A Comparative Analysis of Cognitive Tutoring and Constraint-Based Modeling

Antonija Mitrovic[1], Kenneth R. Koedinger[2], and Brent Martin[1]

[1]Intelligent Computer Tutoring Group,
University of Canterbury, Christchurch, New Zealand
{tanja, brent}@cosc.canterbury.ac.nz
[2]Human-Computer Interaction Institute, Carnegie Mellon University
koedinger@cmu.edu

Abstract. Numerous approaches to student modeling have been proposed since the inception of the field more than three decades ago. What the field is lacking completely is comparative analyses of different student modeling approaches. In this paper we compare Cognitive Tutoring to Constraint-Based Modeling (CBM). We present our experiences in implementing a database design tutor using both methodologies and highlight their strengths and weaknesses. We compare their characteristics and argue the differences are often more apparent than real: for specific domains one approach may be favoured over the other, making them viable complementary methods for supporting learning.

1 Introduction

Student modeling is one of the crucial components of Intelligent Tutoring Systems (ITS). Numerous modeling approaches have been devised over the years, such as overlay modeling, enumerative bug modeling, generative and reconstructive modeling, and constraint-based modeling [4]. Early ITS projects focused on the development of student modeling approaches, and rarely evaluated the methods properly. Although the percentage of papers that include evaluation results has been growing steadily, they always relate to a single student modeling approach in isolation. For the maturation of the field, it is of critical importance to perform comparative analyses of various approaches. Unfortunately, such comparative evaluations are extremely difficult; it is a major undertaking to develop any ITS, let alone two for the same domain.

In this paper we explore the student modeling approaches used in cognitive tutors and constraint-based tutors. We report on an initial case study in which we reimplemented part of an existing constraint-based tutor as a cognitive tutor to compare and contrast the various features of these two approaches. We briefly overview cognitive tutors and constraint-based tutors. In section 4 we present the case study, followed by a comparative analysis of the two approaches. We give the conclusions in the final section.

P. Brusilovsky et al. (Eds.): UM 2003, LNAI 2702, pp. 313–322, 2003.

2 Cognitive Tutors

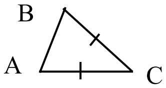
Cognitive Tutors [5] are some of the most successful ITS today. They have been developed for a number of domains including algebra, geometry and LISP. Cognitive Tutors are based on the ACT-R theory of cognition [2], which claims that there are two long-term memory stores: declarative and procedural. The theory explains human learning as going through several phases. The first involves learning declarative knowledge, including factual knowledge (such as theorems in a mathematical domain), which is represented as *chunks*. Declarative knowledge is later turned into procedural knowledge, which is goal-oriented and therefore more efficient to use. Procedural knowledge is represented in the form of production rules. In the last phase, the production rules are further optimised when the student becomes an expert. The fundamental assumption of ACT-R is that cognitive skills are realised by production rules. In order to support students to learn a specific task, that is, to learn a specific set of production rules that will enable students to perform the tasks correctly, cognitive tutors organize instruction around the underlying production rules.

A generic student model is produced and used in a process called *model tracing*, while a student-specific model is produced by *knowledge tracing*. A cognitive tutor is based on a cognitive model of the domain expertise, represented by production rules, which describes the domain knowledge needed to perform tasks like good (and perhaps poor) students. Cognitive tutors generate immediate feedback, i.e. they react to each step the student makes while solving a problem. An error is detected either when a student step does not match any rule, or it does match one of the *buggy rules,* which represent typical mistakes. Model tracing thus checks whether or not the student is performing correctly by comparing each student's step directly with one or more correct or incorrect steps generated dynamically by the production system.

To illustrate, consider a set of production rules for finding the angles in geometry problems like the one shown at the top of Figure 1. The first two production rules can be used in sequence to first find angle B (because angles opposite equal sides are equal) and then to find angle C (because the sum of the angles in a triangle is 180). Once the first rule fires and finds the value for angle B, it is possible for the next rule to fire and find angle C (note that the angle label names are arbitrary: these rules apply to a triangle with any point labels.) The last rule in Figure 1 is an example of a buggy rule used to detect particular common mistakes. In geometry, students often over-generalize from common orientations of figures. This buggy production represents the shallow inference that the angles at the bottom of an isosceles triangle are always equal.

