Chapter 4
Modeling Domains and Students with Constraint-Based Modeling

Antonija Mitrovic

Intelligent Computer Tutoring Group, Department of Computer Science & Software Engineering, Privte Bag 4800, University of Canterbury, Christchurch, New Zealand
tanja.mitrovic@canterbury.ac.nz

Abstract. Stellan Ohlsson proposed CBM in 1992 as a way to overcome some problems in student modeling. Since then, the approach has been extended and used in numerous intelligent tutoring systems, which we refer to as constraint-based tutors. CBM is now an established methodology for modeling instructional domains, representing students' domain knowledge and also higher-level skills. Authoring support for constraint-based tutors is now available, as well as mature, well-tested deployment environments. We present CBM, its foundation and extensions, various types of instructional domains we applied it to, and conclude with avenues for future research.

4.1 Introduction

It has been fifteen years since the initial work on SQL-Tutor, the first ever constraint-based tutor, started. Since then, we have extended CBM from the original, theoretical proposal (Ohlsson 1992) to a thoroughly tested and widely used methodology and have accumulated significant experience developing constraint-based tutors in a variety of instructional domains (Mitrovic et al. 2007). CBM has attracted significant attention and debate, and more and more researchers are adopting it for developing their own Intelligent Tutoring Systems (ITSs) – see e.g. (Galvez et al. 2009; Oh et al. 2009; Le et al. 2009; Siddappa and Manjunath 2008, Menzel 2006; Petry and Rosatelli 2006; Riccucci et al. 2005; Rosatelli and Self 2004).

In this chapter, we start from the basic idea and the psychological foundation of CBM, and present various extensions we have made over the years. We focus on how domain models can be represented in terms of constraints, and then present various kinds of student models based on them. We then turn to the implications of CBM for pedagogical decision making, and discuss various approaches to providing feedback, selecting problems, supporting higher-order skills and affect. CBM is a flexible approach which can be used in a wide variety of instructional domains, to support many teaching strategies.

4.2 CBM: The Basic Idea

At the time when CBM was proposed (Ohlsson 1992), model/knowledge tracing approach championed by the CMU researchers (Anderson et al. 1990; Anderson et al. 1995, Koedinger et al. 1997) was the clear winner on the ITS scene; in fact, it is still the most widely used approach for developing ITSs. Ohlsson proposed CBM as away to avoid some limitations of model-tracing, such as having runnable models of the expert and the student. There are several problems with developing such runnable models, expressed as sets of production rules. He noted the complexity of developing the production set, which is necessary to generate the solution to be compared to the student's actions. For some instructional tasks it might even be impossible to come up with the production set as the domain or the task itself may be ill-defined. Furthermore, if the system is to respond to errors intelligently, buggy rules are necessary. A buggy rule generates an incorrect action, and when it matches the student's action, the tutor can provide remedial feedback.

Enumerating mistakes students can (and do) make is time-consuming and intractable, as the space of incorrect knowledge is vast. Instead of capturing mistakes, CBM focuses on domain principles that *every* correct solution must follow. The fundamental observation CBM is based on is that all correct solutions (to any problems) share the same feature – they do not violate any domain principles. Therefore, instead of representing both correct and incorrect space, it is enough to represent the correct space by capturing domain principles in order to identify mistakes. Any solution (or action) that violates one or more domain principles is incorrect, and the tutoring system can react by advising the student on the mistake even without being able to replicate it.

CBM represents the solution space in terms of abstractions. All solutions states that require the same reaction from the tutor (such as feedback) are grouped in an equivalence class, which corresponds to one constraint. Therefore, an equivalence class represents all solutions that warrant the same instructional action. The advantage of this approach is in its modularity; rather than looking for a specific way of solving the problem (correct or incorrect), each constraint focus on one small part of the domain, which needs to be satisfied by the student's solution in order to be correct. An important assumption is that the actual sequence of actions the student took is not crucial for being able to diagnose mistakes: it is enough to observe the current state of the solution. The student model does not represent the student's action, but the effects of his/her actions instead.

A constraint is an ordered pair an ordered pair (C_r, C_s), where C_r is the relevance condition and C_s is the satisfaction condition. The relevance condition specifies a (simple or compound) test specifying the features of the student solution for which the constraint is of importance. For example, the constraint might be applicable to situations when the student has added two fractions which have the common denominator. The satisfaction condition specifies additional test(s) that must be met by the student solution in order for it to be correct. For the same example, the student's solution is correct if the denominator of the resulting fraction is equal to the denominators of the two given fractions, and the numerator is equal to the sum of the numerators of the given fractions. If the relevance condition is met, but the satisfaction condition is not, then the student's solution is incorrect. Therefore, the general form of a constraint is:

If <relevance condition> is true,
Then <satisfaction condition> had better also be true.

