Problem solving support in constraint-based tutors

Antonija Mitrovic[1]
Stellan Ohlsson[2]
Brent Martin[1]

[1]*Intelligent Computer Tutoring Group, Department of Computer Science
University of Canterbury, Christchurch, New Zealand
{tanja,brent}@cosc.canterbury.ac.nz*

[2]*Department of Psychology, University of Illinois at Chicago
stellan@uic.edu*

Abstract: This paper discusses various ways problem-solving is supported in constraint-based tutors. We briefly discuss the underlying learning theory, and several constraint-based tutors developed at ICTG. We then discuss the various ways problem-solving is supported in these tutors. The interfaces play an important role, as they provide the student with domain-related information, describing the important domain concepts or providing means of obtaining additional information about problems. The interface also supports good practices in the domain, and visualizes the goal structure. Constraint-based tutor diagnose student's solution, and provide feedback on the basis of this analysis. Feedback provides long and short-term learning advantages through revision of faulty knowledge in the context of learners' errors. Our tutors provide progressive levels of feedback, with each new message specifying more detail about the error and the basic domain principle that was violated. Finally, we also engage the student in discussion about problem-solving actions, which support the learner to reflect on his/her actions and strengthen relationships between procedural and declarative knowledge.

1. Introduction

Intelligent tutoring systems (ITS) have been proven to provide significant learning gains for students in a variety of instructional domains. In order to provide individualized instruction, ITSs diagnose students' actions, and maintain student models, which they use to provide problem-solving support and generate appropriate pedagogical decisions. ITSs are problem-solving environments; hence problem-solving support is one of the main supporting technologies (Brusilovsky & Peylo, 2003).

In this paper, we report on the various ways in which problem solving is supported in constraint-based tutors. Our ITSs are based on the theory of learning from performance errors (Ohlsson 1996), which proposes that we often make mistakes when performing a task, even when we have been taught the correct way to do it. According to this theory, we make mistakes because the declarative knowledge we have learned has not been internalized in our procedural knowledge, and so the number of decisions we must make while performing the procedure is

sufficiently large that we make mistakes. By practicing the task, 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 declarative knowledge about the task, and so the number of mistakes we make is reduced. The theory views learning as consisting of two phases: *error recognition* and *error correction*. A student needs declarative knowledge in order to detect an error. Only then can the error be corrected so that the solution used is applicable only in situations in which it is appropriate. The theory of learning from performance errors therefore suggests that feedback should provide long and short-term learning advantages through revision of faulty knowledge in the context of learners' errors.

Constraint-Based Modeling (CBM) is a student modeling approach (Ohlsson 1994; Mitrovic & Ohlsson 1999) arising from the above theory. CBM starts from the observation that all correct solutions for a problem are similar in that they do not violate any domain principles. CBM is not interested in the exact sequence of states in the problem space the student has traversed, but in what state they are currently in. As long as the student never reaches a state that is known to be wrong, they are free to perform whatever actions they please. Constraints define equivalence classes of problem states. An equivalence class triggers the same instructional action; hence all states in an equivalence class are pedagogically equivalent. It is therefore possible to attach feedback messages directly to constraints. The domain model is therefore a collection of state descriptions of the form: *If <relevance condition> is true, then <satisfaction condition> had better also be true, otherwise something has gone wrong.* In other words, if the student solution falls into the state defined by the relevance condition, it must also be in the state defined by the satisfaction condition in order to be correct.

Constraint-based tutors evaluate student solutions by matching them against the constraint set (i.e. the domain model). Firstly, all relevance patterns are matched against the problem state. Secondly, the satisfaction components of 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 satisfied and violated constraints. Long-term student model consists of the list of all constrains used by the student and the history of constraint usage.

Constraint-based tutors support problem-solving in three ways. First, the interface provides information about the domain of instruction, and its design heavily depends on the chosen domain; the next section discusses how the interface supports the learner while solving problems. Second, problem-solving is supported via the feedback that the system provides, which is discussed in section 3. Finally, our systems support dialogues with the learners about his/her problem-solving actions; this is detailed in Section 4.

2. Supporting problem solving via the interface

In addition to serving as a communication medium, the interface of an ITS also serves as a means of supporting problem solving. Much emphasis is placed upon providing support for the student when designing the interfaces for our constraint-based tutors. This support may take many forms. First, the interface provides information about the domain of study. In SQL-Tutor, the first constraint-based tutor (Mitrovic 1998; Mitrovic & Ohlsson 1999), the interface shows the schema of the database the learner is currently working on, in addition to showing the text of the problem. Visualization of the schema is very important, as it frees the learner from having to memorize the names of relations and attributes. The learner can also obtain more information about the semantics of attributes, data types used, integrities specified on attributes etc. Having

all this information available reduces the cognitive load, and enables the student to concentrate on the task (i.e. query formulation). KERMIT, our database design tutor (Suraweera & Mitrovic 2004) provides all the components of the Entity Relationship model to the student, while NORMIT (Mitrovic 2003) additionally provides help about each step in the data normalization procedure.

