Constraint-Based Knowledge Representation for Individualized Instruction

Stellan Ohlsson and Antonija Mitrovic

[1]Department of Psychology, University of Illinois at Chicago
1007 West Harrison Street, Chicago, IL 60607
stellan@uic.edu
[2]Intelligent Computer Tutoring Group, Computer Science Department
University of Canterbury, Private Bag 4800, Christchurch, New Zealand
tanja@cosc.canterbury.ac.nz

Abstract. Traditional knowledge representations were developed to encode complete, explicit and executable programs, a goal that makes them less than ideal for representing the incomplete and partial knowledge of a student. In this paper, we discuss *state constraints*, a type of knowledge unit originally invented to explain how people can detect and correct their own errors. Constraint-based student modeling has been implemented in several intelligent tutoring systems (ITS) so far, and the empirical data verifies that students learn while interacting with these systems. Furthermore, learning curves are smooth when plotted in terms of individual constraints, supporting the psychological appropriateness of the representation. We discuss the differences between constraints and other representational formats, the advantages of constraint-based models and the types of domains in which they are likely to be useful.

1. Introduction

The promise of Artificial Intelligence (AI) is to make artifacts responsive to human needs and to varying conditions. Thirty years ago, Allen Newell [31] envisioned an enchanted world in which brakes know how to stop a car on wet pavement, bridges watch out for the safety of those who cross them, instruments converse with their users and street lights help people find their way. The ability of AI-enhanced artifacts to behave conditionally is a source of magic, if we can only learn to harness it to our purposes.

In the context of education, the purpose of putting AI inside instructional materials--electronic books, intelligent tutoring systems, training simulations, etc.--is to make those materials responsive to the individual learner. Learners vary in amount and kind of prior knowledge, cognitive ability, learning style, natural pace, working memory capacity and other aspects. Consequently, one and the same instructional sequence cannot provide optimal learning for all

learners. Educational theorists recognized the value of individualized instruction decades ago [11], but they had no technology for delivering such instruction on a massive scale. Today, we do.

In the absence of AI, the only alternative to fixed and predefined instructional sequences are experiences that are shaped by the learner: exploratory activities, group discussions, self-directed inquiry and so on. There is little doubt that open-ended instruction of this sort can engage students, increase their motivation and support mastery of many subject matter topics.

However, open-ended instruction also has drawbacks and limits. An instructional activity that is under the control of the learner might veer away from the targeted subject matter. The students are likely to learn something, but the instructor might not be able to steer them towards a particular topic, making this type of instruction difficult to use in public schools and other instructional institutions that assume a prespecified curriculum. Also, open-ended instruction is not appropriate for all types of subject matter. It is most appropriate in domains where there are no right or wrong answers; less so in, for example, physics and programming. Furthermore, open-ended instruction tends to be time consuming, sometimes requiring hours of instruction to teach a small fraction of a crowded instructional agenda. By asking each learner to re-discover well-known facts and principles on their own, open-ended instruction negates the advantages of cultural transmission of knowledge. Finally, the educational research literature does not contain a wealth of hard data that confirm the pedagogical efficiency that advocates often claim for discovery, exploration, free inquiry and related forms of instruction.

Artificial intelligence offers an escape from the dilemma of choosing between teacher-specified instruction and student-controlled learning. AI techniques can be used to construct interactive instructional materials that respond flexibly to the student. The instruction provided by such materials need not consist of a predefined instructional sequence, nor need it be entirely under the student's control. Instead, it provides individualized guidance on the path to mastery of a specified subject matter.

To date, this potential has primarily been realized with respect to cognitive skills in well-defined domains. The set of such domains includes many school topics such as algebra, arithmetic, calculus, geometry, physics and the theory of electricity [14; 49], but also topics such as formal logic and computer programming [46]. This is a wide class of worthwhile instructional targets.

The key step in the AI approach to individualized instruction is to equip the instructional system with formal representations of both the target subject matter and the learner. The response of the system at each moment in time is computed on the basis of those representations. This feature differentiates AI-based systems from other types of computer-based instructional systems.

Although explicit knowledge representation is the source of AI power, it is also the main bottleneck in system development. Representing the target subject matter might pose difficult or unsolvable research problems. How to represent time and space are examples. In general, instructional domains which blend seamlessly into common sense and which require unrestricted

natural language capabilities are currently beyond the reach of the AI approach. However, within the large universe of well-defined problem solving domains, the representational problems can usually be solved.

The harder question is how to represent the student. There are three main difficulties. First, the student's knowledge is partially incorrect and partially correct. Knowledge representation formalisms were developed to represent correct or expert knowledge. Adapting them to represent incorrect knowledge is not trivial. Second, the universe of incorrect knowledge is vast, and to diagnose exactly which incorrect representation of the target domain is controlling the learner's behavior is a difficult problem [33, 35, 43]. Finally, the student's knowledge, unlike the target subject matter, is a dynamical entity. It changes in the course of an instructional session -- or so one would hope -- so an AI-based system needs a systematic technique for updating its representation of the student on line.

Artificial intelligence researchers have invented several knowledge representations that are by now well understood in a formal sense and embodied in languages and tools for system development. However, those knowledge representations were not developed specifically for use in instructional systems and consequently there is no reason to expect them to be optimal for that purpose.

But there is no reason to limit educational systems to the traditional knowledge representations. In past work [35], we proposed a new format for representing knowledge, based on the notion of a state constraint. In this paper, we first set the stage for Constraint-Based student Modeling (CBM) by identifying the central weaknesses in the classical knowledge representations. Second, we develop the basic ideas behind constraint-based knowledge representation, and then discuss how constraint-based modeling can be used in educational systems. Third, we discuss the appropriateness of constraints as knowledge representation with respect to empirical evaluations performed on some constraint-based tutors. We end with a general discussion of the characteristics of the instructional domains in which constraint-based modeling is likely to be useful.

2. Traditional Knowledge Representations

It is convenient to distinguish between three traditional knowledge representations: propositions, procedures and rules. *Propositional representations* are descendants of formal logic and linguistics. Logic-based programming languages represent attempts to embed all knowledge within the concept of propositional knowledge, an approach which has devoted followers but which has not come to dominate AI work. However, propositions are fundamental in the sense that they are used to encode declarative knowledge even in systems that use other means than propositions to represent procedural knowledge. The system developed by Sleeman, Kelly, Martinak, Ward and Moore [45] and Sleeman, Hirsch, Ellery and Kim [44] for

high-school algebra illustrates the use of logic programming in student modeling.