The origin of a mistake is not crucial for generating pedagogical interventions, as the tutoring system can give feedback to the student about the domain principle that was violated, without knowing exactly what kind of incorrect knowledge generated the mistake. Constraint-based tutors normally attach feedback messages directly to constraints.

It is important to point out the difference between constraints and production rules. Although the (English) statement above seems similar to an IF-THEN production rules, it is of a very different nature. A production rule proposes an action to be taken if the goal and the situation specified in the IF part are met. On the contrary, a constraint specifies conditions for a solution state to be correct. The two conditions in a constraint are not linked with logical implication, but with the "ought to" connective, as in if C_r is true, C_s ought to be true as well (Ohlsson and Mitrovic 2007). Production rules are of generative nature, while constraints are evaluative, and can be used for making judgment.

A set of constraints represent features of correct solutions explicitly. Any solution violating one or more constraints is incorrect; therefore the constraint set models errors indirectly, without enumerating them. As the consequence, CBM allows the student to explore the solution space freely; any correct approach to solving a problem will be supported, as there are no constraint violations. CBM is not sensitive to the radical strategy variability (Ohlsson and Bee 1991), which is the observation that students often use different strategies for solving the same problems. CBM allows for creativity: even those solutions that have not been considered by the system designers will be accepted by constraints, as they do not violate any domain constraints. Any problem-solving strategy resulting in a correct state will be recognized as such by CBM.

4.3 The Theoretical Foundations of CBM

CBM is based on Ohlsson's theory of learning from performance errors (Ohlsson 1996). This theory assumes the existence of procedural and declarative knowledge, as common to many theories of learning. Learning starts with accumulating declarative knowledge, which is later converted to procedural knowledge through practice. Procedural knowledge is necessary for generating actions, while declarative knowledge has an important function in evaluating consequences of actions.

The theory states that people make errors because their procedural knowledge is missing or is faulty. Faulty knowledge might be too general or too specific. The theory focuses on learning from errors, which consists of two phases: error detection and error correction. A person may be aware of the error he/she made because the actual outcomes of the action do not match the expected ones; this is the situation when the declarative knowledge (the expectancy of the results of the performed action) allows the person to identify the error themselves. In other situations, if declarative knowledge is missing, the person cannot identify the mistake on his/her own, and would need help, provided in terms of feedback. This feedback might come from the environment itself, or from the teacher, and enables the student to correct the procedural knowledge. The feedback from a tutor, be it a human or an artificial one, consists of identifying the part of the action/solution which is incorrect (blame assignment) and the domain principle that is violated by it. Therefore, the theory states that declarative knowledge is represented in the form of constraints on correct solutions. Such knowledge can be used to correct faulty knowledge, by making it more general or more specific. Computer simulations have shown that using constraints for correcting procedural knowledge is a plausible mechanism. For a detailed account of how constraints allow the corrections of procedural knowledge, please see (Ohlsson 1996).

4.4 Domain Modeling

Domain modeling s widely recognized as being time-consuming and critical for the success of an ITS. In this section, we discuss various types of constraints necessary for a constraint-based system, as well as some authoring considerations.

4.4.1 Syntax vs. Semantic Constraints

The original idea for CBM, as presented in (Ohlsson 1992), viewed constraints as only representing syntax knowledge, i.e. problem-independent domain principles. An example constraint presented in the 1992 paper is from the area of fraction addition:

> *If the current problem is a/b + c/d, and the student's solution is (a+c)/n, then it had better be the case that n=b=d*

The constraint is relevant for the situation when the student added two fractions by adding the numerators; this is only allowed when the denominators of the two fractions and the denominator of the resulting fraction are equal. We refer to such constraints as s*yntax constraints*. These constraints allow an ITS to identify errors in student solutions which violate the general domain principles. Another simple example is *If you are driving in New Zealand, you better be on the left side of the road.*

Our research on CBM started with SQL-Tutor, a constraint-based tutor which teaches university-level students to specify queries in the SQL language. This task is a design task: the student is given a problem in the natural language, and the student needs to convert this into an SQL Select statement. There is no algorithm for performing the task. Furthermore, the natural language text could be ambiguous. Additional complexity inherent in the task is that the student needs to be familiar with the relational data model and the specific relational database the problem is based on. Students typically find the task very difficult (Mitrovic 1998b).

Some examples[1] of syntax constraints from SQL-Tutor are given in Figure 4.1. Constraint 2 is the simplest constraint in this system: the relevance condition is always true, and therefore the constraint is relevant to all solutions. Its satisfaction condition requires that the SELECT clause is specified. In other words, every query must list some expression(s) to be returned from the database.

