 specifications. A simpler network equivalent to Figure 7 (*Small*) was also trained from this data, although in this case the structure was already specified and only the conditional probability $p(N_i|L_i,Problem,Constraint)$ had to be learned. This entire process of training was repeated three times (j=1..3), each time with a different randomly generated training/test dataset divisions. The total number of different networks that were generated, therefore, was 21.

The first question to answer was whether or not the larger networks were better predictors of student performance than the small ones on the test data. Each of the 6 large networks generated from the jth training set was compared to the *Small* network generated from the jth training set in the following way. For each problem attempt in the jth test set, the large and small networks were given the values for $L_1..L_{25}$. The large networks had their $P_1..P_{25}$ nodes instantiated to NR for each constraint not relevant to the attempt's problem. The values of the remaining uninstantiated nodes in $P_1..P_{25}$ were then predicted. For the large networks, this required one evaluation of the network, and for *Small*, one evaluation per relevant constraint was necessary. The standard junction tree algorithm for Bayesian network inference was used (Lauritzen & Spiegelhalter, 1988). Then, for each P_i representing a relevant constraint, the predicted value of P_i was compared to the actual value of P_i. The total number of correct predictions was counted. A correct prediction was deemed to occur if the predicted outcome with maximum probability matched the actual outcome. To clarify what was done further, the output of each comparison was essentially a table of tuples of the form <*i*, $L_j(i)$, $S_j(i)$, $T_j(i)$>, where i=1..n_j is the attempt (n_j is the number of attempts in the *j*th test set), $L_j(i)$ is the number of correct predictions given by the large network, $S_j(i)$ is the number of correct predictions given by the small network, and $T_j(i)$ is the maximum number of correct predictions (simply the total number of relevant constraints on that attempt).

The coefficient of determination (r^2) was calculated for each network by taking the number of correctly predicted constraints as a function of the number of relevant constraints. The results are summarised in Table 7. The r^2 values of the large networks are higher than those of the small networks, suggesting that the large specifications are better.

Table 7. The coefficients of determination characterising the number of correct predictions as a function of the total number of relevant constraints, for each network and training/test dataset.

	TestData$_1$ (n=639)	**TestData$_2$ (n=608)**	**TestData$_3$ (n=686)**
PLarge(10)	0.7649	0.7434	0.7638
PLarge(6)	0.7562	0.7385	0.7615
PLarge(4)	0.7101	0.7208	0.7417
Large(10)	0.7446	0.7464	0.7495
Large(6)	0.749	0.7347	0.7511
Large(4)	0.7117	0.7236	0.7387
Small	0.7312	0.7086	0.7314

Next, we tested to see if the large networks were statistically significantly better predictors of student performance than the small networks. Note that for each of the 18 comparisons, the number of correct predictions made by the small and large networks are paired. That is, for each attempt i=1..n_j in each of the 18 tests, both an $S_j(i)$ and a $L_j(i)$ were generated, both of which can be considered stochastic functions of *i*. Therefore, the samples are pair-wise dependent. A paired-difference experiment (McClave & Benson, 1991, pp. 421-7) was used to test for significant differences.

Table 8 shows that the *PLarge(10)*, *PLarge(6)* and *Large(6)* specifications all produce Bayesian networks that are statistically significantly better predictors of student performance than the networks produced by the *Small* specification. The other specifications each had at least one comparison where no statistically significant difference was found (indicated by "Accept H_0"). For these tests, a high *t* value indicates greater significance. For all the networks, the outcomes in the second test set were much more difficult to predict than those of the first and third sets, resulting in lower *t* values. The exception to this is *Large(10)*, which happened perform barely well on the second test set but not the first and third. From these results, we were able to rule out *Small* as a worthwhile specification to continue with.

Table 8. Results of two-tailed paired difference experiments comparing each large network against *Small*. H_0 is the hypothesis that there is no difference in the mean number of correct predictions made by both networks. A positive *t* value indicates that the large network is better than *Small*. The rejection region for all the datasets is approximately ±2.58 for 99% confidence, and ±1.96 for 95% confidence.

	TestData$_1$ (n=639)	**TestData$_2$ (n=608)**	**TestData$_3$ (n=686)**
	Small	**Small**	**Small**
PLarge(10)	t=5.75, α=0.01	t=3.95, α=0.01	t=5.46, α=0.01
PLarge(6)	t=5.17, α=0.01	t=3.30, α=0.01	t=5.06, α=0.01
PLarge(4)	Accept H_0	Accept H_0	t=2.1, α=0.05
Large(10)	Accept H_0	t=1.98, α=0.05	Accept H_0
Large(6)	t=4.01, α=0.01	t=2.78, α=0.01	t=3.93, α=0.01
Large(4)	Accept H_0	Accept H_0	t=2.05, α=0.05

The next task was to determine the most accurate large network. In particular, does the selection of the minimal threshold or the addition of prior knowledge result in improved performance? For this analysis, the three best large networks (*PLarge(10)*, *PLarge(6)* and *Large(6)*) were compared. No statistically significant difference was found between *PLarge(10)* and *PLarge(6)* on any of the training/testing dataset divisions. However, significant differences were found between *PLarge(10)* and *Large(6)* as Table 9 shows.

Table 9. Results of two-tailed paired difference experiments comparing *PLarge(10)* against *Large(6)*. H_0 is the hypothesis that there is no difference in the mean number of correct predictions of made by both networks. A positive *t* value indicates that the *PLarge(10)* is better than *Large(6)*. The rejection region for all the datasets is approximately ±2.58 for 99% confidence, and ±1.96 for 95% confidence.

	TestData$_1$ (n=639)	**TestData$_2$ (n=608)**	**TestData$_3$ (n=686)**
	Large(6)	**Large(6)**	**Large(6)**
Plarge(10)	t=3.13, α=0.01	Accept H_0	t=2.08, α=0.05

To conclude step 2, *PLarge(10)* was selected as the best specification to proceed with because the *t* values for *PLarge(10)* were, on average, greater than those of *PLarge(6)* in Table 8. The training and testing datasets were combined into a single dataset and a Bayesian network with the *PLarge(10)* specification was learned. One of the desirable features discussed earlier was to take advantage of the unique ability of Bayesian network to integrate prior knowledge and data; this has been shown to improve predictive accuracy here.

Step 3: Decision-Theoretic Strategy

Step 3 is the implementation of decision-theoretic PAS strategies. The key task here is to define a utility function U(x,d) specific to the PAS strategy that can be substituted into Equation 1 to yield a task-specific expected utility function. For CAPIT, we were interested in two tasks: next problem selection, and error message selection following an attempt in which multiple constraints are violated.

The value of the next problem $d \in$ {`Problem_1`, …, `Problem_45`} is determined by predicting the student's performance on the problem with the Bayesian network. An appropriate problem can be considered to fall 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 of more capable peers". We interpret this as stating that a student should be given a problem slightly above their current level but not so difficult as to be discouraging. This principle implies that utility should be maximised for problems where one or two errors are likely (reflecting a challenging problem), but minimised for problems whose outcome is no errors (being too easy) or several errors (being too hard). This utility function is defined in Table 10.

Table 10. The utility function for problem selection. Utility is maximised for problems resulting in one or two errors only.

x	**U(*x*,*d*)**
`No-Errors`	0.0
`1-Error`	1.0
`2-Errors`	1.0
`3+Errors`	0.0

The cost of all problems is assumed to be zero. Let us also assume that ξ comprises the student history (i.e. the instantiations of $L_1..L_{25}$) and the instantiations of N_i to `NR` for those constraints not relevant to *d*. Substituting into Equation 1 yields the expected utility of problem *d*:

$$\begin{aligned} E[U(x,d)\,|\,\xi] &= p(\texttt{No-Errors}|\, d,\xi)\, \mathrm{U}(\texttt{No-Errors}, d) \\ &\quad + p(\texttt{1-Error}|\, d,\xi)\, \mathrm{U}(\texttt{1-Error}, d) \\ &\quad + p(\texttt{2-Errors}|\, d,\xi)\, \mathrm{U}(\texttt{2-Errors}, d) \\ &\quad + p(\texttt{3+Errors}|\, d,\xi)\, \mathrm{U}(\texttt{3+Errors}, d) - 0 \\ &= p(\texttt{1-Error}|\, d,\xi) + p(\texttt{2-Errors}|\, d,\xi) \end{aligned}$$

Now we need to calculate $p(\texttt{1-Error}|\ d,\xi)$ and $p(\texttt{2-Errors}|\ d,\xi)$ from the Bayesian network. This is not straightforward because the predicted outcomes $N_1..N_{25}$ are not necessarily mutually or conditionally independent. In fact, the best way to deal with this computation is to extend the Bayesian network itself at runtime by adding a deterministic function *NumErrors* ∈ {`No-Errors`, `1-Error`, `2-Errors`, `3+Errors`} to the network, which is dependent on the relevant constraints only. The function simply counts the number of its parents that are violated, but because the parents of *NumErrors* are uncertain, the uncertainty is transferred to *NumErrors* by the Bayesian network inference algorithm (Lauritzen & Speigelhalter, 1988) in the correct way. The addition of *NumErrors* to the example large network is illustrated in Figure 9. The probabilities of Equation 1 can now be determined by querying the posterior distribution over the *NumErrors* variable.

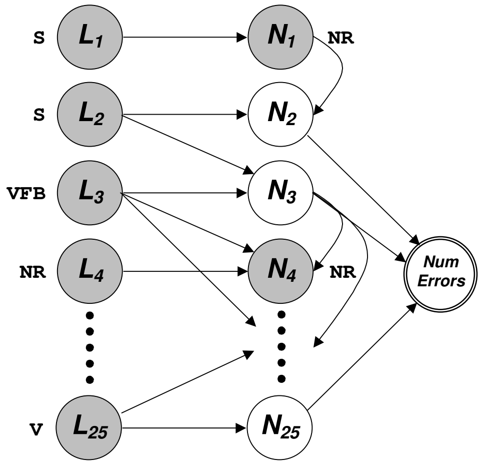


Figure 9. The same large network as depicted in Figure 8, but with *NumErrors* added as a child of all the N_i nodes representing relevant constraints.