For brevity, we lump functional languages like Lisp--the classical AI language--together with Pascal and other procedural languages that represent functions (procedures) as combinations of, or calls on, other functions (subprocedures). Such *procedural representations* are particularly suitable for encoding hierarchical plans or strategies when the ordering of problem solving steps is crucial for correct performance. Instructional examples include Buggy and Debuggy, the paradigm-creating systems in the context of which the question of student modeling was first posed [6, 7].

Other instructional systems are based on *rule-based representations*. Such representations consist of a declarative (propositional) knowledge base, a rule set and an interpreter that executes the rules vis-à-vis the knowledge base. This approach provides flexibility in execution and is thus particularly suitable for domains in which the ordering of problem solving steps can vary. The series of instructional systems built by John R. Anderson, Kenneth Koedinger, Albert Corbett and their co-workers at Carnegie-Mellon University illustrate the power of rule-based knowledge representations for delivering individualized instruction [2, 3, 14].

Although these three types of knowledge representation offer varying advantages and strengths, they share a common weakness that we refer to as *overspecificity*. A knowledge base consisting of Horn clauses, Lisp functions or production rules is a highly articulate and detailed model of what a student knows. For example, if a rule-based student model consists of a hundred rules (a conservative estimate) and each rule consists of ten atomic expressions (also a conservative estimate), then such a model makes no less than 1,000 micro-claims about what is in the student's head.

This level of specificity in the student model cannot be supported by the empirical data available to an instructional system. Observable problem solving behavior unfolds at a time scale that is one or two orders of magnitude larger than the time it takes to execute a primitive procedure or a rule. *The main obstacle to building an instructional system that can infer students' knowledge on-line is that the behavioral data available to the system cannot discriminate between the multiple possible models at the level of specificity required by the traditional AI knowledge representations.* This is the main problem that has limited the application and usefulness of AI to instructional systems to date.

The second problem associated with traditional knowledge representations is that they were designed to encode executable programs. However, executability forces a level of consistency and completeness onto a student model that is unrealistic, both in the sense that the learner is unlikely to exhibit that level of consistency and completeness, and in the sense that inferring a consistent and complete model from student responses to practice problems is a computationally intractable task. Overspecificity and executability are closely related. Traditional knowledge representations demand specificity because they are meant to be executable, and a program cannot be executed unless it is complete, as novice programmers quickly discover.

The upshot is that instructional AI systems that operate with traditional knowledge representations create models of students that are underdetermined by the empirical data and yet fully specified. Techniques for student modeling can be conceptualized as attempts to deal with this dilemma. For example, model tracing, the highly successful technique used in the array of CMU tutors [2, 14], makes only local inferences from a single step to a single underlying rule and thus side steps the need for executability of the rule set as a whole. (This move also negates the main reason to be interested in rule-based representations, namely the potential for flexible execution.) Alternative approaches to the overspecificity problem include Bayesian networks which assign and update probabilities to the individual knowledge units [10, 20, 22] and fuzzy logic [13].

The problem of empirically underdetermined specificity is unproductive because the level of specificity demanded by the traditional knowledge representations is not pedagogically motivated. The task of an instructional system is to map pedagogical situations onto instructional actions. But no intelligent tutoring system has an infinitely fine grained repertoire of instructional actions. In the basic case, the ITS has a fixed number of instructional messages and an instructional action consists of presenting one of those messages, perhaps with more or fewer details included. In some cases, the system can also choose which practice problem or exercise to present next. A few systems can choose to quiz the student instead of presenting more instruction.

To support pedagogical choices of this sort, the level of specificity presumed by the traditional knowledge formats is not needed. To decide that instructional message X is appropriate, the system does not need to know exactly what knowledge (correct or incorrect) the student has. There is a large set of cognitive states in which hearing or reading X might be beneficial. To decide to present message X, an instructional system only needs to know that the student is in one of those states. To discriminate between knowledge states at a finer grain of detail does not enhance the pedagogical power of the system.

In summary, the classical AI knowledge representations are unsuitable for student modeling because they require specificity and are designed for executability. These two related features are the main causes of the intractability of the student modeling problem. Instructional applications of AI might benefit from a representation that allows pedagogically relevant forms of abstraction. We next introduce such a representation.

3. Constraint-Based Knowledge Representation

A constraint-based model represents knowledge about a domain as a set of constraints on correct solutions in that domain. The constraints select, out of the universe of all possible solutions, the set of correct solutions. More

precisely, they partition the universe of possible solutions into the correct and the incorrect ones.

We need not assume a formal criterion or definition of correctness. Constraint-based modeling can be applied to any type of behavior. Constraints can represent aesthetic or moral judgments as well as, for example, arithmetic correctness. Indeed, the notion of correctness used in a given application can be seen as operationally *defined* by the set of constraints.

One consequence of partial mastery of a domain is that the learner is incapable of recognizing some incorrect solutions as incorrect. That is, the learner makes errors. We can represent such partial mastery of a domain with an incomplete set of constraints. An incomplete set will not strictly circumscribe the set of correct solutions, but a larger universe of solutions. A set of constraints that identifies precisely those solutions that the learner recognizes or believes to be correct is a model of what the learner knows about the domain.

Constraint-based knowledge representations have unfamiliar but useful features. They were invented in response to a deep puzzle in the theory of skill acquisition. We summarize the theoretical rationale before describing a formal notation for constraints and its instructional applications.

3.1. Theoretical Rationale

Human beings can catch themselves making errors. For example, in making a speech error such as saying “left” instead of “right”, the speaker often corrects his or her error in the next sentence or phrase (“No, wait, I meant ‘right’ “). See Norman [32], Ohlsson [37] and Reason [42] for further development of this observation.

The ability to catch one’s own errors is paradoxical. If a person does not have enough knowledge to recognize an action or discourse as erroneous, then how can he or she catch the error? But if the person does have enough knowledge to recognize it as erroneous, then why is that action or discourse issued in the first place? The ability to catch one’s errors forces a distinction between *generative* and *evaluative* knowledge [32, 37]. On the one hand, there must be a system (e.g., a rule set) for generating actions that have some probability of being appropriate, correct or useful in the current context. When this knowledge base is incomplete or incorrect, errors result. On the other hand, there must be a separate knowledge base for evaluating the (outcome of) an action and judging it as correct or incorrect, as the case might be. These two knowledge bases are independent in the sense that a piece of knowledge that appears in one does not necessarily also appear in the other. The interaction between these two knowledge systems is what we subjectively experience as catching ourselves making an error.