3 Constraint-Based Tutors

Constraint-Based Modeling (CBM) is an approach proposed by Ohlsson [11] as a way of overcoming the intractable nature of student modeling [14]. CBM arises from Ohlsson's theory of learning from performance errors [12], which proposes that we often make mistakes when performing a task, even when we have been taught the correct way to do it. According to this theory, we make mistakes because the declarative knowledge we have learned has not been internalized in our procedural



Angle A is 65.
What is angle C?

Two correct production rules:

IF goal is to find an angle in an isosceles triangle ABC and AC = AB
and angle A is known
THEN set the value of angle B to A.

IF goal is to find an angle in a triangle ABC and angles A and B are known
THEN set the value of C to 180-A-B

Buggy production rule:

IF goal is to find an angle in an isosceles triangle ABC
and angle A and C are at the bottom of the triangle and angle A is known
THEN set the value of angle C to A.

Fig. 1. Three production rules for computing the size of an angle

knowledge, and so the number of decisions we must make while performing the procedure is sufficiently large that we make mistakes. However, by practicing the task and catching ourselves (or being caught by a mentor) making mistakes, we modify our procedure to incorporate the appropriate rule that we have violated. Over time we internalize all of the declarative knowledge about the task and so the number of mistakes we make is reduced. Ohlsson describes the process of learning from errors as consisting of two phases: *error recognition* and *error correction*. After detection, an error can be corrected so that the solution used is applicable only in situations in which it is appropriate. A student needs declarative knowledge to detect an error. If the student does not possess such declarative knowledge, an ITS may play the role of a mentor and inform the student of the mistake. A carefully designed sequence of feedback messages that reflects the action of a human teacher helps the student to overcome problems in his/her knowledge.

The starting point for CBM is that correct solutions are similar to each other in that they satisfy all the general principles of the domain. No correct solution can be arrived at by traversing a problem state that violates a fundamental principle of the domain. In CBM, we are not interested in what the student has done, but in what *state* they are currently in. As long as the student never reaches a state that is known to be wrong, they are free to perform whatever actions they please.

Constraints define equivalence classes of problem states. An equivalence class triggers the same instructional action; hence all states in a class are pedagogically equivalent. It is therefore possible to attach feedback messages directly to constraints. The domain model is therefore a collection of state descriptions of the form:

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

In other words, if the student solution falls into the state defined by the relevance condition, it must also be in the state defined by the satisfaction condition. A violated constraint signals an error, which translates to incomplete or incorrect knowledge.

Consider the same example of calculating angles of a triangle, used in Section 2. Figure 2 illustrates the constraints that can be used to diagnose students' solutions. The first two constraints jointly define that (only) the two base angles in an isosceles triangle have the same size. The third constraint is equivalent to the second rule in Figure 1. Constraint 2 catches the same error as the buggy rule from Figure 1.

First, all relevance patterns are matched against the problem state. Then, the satisfaction components of constraints that matched the problem state in the first step (i.e., the relevant constraints) are tested. If a satisfaction pattern matches the state, the constraint is satisfied. Otherwise, it is violated. The short-term student model consists of all violated constraints.

We believe that CBM is neutral with respect to the domain. Within the ICTG group, we have developed SQL-Tutor, a tutor for teaching the declarative language SQL [10], CAPIT, a system that teaches the rules of punctuation and capitalization in English [8], KERMIT, a system for database design [15], and NORMIT [9], an ITS that teaches the procedural task of data normalization. We have experienced no problems expressing the knowledge in these domains in terms of constraints.

CBM, as proposed in [11] is a method for diagnosing students' solutions. The approach identifies errors, which is extremely important for students lacking declarative knowledge, because they are unable to detect errors themselves. We have also shown that CBM can be extended to allow for long-term modeling of students' knowledge, and alternatives for generation of pedagogical actions [7].

4 Case Study: Teaching Database Design

In this case study we re-implemented a part of the KERMIT tutor using the cognitive tutors methodology. This allows us to compare the two approaches. We start by briefly introducing KERMIT, the context of the study, and then present our experiences.