The strategy for decision-theoretic error message selection is slightly different. In this case, *d* ∈ {$\texttt{FB}_\texttt{i}$ | Constraint *i* was relevant and violated on the last attempt} where $\texttt{FB}_\texttt{i}$ is the decision to give feedback on the *i*th constraint. It is assumed that an error message about a constraint can influence the outcome of the next attempt at the constraint, resulting in a satisfaction (desired) or a violation (not desired). Table 11 characterises this as a utility function

Table 11. The utility function for feedback selection.

x	U(*x*,*d*)
N_i=V	0.0
N_i=S	1.0

Because the system gives feedback on only one violated constraint per attempt, the probabilities of these outcomes can be read directly from the network by "pretending" that feedback was given on the *i*th constraint. That is, we instantiate L_i to `VFB` instead of `V` and query N_i to obtain $p(N_i=\mathrm{V}|d, \xi)$ and $p(N_i=\mathrm{S}|d, \xi)$. However, the cost of each feedback message cannot be assumed to be zero, because each constraint will have a different probability of being satisfied without feedback, anyway. This probability of satisfaction without feedback can be considered the "opportunity cost" of giving feedback on the *i*th constraint, which is therefore defined as:

$$cost(d) = p(N_i=\mathrm{S}|\xi)$$

Substituting these values into Equation 1 yields the expected utility for feedback:

$$\begin{aligned} E[U(x,d)\,|\,\xi] &= \big(p(N_i=\mathrm{S}\,|\,d,\xi)U(N_i=\mathrm{S}) + p(N_i=\mathrm{V}\,|\,d,\xi)U(N_i=\mathrm{V})\big) - p(N_i=\mathrm{S}\,|\,\xi) \\ &= p(N_i=\mathrm{S}\,|\,d,\xi) - p(N_i=\mathrm{S}\,|\,\xi) \end{aligned}$$

The expected utility of an error message is therefore the posterior gain in probability of the constraint being satisfied that the message results in.

This step illustrates the framing of two simple learning theories as utility functions. To capture the more general notions such as a curriculum, other learning theories could be represented as more complex utility functions and thus integrated into the normative framework.

Step 4: On-line adaptation

The next challenge was implementing an on-line learning algorithm so that the Bayesian student model would adapt to the student. Heckerman (1999) shows how to calculate conditional probabilities for a Bayesian network from data. Let $X=x_k|Pa_x=pa_x$ represent an observation of variable X in state x_k when its parents Pa_x are in configuration pa_x. A normal Bayesian network maintains, for each possible $X=x_k|Pa_x=pa_x$, a single conditional probability $p(X=x_k|Pa_x=pa_x)$. The Dirichlet priors approach treats $p(X=x_k|Pa_x=pa_x)$ itself as an uncertain variable, and calculates its expected value from the data. It turns out that by assuming that the probability distribution over $p(X=x_k|Pa_x=pa_x)$ is a Dirichlet distribution, the expected value corresponds to the frequency. Suppose $X=x_k|\ Pa_x=pa_x$ has been observed α_k times while $Pa_x=pa_x$ has been observed α times. Obviously $\alpha_k \le \alpha$. Then, it shown by Heckerman (1999) that the expected value of $p(X=x_k|Pa_x=pa_x)$ is:

$$E[p(X=x_k|Pa_x=pa_x)] = \frac{\alpha_k}{\alpha}$$

If $Pa_x=pa_x$ is observed a further N times, while N_k further observations of $X=x_k|pa_x$ are made, the expected value of the conditional probability updates simply to:

$$E[p(X=x_k|Pa_x=pa_x)|\text{observations}] = \frac{\alpha_k + N_k}{\alpha + N}$$

The parameters α and α_k are known as sufficient statistics, because they are adequate to define a Bayesian network; once they have been calculated, the rest of the training data can be discarded.

The problem with this algorithm is that it does not take into account the temporal ordering of the cases, and therefore there is no way to "bias" the conditional probabilities towards to most recent cases. This is an incorrect assumption for an intelligent tutor to make because the student's state is expected to change constantly. The system therefore needs a way of gradually discounting the effects of old data. However, the effect of the standard approach would be that as α gets increasingly large, the influence of new cases on the conditional probabilities decreases. To illustrate, CAPIT's population student model was learned from records of approximately 3300 problem attempts. The average student is likely to make only 50-100 problem attempts. Therefore, the standard Dirichlet priors approach would not be expected to adapt the network's parameters to the student to the desired extent.

Fortunately, the standard approach can be modified to prefer more recent observations. Our solution is to reduce α to a value such that the effect of new cases becomes significant. Let that value be α_{MAX}. The sufficient statistics α and α_k can now be replaced by two new statistics, α' and α_k', defined as:

$$\alpha' = \alpha_{MAX},\ \alpha_k' = \alpha_{MAX}(\alpha_k/\alpha)$$

The lower the constant α_{MAX}, the more significance new cases will have on the conditional probabilities. In CAPIT, the conditional probabilities are updated after every attempt (so N=1). The update rule, therefore, simplifies to:

$$E[p(X=x_k|pa_x)|\text{One observation of } X=x_k \text{ when } Pa_x=pa_x] = \frac{\alpha_k'+1}{\alpha'+1}$$

$$E[p(X=x_k|pa_x)|\, j \neq k, \text{ One observation of } X=x_j \text{ when } Pa_x=pa_x] = \frac{\alpha_k'}{\alpha'+1}$$

A value for α_{MAX} was chosen by trials with simulated students. Two students were simulated: a "good" student who got every problem correct, and a "bad" student who made numerous mistakes and frequently abandoned problems. A domain expert analysed the sequence of problems that was selected for each student. It was found that when $\alpha_{MAX} > 5$, the system was not quick enough to present challenging problems to the good student even after several problems were solved correctly in a single attempt. This occurred simply because the conditional probabilities did not update fast enough. For the bad student, simple problems were repeatedly selected regardless of the value of α_{MAX}. A value of $\alpha_{MAX} = 5$ was therefore selected.

Step 5 Evaluation

Two evaluations were performed; a simple, informal evaluation to ensure that the system was behaving reasonably, followed by an extensive classroom evaluation with school students.

Simulated Students Evaluation

The first step in the evaluation was an informal observation of the behaviour of the decision-theoretic version of CAPIT. The observations were noted during the trial run with the simulated good and bad students. The system always started with the easiest problem, which involved merely dividing the text into sentences and inserting capital letters at the start, and periods at the end, of each sentence. For the good student, further problems typically introduced new constraints one at a time until a certain point was reached (probably when the posterior probability of the student satisfying the most common constraints was sufficiently high), after which more difficult problems (e.g. direct speech problems) introducing several new constraints at a time were selected. This is similar to a human tutor assessing a good student's capabilities initially with easier problems, before moving more directly to challenging problems. For the bad student who repeatedly made mistakes and abandoned problems, the tutor appeared to repeatedly select problems from a pool of 3-4 easier problems but never selected the same problem twice consecutively. Again, a similar strategy to that of a human tutor returning to previously abandoned problems while maintaining some variety. Note that problems in this system do not have explicit levels – all unsolved problems are available to be selected at any one time. Feedback selection was also observed. In the extreme case of a bad student who repeatedly submitted the same (incorrect) solution with multiple violated constraints, the selection of feedback messages seemed to cycle from the most to the least specific constraints, and back again, with each attempt. Again, there is no explicit rule programmed into the tutor to make it do this.

Classroom Evaluation

Three classes of 9-10 year olds at Ilam School in Christchurch, New Zealand, participated in a four-week evaluation of CAPIT. The first class (Group A) did not use the tutor at all. The purpose of this group was to provide a baseline for comparing the pre and post test results of students that did use a tutor in the domain with those that did not. The second class (Group B) used the initial version of the tutor with randomised problem and error message selection, and the third class (Group C) used the full version of the tutor with decision-theoretic PAS and the adaptive Bayesian student model. The groups using the tutor, B and C, had one 45-minute session per week for the duration of the study, and they worked in the same pairs each week. (Working in pairs was necessary because of the limited availability of computers.) Every interaction was logged. Pre and post tests were also completed, with students completing the tests in the same pairs.

Some significant attributes of the performance of Groups B and C during the evaluation are summarised in Table 12. Pairs of students in Group C used CAPIT for approximately 34 minutes more on average than those in Group B, which was a result of a teacher cutting one of the sessions short. As a result, Group C made many more attempts, and asked for more *Why?* explanations, than Group B. The average time per attempt for both groups is approximately 43 seconds. However, despite the additional interaction time, Group C attempted and solved less problems than Group B, and abandoned more problems. This is probably due to Group C being a less-able class than Group B, an observation that was confirmed by the teachers. In hindsight, pairs of students should have been assigned to groups randomly rather than by class. An interesting discrepancy is the mean number of attempts per solved problem. Group C performed better here, perhaps suggesting that the feedback messages in their case were better adapted.

Table 12. Averages describing the behaviour of Groups B and C's.

	Group B	**Group C**
Number of pairs	16	14
Ave. interaction time per pair (mins)	80.9	115
Ave. # attempts	109.7	167.3
Ave. # solved problems per pair	29	22
Ave. # attempted problems per pair	34	30
Ave. # attempts per solved problem	5.8	5.5
Ave. # expl. asked for per pair	10.3	18

Further analysis was performed at the level of the individual constraints. Figure 10 gives the average number of times each constraint was relevant per user. This reflects the higher number of attempts made by Group C, and highlights the constraints that are common to most of the problems, for example, constraint 4 (a sentence must start with a capital letter) and constraint 7 (a sentence must end with a period). This table can be compared to Figure 11, the frequency with which constraints were violated when relevant. It shows that Group C violated proportionately more constraints that Group B, which corresponds with the averages in Table 11. However, it is interesting to note that the constraints defining the correct punctuation of direct speech (constraints 17-25) were violated proportionately less by Group C. This suggests that the decision-theoretic problem sequencing placed problems involving direct speech (which are more difficult) later in the sequence, after the student had mastered the other constraints, therefore allowing them to focus on learning direct speech punctuation.