Constraint 110 focuses on the FROM clause; the constraint is relevant when this clause contains the JOIN keyword. This keyword is used to specify a join condition, and the syntax requires the ON keyword to be specified in the same clause. Notice that this constraint does not check for other elements of the join condition – it simply specifies that those two keywords must appear in the same clause. There are other constraints in SQL-Tutor that check for other elements of the join condition. Such low-level of granularity allows for feedback messages to be very specific. In the case of constraint 110, the feedback message will remind the student that the JOIN and ON keywords need to be used at the same time.

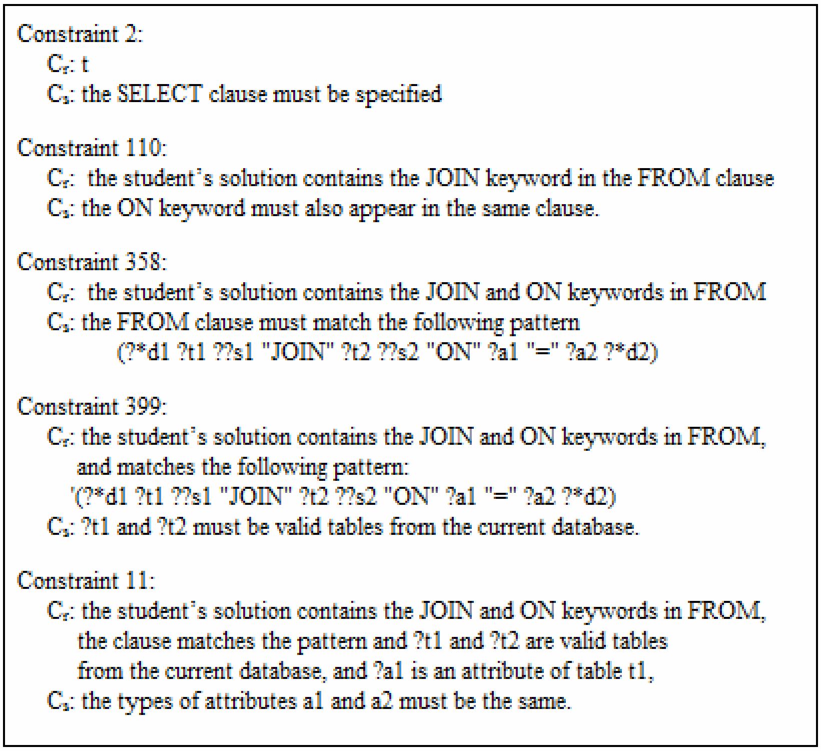
This point is further illustrated by other constraints from Figure 4.1. Constraint 358 builds upon constraint 110: both conditions from 110 are in the relevance condition of constraint 358, and therefore its relevance condition is more restrictive than the one of constraint 110. Its satisfaction condition matches the FROM clause to the given pattern, which specified the general form of a join condition. It allows any number of elements at the beginning and at the end of the FROM clause (wildcards ?*d1 and ?*d2), but somewhere in the clause there must be a value that will be allocated to variable t1, optionally followed by another value (s1, which corresponds to an alias assigned to a table), which is followed by the JOIN keyword, another value (which will be assigned to t2), another optional value (which, if exists, will be assigned to s2), the ON keyword, one value (assigned to a1), the equal sign, and another value (a2). This constraint does not check the values assigned to the variables, but simply checks that the clause is of the specified form. The feedback message attached to this constraint can be more specific about how join conditions are specified.

The following constraint (399) is even more specific: now the relevance condition requires the student's FROM clause to match general form of the join condition, and the satisfaction condition checks that both t1 and t2 are valid table names from the current database. There are other constraints in SQL-Tutor that further check that a1 and a2 are valid attributes, and that the types of those two attributes match each other.

Constraint 11 has all the previously discussed tests in its relevance condition, and its satisfaction condition checks whether the types of attributes used in the join condition are the same. All of those checks can be done on the basis of syntax knowledge, independently of the problem requirements.

However, students do not make only syntax errors. Often their solutions are syntactically correct but the answer is not correct for the problem at hand, and the student should be alerted about that too. Therefore, it is necessary to check the semantic correctness of the solution, and for that reason we introduced semantic

[1] We present the constraints in their English form. The constraints as specified in the constraint language used in SQL-Tutor are given in (Mitrovic 1993).



Constraint 2:
C_r: t
C_s: the SELECT clause must be specified

Constraint 110:
C_r: the student's solution contains the JOIN keyword in the FROM clause
C_s: the ON keyword must also appear in the same clause.