C_{r1}: A base angle of an isosceles triangle is known (θ_1),
And the student has calculated the size of the other base angle θ_2
C_{s1}: The size of θ_2 is θ_1

C_{r2}: A base angle of an isosceles triangle is known (θ_1),
And the student has calculated that the size of another angle θ_2 that equals θ_1,
C_{s2}: θ_2 is a base angle

C_{r3}: Two angles of a triangle are known (θ_1 and θ_2),
And the student has calculated the size of the third angle θ_3
C_{s3}: The size of θ_3 is $(180-\theta_1-\theta_2)$

Fig. 2. Three constraints that check whether the size of an angle is correct

4.1 KERMIT: A Constraint-Based Tutor for Database Design

KERMIT (Knowledge-based Entity Relationship Modeling Intelligent Tutor) is an ITS for teaching database design that was developed at the ICTG group. One of the goals of the system was to test the ICTG's methodology for building ITS, because database design is a domain whose characteristics are different from those of the domains previously worked on. Database design is a particularly open-ended task: although there is an outcome defined in abstract terms, there is no single "correct" procedure to obtain that outcome. For a detailed discussion of the system, see [15]. KERMIT is a problem-solving environment in which students practice database design using the Entity Relationship (ER) data model.

KERMIT consists of an interface, a pedagogical module that determines the timing and content of pedagogical actions, and a constraint-based modeller, which analyses student answers and generates student models. The interface displays the current problem and provides controls for stepping between problems, submitting a solution and selecting the level of feedback. It also contains the main working area, in which the student draws the ER diagram. Feedback is presented on request. KERMIT contains a set of problems and the ideal solutions to them, but has no problem solver.

In order to check the correctness of the student's solution, KERMIT compares it to the correct solution using domain knowledge represented in the form of more than 90 constraints. The constraints cover both syntactic and semantic knowledge. The syntactic constraints are concerned with syntactic details in a student's solution. An example of such a constraint is "A regular entity must have at least one key attribute." Semantic constraints relate the student's solution to the system's ideal solution. For example, there are constraints that check for equivalent, but not identical, ways of modeling a database in the student's and ideal solution.

4.2 Re-implementing KERMIT as a Model-Tracing Tutor

To compare these two approaches, we implemented a subset of KERMIT using model tracing. We refer to this implementation as KERMIT-MT. Table 1 summarizes the differences between the two implementations. In the following discussion we will use this ER modelling problem: "*Some students live in student halls. Each hall has a unique name, and each student has a unique number.*"

Table 1. A comparison of the two implementations

Feature	KERMIT-MT	KERMIT
Problem representation	Text + chunks (words + world knowledge)	Text + tags
Ideal solution	Not stored	Entity and relationship lists
Domain knowledge	Production rules: Entities: 2 Attributes: 4 Relationships: 14 Done: 5	Matching constraints: Entities: 5 Attributes: 9 Relationships: 9

Beginning with problem representation, KERMIT requires the problem text to be stored together with tags that identify phrases corresponding to the constructs in the ideal solution. For example, the word "*student*" would have a tag that identifies it as corresponding to an entity type in the ideal solution. The ideal solution and the student's solution are represented in the same way. There is also a list of entities, where each entity is described in terms of its name, type and a list of attributes. Similarly, there is a list of relationships, containing the name of each relationship, its type, the names of participating entities and possibly a list of attributes.

KERMIT-MT, on the other hand, requires the problem text plus additional structures that represent what such a system might output for this problem. In addition, the problem author specifies declarative chunks to represent the semantics of all relevant words (nouns, verbs, or modifiers) appearing in the problem text. Interestingly, the text of an ER problem often does not contain everything the student needs to know in order to solve the problem: some chunks come from the student's world knowledge. In KERMIT-MT, such elements were specified explicitly in order to enhance the ability to give advice. In contrast, this difference between what a student can infer from the text and what they must know is not represented in KERMIT, and thus it cannot provide specific instruction on the difference, or diagnose a student with difficulties with one kind of inference but not the other.

KERMIT requires an ideal solution to be stored, while there is no such requirement in KERMIT-MT. However, it is easier to create a representation of the ideal solution than to specify all the chunks needed in KERMIT-MT. On the other hand, the solution representation has less information and thus less to draw on when creating meaningful advice for students.