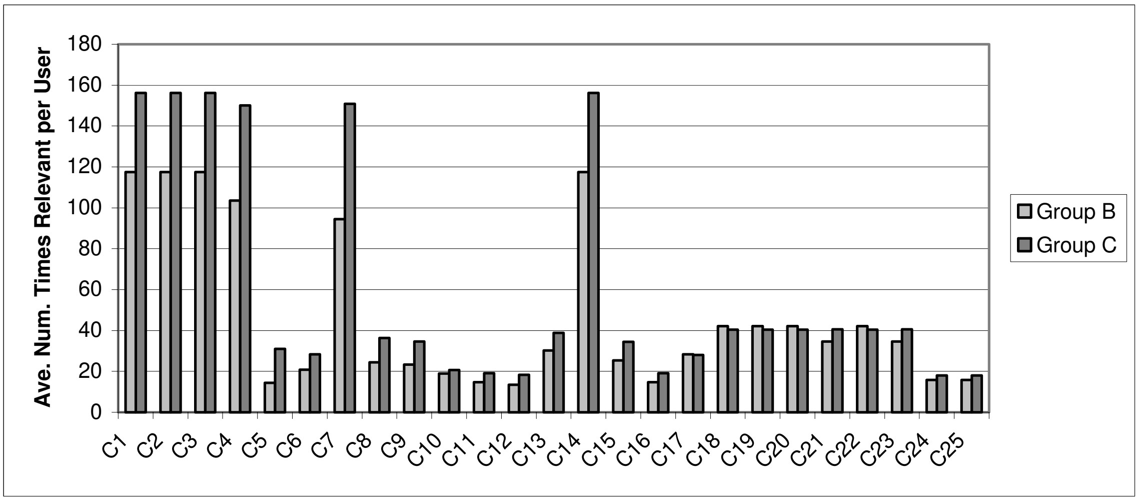


Figure 10. The frequency of constraint relevance to selected problems.

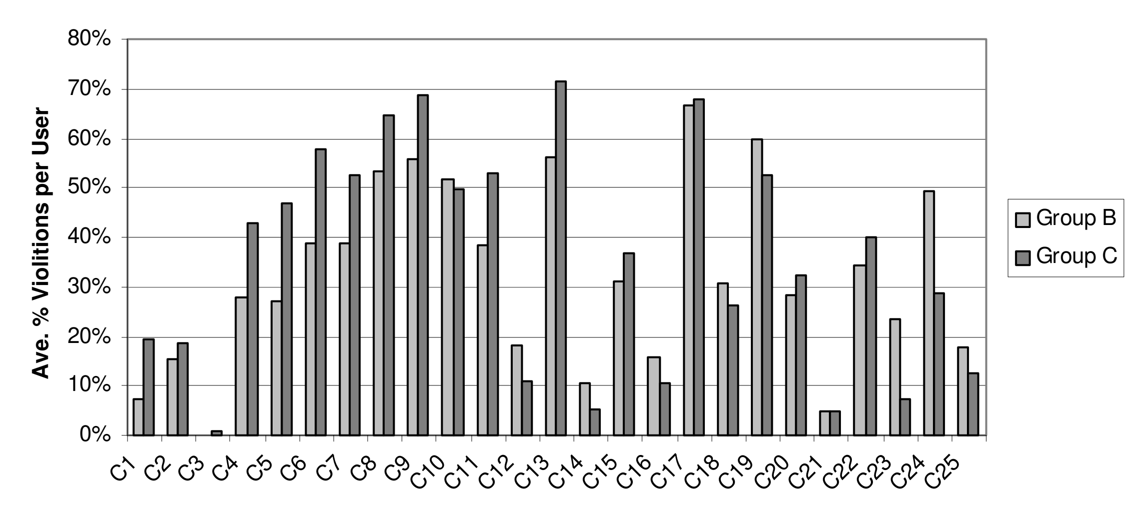


Figure 11. The frequency of constraint violation when relevant.

The pre- and post-tests were comparable (and challenging) and consisted of eight completion exercises similar to those presented by CAPIT, but done manually with pencil-and-paper. Students worked in their assigned pairs to complete the test. The score for each test was calculated by subtracting the number of punctuation and capitalisation errors from the number of punctuation marks and capital letters required for a perfectly correct solution; it was thus possible for a pre- or post-test to have a negative score (fortunately none of the students were that bad). The mean scores and standard deviations (the Y error bars) are shown in Figure 12. The mean pretest score for Group C is almost 10% lower than that of Group B. Both Group B and C show an improvement in mean test scores, although the improvement is more marked for Group C. Group A, the class that did not use the tutor, actually regressed.

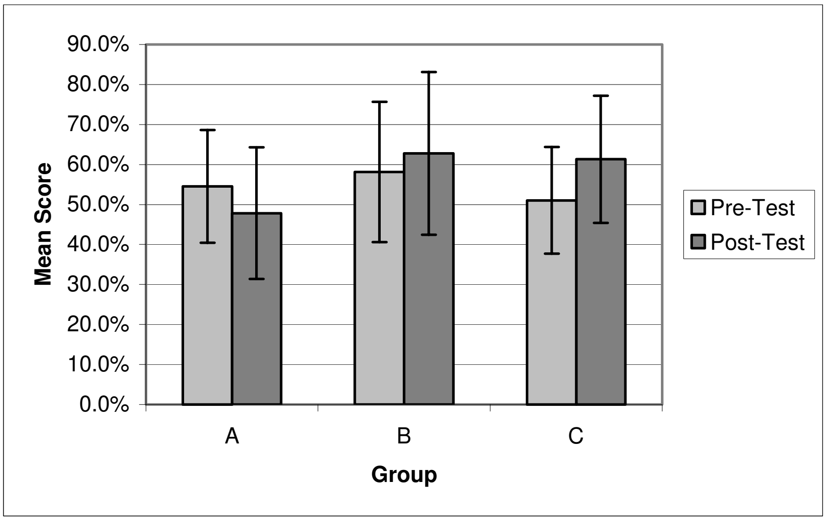


Figure 12. Mean pre- and post-test scores.

Statistical significance tests were also performed to compare the individually matched improvements of Groups B and C from pre-test to post-test. Because the same pair of students in each group completed both a pre- and a post-test, a one-tailed paired difference experiment (McClave & Benson, 1991, pp. 421-7) was performed to gauge the significance of the improvement. With H_0 being the proposition that a group did not improve, it was found that Group B improved with 95% confidence ($\alpha = 0.05$, $t = 1.86$, rejection region ± 1.75) while Group C improved with 99% confidence ($\alpha = 0.01$, $t = 3.4$, rejection region ± 2.6). The improvement is thus much more significant for Group C, which used the decision-theoretic strategies.

We also calculated the effect size, which is defined as the difference in the mean gains of the control (Group B) and experimental groups (Group C), divided by the standard deviation of the mean gain of the control group. This measure gives the magnitude of the change attributable to the intelligent PAS strategy as opposed to the random one. The effect size is 0.557, a value that is comparable to the effect size of 0.63 found by Albacete & VanLehn (2000) after a two-hour session with their tutor. (The average total interaction time in our case was less than two hours for both groups.)

The pre- and post-tests analysis, and the frequencies in Figure 11, confirm that Group C was initially less able than Group B, but learned the constraints at a faster rate. We decided to investigate the constraint violation frequencies further. Each attempt at a problem was analysed, and the total proportion of violated constraints was calculated for each attempt. This was averaged over all students in each group, and the result is depicted in Figure 13. The scatter plot shows that Group C initially made more errors than Group B, but that the rate of constraint violation decreased much faster for that group, supporting the hypothesis that Group C learned the rules of the domain more quickly. Figure 14 shows the results of the same analysis, as an example, for constraints 4 and 7 only, which both depend on the child's cognitive ability to separate the problem text into sentences. The difference is much more marked for these constraints than for the average of all the constraints, but the trend is the same. For both scatter diagrams, a cut-off point of 125 attempts was selected because approximately half of the pairs of students reached this number of attempts, and beyond this number statistical effects arising from the smaller number of pairs tend to corrupt the trend.

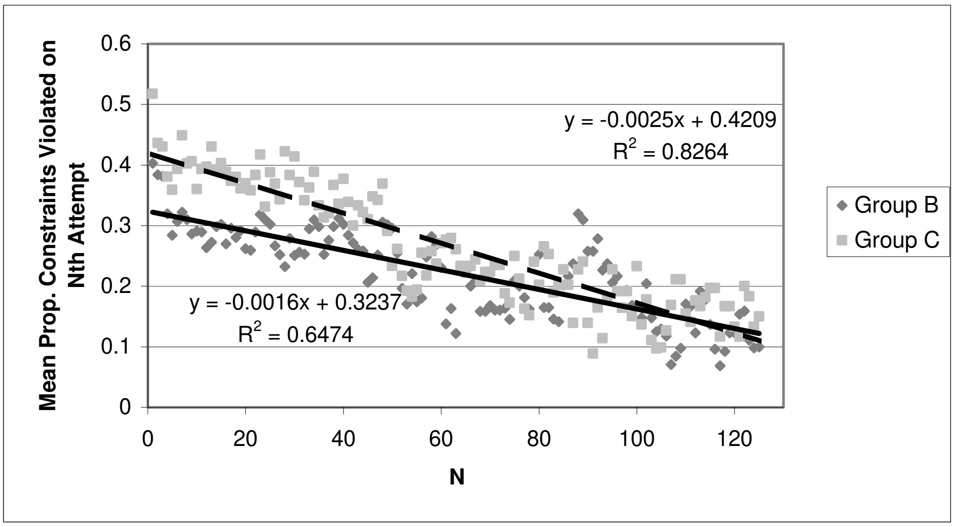


Figure 13. Rate of constraint violation by attempt, for all constraints.