Constraint 358:
C_r: the student's solution contains the JOIN and ON keywords in FROM
C_s: the FROM clause must match the following pattern
(?*d1 ?t1 ??s1 "JOIN" ?t2 ??s2 "ON" ?a1 "=" ?a2 ?*d2)

Constraint 399:
C_r: the student's solution contains the JOIN and ON keywords in FROM, and matches the following pattern:
'(?*d1 ?t1 ??s1 "JOIN" ?t2 ??s2 "ON" ?a1 "=" ?a2 ?*d2)
C_s: ?t1 and ?t2 must be valid tables from the current database.

Constraint 11:
C_r: the student's solution contains the JOIN and ON keywords in FROM, the clause matches the pattern and ?t1 and ?t2 are valid tables from the current database, and ?a1 is an attribute of table t1,
C_s: the types of attributes a1 and a2 must be the same.

Fig. 4.1 Syntax constraints from SQL-Tutor

constraints (Mitrovic 1998a, 1998b, 1998c; Mitrovic and Ohlsson 1999). A semantic constraint compares the student solution and the correct solution to the same problem, again focusing on a single domain principle. Semantic constraints check whether the student's solution is the correct solution for the problem at hand. The semantics of the particular problem is captured in the ideal solution; in SQL-Tutor, the teacher specifies an ideal solution for each problem. This ideal solution is carefully selected to illustrate some important features of the language.

However, in SQL there are often several correct solutions for the same problem, as the language is redundant. Therefore, semantic constraints cannot simply check whether the student's solution is identical to the ideal one; they need to check for equivalent ways of solving the problem.

Figure 4.2 illustrates some semantic constraints from SQL-Tutor. Constraint 207, for example, is relevant when the ideal solution has a join condition specified in the FROM clause, and the student's solution has an empty WHERE clause and more than one table in FROM. In general, if a query uses more than one table in FROM, a join condition is necessary. Join conditions in can be specified either in FROM or in WHERE; this constraints focuses on the FROM clause only, and requires (in the satisfaction condition) that the student specified the join condition in

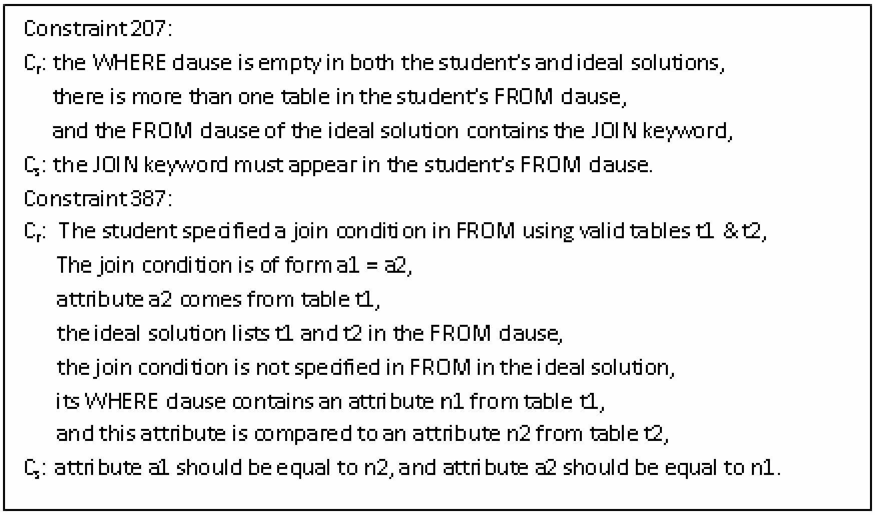

Constraint 207:
C_r: the WHERE clause is empty in both the student's and ideal solutions,
there is more than one table in the student's FROM clause,
and the FROM clause of the ideal solution contains the JOIN keyword,
C_s: the JOIN keyword must appear in the student's FROM clause.
Constraint 387:
C_r: The student specified a join condition in FROM using valid tables t1 & t2,
The join condition is of form a1 = a2,
attribute a2 comes from table t1,
the ideal solution lists t1 and t2 in the FROM clause,
the join condition is not specified in FROM in the ideal solution,
its WHERE clause contains an attribute n1 from table t1,
and this attribute is compared to an attribute n2 from table t2,
C_s: attribute a1 should be equal to n2, and attribute a2 should be equal to n1.

Fig. 4.2 Semantic constraints from SQL-Tutor

FROM. Note that the satisfaction condition does not check for the complete join condition: it only requires the JOIN keyword to be used in FROM. If the student made a syntax error when specify the join condition in FROM, it would be caught by the syntax constraints we discussed previously.

Constraint 387 is relevant when the student specified a join condition in the FROM clause, but the ideal solution contains a join condition in the WHERE clause. The relevance condition of this constraint establishes correspondences between the table and attribute names used in the ideal and the student solution, and the satisfaction condition checks that the corresponding attributes are equal. Note that there are other constraints that will be checking for different combination of attributes.