The distinction between generative and evaluative knowledge suggests a particular hypothesis about how cognitive skills are learned [36, 37, 38, 39,

40]. A learner approaches an unfamiliar task with some set of dispositions, heuristics, orienting attitudes and strategies acquired in past experience. These suffice to generate task relevant actions but they do not guarantee that those actions are correct (or else the task is not unfamiliar after all). The result is exploratory, tentative behavior (heuristic search; trial and error). Evaluative knowledge is used to judge the outcomes and consequences of such behavior.

When the outcomes are found to be inappropriate, useless or incorrect (in some sense that depends on the task domain), then the response on the part of the learner is to revise the generative knowledge so as to avoid repeating that same error in the future. Over time, the information in the evaluative knowledge is gradually incorporated into the generative knowledge and the capability of the latter to generate correct actions gradually increases. The central process in skill acquisition is the migration of structure from the evaluator to the generator.

This hypothesis only applies in scenarios in which the learner has *some* declarative knowledge of what the correct solutions looks like, even before he or she is able to reliably generate them. This is in accord with intuition as well as instructional practice. Consider the task of recovering from being lost in unfamiliar surroundings. If the surroundings are truly unfamiliar, then one cannot tell, after walking for a while, whether one is closer or further from one's goal. Without ability to recognize the correct path, the walker cannot make an informed judgment about the path he or she is taking. Descriptive knowledge of what the desired solution looks like is crucial for catching oneself making errors and correcting wrong choices.

Instructional practice implicitly recognizes this fact. In most instructional scenarios learners are given verbal descriptions of the desired performance before working on practice problems, usually in the form of a lecture or review of a solved practice problem. This is peculiar, because it is well known that a verbal description of a skill does not confer proficiency in that skill; why then provide such a description? According to the theory of learning from error the function of such descriptions is to provide the learner with evaluative knowledge, i. e., a set of constraints by which he or she can catch his or her own errors and thereby propel himself or herself down the learning curve.

A detailed statement of the theory of learning from error is available in [37] and its instructional implications are developed in [38]. Computer simulations that model learning from error in arithmetic and college chemistry have been reported in Ohlsson [36], Ohlsson, Ernst and Rees [39] and Ohlsson and Rees [40].

3.2. A Formalism for State Constraints

Ohlsson and Rees [40] introduced a formal notation for constraints. The unit of knowledge is called a *state constraint*. Each state constraint is an ordered pair

$$<C_r, C_S>,$$

where C_r, the *relevance condition*, identifies the class of problem states for which the constraint is relevant, and C_S, the *satisfaction condition*, identifies the class of (relevant) states in which the constraint is satisfied. Each member of the pair can be thought of as a set of features or properties of a problem state. Thus, the semantics of a constraint is: *if the properties C_r hold, then the properties C_s have to hold also* (or else something is wrong).

The following example is taken from a well-known puzzle problem, the Tower of Hanoi[1]:

```
If disc X is on peg Z and disc Y is on peg Z and X is
on top of Y,

then X is smaller than Y (or else there is an error).
```

In this example, C_r, the relevance criterion, is the complex predicate *disc X is on peg Z and disc Y is on peg Z and X is on top of Y,* and C_s, the satisfaction criterion, is the predicate *X is smaller than Y.*

A simple example from the domain of Lisp programming is the following constraint:

```
If the code for a Lisp function has N left parentheses,

there has to be N right parentheses as well (or else
there is an error).
```

In this example, *the code has N left parentheses* is the relevance criterion and *the code has N right parentheses* is the satisfaction criterion. This example has the unusual feature that the relevance criterion is always satisfied, so the constraint is always relevant. In practice, this is not typical. For example, in SQL-Tutor, the system discussed in [30], approximately 10% of the constraints turn out to be relevant in every problem state.

A state constraint can be implemented in a variety of ways. The most obvious way is to code each constraint as a pair of *patterns*, where each pattern is a list (conjunction or disjunction) of elementary propositions which may or may not contain variables. In this implementation, each half of a constraint is analogous to the condition side of a production rule. Alternatively, state constraints can be implemented as pairs of Lisp predicates. The important point is that each state constraint is a pair of (possibly complex) tests on problem states.

Constraints are modular. Unlike procedures, constraints do not interact directly with each other--a constraint does not pass results to other

---

[1] In the Tower of Hanoi problem, a stack of discs with holes in the center are to be moved from one peg to another in accordance with a set of rules: Only one disc is to be moved at a time, a larger disc cannot be on top of a smaller one, and at the end, the stack of discs should be on a specified peg.

constraints--so they are comparatively easy to implement and debug. Constraints are also general. They express criteria of correctness that hold throughout the target domain rather than being specific to a particular problem. All constraints are conceptualized as being at the same level of abstraction and they are applied in parallel. No hierarchy or other types of organization is imposed on the constraint base. However, such a structure may be beneficial when the instructional system supports metacognitive skills [12].

We distinguish between two types of constraints. Constraints of the first type represent syntactic properties of the domain, and are typically simple and easy to formulate. They refer only to the student's solution. Constraints of the second type represent semantic properties of the domain. They operate on the relation between the student's solution and the ideal solution. Semantic constraints are typically more complex than syntactic constraints. Of course, the distinction between the two kinds of constraints is not strict and some constraints inspect both the syntax and the semantics of the student's solution.

The Tower of Hanoi constraint and the Lisp constraint discussed previously are examples of syntactic constraints they apply to every problem state in their corresponding domains. The following constraint from SQL-Tutor is an example of syntax constraints:

```
If the student's query contains a nested SELECT,

Then it must be preceded with a comparison operator or
a predicate (IN, ANY, ALL or EXISTS).
```

This constraint specifies a particular syntax rule from SQL. An example semantic constraint from the same domain is:

```
If the ideal solution contains a condition using the
BETWEEN predicate, and the student's query tests
whether the same attribute is less than or equal to a
constant,

There should be another search condition in the
student's query, checking whether the attribute's value
is greater or equal to another constant, and the two
constants should be the same ones used in the BETWEEN
condition in the ideal solution.
```

This semantic constraint allows the student to use an alternative solution, different from the ideal solution, by checking that all the necessary conditions have been specified by the student correctly.