The interfaces also support problem solving by visualizing the goal structure. SQL-Tutor shows the six clauses of the SELECT statement, thus making the structure of the query obvious to the student. Likewise, every page in NORMIT shows the structure of the current step of the data normalization procedure, summarizing the work the student has done on the previous steps.

The interfaces also enforce good practice in the chosen instructional domain. For example, for each new construct added to an ER diagram in KERMIT, the student must highlight a phrase from the problem text as the name of that construct. It is not possible to name a construct by typing. Although this requirement might seem too constraining at the first glance, it enforces two of the most important practices in database design: using the end-users language and reflecting on the requirements. By selecting names for various components of the schema directly from the problem text, the student has to think about the requirements. The interface highlights the previously selected parts of the problem text that correspond to various types of ER constructs using different colours, making it easier for the student to review how much of the problem has been covered.

Finally, interfaces provide integrated problem solving environments and provide an experience for the student that should be close to the real-world experience. In case of SQL-Tutor, for example, we provide the student with an option of running their query on a real database. At any point during problem solving, the student may run the query and observe the result from the database. This is additional source of information for the student to reflect on his/her solution and feedback received from the tutor.

3. Supporting problem solving via feedback

The theory of learning from performance errors, as previously discussed, views declarative knowledge as fundamental for learning. A learner can identify an error by comparing the desired effects of an action to the real effects. However, if declarative knowledge is missing, the learner will not be able to identify errors made. This is especially important in early phases of learning. A novice learner is in a vicious circle: trying to improve performance in some skill, the learner naturally does not intend to make errors but he/she is unable to detect errors, because of the lack of experience and knowledge. The same problem applies to the error correction stage: the learner must revisit faulty knowledge, but knowing very little about the domain of instruction, the learner will have difficulty identifying relevant knowledge to correct. In such cases, an ITS makes learning possible by augmenting students' declarative knowledge: the ITS points out the errors for the learner. A constraint-based tutors analyses students' solutions by matching them to constraints. Violated constraints correspond to mistakes, and the system can give feedback.

According to the theory of learning from performance errors, the system should refer the learner to the relevant part of the domain knowledge. Consequently, an effective feedback message should tell the user (a) where exactly the error is, (b) what constitutes the error (perform blame allocation), and (c) refer the user to the underlying concept of the correct solution (revise underlying knowledge). Every constraint has one or more predefined feedback messages, which are given to the student when the constraint is violated.

Our constraint-based tutors provide several levels of feedback. On first submission, the tutor offers positive/negative feedback, stating only whether the student's solution is correct or not. The level of feedback for the second submission will be automatically increased to *Error Flag*, which point to the incorrect part of the solution, still leaving it to the student to figure out the exact mistake. On the next level, a hint message provides details about a single error which violates one of the domain constraints. The student can additionally obtain higher level messages on demand, such as *All Errors* (which lists hints for all violated constraints), *Partial Solution* (which specified the correct version of a part of the solution*) and *Complete Solution. Partial Solution* is particularly important in that it shows the student a correct version of one part of their answer that is currently wrong (e.g. one of the six clauses in SQL) without giving them the whole answer; this type of feedback appears to be very effective, whereas showing the full solution provides very limited gain (Mitrovic, Martin and Mayo 2002). Because the student's problem-solving strategy may differ from the system's solution, this feedback must be generated for each specific case (Martin and Mitrovic 2000).

We have also evaluated the effect of the feedback style. In a recent study performed with our database design tutor (Zakharov et al. 2005) we analyzed the effect of intuitive, common-sense feedback messages that tell the student how to correct the error, compared to carefully reengineered feedback messages that comply with the underlying learning theory. The rephrased messages always indicate the error, and then discuss the violated domain principle, without clearly saying how to correct the error. 105 students used two versions of the tutor over two weeks. The results show that students who received theory-based feedback learned more domain concepts and learned them faster than their peers receiving free-style feedback.

All of our constraint-based tutors are available on the Web, although for some of them we initially developed stand-alone systems. We have adopted a centralized architecture, with a thin client, supporting very limited functions, and the server performing all pedagogical functions. Most importantly, the server is responsible for intelligent analysis of the student solution. However, our database design tutor has a distributed architecture. Because the nature of the task (i.e. database design) is much more demanding and interactive than other tutors, which require a purely textual solution, we have distributed pedagogical actions between the client and the server. The client intervenes in situations when the student makes some syntax errors (such as submitting a diagram with missing component names), to make interaction faster. We have also developed a tutoring engine, WETAS that supports the rapid development of new constraint-based ITS (Martin and Mitrovic 2002).