The constraints need to be specified on a low-level of granularity in order for feedback to be specific. As the consequence, a set of constraints is required to fully specify one domain principles. SQL-Tutor contains a large number of constraints (currently over 700) and it still does not cover all of SQL; the completeness of a constraint set is not necessary for the ITS to be useful. It is important that the constraint set allows for the diagnosis of solutions for a given set of problems. The addition of new problem types would require the constraint set to be extended. But it is easy to add problems of similar types: at the moment SQL-Tutor supports 13 databases, and about 300 problems defined for them. To add another database and the corresponding set of problems, all that is necessary is to add problem text for each problem, its solution, and the information about the database.

4.4.2 Applying CBM to Ill-Defined and Well-Defined Tasks

As discussed previously, the task of specifying queries in SQL-Tutor is a design task, and such tasks are ill-defined. We developed other constraint-based tutors for design tasks. EER-Tutor (Suraweera and Mitrovic 2002, 2004; Mitrovic et al. 2004; Zakharov et al. 2005) teaches conceptual database design, another ill-defined task. The student needs to design an EER diagram for a specific situation starting from the requirements specified in the form of English text. Although the EER data model is well-designed and relatively simple, the actual task of designing a conceptual schema for a database is ill-defined (Suraweera and Mitrovic 2004). The requirements are often incomplete, and the student needs to use their general knowledge in order to design the EER diagram. There is no algorithm to use to identify the necessary components of the solution. Furthermore, the goal state (i.e. the solution) is defined in abstract terms, as an EER diagram that satisfies the requirements. An example syntax constraint from EER-Tutor checks that every regular entity type has at least one key attribute. Semantic constraints make sure that the student has identified all the necessary entities, relationships and attributes, at the same time allowing for alternatives. For example, an attribute of a 1:N relationship may be alternatively represented as an attribute of the entity on the N side of the relationship. Semantic constraints check for such equivalences between the student and the ideal solution.

Another constraint-based tutor for a design task is COLLECT-UML, a tutor that teaches object-oriented software design, by requiring students to design UML class diagrams from textual descriptions (Baghaei et al. 2006, 2007). In all of those instructional tasks, the domain itself is well defined (i.e. the domain theory is well-specified in the terms of the underlying model), but the instructional task is ill-defined.

However, CBM is not only capable of capturing domain knowledge for design tasks – it can also be used for procedural tasks, for which problem-solving algorithms are known. We have developed several tutors of this type. NORMIT (Mitrovic 2005) is a constraint-based tutor that teaches data normalization in relational databases. The domain is well-defined and so is the task: there is an algorithm that students need to learn and apply correctly. NORMIT breaks the task into a series of steps, and requires the student to apply a step correctly before moving on to the next step. This was a deliberate decision, as we wanted to stress the importance of the correct order in which the steps are applied. However, CBM could also be used in the same task in a less restrictive way, by specifying constraints in a slightly different way. We developed another version of NORMIT, in which the student can apply the steps in any order, but constraints check that the student has calculated all the necessary parts of the solution before attempting ones which depend on the previous steps. We refer to such constraints as the *path constraints*.

ERM-Tutor (Milik et al. 2006) is another example of a constraint-based tutor that teaches a procedural task – this time, the student needs to transform an EER diagram into a relational schema. We also developed a lot of tutors for procedural tasks within ASPIRE, our authoring system discussed later in this chapter. An example of such tutors is CIT, a constraint-based tutor that teaches students how to make decision on capital investments (Mitrovic et al. 2008).

In a recent paper (Mitrovic and Weerasinghe 2009), we presented a classification of ITSs in terms of two dimensions, instructional domains and instructional tasks. Both of these can be ill- or well-defined. CBM has previously been applied to well-defined domains, with both ill-defined tasks (SQL-Tutor, EER-Tutor, COLLECT-UML) and well-defined tasks (NORMIT, ERM-Tutor). In the case of ill-defined domains, the tasks can still be ill- or well-defined. CBM can be applied to well-defined tasks no matter what kind of domain is at hand; an example of ill-defined domain and a well-defined task is psychological assessment. In such cases, CBM would be applicable, but so would model-tracing. The latter approach however cannot be applied in the case of an ill-defined task and an ill-defined domain, such as essay writing or artistic tasks. CBM can still be used in such situations.

Let us consider architectural design. If the task is to design a house with three bedrooms for a given piece of land which needs to be eco-friendly and energy efficient, there can be a set of designs which satisfy the minimal requirements. Constraints that need to be satisfied involve the problem specification and the norms for energy consumption and ecological consequences – but the designs will differ in terms of aesthetics and personal preferences of the designer. The constraint set will capture the minimal requirements, and still allow for a variety of solutions. Therefore, in ill-defined domains the student has the freedom to include solution components to make the solution aesthetically pleasing or more to their preferences, and the ITS will still accept it as a good solution for the problem. It is also possible to have weights attached to constraints, with highest weights being assigned to mandatory constraints, and lower weights assigned to constraints that need not necessarily be satisfied as they correspond to optional elements.