3.3. Using constraints to diagnose students' solutions

As discussed earlier, constraints are used to develop the model of the domain, and are used to evaluate the student's solutions. Hence, a system

using this representation has to have an internal representation of the current problem state. If the state constraints are implemented as patterns, then it is convenient to represent the problem state as a list (conjunction) of elementary propositions. (Such a list is analogous to the working memory of a production system architecture.) Furthermore, the system must update that representation when the state of the problem changes. The actions or processes producing the changes need not necessarily be represented but their effects on the problem state have to be.

Once the problem state and the set of constraints have been encoded, the computations required to test whether a given problem state is consistent with a set of constraints are straightforward: compare the state against all constraints and notice any constraint violations. This is a two step process. In the first step, all the relevance patterns are tested against the problem state to identify those constraints that are relevant in that state. In the second step, the satisfaction patterns of the *relevant* constraints are tested against the problem state. If the satisfaction pattern of a relevant constraint matches the current state, then that constraint is satisfied. If the satisfaction pattern of a relevant constraint is not satisfied, then that state violates the constraint.

The algorithm needed to test the constraints against the current state depends on the type of encoding used. If the constraints are pairs of patterns, then the comparison can be carried out with a standard pattern matching algorithm, e. g., a RETE network [8]. If the constraints are pairs of Lisp predicates, no special algorithm is, in principle, needed, although our experience indicates that the evaluation of a large constraint base might be too slow to be practical without a pattern-matching component.

The short-term model of the student consists of the list of satisfied and the list of violated constraints, which enables a constraint-based tutor to generate feedback to the student. It is also necessary to model long-term knowledge of the student as well, to support other types of pedagogical decisions (such as instructional planning). The simplest way of modeling long-term knowledge of the student is to use overlay models. Such a student model contains the summary information about the student's usage of each individual constraint. Early versions of SQL-Tutor [23, 30] were based on such models. It is also possible to develop probabilistic student models on the basis of constraint-based diagnosis of student's answers [22].

3.4. Using Constraints for Other Pedagogical Tasks

We have emphasized the diagnostic use of a constraint-base because the constraint idea was originally developed in response to the intractability of the student modeling problem. However, a constraints can also be used to support other tasks and functions in an intelligent tutoring system.

WETAS [19] is constraint-based authoring shell that supports the development of constraint-based tutors by supporting user interface, student modeling, problem selection and feedback generation. In order to develop a

new tutor, the author needs to provide a description of the domain, the set of constraints, and the set of problems with their solution. Furthermore, WETAS extends CBM to provide novel functions, such as correcting the student's answer, generating problems and problem solving. Starting from an incorrect student answer, WETAS uses violated constraints to find out the correct fragments to be used in the solution [17]. This allows for better feedback to be given to the student, especially in cases when the student's problem solving strategy differs substantially from the one used in the ideal solution. Problem generation is based on the same idea, but it starts from an empty solution, rather than from an incorrect solution [18]. Finally, WETAS can generate problems starting from constraints. The role of the author is just to make sure that the generated problems (and their solutions) are meaningful, and write the textual description of the problems.

Besides supporting the acquisition of declarative and procedural knowledge (i.e. problem-solving skills), CBM may be also used to support metacognitive skills. Several studies performed on SQL-Tutor [24, 28] and KERMIT [12] extended CBM to support reflection by opening the student model. Furthermore, two studies performed on KERMIT [50, 51] and NORMIT [26, 27] show the CBM may also be used successfully to support students while explaining their actions, thus relating the problem-solving activities to their declarative knowledge.

3.5. Overcoming the Overspecificity Problem

How does the state constraint knowledge representation circumvent the overspecification problem that we have identified as the main weakness in the traditional knowledge representations?

If the constraints are formulated in a psychologically appropriate way, the system will evaluate a student's solutions as correct or incorrect in the same way as the student would, if he or she knew more knowledge about the domain. In effect, by containing the constraints that the student would have, had he or she already attained mastery, the system plays the role of an amplified evaluative knowledge base. The hope is that access to such a knowledge base will speed up and augment the transfer of information to the generative component. This approach is quite different from attempting to model (rather than amplify) the student's generative (rather than evaluative) knowledge, the typical aim of other student modeling techniques.

The state constraint approach circumvents the overspecificity problem by providing two pedagogically relevant forms of abstraction. First, a constraint base enables selective evaluation of problem solving steps. Not all problem solving steps are equally informative or important in diagnosing a student's knowledge. For example, in solving a problem in elementary arithmetic or algebra, the student will almost certainly type an equal sign somewhere in his or her answer. This step might in and of itself contain minimal information about the student's thoughts about the problem. Rather than trying to predict

such a step (i.e., to model the generative knowledge that produced the step), an instructional system might be better off to wait to see what the student does next.

No additional mechanism needs to be implemented to allow a constraint-based system to ignore pedagogically uninformative steps. If the step does not evoke any constraint (i.e., does not cause any relevance condition to match that did not match in the previous state), then the step is *de facto* ignored. Constraints can be written so as to react only to problem states that do contain pedagogically significant information about the learner [35].

Second, a constraint base circumvents the overspecificity problem by allowing an instructional system to operate with classes of *pedagogically equivalent* solution paths. The basic purpose of an instructional system is to map student performances onto instructional actions (e.g., typing out a particular instructional message). Hence, the system needs to group student solutions into classes of solutions that require the same instructional response from the system.

For example, consider the following SQL constraint:

```
Every SQL query must contain relation names in the FROM
clause.
```

It does not matter by which sequence of steps the learner arrived at a query that violates this constraint. All sequences of steps that lead to such a violation require the same instructional response: Talk to the learner about the purpose of the FROM clause. A constraint C implicitly defines a bundle of solution paths, namely all paths that pass through some problem state that violates C. If C is a pedagogically motivated constraint, all those paths should require the same instructional response.

In short, the state constraint representation provides two types of abstraction: selection of informative steps while ignoring others and the ability to bundle incorrect solution paths into pedagogical equivalence classes. Both forms of abstraction help circumvent the overspecificity problem. There is no need to predict or model every problem solving step. The student's behavior is understood in terms of which constraints he or she does and does not violate. The existing constraint-based tutors demonstrate that this type of model suffices to select instructionally appropriate responses in a variety of domains such as database querying [23, 29, 30], database design [48], database normalization [25, 27], software analysis and design [5] and language instruction [22].