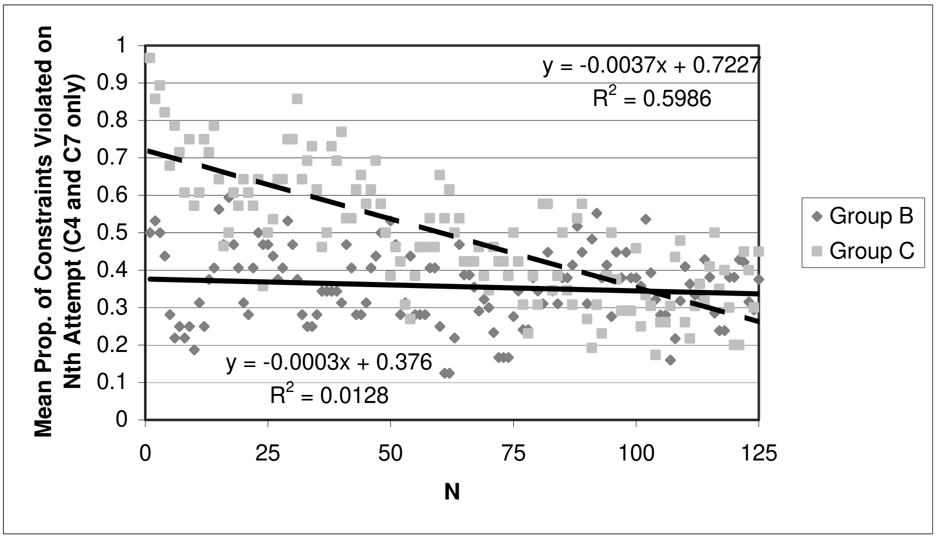


Figure 14. Rate of constraint violation by attempt, for constraint 4 and 7.

Further analysis investigated the mean number of attempts, and the mean time required, to solve the *n*th problem. Figures 15 and 16 show the results of this analysis. Both line graphs show the same basic trend; Group C was less to able initially, but improved at a faster rate than Group B.

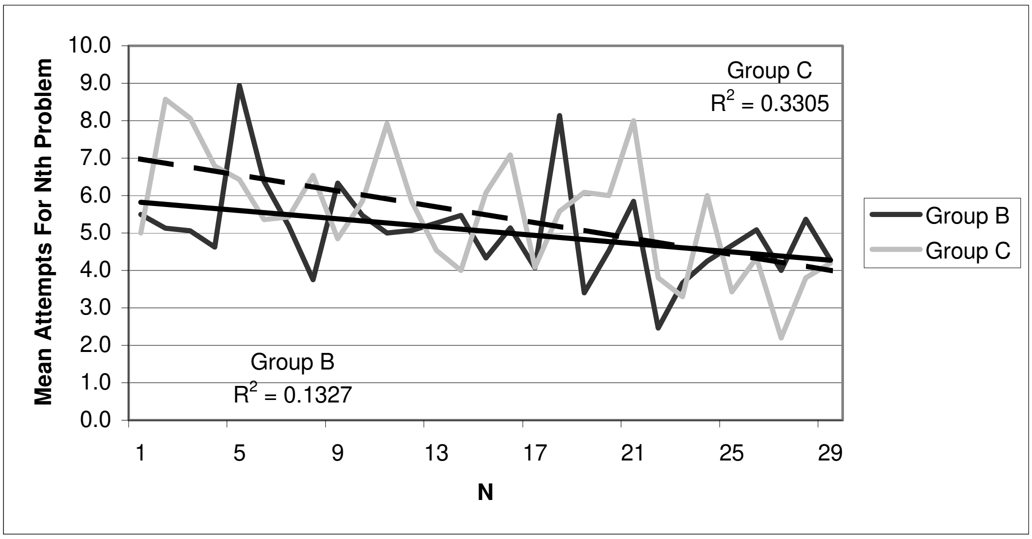


Figure 15. Number of attempts solving the *n*th problem.

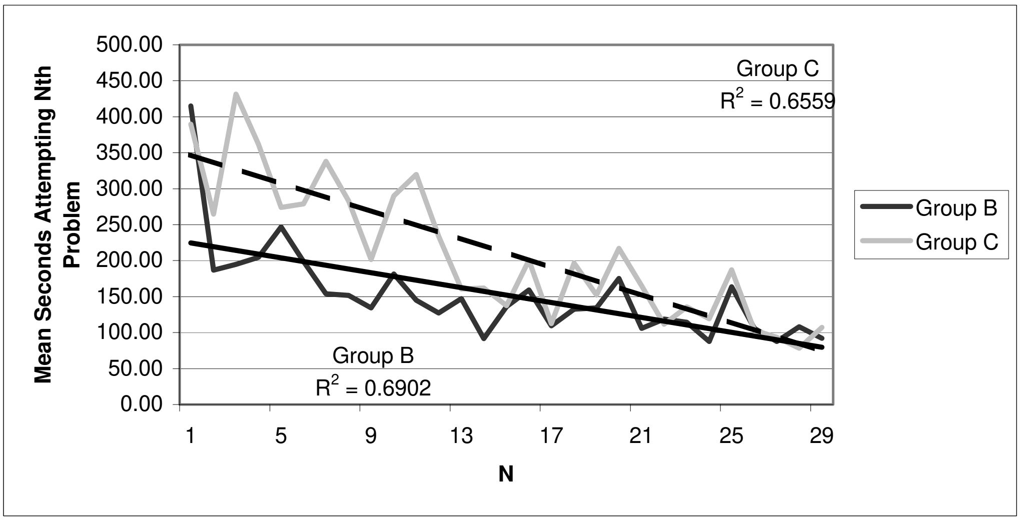


Figure 16. Time required solving the *n*th problem.

To summarise, we have demonstrated that the version of CAPIT with decision-theoretic problem and error message selection, and an adaptive Bayesian network student model, has led to a faster rate of learning than the same system with randomised PAS. This completes the fifth and final step of the application of the general methodology we have proposed to CAPIT.

DISCUSSION AND CONCLUSION

The significant contribution of this paper is the proposal and demonstration of a general methodology for the design and implementation of normative intelligent tutors. CAPIT, a tutor for capitalisation and punctuation, is both a working illustration that decision-theoretic computations in intelligent tutors can be tractable, and evidence that the methodology works. It is therefore possible to build normative intelligent tutors (with Bayesian student models and decision-theoretic PAS) that are guaranteed to be optimal with respect to the normative principles of rational behaviour. We have also made explicit the data-centric approach to building Bayesian network student models, whereby the structure and conditional probabilities are learned from data, and then continuously adapted on-line to the student. We believe that this approach produces student models better able to predict student performance than both the expert- and efficiency-centric approaches. The data-centric approach also results in more compact student models, because it only explicitly models observable variables.

An interesting conclusion of the statistical comparison between different Bayesian networks architectures in Step 2 is that the *Small* specification produced networks that predicted student performance almost as well as the various large network specifications (the r^2 values varied by at most 0.06). The question arises as to why this is so. An informal analysis of the data collected during Step 1 revealed that, on average, a constraint previously satisfied will be satisfied on the next attempt 91% of the time. This regularity may well explain *Small*'s relatively good performance. However, other domains may not exhibit this degree of regularity. For example, in a domain where the constraints are highly interdependent, the probability of a constraint being satisfied on the next attempt may depend much more on the previous (and current) outcomes of other constraints. On the other hand, *Small* can be expected to outperform the large networks on domains where constraint mastery is wholly or mostly probabilistically independent.

This suggests that there must be some careful justification (e.g. the statistical significance tests performed in Step 2) when a larger, complex model is chosen over a much simpler one. This issue is important, and it should not be skirted over when describing the rationale for a particular intelligent tutor architecture. Furthermore, the relatively good performance of a Bayesian network with no explicit model of the student's internal representation at predicting student performance begs the question of which domains in general are suitable for such an approach. We suggest that suitable domains are those where the concepts are ill-defined, or where different students are expected to conceptualise the domain in different ways, (e.g. constructivist environments). Also, as discussed earlier, domains where the conceptualisation is too complex to produce a simple, tractable model might benefit.

Another advantage of this approach is that it bypasses the problem of prior probabilities in Bayesian networks. VanLehn et. al. (1998) report that different choices of prior probabilities for root nodes in a network can significantly influence the posterior probabilities of other nodes. The workaround suggested by VanLehn et. al. is to treat only the difference between a variable's prior and posterior probability as significant. Our Bayesian model circumvents this problem entirely. Whenever the network is evaluated, the root nodes $L_1..L_{25}$ are always known with certainty because they represent the observed student's history. That is, the causality is always directed from the known ($L_1..L_{25}$) to the unknown ($N_1..N_{25}$), and not the other way around. Therefore we do not even need to maintain priors for these variables.

One issue arising from the design of the evaluation study was that it was inevitable that Group C, which had an intelligent version of CAPIT, would outperform Group B, who were using the randomised version. While it would have made for a better comparison if Group B had been using an alternate intelligent version of CAPIT (e.g. perhaps a version using heuristics to select the next problem and error message), the amount of data required to establish any significant differences between the two strategies would have had to have been much greater, requiring a much more large-scale evaluation study.

One area of concern with this approach is scalability, both to larger domains and different domains. In a larger domain, the space of <*State*, *Action*, *Outcome*> triples may be so large as to effectively render network induction impossible. This may be because the size of the *State* variable is very large (much larger than the 25 constraints modelled CAPIT), or there may be a high number of values that *Action* can possibly take (the limit in CAPIT was 45). A possible solution is to manipulate the learning algorithm to compensate for this additional complexity. For example, if the number of variables in the network is higher, then less numbers of edges should be added to the network during the learning process in order to keep the complexity down. Cheng's (1998) algorithm is flexible enough to perform this. With respect to actions, a possible solution is to divide the set of actions into groups, and then apply decision-theoretic action selection twice: firstly to select the group; secondly to select the action within that group. This way, the total number of actions being considered will be equal to the number of groups plus the number of items in the selected group.