In the case of ill-defined domains and ill-defined tasks, more attention needs to be devoted to the feedback provided to the student. In well-defined domains, feedback generation is straightforward: the student violates some constraints, and feedback on violated domain principles is provided. In model-tracing tutors, buggy production rules provide feedback on errors, and hints can be generated on the next step the student is to take. However, in ill-defined domains, the declarative knowledge is incomplete: the constraint set consists of a set of mandatory principles and some heuristics. Therefore, the feedback mechanism needs to be sophisticated, so that feedback does not confuse the student. If the solution is not complete, feedback becomes even more crucial, as the ITS should discuss only the issues the student has worked on so far.

Ill-defined domains and tasks are very complex, and therefore, ITSs need to scaffold learning, by providing as much information as possible without making it trivial. The common ITS techniques can also be used in ill-defined domains (e.g. visualizing goal structure and reducing the working memory load, providing declarative knowledge in the form of dictionaries or on-demand help etc). Furthermore, the ITS can simplify the process by performing one part of the task for the student automatically or by restricting the actions students can take. Furthermore, solution evaluation can be replaced with presenting consequences of student actions or supporting a related, but simpler task, e.g. peer review.

4.4.3 Authoring Domain Models: Constraint Granularity

Authoring domain models for constraint-based tutors consists of specifying syntax and semantic constraints. As discussed previously, constraint sets are different from production models, and generally easier to develop (Mitrovic et al. 2003). However, the process still requires a careful analysis of the target task.

Syntax constraints are generally easier to develop than semantic constraints, but they still require attention. The granularity of constraints is crucial for the effectiveness of the system. If the constraints are on a too coarse level, the feedback would be too general and not useful for the student. For example, we could have had only one constraint instead of a set of constraints such as those presented in Figure 4.1, which focus on the join conditions in the FROM clause. In that case, the only possible feedback message would be that there is something wrong with the FROM clause. Such feedback, of course, is not very useful.

Therefore, for constraints to be pedagogically effective, they need to focus on a very small aspect of a domain principle. There are potentially many constraints necessary to specify all the student should know about a single domain principle. The most important criterion in this process is the pedagogical importance: how specific is the feedback attached to a constraint?

4.4.4 Authoring Support for Constraint-Based Tutors

In addition to developing many constraint-based systems, we also devoted a lot of effort to providing authoring support (Mitrovic et al. 2007). WETAS (Martin and Mitrovic 2003) is the result of early research in this direction: it is an ITS shell that provides all the functionality necessary for a Web-based constraint tutor. In order to develop an ITS in WETAS, the author needs to provide the constraint set and the problems and their solutions. However, the development of a constraint set is still a demanding task. We developed ASPIRE (Mitrovic et al. 2006; Mitrovic et al. 2009), an authoring and deployment environment for constraint-based tutors, which supports the process of developing the constraint set. The author develops a domain ontology, and provides examples of problems with their solutions, from which ASPIRE generates constraints. ASPIRE is not capable of generating all constraints for any instructional domain, but the initial evaluation we performed shows that it is capable of generating the majority of necessary constraints (of the order of 90%). The quality and coverage of the developed constraint set, of course, depends critically on the quality of the author provided information (the ontology and problems/solutions). VIPER (Martin et al. 2009) further simplifies the authoring process by focusing on instructional domains with specific features, thus making the author's task easier.

4.5 Student Modeling

In the first paper on CBM, Ohlsson (1992) focused on short-term student modeling, or the diagnosis of the student's current action. The process starts by matching the relevance conditions of all constraints to the student solution. Then, for relevant constraints, the satisfaction conditions are matched as well. The same process can be applied incrementally, to isolated actions as the student is performing them, or to the whole solution. Therefore the short-term student model consists of the list of satisfied constraints and (potentially) the list of violated constraints. Violated constraints allow the constraint-based tutor to provide feedback to the student. The feedback states that the action/solution is wrong, points out the part of the solution which is wrong, and then specifies the domain principle that is violated. The error correction is left to the student to perform.

Feedback generation is only one of the pedagogical actions ITSs provide. Most often, feedback is generated on the basis of the last action the student performed, although previous performance can also be taken into account. However, ITSs also require long-term student model in order to generate other adaptive actions, such as selecting problems or topics to be covered in an instructional session. We therefore extended CBM by proposing several different ways of representing the long-term model of the student's knowledge.