3.6. Discussion

Constraint-based student modeling differs in significant respects from other approaches. The constraint-base is not a bug library. Each constraint encodes a piece of *correct* domain knowledge. To build a constraint base, it is not

necessary to conduct extensive research to identify and explicitly codify student's bugs.

Although the constraint base represents correct domain knowledge, it is nevertheless not an expert or ideal student model. These differences are due to the fact that the constraints encode the domain knowledge in evaluative rather than generative form and that the purpose of the constraint-base is to amplify, rather than simulate, the student's knowledge.

Unlike intelligent tutoring systems that use the so-called model tracing technique [2], constraint-based tutors do not follow the student step by step and do not give feedback after individual problem solving steps. For example, SQL-Tutor [30] postpones evaluation and feedback until the student submits his or her solution. This is appropriate, because the order of the steps taken while formulating an SQL query is not constitutive of a correct or successful query.

One advantage of constraint-based student modeling over the bug library technique is that bug libraries do not transfer well between different populations of students [41]. A constraint-base, on the other hand, encodes correct domain knowledge, which of course is the same across student populations.

A second strength of the constraint-based approach is that it can recognize a correct solution submitted by the student, even if that solution is different from the ideal solution. If no constraint is violated, then the student's solution is correct with respect to the notion of correctness embodied in the constraint base. Exclusive reliance on this technique has the disadvantage that if the constraint base is incomplete, some incorrect solutions might mistakenly be classified as correct. This is pedagogically undesirable. Hence, some constraint-based tutors use a stronger recognition technique. For example, in SQL-Tutor there are constraints that compare the student's solution to the ideal solution and check the equivalence of the constructs used to formulate the queries. If there are no constraint violations and the constructs used to formulate the query are equivalent to those in the ideal solution, then the student's solution is accepted as correct. Hence, SQL-Tutor is not thrown off track by correct but creative or unusual solutions, a common problem with other student modeling techniques.

4. Empirical Evaluation: The Smooth Curve Criterion

The ultimate proof of an intelligent tutoring system is that the students learn more effectively while interacting with the system than they do in the context of other types of instruction. The goals of the empirical study reported here were to document that students learn while interacting with the system and to evaluate the appropriateness of the knowledge representation. We begin with a general discussion of how one determines that a knowledge representation is appropriate.

A knowledge representation for an intelligent tutoring system is psychologically appropriate if the units in which the subject matter knowledge have been encoded are also the units in which that knowledge are encoded in the student's head. How does one decide whether this is the case? An interesting answer has been proposed by the research team led by John R. Anderson at Carnegie-Mellon University: Appropriate knowledge representations are indicated by smooth learning curves.

If a performance measure that refers to phenomenological units--e.g., time to complete a practice problems or correctness of a solution--is plotted as a function of the amount of practice, also measured in phenomenological[2] units (e.g., number of practice problems), then the result is often a highly irregular learning curve. This is particularly true for individual learning curves, but large amounts of variability remains even when the data are averaged across individuals.

This should not be surprising. In real instructional settings, as opposed to laboratory experiments, practice problems differ radically in difficulty and in exactly which knowledge units they require for successful solution. Also, the students typically have some control over which practice problems they attempt, so two students who both have solved N practice problems will not have had the same training history. Hence, there is no reason to expect them to have acquired the same knowledge.

In the course of the ACT project, it was discovered that if learning data are plotted in terms of the relevant knowledge units, i.e., the individual production rules hypothesized to be acquired during practice, then learning curves tend to be smooth and regular. The relevant measure of learning is not total solution time or quality of final solution, but the speed or correctness by which a particular knowledge unit (production rule) is applied. If this measure is plotted as a function of the amount of practice *on that particular knowledge unit* (rule), then learning is a smooth, negatively accelerated curve that closely approximates a so-called power law ([1] Figs. 2.2, 2.3; [4] Figs. 2.1, 2.2). It is not the amount of practice on the target skill as a whole but the amount of practice per knowledge unit that determines the level of mastery (of that unit). This fact is consistent with the idea that knowledge units are learned one by one, independently of each other, and that the acquisition of any one unit is a regular process.

This conclusion implies that we can use the character of a learning curve as an indicator of the psychological appropriateness of the knowledge representation in terms of which the curve is plotted. An irregular learning curve indicates an inappropriate representation, while a regular curve indicates an appropriate representation. From this point of view, the irregular learning curves typically obtained when learning data are plotted in terms of

---

[2] The term "phenomenological" is used here in the sense used by physicists, i.e., as referring to categories that are directly given in experience, rather than in the sense used by philosophers, i. e., as referring to a particular epistemology.

practice problems imply that students are not acquiring problem solutions as indivisible wholes.

To apply the smooth curve criterion, one must have a way of measuring the acquisition of individual knowledge units and the amount of practice per knowledge unit. In a tutoring system that uses the model tracing technique, such measures are readily available because each observed problem solving step corresponds to one rule in the knowledge representation. Hence, response latencies and the correctness of the individual steps measures the mastery of the individual rule, and the number of opportunities to apply a rule measures the amount of practice of that rule. If model tracing is not used, or if knowledge is encoded in some other way than in rules, other techniques must be used.

In SQL-Tutor, knowledge is represented in terms of constraints. If those constraints represent psychologically appropriate units of knowledge, then learning should follow a smooth curve when plotted in terms of those constraints. Because SQL-Tutor does not model generative knowledge, we cannot use speed of application as the relevant measure of learning. However, the probability of a constraint violation can be estimated from records of students' interactions with the system.