Scaling the system to different domains is another issue. A limitation of CBM is that constraints must either match or fail to match; the constraint author decides beforehand the conditions of satisfaction and violation. While ambiguity can be encoded into individual constraints (e.g. in CAPIT, commas separating short clauses can be made optional), higher-level ambiguity is not handled by CBM. To illustrate, the possessive pronoun *teachers* could be punctuated to either *teacher's* or *teachers'*. The latter is less likely (unless there is specific contextual evidence that there is more than one teacher), but both are technically correct. CAPIT resolves the problem by accepting only the single most likely solution as the correct solution (*teacher's* in this example), and treating other solutions as incorrect. This is acceptable for a system designed for children, because the system needs to control what its students are actually learning. For example, it is not ideal for a child to continually punctuate possessive nouns such as *teachers* to *teachers'* when the goal of the problem is to teach the correct punctuation of singular possessive nouns. However, in other domains, it may be that the system needs to know when a solution is technically correct. This would require a comprehensive problem-solving module. In a literacy domain, advanced natural language processing would be needed in order to enumerate possible correct solutions. This is beyond the scope of CBM.

To reiterate, the results of the evaluation study are positive and show that the application of normative theories to intelligent tutoring is effective. The log analysis shows that the class using the decision-theoretic version of CAPIT learned the constraints of the domain at a faster rate than another class using the randomised version. The pre- and post-test results support this. Furthermore, on the post-test, both classes outperformed another class that did not have access to the tutor at all.

To conclude, this paper has introduced a methodology for the design of Bayesian long-term student models and decision-theoretic PAS strategies for intelligent tutors. The methodology encompasses a number of new features not present in other systems such as the integration of prior knowledge and data into a single Bayesian network; the learning of both the structure and parameters of the network from data; on-line adaptation; and, decision-theoretic PAS. Furthermore, this methodology is compatible with existing methods for domain knowledge representation and short-term student modelling, such as CBM. The methodology has been found to be effective in the domain of English capitalisation and punctuation, as demonstrated by our new intelligent tutoring system CAPIT.

ACKNOWLEDGEMENTS

This work has been partly supported by the Department of Computer Science, University of Canterbury, Christchurch, New Zealand. The authors would like to thank Westburn and Ilam Schools, Christchurch, for their participation in the evaluations of CAPIT, Jane McKenzie for assisting us to tailor CAPIT's interface and feedback messages for the schoolchildren, and Microsoft for use of their Microsoft Belief Networks (MSBN) library.

REFERENCES

Albacete P. and VanLehn K. (2000). The Conceptual Helper: An Intelligent Tutoring System for Teaching Fundamental Physics Concepts. In Gauthier G., Frasson C., and VanLehn K. (Eds.), *Proceedings of 5th International Conference on Intelligent Tutoring Systems*, Springer-Verlag, pp. 564-573.

Bauer E., Koller D. and Singer Y. (1997). Update rules for parameter estimation in Bayesian networks. In Geiger D. and Shenoy P. (Eds.) *Proceedings of the Thirteenth Conference on Uncertainty in Artificial Intelligence*, Morgan Kaufmann Publishers Inc., San Francisco, pp. 3-13.

Bayes, Rev. T. (1763). An Essay Toward Solving a Problem in the Doctrine of Chances, *Philos. Trans. R. Soc. London* 53, pp. 370-418; reprinted in *Biometrika* 45, pp. 293-315 (1958), and *Two Papers by Bayes*, with commentary by W. Edwards, Deming, New York, Hafner, 1963.

Beck J. and Woolf B.P. (2000). High-level student modeling with machine learning. In Gauthier G., Frasson C., and VanLehn K. (Eds.), *Proceedings of 5th International Conference on Intelligent Tutoring Systems*, Springer-Verlag, pp. 584-593.

Bouwer A. (1998). An ITS for Dutch Punctuation. In Goettle B., Halff H., Redfield C., and Shute V. (Eds.) *Proceedings of the Fourth International Conference on Intelligent Tutoring Systems*, Springer-Verlag, pp. 224-233.

Brusilovsky, P. (2000). Course Sequencing for Static Courses? Applying ITS Techniques in Large-Scale Web-Based Education. In Gauthier G., Frasson C., and VanLehn K. (Eds.), *Proceedings of 5th International Conference on Intelligent Tutoring Systems*, Springer-Verlag, pp. 625-634.

Cheng J., Bell D. and Liu W. (1998). *Learning Bayesian Networks from Data: An Efficient Approach Based on Information Theory.* On World Wide Web at http://www.cs.ualberta.ca/~jcheng/bnpc.htm

Collins J., Greer J., and Huang S. 1996. Adaptive Assessment Using Granularity Hierarchies and Bayesian Nets. In Frasson C., Gauthier G., and Lesgold A. (Eds.) *Proceedings of the Third International Conference on Intelligent Tutoring Systems (ITS-96),* Springer-Verlag, pp. 569-577.

Conati C., Gertner A., Van Lehn K., and Druzdel M. (1997). On-Line Student Modeling for Coached Problem Solving Using Bayesian Networks. In Kay J. (Ed.), *Proc. of the Seventh International Conference on User Modeling (UM99)*, Springer-Verlag, pp. 231-242.

Corbett A. and Anderson J. (1992). Student Modeling and Mastery Learning in a Computer-Based Programming Tutor. In Gauthier G. (Ed.), *Proceedings of the Second International Conference on Intelligent Tutoring Systems,* Springer-Verlag, pp. 413-420.

Corbett A. and Bhatnagar A. (1997). Student Modeling in the ACT Programming Tutor: Adjusting a Procedural Learning Model with Declarative Knowledge. In Jameson A., Paris C., and Tasso C. (Eds.), *User Modeling: Proceedings of the Sixth International Conference, UM97*, pp. 243-254.

Cowell R. (1999). Introduction to Inference in Bayesian Networks. In Jordan M. (Ed.) *Learning in Graphical Models*, pp. 9-26. MIT Press.

D'Ambrosio, B. (1999). Inference in Bayesian Networks. *AI Magazine*, 20(2), pp. 21-36.

Everson H. (1995). Modeling the student in ITS: the promise of a new psychometrics. *Instructional Science* 23, 433-52.

Ganeshan R., Lewis Johnson W., Shaw E., and Wood, B.P. (2000). Tutoring Diagnostic Problem Solving. In Gauthier G., Frasson C., and VanLehn K. (Eds.), *Proceedings of 5th International Conference on Intelligent Tutoring Systems*, Springer-Verlag, pp. 33-42.

Gärdenfors P. (1989). The dynamics of normative systems. In Martino A., *Proceedings of the 3rd International Congress on Logica, Informatica, Dritto*, pp. 293-299. Consiglio Nazionale delle Ricerche, Florence 1989. Also published in A.A. Martino (Ed.) *Expert Systems in Law,* Elsevier, pp. 195-200, 1991.

Gertner A. (1998). Providing feedback to equation entries in an intelligent tutoring system for Physics. In Goettle B., Halff H., Redfield C., and Shute V. (Eds.) *Proceedings of the Fourth International Conference on Intelligent Tutoring Systems*, Springer-Verlag,, pp 254-263.

Gertner A. and VanLehn K. (2000). Andes: A Coached Problem Solving Environment for Physics. In Gauthier G., Frasson C., and VanLehn K. (Eds.), *Proceedings of 5th International Conference on Intelligent Tutoring Systems*, Springer-Verlag, pp. 133-142.

Gertner A., Conati C., and VanLehn K. (1998). Procedural help in Andes: Generating hints using a Bayesian network student model. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*, MIT Press, pp. 106-111.

Heckerman D. (1999). A Tutorial on Learning with Bayesian Networks. In Jordan M. (Ed.) *Learning in Graphical Models*, pp. 301-354. MIT Press.

Horvitz E., Breese J. and Henrion M. (1988). Decision Theory in Expert Systems and Artificial Intelligence. *International Journal of Approximate Reasoning* 2:247-302; also On World Wide Web at http://www.auai.org/auai-tutes.html

Jensen, F., Lauritzen, S. and Oleson, K. (1990). Bayesian Updating in Causal Probabilistic Networks by Local Computations. *Computational Statistics Quarterly, 4*, 269-282.

Krause P. (1998). *Learning Probabilistic Networks*. On World Wide Web at http://www.auai.org/auai-tutes.html

Lauritzen S. and Spiegelhalter D. (1988). Local computations with probabilities on graphical structures and their application to expert systems (with discussion). *Journal of the Royal Statistical Society Series B*, 50, pp. 157-224.

Mayo M. and Mitrovic A. (2000). Using a Probabilistic Student Model to Control Problem Difficulty. In Gauthier G., Frasson C., and VanLehn K. (Eds.), *Proceedings of 5th International Conference on Intelligent Tutoring Systems*, Springer-Verlag, pp. 524-533

Mayo M., Mitrovic A. and McKenzie J. (2000). CAPIT: An Intelligent Tutoring System for Capitalisation and Punctuation. In Kinshuk, Jesshope C. and Okamoto T. (Eds.) *Advanced Learning Technology: Design and Development Issues*, Los Alamitos, CA: IEEE Computer Society (ISBN 0-7695-0653-4), pp. 151-154.

McClave J. and Benson P. (1991). *Statistics for Business and Economics (Fifth Edition).* Dellan Publishing Company.

Millán E., Pérez-de-la-Cruz J., and Suárez E. (2000). Adaptive Bayesian Networks for Multilevel Student Modelling. In Gauthier G., Frasson C., and VanLehn K. (Eds.), *Proceedings of 5th International Conference on Intelligent Tutoring Systems*, Springer-Verlag, pp. 534-543.

Mislevy R. and Gitomer D. (1996). *The Role of Probability-Based Inference in an Intelligent Tutoring System.* On World Wide Web at http://cresst96.cse.ucla.edu/CRESST/pages/ reports.htm.