In our early work (Mitrovic, 1998a, 1998b, 1998c), the long-term model of the student's knowledge was represented in terms of the overlay model. This is the logical extension of Ohlsson's initial proposal of CBM. A set of constraints represents what is true in the domain. Therefore the model of an expert would be equivalent to the whole constraint set, while a novice will only know some constraints. For each constraint the student has used, our tutors store the history of its usage, which allows us to track the student's progress on that constraint. Of course, over time the student's knowledge improves, and therefore the system cannot use the complete history always. A simple way to use such histories is to select a window of a particular size – say last five attempts on a constraint – and calculate the frequency of correct usage. This can be done for each constraint in the student model, and an estimate of the student's knowledge can be based on that. We have used such simple long-term models in most of our constraint-based tutors.

A more sophisticated approach is to develop a probabilistic, Bayesian model of the student's knowledge, as we have done for SQL-Tutor (Mayo and Mitrovic 2000) and CAPIT, a system that teaches elementary school children about punctuation and capitalization rules in English (Mayo and Mitrovic 2001). We also experimented with training an artificial neural network and using it for problem selection (Wang and Mitrovic 2002).

4.6 Pedagogy: What Can CBM Support?

CBM is neutral with respect to pedagogy. Ohlsson (1992) pointed out that CBM can be used offline, for diagnosing students' solution after each session, or online, to generate pedagogical actions. We have used CBM in our tutors with a variety of teaching strategies, to provide feedback to students, to select problems, support students' meta-cognitive skills and collaborative learning.

4.6.1 Feedback Generation in Constraint-Based Tutors

Constraint-based tutors use constraints to augment the student's declarative knowledge. If the student cannot detect errors by him/herself, the system will alert them to the domain principles which are violated. In this way, CBM can be used to provide feedback to students. Such feedback can be provided immediately, after each action the student performs, or in a delayed fashion, after the student is done with a problem. The choice of the timing of feedback is therefore flexible.

As stated previously, feedback is attached to constraints; when the student violates a constraint, the attached feedback message can be given to student. However, there are many additional considerations taken into account when presenting feedback, such as timing of feedback, content, type, presentation and adaptation.

Timing of feedback: In our tutors that teach design tasks, the student can decide when they want to get feedback. The solution is analyzed on student's request, and the whole solution is analyzed at once. The tutor does not diagnose individual student actions. Such an approach puts the student in control of their learning, while still providing necessary guidance when the student requests it. For procedural tasks, we typically break them into steps, and analyze a group of activities within a step. The student is required to complete one step correctly before going on to the next step. CBM can also be used to provide immediate feedback, by analyzing each action the student performs. The decision on the timing of feedback depends on the nature of the task performed: for design tasks it is more natural to diagnose the whole solution at once.

Amount of feedback: Another important pedagogical decision is related to feedback: how much information should be given to the student? Our tutors typically provide several levels of feedback. For example, SQL-Tutor offers the following feedback levels*: correct/incorrect, error flag, hint, all errors, partial solution,* and *complete solution*. On the first submission, the feedback only informs the student whether the solution is correct or not. The following submission points out the part of the solution which is incorrect; for example, the system might identify the FROM clause as being wrong. Such feedback is useful for the student to correct slips. If there is a deeper misunderstanding, the hint-level feedback will present the message attached to the violated constraint. If there are several violated constraints, the student can see the hint messages attached to them at the *All errors* level. The partial solution provides the correct version of the clause that was not right in the student's solution, while the full solution is available on the highest level of feedback. The system will automatically increase the feedback level on each submission until it reaches the *Hint* level; it will then stay on that level. The student, however, can ask for higher-level feedback whenever he/she desires.

Feedback content: The feedback messages we defined for early versions of our tutors were based on our intuition – we were thinking of what a good human tutor would say to the student violating a particular constraint. Such intuitive feedback does not have a theoretical foundation, and can result in feedback of variable quality. We therefore turned to the psychological theory of learning CBM was derived for. The theory says that effective feedback should tell the student here the error is, what constitutes the error and re-iterate the domain principle violated by the student. For example, the feedback message might say that the error is in the sum that the student computed, and point out that the sum is 93, but since the numbers are percentages, they should add up to 100.

We re-engineered feedback for EER-Tutor with these theoretical guidelines in mind (Zakharov et al. 2005). As an example, the original feedback message attached to constraint 23 was: "*Check whether each identifying relationship has an owner entity, which must be a regular entity type.*" The new, theory-based feedback for the same constraint is: "*An identifying relationship type must be connected to a regular entity type, which is the owner of the weak entity type. The highlighted identifying relationship is not connected to a regular entity type."* When this feedback is given to the student, the incorrect construct (i.e. the identifying relationship) is highlighted in the diagram. The results of evaluation show that theoretical feedback is more effective in supporting learning than intuitive one, by increasing the learning rate.