The domain knowledge in KERMIT-MT covers only a part of the domain covered by KERMIT, so in Table 1 we include only the relevant part of the domain model (KERMIT currently contains over 90 constraints, but covers the complete ER domain). KERMIT-MT has 25 production rules, which cover simple and key attributes, regular entities and regular binary relationships only. These were written by Koedinger in about 20 hours and could be refined to fewer rules with more time. There are 23 constraints in KERMIT that correspond to KERMIT-MT's rules. However, these constraints are more general, in that they deal with all kinds of attributes (including composite and multi-valued) so they cover more of the domain than the rules in KERMIT-MT. Further, KERMIT-MT uses buggy rules to generate error-specific feedback to students, while KERMIT does not. On the other hand, KERMIT only provides error feedback, whereas KERMIT-MT can provide hints when a student is stuck about the thinking process to pursue to perform the next step.

5 Discussion

Table 2 summarizes the main differences between MT and CBM. The learning theories that underlie these approaches—ACT-R (MT) and "learning from performance errors" (CBM)—are both based on the distinction between declarative and procedural knowledge, and the view that learning consists of two main phases: in the first, declarative knowledge is encoded; in the second phase, this declarative knowledge is turned into more efficient procedural knowledge. The difference between the theories is in the amount of effort that is assumed in each phase, and also in the focus of instruction based on each theory. ACT-R assumes that the encoding of declarative knowledge is a straightforward process, where experiences are stored in an unchanged form, e.g. examples, successes and failures of attempts. Therefore, efforts are needed in the second phase, when declarative knowledge is proceduralized. In contrast, Ohlsson [11,12] claims that we make mistakes if we do not have sufficient declarative knowledge to detect errors. Consequently, cognitive tutors tend

Table 2. Comparative analysis of CBM and MT

Property	Model Tracing	Constraint-Based Modeling
Knowledge representation	Production rules (procedural)	Constraints (declarative)
Cognitive fidelity	Tends to be higher	Tends to be lower
What is evaluated	Action	Problem state
Problem solving strategy	Implemented ones	Flexible to any strategy
Solutions	Tend to be computed, but can be stored	One correct solution stored, but can be computed
Feedback	Tends to be immediate, but can be delayed	Tends to be delayed, but can be immediate
Problem-solving hints	Yes	Only on missing elements, but not strategy
Problem solved	'Done' productions	No violated constraints
Diagnosis if no match	Solution is incorrect	Solution is correct
Bugs represented	Yes	No
Implementation effort	Tends to be harder, but can be made easier with loss of other advantages	Tends to be easier, but can be made harder to gain other advantages

to focus on generative knowledge (production rules), while constraint-based tutors teach evaluative knowledge (constraints). However, more recent cognitive tutors have successfully addressed evaluative knowledge [6] and declarative knowledge more generally [1]. Other researchers also stress the importance of declarative knowledge. For example, Chi et al [3] see performance in problem solving as largely determined by the completeness of declarative knowledge, rather than the efficiency of procedural knowledge.

MT tutors represent domain knowledge as production rules. This knowledge has high cognitive fidelity, because it is an explicit model of the reasoning that the learner must acquire. High cognitive fidelity is a strong advantage of Cognitive Tutors. In contrast, CBM tutors represent domain knowledge in a declarative form. It is interesting to note similarities between constraints and the inference rules Chi and colleagues discuss in [3]: they find that students who engage in self-explanation learn better, as a consequence of forming inference rules. Chi states that inference rules are more operational than general principles conveyed in traditional instruction, because the conditions are more specific, and inference rules are more decomposed than the general principles. The same reasoning applies to constraints: it takes a number of constraints to equal a general principle of a domain.

Model tracing tutors have been criticised for being too rigid, in that they force students to follow a fixed set of desirable approaches to solving problems [16]. For example, the Lisp tutor requires that the top-down approach be used for writing functions. One might speculate that this may be more beneficial to novices and less so for more knowledgable students, however evaluations of the LISP tutor have tended to show fairly uniform learning improvements for students at all levels. In other words, the evidence suggests the LISP tutor does work well both for novice and more knowledgable students. On the other hand, CBM neither imposes nor supports any particular strategy, since it evaluates the current state in problem solving (as opposed to the current action which is evaluated in Cognitive Tutors). By ignoring the procedures used to solve problems, CBM allows for inconsistencies in problem-solving strategies. The downside is that CBM systems typically are not capable of giving strategic planning advice.