4. Supporting problem solving via dialogs

Another kind of support for problem solving involves dialogues with the student. We have implemented support for self-explanation in two of our tutors: the database design and data normalization tutors. In both systems the student is asked to explain incorrect actions. In KERMIT-SE (Weerasinghe & Mitrovic 2004), the initial question about the erroneous action is problem-specific (e.g. why is a particular attribute a key of an entity type), which is followed by more general questions if the student is not able to provide a satisfactory answer and correct the answer. The dialogue then continues with more questions discussing the relevant domain concepts and leading the student towards the correct answer. On the other hand, NORMIT asks the student to explain the action performed for the first time and incorrect actions. The first question is also problem-specific: the student is asked to specify the reason for performing a

particular action. If the student cannot specify the reason correctly, he/she will then be asked to define the underlying domain concept.

In both systems, the student selects an answer from a list of predefined options, as we wanted to avoid the complexity of natural language processing. Such dialogues still support self-explanation satisfactorily, as pre-defined answers are designed so to require recall and not only recognition. Evaluation studies show that the dialogues improve problem-solving as well as declarative knowledge acquired by students (Mitrovic 2003; Weerasinghe & Mitrovic 2004).

5. Summary

Intelligent Tutoring Systems provide a rich environment for practising problem solving by modelling the domain being learned and the student learning it. Constraint-based modelling is a technique that focuses on error detection and correction. The ICTG has developed a methodology for building constraint-based tutors that is efficient and which provides a rich learning environment. Problem solving is supported in many ways, such as goal visualisation, scaffolding (by making necessary declarative knowledge available), enforcing good practice (by constraining the interface), providing rich domain-specific feedback and engaging in dialog. These tutors have demonstrated significant gains in student performance, both over comparable students who did not use the systems, and over systems that do not provide the same rich feedback and assistance. Current research is focused in making these tutors even easier to build, in the hope that one day educators will consider building ITS no different to preparing course material or writing a text book.

References

1. Brusilovsky, P., Peylo, C. Int. J. Artificial Intelligence in Education, 2003, 156-169.
2. Martin, B., Mitrovic, A., WETAS: A Web-Based Authoring System for Constraint-Based ITS. In De Bra, P., Brusilovsky, P.L., Conejo, R. (eds), Proc. International Conference on Adaptive Hypermedia and Adaptive Web-Based Systems, 2002, pp. 543-546.
3. Martin, B., Mitrovic, A., Tailoring Feedback by Correcting Student Answers. In Gauthier, G., Frasson, C., VanLehn, K. (eds), Proc. Intelligent Tutoring Systems Conference, 2000, pp. 383-392.
4. Mitrovic, A., Experiences in Implementing Constraint-Based Modeling in SQL-Tutor. In B. Goettl, H. Halff, C. Redfield, V. Shute (eds.), Proc. Intelligent Tutoring Systems Conference, 1998, pp. 414-423.
5. Mitrovic, A., Ohlsson, S. Evaluation of a constraint-based tutor for a database language. Int. J. Artificial Intelligence in Education, 10(3-4), 1999, 238-256.
6. 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, 2004, pp. 565-577.
7. Mitrovic, A., Martin, B., Mayo, M., Using evaluation to shape ITS design: Results and experiences with SQL-Tutor. User Modelling and User Adapted Interaction, v12 no 2-3, pp. 243-279.
8. Ohlsson, S. Constraint-Based Student Modelling. In J. Greer and G. McCalla (eds) Student Modelling: The Key to Individualised Knowledge-based Instruction, vol. 125 Computer Systems and Sciences, NATO ASI, Springer-Verlag, 1994, pp. 167–189.
9. Ohlsson, S. Learning from Performance Errors. Psychological Review, 103(2), 1996, 241-262.
10. Suraweera, P., Mitrovic, A. An Intelligent Tutoring System for Entity Relationship Modelling. Int. J. Artificial Intelligent in Education, v14no3-4, 2004, 375-417.
11. Weerasinghe, A., Mitrovic, A. Supporting Self-Explanation in an Open-ended Domain. In: M.Gh.Negoita, R. J. Howlett, L.C. Jain (eds) Proc of the 8th Int. Conf. Knowledge-Based Intelligent Information and Engineering Systems (KES 2004), Berlin: Springer LNAI 3213, 2004, pp. 306-313.
12. Zakharov, K., Ohlsson, S., Mitrovic, A. Feedback Micro-engineering in EER-Tutor. 12th Int. Conf. Artificial Intelligence in Education, Amsterdam, July 2005.