If the constraint base of a tutor is psychologically appropriate--i.e., if the constraints correspond to units of knowledge that tend to be learned independently of each other--then we would expect the data to be regular when plotted in terms of constraints rather than practice problems or solution attempts. In previous work [30] we evaluated this expectation by randomly selecting 100 constraints from SQL-Tutor among those constraints that were relevant at least once during the study[3]. For each of the selected constraints, the problem states in which that constraint was relevant were identified in each student's record and rank ordered from 1 through R. We refer to these as *occasions of application*. For each occasion, it was recorded whether the relevant constraint was violated or satisfied. This analysis was repeated for each subject. The probability of violating a constraint decreased in a negatively accelerated fashion with increasing number of opportunities to acquire the knowledge embedded in that constraint. The fact that the relation looks smooth when plotted in terms of the individual constraints, but not when plotted in terms of practice problems, provides a measure of support for the appropriateness of the constraint representation and, indirectly, for the theory of learning from error. Furthermore, the proportion of subjects who do not violate any constraints increases smoothly and rapidly across occasions of applicability. Both analyses verify that the students learned something from their interactions with SQL-Tutor. We have obtained smooth learning curves in all constraint-based tutors we developed [5, 27, 29, 30, 48, 51, 52].

---

[3] Because different constraints are relevant for different problems, a small group of constraints were never relevant for the particular practice problems the students attempted. We excluded those from the analysis.

5. General Discussion

Formal representation of knowledge is the constituting idea of AI. The ambition to explore the technological potential of such representations makes AI a bridge between technology and the symbolic traditions in philosophy, psychology and other cognitive sciences.

In the context of education, this ambition merges easily with the old [11] but as yet largely unexploited idea of individualized instruction. By embedding AI in instructional artifacts and materials, we can make those artifacts and materials responsive to the individual learner.

This agenda, originally posed by John Seely Brown and co-workers at Xerox PARC in the 1970s, requires solutions to three difficult problems. First, how is the subject matter to be represented? The short answer is that someone who understands the domain has to sit down and lay out the appropriate representation. AI researchers have gathered considerable experience with this so-called knowledge acquisition problem. For instructional domains of low to intermediate complexity, this problem is not an obstacle to progress.

Second, how is the student's knowledge of the domain-- partially correct and partially incorrect--to be represented? Third, how is that representation to be updated on line, in response to student behavior? These two problems are of a different magnitude and workable solutions have been slow to emerge.

We claim that the intractability of these problems is caused, in part, by the particular knowledge representations that have been used in intelligent tutoring systems to date. Horn clauses, Lisp functions and production rules require complete specificity because they were designed for the implementation of executable programs. But the information about a student available to an instructional system cannot support inferences at that level of specificity. Instead, an instructional system needs to represent a student in terms of pedagogically motivated equivalence classes of knowledge states.

A knowledge representation for instruction should not be designed with only technological demands in mind. It is highly likely that an AI-based instructional system will be the more effective, the higher the psychological realism of its knowledge representation [34].

The state constraint representation fulfills both the technological and the psychological requirements. Although each individual state constraint has to be fully specified to be applied, dealing with student behavior in terms of constraint violations provides two useful forms of abstraction. First, problem solving steps that do not trigger constraints are effectively ignored. Second, all solutions that violate a particular constraint indicate a need to teach the knowledge embedded in that constraint. The student's knowledge can be described in terms of the constraints that he or she does and does not violate in the course of an instructional session.

This type of description is appropriate, because research on skill acquisition--particularly people's ability to detect and correct their own errors--indicate that learners have two separate knowledge bases, one for generative

knowledge and one for evaluative knowledge. Skill acquisition can be conceptualized as a process of moving information from the latter to the former [37]. A constraint-based instructional system functions as an amplified evaluative knowledge base that feeds additional information into the student's natural learning process.

How does one assess the validity of a claim to psychological realism? The smooth learning curve criterion proposed by John Anderson, Albert Corbett and Kenneth Koedinger and their co-workers is the only criterion suggested to date. Irregular learning curves indicate an inappropriate representation, while regular curves signal an appropriate representation.

The empirical data reported in [5, 16, 18, 21, 22, 26, 27, 29, 30, 47, 48, 51] show that the constraint-based representation is doing well on this criterion. When learning is measured in terms of the number of errors per practice problem, then the data are highly irregular, as one would expect from the fact that the students had considerable control over which practice problems they attempted and in which order. However, when learning is measured in terms of probability of violating a particular constraint, the learning process is revealed to be highly regular. The number of opportunities to violate a constraint, and hence to acquire the knowledge encoded in that constraint, is a strong determinant of the probability of further violations of that constraint.

The state constraint representation contrasts with traditional knowledge representations. It is not designed to encode executable programs. A set of state constraints normally cannot accomplish tasks or carry out computations. Although we use the "if-then" construct in English transcriptions of state constraint formulas, the state constraint idea cannot be absorbed either into the material implication of logic-based programming or the rule construct of production system architectures. A state constraint is a piece of evaluative knowledge. It is a tool for *passing judgment*, not for computing new results or inferring new conclusions.

Although sets of state constraints encode correct domain knowledge, they have some of the same functions as bug libraries. A constraint violation provides the same information as a positive match between a theoretical bug description and a student solution. The set of all possible violations of a constraint base represents the same universe of behaviors as the set of all positive matches to a bug library, but the representation is implicit rather than explicit. Consequently, constraint-based modeling does not require extensive research to identify and represent the common bugs in a given student population.

Although conditional branching and pattern matching are familiar ideas in AI, the bipartite structure of state constraints provide a new twist. The matching of the satisfaction pattern is conditional on a successful prior match of the relevance pattern. Nothing like this double conditionality occurs in the execution of Horn clauses and production rules, but it is the source of the power of the state constraint representation. A state constraint cannot be replaced by an unstructured pattern.

The idea of a constraint and the associated processes like constraint relaxation are, of course, familiar in AI research [9]. However, in the standard

formulation, a constraint is a relation between two variables, such that specifying the value of one variable restricts the possible values of the other. In the typical case, the variables refer to quantities and the constraint itself is an equation linking those quantities [15].

The state constraint idea and the standard notion of a constraint appear different at first glance, but a closer look reveals that they are related. The state constraint concept generalizes the common notion of a constraint to variables that refer to truth values. If certain propositions (namely, those in the relevance condition) turn out to be true, then that restricts the possible truth values of some other propositions (namely, those in the satisfaction condition). Although they are similar at this high level of description, implementations of the state constraint idea and of the standard constraint notion have such different flavors that they are better considered distinct representations in the context of system implementation.

Although we have emphasized the strengths of the state constraint idea throughout this article, we do not claim that it provides the final solution to the problem of student modeling and hence to AI-based instruction. On the contrary, we regard constraint-based modeling as one technique among others, with its own unique profile of strengths and weaknesses that is likely to work well in some task domains but not in others. First, constraint-based modeling is likely to be useful in domains with problem states with complex internal structures that accumulate detail as problem solving proceeds. In such domains, student errors typically result in inconsistencies between parts of problem states and those inconsistencies can be detected by state constraints.