Typical CBM-based systems generate instructional actions without being able to solve problems, by comparing the ideal solution (specified by the human teacher) to the student's solution. If there are alternative solutions, they are recognized as such by constraints that check for the necessary elements in the solution. We have developed an extension to CBM that allows problems to be solved (and student solution errors to be corrected) directly from the constraint set, and implemented it for SQL-Tutor [7]. However, it requires that the constraint set be more complete, otherwise erroneous solutions may be generated, but has the benefit of being optional. In contrast, Cognitive Tutors typically are able to solve problems, and having some simulation of problem solving is important for providing planning advice. However, it is also possible for Cognitive Tutors, like CBM, to store solutions (or approximations thereof) and write rules that work from them. In both cases, an incremental development strategy is possible whereby one starts by implementing the ITS using stored solutions and then adds problem solving capabilities as needed (ideally driven by student usage data).

Cognitive Tutors typically offer immediate feedback (usually only implicitly by "flagging" an error when it occurs), while constraint-based tutors provide feedback on demand. However, both approaches are capable of providing the other type of feedback, so this difference is somewhat superficial. Further, Cognitive Tutors can offer strategic problem-solving hints in terms of the next step to perform in a plan. These hints are typically provided on demand or after the student has made more errors on a goal than a teacher-set error threshold. Hints are generated by running the production set. Constraint-based tutors are, in general, not able to solve problems, but they can provide feedback on missing elements of the solution.

Another important issue is the completeness of the knowledge base. It is widely accepted that the quality of the knowledge base is the determining factor for the quality of instruction and diagnosis. In model tracing, an incomplete knowledge base may mean that there are some correct or buggy rules missing. When a student performs a step that matches neither a correct or buggy rule, that step is assumed to be incorrect. While the system cannot explain why (because there is no buggy rule), it is able to point to the particular step in the whole solution that is in error. Cognitive Tutor developers work hard to avoid it, but it is possible that a correct rule is missing in such a case, and that the student's step is actually correct. Thus, it is possible in Cognitive Tutors that a correct solution is rejected. In CBM, the default no-match diagnosis is that the student's solution is *correct*, even though this may be false. The default behaviour is therefore "*innocent until proven guilty*", versus "*guilty until proven innocent*" for Cognitive Tutors. In both cases, it is important to perform careful engineering and iterative student testing to prevent such situations.

Although the approach of modeling all possible solutions works well for well-defined domains such as mathematics, it may not be a realistic solution when the domain is ill defined. However, as discussed above, this purported difference between CBM and Cognitive Tutors is more apparent than real. The task of composing a collection of buggy rules is also a major undertaking. Studies have shown that bug libraries do not transfer well to new population of students; if a bug library is developed for a certain group of students, it may not cover the bugs that another group of students may make [13]. Neither Cognitive Tutors nor CBM require studies of student bugs, but both can benefit from such study. Whereas student bugs are represented directly in Cognitive Tutors, in CBM they can be used to help determine the pedagogical states to represent.

6 Conclusions

This paper presented a comparative analysis of two student modeling approaches: model tracing and constraint-based modeling. We discussed the characteristics and the underlying learning theories of these two approaches, and presented a case study where two tutors were developed for the same domain. We also analysed the two approaches in terms of their main attributes. Creating constraint-based modeling systems tends to require less time and effort, but the result tends to be less comprehensive in terms of specific advice-giving capabilities. Creating model-tracing tutors tends to require more time and effort, but this results in more specific advice-giving capabilities. This is apparent from the case study where it was arguably harder to develop a model-tracing tutor for database design than a constraint-based system. On the other hand, the model-tracing tutor has capabilities to give planning hints and advice expressed more in terms of what a student needs to think to generate a part of the answer than in terms of the desired features of that part of the answer. We emphasize that the stated differences are tendencies and are not hard and fast. It is possible to write constraint-based systems that generate solutions [7] and thus can be used to provide planning advice. Of course, doing so takes more time and effort. Conversely, it is possible to more quickly and easily build a model-tracing tutor by writing production rules in a diagnostic mode that focus more on whether student steps are correct or not and less on how to generate those steps. But, again, such a change does not come without cost: the resulting cognitive tutor will not be able to provide planning advice.