Mitrovic, A. and Ohlsson, S. (1999). Evaluation of a Constraint-Based Tutor for a Database Language. *Int. J. on Artificial Intelligence in Education*, 10(3-4), 238-256.

Murray R. and VanLehn K. (2000). DT Tutor: A Decision-Theoretic, Dynamic Approach for Optimal Selection of Tutorial Actions. In Gauthier G., Frasson C., and VanLehn K. (Eds.), *Proceedings of 5th International Conference on Intelligent Tutoring Systems*, Springer-Verlag, pp. 153-162.

Murray W. (1998). A Practical Approach to Bayesian Student Modeling. In Goettle B., Halff H., Redfield C., and Shute V. (Eds.) *Proceedings of the Fourth International Conference on Intelligent Tutoring Systems*, Springer-Verlag, pp. 424-433.

Murray W. (1999). An easily implemented, linear time algorithm for Bayesian student modeling in multi-level trees. In Lajoie S. and Vivet M., *Proceedings of the 9th International Conference on Artificial Intelligence and Education (AI-ED 99)*, IOS Press, pp.413-420.

Ohlsson S., (1994). Constraint-based Student Modeling. In: Greer, J.E., McCalla, G.I. (eds.): *Student Modeling: the Key to Individualized Knowledge-based Instruction*, pp. 167-189. NATO.

Pearl J. (1988). *Probabilistic reasoning in intelligent systems: networks of plausible inference (revised 2nd edition)*. Morgan Kauffman, USA

Reye J. (1998). Two-Phase Updating of Student Models Based on Dynamic Belief Networks. In Goettle B., Halff H., Redfield C., and Shute V. (Eds.) *Proceedings of the Fourth International Conference on Intelligent Tutoring Systems*, Springer-Verlag, pp. 274-283.

Russell S. and Norvig P. (1995). *Artificial Intelligence: A Modern Approach.* Prentice-Hall.

Savage L. (1954). *The Foundations of Statistics.* Wiley.

Shafer, G. (1986). Probability judgement in artificial intelligence. In L. Kanal and J. Lemmer (Eds.), *Uncertainty in Artificial Intelligence*. New York: North-Holland.

Stern M., Beck J., and Woolf B. (1999). *Naïve Bayes Classifiers for User Modeling.* Center for Knowledge Communication, Computer Science Department, University of Massachusetts.

VanLehn K., Niu Z., Siler S. and Gertner A. (1998). Student Modeling from Conventional Test Data: A Bayesian Approach without Priors. In Goettle B., Halff H., Redfield C., and Shute V. (Eds.) *Proceedings of the Fourth International Conference on Intelligent Tutoring Systems*, Springer-Verlag, pp. 434-443.

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

Zadeh, L. (1983). The role of fuzzy logic in the management of uncertainty in expert systems. *Fuzzy Sets and Systems, 11*, 199-227.

Appendix H

Relative Contributions To Published Papers

Regulation 8(c) of the Degree of Doctor of Philosophy section in the 2000 University Calender states that "*where the published work has more than one author it shall be accompanied by a statement signed by the candidate identifying the candidate's own contribution*." The contributions are as follows.

For all the papers, Dr Mitrovic contributed useful practical advice on how to conduct evaluation studies, and how to analyse results. She also provided a wealth of information about ITS research and some of the psychological theories underlying them, which was exceedingly useful.

For the papers specific to SQL-Tutor, Mayo & Mitrovic (1999) and Mayo & Mitrovic (2000), Dr Mitrovic is the author of the SQL-Tutor system and performed some of the statistical analysis of the evaluation results. Her programmer, Kurt Hausler, also assisted in joining my student modelling/problem selection model to the rest of the system and setting up the evaluation study. The rest of the work, including research on Bayesian networks, the bulk of the implementation, and the rest of the results analysis, was carried out by myself.

The papers specific to CAPIT are Mayo & Mitrovic (2001), and Mayo et al. (2000). There are a total of three authors of these papers: myself, Dr. Mitrovic, and Jane McKenzie. I designed the methodology and built CAPIT, organised and carried out the evaluation studies, and analysed the results. Dr Mitrovic provided advice. Jane McKenzie is a teacher who assisted in tailoring the interface and feedback messages for 8-10 year olds. She also gave valuable advice about which parts of the domain the constraints should cover.

Signed __________________________ Date ____________________

References

Akhras F. and Self J. (2000). System intelligence in constructivist learning. *International Journal of Artificial Intelligence in Education* 11, to appear.

Albacete P. and VanLehn K. (2000). The Conceptual Helper: An Intelligent Tutoring System for Teaching Fundamental Physics Concepts. In Gauthier G., Frasson C., and VanLehn K. (Eds.), *Proceedings of Fifth International Conference on Intelligent Tutoring Systems*, Springer-Verlag, pp. 564-573.

Aleven V. and Koedinger K. (2000). Limitations of Student Control: Do students know when they need help? In Gauthier G., Frasson C., and VanLehn K. (Eds.), *Proceedings of Fifth International Conference on Intelligent Tutoring Systems*, Springer-Verlag, pp. 292-303.

Aleven V., Koedinger K., Sinclair H. and Snyder J. (1998). Combatting Shallow Learning in a Tutor for Geometry Problem Solving. In Goettle B., Halff H., Redfield C., and Shute V. (Eds.) *Proceedings of the 4th International Conference on Intelligent Tutoring Systems*, Springer-Verlag, pp. 364-373.

Andersen, S., Oleson, K., Jensen, F. and Jensen, F. (1989). HUGIN - a shell for building belief universes for expert systems, *Proceedings of the 11th International Joint Conference on Artificial Intelligence.*

Anderson J. (1983). *Rules of the mind.* Hillsdale, N.J: L. Erlbaum Associates.

Anderson J. and Lebiere C. (1998). *The atomic components of thought.* Hillsdale, N.J: L. Erlbaum Associates.

Anderson J., Corbett A., Koedinger K. and Pelletier R. (1996). Cognitive Tutors: Lessons Learned. *Journal of the Learning Sciences* 4(2) 167-207.

Arnborg, R., Corneil D. and Proskurowski A. (1987). Complexity of findings in a k-tree. *SIAM Journal on Algebraic and Discrete Methods, 8*(2), 277-284.

Bauer E., Koller D. and Singer Y. (1997). Update rules for parameter estimation in Bayesian networks. In Geiger D. and Shenoy P. (Eds.) *Proceedings of the*

Thirteenth Conference on Uncertainty in Artificial Intelligence, Morgan Kaufmann Publishers Inc., San Francisco, pp. 3-13.

Bayes, Rev. T. (1763). An Essay Toward Solving a Problem in the Doctrine of Chances, *Philos. Trans. R. Soc. London* 53, pp. 370-418; reprinted in *Biometrika* 45, pp. 293-315 (1958), and *Two Papers by Bayes*, with commentary by W. Edwards, Deming, New York, Hafner, 1963.

Beck J. (2000). *What (and how) are we trying to learn?* Computer Science Department, University of Massachusetts.

Beck J. and Woolf B. (2000). High-Level Student Modeling with Machine Learning. In Gauthier G., Frasson C., and VanLehn K. (Eds.), *Proceedings of Fifth International Conference on Intelligent Tutoring Systems*, Springer-Verlag, pp. 584-593.

Bloom C., Linton F. and Bell B. (1997). Using evaluation in the design of an intelligent tutoring system. In *Journal of Interactive Learning Research*, 8(2): 235-76.

Blythe J. (1999). Decision-Theoretic Planning. *AI Magazine* 20(2): 37-54.

Bonar J. and Soloway E. (1985). Pre-programming knowledge: a major source of misconceptions in novice programmers. *Human-Computer Interaction* 1: 133-161.

Bos E. and van de Plassche J. (1994). A knowledge-based, English verb-form tutor. In *Journal of Artificial Intelligence in Education* 5(1): 107-129.

Bouwer A. (1998). An ITS for Dutch Punctuation. In Goettle B., Halff H., Redfield C., and Shute V. (Eds.) *Proceedings of the 4th International Conference on Intelligent Tutoring Systems*, pp. 224-233, Springer-Verlag.

Brecht B. and Jones M. (1988). Student models: the genetic graph approach. *International Journal of Man-Machine Studies* 28:483-504.

Breese J. and Heckerman D. (1996). Decision-Theoretic Troubleshooting: A Framework for Repair and Experiment. In Horvitz E. and Jensen F. (Eds.) *Proceedings of the Twelfth Conference on Uncertainty in Artificial Intelligence,* San Francisco: Morgan Kaufmann Publishers, Inc.

Brusilovsky, P. (2000). Course Sequencing for Static Courses? Applying ITS Techniques in Large-Scale Web-Based Education. In Gauthier G., Frasson C., and VanLehn K. (Eds.), *Proceedings of Fifth International Conference on Intelligent Tutoring Systems*, Springer-Verlag, pp. 625-634.

Burton J. (1982). Diagnosing bugs in a simple procedural skill. In Sleeman D and Brown J. (Eds.) *Intelligent Tutoring Systems*, New York, NY: Academic Press.

Burton R. and Brown J. (1978). A tutoring and student modelling paradigm for gaming environments. *ACM SIGCSE Bulletin* 8(1): 236-46.

Chao-Lin L. and Wellman M. (1998). Incremental Tradeoff Resolution in Qualitative Probability Networks. In Cooper G. and Moral S. (Eds.) *Fourteenth conference on* *uncertainty in artificial intelligence*. Morgan Kaufmann Publisher, San Francisco, pp. 338-345. Available on the web at http://www2.sis.pitt.edu/~dsl/UAI/uai98.html.

Cheng J., Bell D. and Liu W. (1997). An Algorithm for Bayesian Belief Network Construction from Data, *Proceedings of the 6th International Workshop on AI and Statistics (AI and Stat. '97).*

Cheng J., Bell D. and Liu W. (1998). *Learning Bayesian Networks from Data: An Efficient Approach Based on Information Theory.* On World Wide Web at http://www.cs.ualberta.ca/~jcheng/bnpc.htm

Clancey W. (1983). GUIDON. *Journal of Computer-Based Instruction* 10(1): 8-14.