Second, constraint-based modeling is likely to be advantageous in domains in which the order of problem solving steps is not crucial for the correctness of the final solution. In such domains, the sequential information in a behavioral record is not diagnostic and all diagnosis has to be done on the basis of the final outcome, a situation that provides a good fit to the state constraint technique. Formulating database queries and database design are domains of this sort. Of course, when the sequential information is diagnostic, then the constraint-based technique can make use of it, as its success in arithmetic [39, 40] and data normalization [25, 27] shows, but it is not essential. The articles in this issue provide additional information about the application of constraint-based tutoring to a diverse set of task domains.

The general lesson of constraint-based modeling is that progress in embedding AI in instructional artifacts requires the invention of novel knowledge representations especially designed for this purpose. There is no reason to limit knowledge representation to the traditional representations, nor is there any reason to believe that the state constraint representation will remain the last word. The universe of possible knowledge representations is large and we have only explored a small fraction of it. Innovative exploration, rather than ever more rigorous analysis and application of entrenched techniques is likely to open doors into the enchanted world of adaptive and helpful artifacts that Allen Newell envisioned two decades ago.

Acknowledgments

The preparation of this article was supported, in part, by Grant No. N00014-97-1-0826 from the Cognitive Science Program of the Office of Naval Research, US Navy, to the first author, and by research grants U6430 and U6532 from the University of Canterbury to the second author.

References

1. Anderson, J. R.: Rules of the mind. Hillsdale, NJ: Erlbaum (1993)
2. Anderson, J. R., Boyle, C. F., Corbett, A. T., Lewis, M. W.: Cognitive modeling and intelligent tutoring. Artificial Intelligence, Vol. 42, 7-49 (1990)
3. Anderson, J. R, Corbett, A. T., Koedinger, K. R., Pelletier, R.: Cognitive tutors: Lessons learned. Learning Sciences, Vol. 4, No. 2, 167-207 (1995)
4. Anderson, J. R., Lebiere, C.: The atomic components of thought. Mahwah, NJ: Erlbaum (1998)
5. Baghaei, N., Mitrovic, A., Irwin, W.: A Constraint-Based Tutor for Learning Object-Oriented Analysis and Design using UML. In: C.K. Looi, D. Jonassen, M. Ikeda (eds), Proc. Int. Conf. Computers in Education, 11-18 (2005)
6. Brown, J. S., Burton, R. R.: Diagnostic models for procedural bugs in basic mathematical skills. Cognitive Science, Vol. 2, 155-192 (1978)
7. Burton, R.: Diagnosing bugs in a simple procedural skill. In D. Sleeman & J. S. Brown (eds.) Intelligent tutoring systems. New York, NY: Academic Press, 17-183 (1982)
8. Forgy, C. L.: Rete: A fast algorithm for the many pattern/many object pattern match problem. Artificial Intelligence, Vol. 19, 17-37 (1982)
9. Freuder, E., Mackworth, A. (eds.): Constraint-based reasoning. Cambridge, MA: MIT Press (1994)
10. Gentner A., Conati C., VanLehn, K.: Procedural help in Andes: Generating hints using a Bayesian network student model. In Proc. 15th National Conf. Artificial Intelligence (AAAI-98), MIT Press, 106-111 (1998)
11. Glaser, R.: Some implications of previous work on learning and individual differences. In R. Gagne (Ed.), Learning and individual differences. Columbus, OH: Merrill , 1-18 (1967)
12. Hartley, D., Mitrovic, A.: Supporting learning by opening the student model. In: S. Cerri, G. Gouarderes and F. Paraguacu (eds.) Proc. 6th Int. Conf on Intelligent Tutoring Systems, Biarritz, France, LCNS 2363, 2002: 453-462 (2002)
13. Hawkes, L. W., Derry, S. J.: Error diagnosis and fuzzy reasoning techniques for intelligent tutoring systems. Artificial Intelligence in Education, Vol. 1, 43-56 (1989/90).
14. Koedinger, K. R., Anderson, J. R., Hadley, W. H., Mark, M. A.: Intelligent Tutoring Goes to School in the Big City. Artificial Intelligence in Education, Vol. 8, 30-43 (1997)
15. Leler, W.: Constraint programming languages. Reading, MA: Addison-Wesley (1988)
16. Martin, B.: Intelligent Tutoring Systems: The practical implementation of constraint-based modelling. PhD Thesis, University of Canterbury (2002)