We conclude that both approaches have their strengths and weaknesses. Model tracing is an excellent choice for domains where appropriate problem solving strategies are well-defined, and where comprehensive feedback on them is desirable. On the other hand, CBM offers a workable alternative when such strategies are not available or appropriate, or there is too little time or resources to build a model-tracing knowledge base. CBM and model-tracing are viable, complementary approaches to building real-world tutors.

Acknowledgments. KERMIT was developed by Pramuditha Suraweera. The authors thank the Erskine fund of the University of Canterbury for funding the visit of Kenneth Koedinger to New Zealand. The work presented here was supported by the University of Canterbury research grant U6430. This research could not have been done without the support of past and present members of HCII and ICTG.

References

1. Aleven, V., Koedinger, K.: An Effective Metacognitive Strategy: Learning by Doing and Explaining with a Computer-based Cognitive Tutor. Cognitive Science 26 (2002) 147–179
2. Anderson, J. R., Lebiere, C.: The Atomic Components of Thought. Mahwah, NJ: Erlbaum (1998)
3. Chi, M. T. H., Bassok, M., Lewis, W., Reimann, P., Glaser, R.: Self-Explanations: How Students Study and Use Examples in Learning to Solve Problems. Cognitive Science, 13 (1989) 145–182
4. Holt, P., Dubs, S., Jones, M., Greer, J.E.: The State of 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 (1994) 3–35
5. Koedinger, K. R., Anderson, J. R., Hadley, W. H., Mark, M. A.: Intelligent Tutoring Goes to School in the Big City. Int. J. Artificial Intelligence in Education, 8 (1997) 30–43
6. Mathan, S., Koedinger, K.: An Empirical Assessment of Comprehension Fostering Features in an Intelligent Tutoring System. In: S. Cerri, G. Gouarderes, F. Paraguacu (eds.) Proc. ITS 2002, LNCS Vol. 2363 Springer-Verlag, (2002) 330–343
7. Martin, B, Mitrovic, A.: Tailoring Feedback by Correcting Student Answers. Proc. ITS'2000, LNCS Vol. 1839, Springer-Verlag, (2000) 383–392
8. Mayo, M., Mitrovic, A.: Optimising ITS Behavior with Bayesian Networks and Decision Theory. Int. J. Artificial Intelligence in Education, 12 (2001) 124–153
9. Mitrovic, A.: NORMIT, a Web-enabled Tutor for Database Normalization. Kinshuk, R. Lewis, K. Akahori, R. Kemp, T. Okamoto, L. Henderson, C-H Lee (eds.) Proc. ICCE 2002 (2002) 1276–1280
10. Mitrovic, A., Ohlsson, S.: Evaluation of a Constraint-Based Tutor for a Database Language. Int. J. on Artificial Intelligence in Education 10 (3–4) (1999) 238–256
11. Ohlsson, S.: Constraint-based Student Modeling. In Student Modeling: the Key to Individualized Knowledge-based Instruction. Springer (1994) 167–189
12. Ohlsson, S.: Learning from Performance Errors. Psychological Review 103 (1996) 241–262
13. Payne, S., Squibb, H.: Algebra Mal-rules and Cognitive Accounts of Errors. Cognitive Science, 14 (1990) 445–481
14. Self, J. A.: Bypassing the Intractable Problem of Student Modeling. In: C. Frasson and G. Gauthier (eds.), Intelligent Tutoring Systems: at the Crossroads of Artificial Intelligence and Education. Norwood: Ablex (1990) 107–123
15. Suraweera, P., Mitrovic, A.: KERMIT: a Constraint-based Tutor for Database Modeling. In: S. Cerri, G. Gouarderes, F. Paraguacu (eds.) Proc. ITS 2002, LNCS Vol. 2363 Springer-Verlag, (2002) 377–387
16. VanLehn, K. et al.: Fading and Deepening: the Next Steps for Andes and other Model-Tracing Tutors. In: Proc. ITS'2000, LNCS Vol. 1839, Springer-Verlag, (2000) 474–483