Clancey W. (1987). *Knowledge-based tutoring: The GUIDON program.* Cambridge, MA: MIT Press.

Collins J., Greer J., and Huang S. 1996. Adaptive Assessment Using Granularity Hierarchies and Bayesian Nets. In Frasson C., Gauthier G., and Lesgold A. (Eds.) *Proceedings of the Third International Conference on Intelligent Tutoring Systems (ITS-96),* Springer-Verlag, pp. 569-577.

Conati C., Gertner A., VanLehn K. and Druzdel M. (1997). On-Line Student Modeling for Coached Problem Solving Using Bayesian Networks. In Kay J. (Ed.), *Proceedings of the Seventh International Conference on User Modeling (UM99)*, Springer-Verlag, pp. 231-242.

Corbett A. and Anderson J. (1992). Student Modeling and Mastery Learning in a Computer-Based Programming Tutor. In Gauthier G. (Ed.), *Proceedings of the Second International Conference on Intelligent Tutoring Systems,* Springer-Verlag, pp. 413-420.

Corbett A. and Anderson J. (1995). Knowledge tracing: modeling the acquisition of procedural knowledge. *User Modeling and User-Adapted Interaction* 4: 253-278.

Corbett A. and Bhatnagar A. (1997). Student Modeling in the ACT Programming Tutor: Adjusting a Procedural Learning Model with Declarative Knowledge. In Jameson A., Paris C., and Tasso C. (Eds.), *User Modeling: Proceedings of the Sixth International Conference, UM97*, pp. 243-254.

Corbett A., Trask H., Scarpinatto K. and Hadley W. (1998). A formative evaluation of the PACT Algebra II Tutor: Support for Simple Hierarchical Reasoning. In Goettle B., Halff H., Redfield C., and Shute V. (Eds.) *Proceedings of the 4th International Conference on Intelligent Tutoring Systems*, Springer-Verlag, pp. 374-383.

Cowell R. (1999). Introduction to Inference in Bayesian Networks. In M. Jordon (Ed.), *Learning in Graphical Models* (pp. 9-26). Cambridge, Massachusetts: MIT Press.

Cowell R., Dawid A., Lauritzen S. and Spiegelhalter D. (1999). *Probabilistic networks and expert systems.* New York, NY: Springer.

D'Ambrosio B. (1999). Inference in Bayesian Networks. *AI Magazine*, 20(2), pp. 21-36.

Dean T. (1991). *Planning and control.* California: Morgan Kaufman.

Dimitrova V., Self J. and Brna P. The interactive maintenance of open learner models. In Lajoie S and Vivet M (eds.) *Artificial Intelligence and Education (AI-ED'99)*, pp. 405-412. Amsterdam: IOS Press.

Elsom-Cook M. (1990). *Guided discovery tutoring: a framework for ICAI Research.* London, UK: Paul Chapman.

Everson H. (1995). Modeling the student in ITS: the promise of a new psychometrics. *Instructional Science* 23, 433-52.

Fargier H. and Perny P. (1999). Qualitative models for decision making under uncertainty without the commensurability assumption. In Laskey K. and Prade H. (Eds.) *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence,* Morgan Kaufmann Publishers, Inc., San Francisco.

Farrell R., Anderson J. and Reiser B. (1984). An interactive computer-based tutor for LISP tutoring. In *Proceedings of the 6th Cognitive Science Society Conference,* Boulder, CO, pp. 152-55.

Forgy C.L. (1982). Rete: a Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem. *Artificial Intelligence* 19, 17-37.

Freedman R., Penstein Rose C., Ringenberg M. and VanLehn K. (2000). ITS Tools for natural language dialogue: a domain-independent parser and planner. In Gauthier G., Frasson C., and VanLehn K. (Eds.), *Proceedings of Fifth International Conference on Intelligent Tutoring Systems*, Springer-Verlag, pp. 433-442.

Ganeshan R., Lewis Johnson W., Shaw E. and Wood, B.P. (2000). Tutoring Diagnostic Problem Solving. In Gauthier G., Frasson C., and VanLehn K. (Eds.), *Proceedings of Fifth International Conference on Intelligent Tutoring Systems*, Springer-Verlag, pp. 33-42.

Gärdenfors P. (1989). The dynamics of normative systems. In Martino A.,*Proceedings of the 3rd International Congress on Logica, Informatica, Dritto*, pp. 293-299. Consiglio Nazionale delle Ricerche, Florence 1989. Also published in A.A. Martino (Ed.) *Expert Systems in Law,* Elsevier, pp. 195-200, 1991.

Gertner A. (1998). Providing feedback to equation entries in an intelligent tutoring system for Physics. In Goettle B., Halff H., Redfield C., and Shute V. (Eds.) *Proceedings of the 4th International Conference on Intelligent Tutoring Systems*, Springer-Verlag,, pp 254-263.

Gertner A. and VanLehn K. (2000). Andes: A Coached Problem Solving Environment for Physics. In Gauthier G., Frasson C., and VanLehn K. (Eds.), *Proceedings of*

Fifth International Conference on Intelligent Tutoring Systems, Springer-Verlag, pp. 133-142.

Gertner A., Conati C. and VanLehn K. (1998). Procedural help in Andes: Generating hints using a Bayesian network student model. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*, MIT Press, pp. 106-111.

Gluck K., Shute V., Anderson J. and Lovett M. (1998). Deconstructing a computer-based tutor: striving for better learning efficiency in Stat-Lady. In Goettle B., Halff H., Redfield C., and Shute V. (Eds.) *Proceedings of the 4th International Conference on Intelligent Tutoring Systems*, Springer-Verlag, pp. 66-75.

Good T. and Brophy J. (1990). *Educational psychology: A realistic approach* (4th Ed.) White Plains, NY: Longman.

Harmer B. (1998). *Too many New Zealanders have poor literacy skills.* On World Wide Web at http://www.nz.com/NZ/News/WYSIWYG/1998_News/1998November15.html.

Heckerman, D. (1999). A Tutorial on Learning with Bayesian Networks. In M. Jordon (Ed.), *Learning in Graphical Models* (pp. 301-354). Cambridge, Massachusetts: MIT Press.

Holt P., Dubs S., Jones M. and Greer J. (1994). The State of Student Modelling. In: Greer, J.E., McCalla, G.I. (Eds.): *Student Modelling: the Key to Individualized Knowledge-based Instruction*, pp. 3-38. NATO.

Horvitz E. (2000). *Uncertainty, Utility, and Understanding.* Invited Presentation given at ITS 2000, the Fifth International Conference on Intelligent Tutoring Systems.

Horvitz E., Breese J. and Henrion M. (1988). Decision Theory in Expert Systems and Artificial Intelligence. *International Journal of Approximate Reasoning* 2:247-302; also On World Wide Web at http://www.auai.org/auai-tutes.html

Howard R.A. and Matheson J.E. (1984). Influence diagrams. In Howard R.A. and Matheson J.E. (Eds.), *Readings on the principles and applications of decision analysis*, vol. 2, 719-62. SDG, California.

Jensen, F., and Jensen, F. (1994). Optimal Junction Trees, *Proceedings of the 10th Conference on Uncertainty in Artificial Intelligence (UAI-94).*: Morgan Kaufman.

Jensen, F., and Lauritzen, S. (2000). Probabilistic Networks. In *Algorithms for Uncertainty and Defeasible Reasoning* (Vol. 5, pp. 289-320): Kluwer Academic Publishers.

Jensen, F., Lauritzen, S. and Oleson, K. (1990). Bayesian Updating in Causal Probabilistic Networks by Local Computations. *Computational Statistics Quarterly, 4*, 269-282.

Jensen, F., Oleson, K. and Andersen, S. (1990). An Algebra for Bayesian Belief Universes for Knowledge-Based Systems. *Networks, 20*, 637-659.

Katz S., Lesgold A., Eggan G., and Gordin M. (1992). Modelling the student in Sherlock II. *Journal of Artificial Intelligence and Education* 3(4): 495-518.

Kay J. (2000a). Stereotypes, Student Models and Scrutability. In Gauthier G., Frasson C., and VanLehn K. (Eds.), *Proceedings of Fifth International Conference on Intellig ent Tutoring Systems*, Springer-Verlag, pp. 19-30.

Kay J. (2000b). Accretion representation for scrutable student modelling. In Gauthier G., Frasson C., and VanLehn K. (Eds.), *Proceedings of Fifth International Conference on Intelligent Tutoring Systems*, Springer-Verlag, pp. 292-303.

Kjaerulff, U. (1992). Optimal decomposition of probabilistic networks by simulated annealing. *Statistics and Computing*, 2, 7-17.

Kjaerulff, U. (1999). Inference in Bayesian Networks using Nested Junction Trees. In M. Jordon (Ed.), *Learning in Graphical Models*, pp. 51-74. Cambridge, MA: MIT Press.

Koedinger K., Anderson J., Hadley W. and Mark M. (1997). Intelligent Tutoring Goes To School in the Big City. In *International Journal of Artificial Intelligence in Education* 8: 30-43.

Kohonen T. (1997). *Self-organizing maps*. New York: Springer.

Kozlov A. and Pal Singh J. (1994). A Parallel Lauritzen-Speigelhalter Algorithm for Probabilistic Inference, *Proceedings of the Conference of Supercomputing '94*, pp. 320-329.

Krause P. (1998). *Learning Probabilistic Networks*. Surrey, United Kingdom: Philips Research Laboratory. On World Wide Web at http://www.auai.org/auai-tutes.html

Langley P., Wogulis J., and Ohlsson S. (1990). Rules and principles in cognitive diagnosis. In N. Fredericksen, R. Glaser, A. Lesgold, and M. G. Shafto (Eds.), *Diagnostic monitoring of skill and knowledge acquisition.* Hillsdale, NJ: Lawrence Erlbaum.

Lauritzen S. and Spiegelhalter D. (1988). Local computations with probabilities on graphical structures and their application to expert systems (with discussion). *Journal of the Royal Statistical Society Series B*, 50, pp. 157-224.