17. Martin, B., Mitrovic, A.: Tailoring Feedback by Correcting Student Answers. In: G. Gauthier, C. Frasson and K. VanLehn (eds), Proc. ITS'2000, Springer, 383-392 (2000)
18. Martin, B., Mitrovic, A.: Automatic Problem Generation in Constraint-Based Tutors. In: S. Cerri, G. Gouarderes and F. Paraguacu (eds.) Proc. 6th Int. Conf on Intelligent Tutoring Systems ITS 2002, Biarritz, France, LCNS 2363, 388-398 (2002)
19. Martin, B., Mitrovic, A.: Domain Modeling: Art or Science? In: U. Hoppe, F. Verdejo & J. Kay (ed) Proc. 11th Int. Conference on Artificial Intelligence in Education, IOS Press, 183-190 (2003)
20. Martin, J., VanLehn, K.: OLAE: Progress toward a multi-activity, Baysian student modeler. In S. Brna, S. Ohlsson & H. Pain (Eds.), Artificial intelligence in education: Proceedings of AIED93, Charlottesville, VA: AACE, 410-417 (1993)
21. Mayo, M.: Bayesian Student Modelling and Decision-Theoretic Selection of Tutorial Actions in Intelligent Tutoring Systems, PhD Thesis, University of Canterbury (2001)
22. Mayo, M., Mitrovic, A.: Optimising ITS Behaviour with Bayesian Networks and Decision Theory'. Artificial Intelligence in Education, Vol. 12, No. 2, 124-153 (2001)
23. Mitrovic, A.: SQL-Tutor: a preliminary report. Technical Report No. TR-COSC 08.97. Christchurch, New Zealand: Computer Science Department, University of Canterbury (1997)
24. Mitrovic, A.: Self-assessment: how good are students at it? In: J. Gilbert, R. Hubscher, S. Puntambekar (eds) Proc. AIED 2001 Workshop on Assessment Methods in Web-Based Learning Environments & Adaptive Hypermedia, San Antonio, 2-8 (2001)
25. Mitrovic, A.: NORMIT, a Web-enabled tutor for database normalization. In Kinshuk, R. Lewis, K. Akahori, R. Kemp, T. Okamoto, L. Henderson & C.-H. Lee (Eds.) Proceedings of the Int. Conf. on Computers in Education, ICCE 2002, Los Alamitos, CA: IEEE Computer Society, 1276-1280 (2002)
26. Mitrovic, A.: Supporting Self-Explanation in a Data Normalization Tutor. In: V. Aleven, U. Hoppe, J. Kay, R. Mizoguchi, H. Pain, F. Verdejo, K. Yacef (eds) Supplementary proceedings, AIED 2003, 565-577 (2003)
27. Mitrovic, A.: The Effect of Explaining on Learning: a Case Study with a Data Normalization Tutor. In: C-K Looi, G. McCalla, B. Bredeweg, J. Breuker (eds) Proc. Artificial Intelligence in Education, IOS Press, 499-506 (2005)
28. Mitrovic, A., Martin, B.: Evaluating the effects of open student models on learning. In: P. de Bra, P. Brusilovsky and R. Conejo (eds) Proc. 2nd Int. Conf on Adaptive Hypermedia and Adaptive Web-based Systems AH 2002, Malaga Spain, LCNS 2347, 296-305 (2002)
29. Mitrovic, A., Martin, B., Mayo, M.: Using Evaluation to Shape ITS Design: Results and Experiences with SQL-Tutor. User Modeling and User-Adapted Interaction, Vol. 12, No. 2-3, 243-279 (2002)
30. Mitrovic, A., Ohlsson, S.: Evaluation of a Constraint-Based Tutor for a Database Language. Artificial Intelligence in Education, Vol. 10, No. 3-4, 238-256 (1999)
31. Newell, A.: Fairytales: Remarks at the inaugural dinner for the U. A. and Helen Whitaker Professorships. Viewpoint (no. 3). Pittsburgh, PA: Carnegie Mellon University (1976)
32. Norman, D.: Categorization of action slips. Psychological Review, Vol. 88, 1-15 (1981)
33. Ohlsson, S.: Some principles of intelligent tutoring. Instructional Science, Vol. 14, 293-326 (1986)

34. Ohlsson, S.: System hacking meets learning theory: Reflections on the goals and standards of research in Artificial Intelligence and education. Artificial Intelligence in Education, Vol. 2, No. 3, 5-18 (1991)
35. Ohlsson, S.: Constraint-based student modeling. Artificial Intelligence and Education, Vol. 3, No. 4, 429-447 (1992)
36. Ohlsson, S.: The interaction between knowledge and practice in the acquisition of cognitive skills. In A. Meyrowitz and S. Chipman (Eds.), Foundations of knowledge acquisition: Cognitive models of complex learning. Norwell, MA: Kluwer, 147-208 (1993)
37. Ohlsson, S.: Learning from performance errors. Psychological Review, Vol. 103, 241-262 (1996)
38. Ohlsson, S.: Learning from error and the design of task environments' Educational Research, Vol. 25, No. 5, 419-448 (1996)
39. Ohlsson, S., Ernst, A., Rees, E.: The cognitive complexity of doing and learning arithmetic. Research in Mathematics Education, Vol. 23, No. 5, 441-467 (1992)
40. Ohlsson, S., Rees, E.: The function of conceptual understanding in the learning of arithmetic procedures. Cognition & Instruction, Vol. 8, 103-179 (1991)
41. Payne, S., Squibb, H.: Algebra mal-rules and cognitive accounts of errors' Cognitive Science, Vol. 14, 445-481 (1990)
42. Reason, J. T.: Cognitive underspecification: Its variety and consequences' In B. J. Baars, (Ed.), Experimental slips and human error: Exploring the architecture of volition. New York, NY: Plenum Press, 71-91 (1992)
43. Self, J. A.: Bypassing the intractable problem of student modeling. In C. Frasson & G. Gauthier (Eds.), Intelligent tutoring systems: At the crossroads of artificial intelligence and education. Norwood, NJ: Ablex, 107-123 (1990)
44. Sleeman, D., Hirsch, H., Ellery, I., Kim, I-Y.: Extending domain theories: Two case studies in student modeling. Machine Learning, Vol. 5, 11-37 (1990)
45. Sleeman, D., Kelly, A. E., Martinak, R., Ward, R. D., Moore, J. L.: Studies of diagnosis and remediation with high school algebra students. Cognitive Science, Vol. 13, 551-568 (1989)
46. Soloway, E., Spohrer, J.: Studying the novice programmer. Hillsdale, NJ: Erlbaum (1989)
47. Suraweera, P.: An Intelligent Teaching System for Database Modelling' MSc thesis, University of Canterbury (2001)
48. Suraweera, P., Mitrovic, A.: KERMIT: a Constraint-based Tutor for Database Modeling. In: S. Cerri, G. Gouarderes and F. Paraguacu (eds.) Proc. 6th Int. Conf on Intelligent Tutoring Systems ITS 2002, Biarritz, France, Springer-Verlag LCNS 2363, 377-387 (2002)
49. VanLehn, K., Lynch, C., Schulze, K., Shapiro, J.A., Shelby, R., Taylor, L., Treacy, D., Weinstein, A., Wintersgill, M.: The Andes Physics Tutoring System: Lessons Learned. Artificial Intelligence in Education, Vol. 15, No. 3, 147-204 (2005)
50. Weerasinghe, A.: Exploring the effects of self-explanation in the context of a database design tutor. MSc thesis, University of Canterbury (2003)
51. Weerasinghe, A., Mitrovic, A.: 2005, 'Facilitating Deep Learning through Self-Explanation in an Open-ended Domain. Knowledge-based and Intelligent Engineering Systems, IOS Press, Vol. 9, (2006, in print)
52. Zakharov, K., Mitrovic, A., Ohlsson, S.: Feedback Micro-engineering in EER-Tutor. In: C-K Looi, G. McCalla, B. Bredeweg, J. Breuker (eds) Proc. Artificial Intelligence in Education, IOS Press, 718-725 (2005)