Mark M. and Greer J. (1993). *Evaluation methodologies for Intelligent Tutoring Systems.* ARIES Laboratory, Department of Computational Science, University of Saskatchewan, Saskatoon, Canada.

Martin B. and Mitrovic A. (2000). Tailoring Feedback by Correcting Student Answers. In Gauthier G., Frasson C., and VanLehn K. (Eds.), *Proceedings of Fifth*

International Conference on Intelligent Tutoring Systems, Springer-Verlag, pp. 383-392

Matz M. (1982.) Towards a process model for high school algebra errors. In Sleeman D. & Brown J. (Eds.) *Intelligent Tutoring Systems*, London: Academic Press, pp. 25-50.

Mayo M. (2000). *A Normative Framework for Intelligent Tutoring Systems.* Poster accepted and presented at *ITS 2000* in Montreal, Canada.

Mayo M. and Mitrovic A. (1999). Estimating Problem Value in an Intelligent Tutoring Systems using Bayesian Networks. In *Proceedings 12th Australian Joint Conference on AI (AI '99)*, pp. 472-473. See also Appendix D, this volume.

Mayo M. and Mitrovic A. (2000). Using a Probabilistic Student Model to Control Problem Difficulty. In Gauthier G., Frasson C., and VanLehn K. (Eds.), *Proceedings of Fifth International Conference on Intelligent Tutoring Systems*, Springer-Verlag, pp. 524-533. See also Appendix E, this volume.

Mayo M. and Mitrovic A. (2001). Optimising ITS Behaviour with Bayesian Networks and Decision Theory. *International Journal of Artificial Intelligence and Education,* 12. See also Appendix G, this volume.

Mayo M., Mitrovic A. and McKenzie J. (2000). CAPIT: An Intelligent Tutoring System for Capitalisation and Punctuation. In Kinshuk, Jesshope C. and Okamoto T. (Eds.) *Advanced Learning Technology: Design and Development Issues*, Los Alamitos, CA: IEEE Computer Society (ISBN 0-7695-0653-4), pp. 151-154. See also Appendix F, this volume.

McClave J. and Benson P. (1991). *Statistics for Business and Economics (Fifth Edition).* Dellan Publishing Company.

Mergel B. (1998). *Instructional Design and Learning Theory.* On World Wide Web at http://www.usask.ca/education/coursework/802papers/mergel/brenda.htm

Microsoft, 2000. Pattern property. In *Visual Basic Scripting Edition Language Reference*, available online at http://msdn.microsoft.com/scripting/vbscript/doc/vspropattern.htm.

Millán E., Pérez-de-la-Cruz J. and Suárez E. (2000). Adaptive Bayesian Networks for Multilevel Student Modelling. In Gauthier G., Frasson C., and VanLehn K. (Eds.), *Proceedings of Fifth International Conference on Intelligent Tutoring Systems*, Springer-Verlag, pp. 534-543.

Milne S., Shiu E. and Cook J. (1996). Development of a model of user attributes and its implementation within an adaptive tutoring system. *User modeling and user-adapted interaction* 6: 303-335.

Mislevy R. and Gitomer D. (1996). *The Role of Probability-Based Inference in an Intelligent Tutoring System.* CSE Technical Report 413, CRESST. On World Wide Web at http://cresst96.cse.ucla.edu/ CRESST/pages/reports.htm.

Mitrovic A. (1998). Experiences in Implementing Constraint-Based Modeling in SQL-Tutor. In Goettle B., Halff H., Redfield C., and Shute V. (Eds.) *Proceedings of the 4th International Conference on Intelligent Tutoring Systems*, Springer-Verlag, pp. 414-423.

Mitrovic A. and Ohlsson S. (1999). Evaluation of a Constraint-Based Tutor for a Database Language. *Int. J. on Artificial Intelligence in Education*, 10(3-4), 238-256.

Mitrovic A., Djordjevic-kajan S. and Stoimenov L. (1996). INSTRUCT: Modeling Students by Asking Questions. *User Modeling and User-Adapted Interaction*, 6(4) 273-302.

Murray R. and VanLehn K. (2000). DT Tutor: A Decision-Theoretic, Dynamic Approach for Optimal Selection of Tutorial Actions. In Gauthier G., Frasson C., and VanLehn K. (Eds.), *Proceedings of Fifth International Conference on Intelligent Tutoring Systems*, Springer-Verlag, pp. 153-162.

Murray W. (1998). A Practical Approach to Bayesian Student Modeling. In Goettle B., Halff H., Redfield C., and Shute V. (Eds.) *Proceedings of the 4th International Conference on Intelligent Tutoring Systems*, Springer-Verlag, pp. 424-433.

Murray W. (1999). An easily implemented, linear time algorithm for Bayesian student modeling in multi-level trees. In Lajoie S. and Vivet M., *Proceedings of the 9th International Conference on Artificial Intelligence and Education (AI-ED 99)*, IOS Press, pp.413-420.

Neal R. and Hinton, G. (1999). A View of the EM Algorithm that Justifies Incremental, Sparse, and Other Variants. In M. Jordon (Ed.), *Learning in Graphical Models* (pp. 355-370). Cambridge:MA: MIT Press.

Ohlsson S. (1994). Constraint-based Student Modelling. In: Greer, J.E., McCalla, G.I. (Eds.): *Student Modelling: the Key to Individualized Knowledge-based Instruction*, pp. 167-189. NATO.

Ohlsson S. (1996). Learning from performance errors. *Psychological Review*, 103, pp. 241-262.

Pearl J. (1988). *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference* (2nd ed.): Morgan Kauffman.

Quinlin J. (1993). *C4.5: Programs for Machine Learning.* San Mateo, CA: Morgan Kauffman.

Reye J. (1998). Two-Phase Updating of Student Models Based on Dynamic Belief Networks. In Goettle B., Halff H., Redfield C., and Shute V. (Eds.) *Proceedings of* *the 4th International Conference on Intelligent Tutoring Systems*, Springer-Verlag, pp. 274-283.

Russell S. and Norvig P. (1995). *Artificial Intelligence: A Modern Approach.* Prentice-Hall.

Savage L. (1954). *The Foundations of Statistics.* Wiley.

Self J. (1988). Bypassing the intractable problem of student modelling. In *Proceedings ITS'88*, pp. 18-24.

Self J. (1994). Formal approaches to student modelling. In Greer, J.E., McCalla, G.I. (Eds.): *Student Modelling: the Key to Individualized Knowledge-based Instruction*, pp. 295-354. NATO.

Shachter R., Andersen S. and Szolovits P. (1991). *The equivalence of exact methods for probabilistic inference on belief networks,* Department of Engineering, Stanford University. Department of Electrical Engineering.

Shafer, G. (1986). Probability judgement in artificial intelligence. In L. Kanal and J. Lemmer (Eds.), *Uncertainty in Artificial Intelligence*. New York: North-Holland.

Simon H. (1976). *Administrative Behavior. A study of decision-making processes in administrative organization.* New York: Free Press.

Singley M. (1995). Promoting Transfer Through Model Tracing. In McKeough A., Lupart J. and Marini A. (Eds.) *Teaching for transfer.* Lawrence Erlbaum: NJ.

Sleeman D. (1982). Assessing aspects of competence in basic algebra. In Sleeman D. & Brown j. (Eds.) *Intelligent Tutoring Systems*, Academic Press: London, pp. 185-199.

Sleeman D. and Smith M. (1981). Modelling students' problem solving. *Artificial Intelligence* 16: 171-187.

Sleeman D., Kelly A., Martinak W. and Moore S. (1989). Studies of diagnosis and remediation with high school algebra students. *Cognitive Science* 13:467-506.

Soloway E. and Johnson W. (1981). Remembrance of blunders past: a retrospective on the development of PROUST. In *Proceedings of the 6th Cognitive Science Society Conference,* Boulder, CO, pp. 57.

Stern M., Beck J. and Woolf B. (1999). *Naïve Bayes Classifiers for User Modeling.* Center for Knowledge Communication, Computer Science Department, University of Massachusetts.

VanLehn K. (1982). Bugs are not enough: Empirical studies of bugs, impasses and repairs in procedural skills. *Journal of Mathematical Behaviour* 3: 3-72.

VanLehn K. (1990). *Mind Bugs.* Cambridge, MA: MIT Press.

VanLehn K. and Martin J. (1997). Evaluation of an assessment system based on Bayesian student modeling. In *International Journal of Artificial Intelligence in Education*, 8, 179-221.

VanLehn K., Niu Z., Siler S. and Gertner A. (1998). Student Modeling from Conventional Test Data: A Bayesian Approach without Priors. In Goettle B., Halff H., Redfield C., and Shute V. (Eds.) *Proceedings of the 4th International Conference on Intelligent Tutoring Systems*, Springer-Verlag, pp. 434-443.

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

Virvou M. and Tsirgara V. (2000.) Involving effectively teachers and students in the life cycle of an Intelligent Tutoring System. *Educational Technology & Society* 3(3): 511-521.

Webb G., Chiu B. and Kuzmycz M. (1997). Comparative evaluation of alternative induction engines for feature based modelling. In *International Journal of Artificial Intelligence in Education*, 8, 97-115.

Wheeler J. and Regian J. (1999). The use of a cognitive tutoring system in the improvement of the abstract reasoning component of word problem solving. In *Computers in Human Behaviour*, 15: 243-254.

Winter M. and McCalla G. (1999). The emergence of student models from an analysis of ethical decision making in a scenario-based learning environment. In Kay J. (Ed.), *Proceedings of the Seventh International Conference on User Modeling (UM99)*, Springer-Verlag, pp. 265-74.

Yannakakis, M. (1981). Computing the minimal fill-in is NP-Complete. *SIAM Journal on Algebraic and Discrete Methods* 2, 77-79.

Zadeh, L. (1983). The role of fuzzy logic in the management of uncertainty in expert systems. *Fuzzy Sets and Systems, 11*, 199-227.