Types of feedback: Feedback messages attached to constraints are feedback on errors – we refer to such feedback as *negative feedback*. Most feedback provided by ITSs is of this type. However, human teachers very often provide positive feedback, i.e. feedback on correct actions. Such feedback is useful as it confirms tentative actions and supports students in strengthening their knowledge. Positive feedback also helps the student to integrate newly acquired with existing knowledge. We developed a version of SQL-Tutor which provided positive feedback in addition to negative (Barrow et al. 2008). The content of positive feedback acknowledges the correct action, and re-iterates the domain principle that was satisfied by it. However, positive feedback is not given on each submission, as that would be overwhelming and repetitive. On the contrary, the system decides when to provide positive feedback, looking for evidence that the student was uncertain, but still managed to solve the problem (or a problem step). Other situations when positive feedback is provided is when he student learns a difficult constraint, uses the constraint correctly for the first time, or solves a difficult problem. The study has shown that students who received positive feedback solved the same number of problems and learnt the same amount of knowledge as the students in the control group but in half the time of the control group.

Feedback presentation: if the student violated several constraints, the system needs to decide which constraints to target, and in which order. The simplest solution is to order the constraints within the domain model and use this order to present feedback, as we have done in early versions of SQL-Tutor. However, the ordering is difficult to implement. It is also possible to select constraints adaptively, on the basis of the student model, as we have done in several systems. Other researchers have suggested similar approaches, for example adding weights to constraints and selecting constraints with the highest weights (Le et al. 2009). Furthermore, constraints can be organized in terms of domain concepts, as done in ASPIRE (Mitrovic et al. 2009) and also in the context of EER-Tutor, when deciding on the tutorial dialogue to engage the student in (Weerasinghe et al. 2009).

Adapting feedback: Feedback provided on violated constraints corresponds to the current solution; however, if two students submit exactly the same solution, they would get the same feedback. For that reason, we enhanced SQL-Tutor to adapt the feedback to the particular student by changing the generality of feedback in relation to the student model (Martin and Mitrovic 2006).

4.6.2 CBM and Problem Selection

CBM also supports problem selection; as stated in the previous section, the long-term model stores the summary of the student's progress on a skill. Numerous problem-selection strategies can be formulated on the basis of such student models. In our early work, we started with a simple problem-selection strategy which focused on a single constraint that the student has most difficulty learning. Such a constraint can be easily identified from the long-term student model, and can guide the selection of the problem. For example, in early versions of SQL-Tutor (Mitrovic 1998a, 1998b, 1998c) the system identified the most often violated constraint, and then selected a problem which was new to the student, at the appropriate level of difficulty[2] (based on the student model) which exercised the chosen constraints. We later used decision theory and the probabilistic student model to select the best problem for the student (Mayo and Mitrovic 2000, 2001). In another version of SQL-Tutor, we trained an artificial neural network to predict the problem which will be at the right level of complexity for the student (Wang and Mitrovic 2002). Later on, we experimented with computing the problem difficulty dynamically, in relation to the student model; the corresponding problem-selection strategy proved to be superior to the one based on the static problem complexities (Mitrovic and Martin 2004). Finally, we also introduced problem templates and presented them to students, during problem selection (Mathews and Mitrovic 2007).

4.6.3 Supporting Higher-Level Skills and Affect with CBM

Constraints represent the domain knowledge, and are used as the basis for representing students' knowledge. We also explored the effect of opening the student model to the student on their higher-level skills, such as self-assessment. In a series of studies done in the context of SQL-Tutor (Mitrovic and Martin 2007), we have shown that students improve their self-assessment skills when having access to relatively simple open student models, in the form of skill meters. Although the student model was presented in a highly abstracted way, it helped student reflect on their knowledge and select more appropriate problems for further work.

Another important higher-level skill is self-explanation, which is known to promote deep learning (Chi 2000). In the studies with our database design tutor (Weerasinghe and Mitrovic 2006, 2008), the students participated in tutorial dialogues aimed at elicitating explanations from students. Such explanations enable students to relate their problem-solving skills to declarative knowledge (Mitrovic 2005).

Constraints can also be used to represent collaborative skills. COLECT-UML supports teams of students developing an UML diagram collaboratively (Baghaei et al. 2007). The system provides domain-level feedback based on the analysis of individual and group solutions. In addition, it also provides feedback on collaboration by analyzing the student's participation in group activities. The model of ideal

[2] Each problem in SQL-Tutor has a static problem complexity level specified by the teacher (ranges from 1 to 9).

collaboration was presented in the form of a set of meta-constraints. The system was successful in supporting both students' learning of domain knowledge and collaborative skills.

A lot of research has been done during the last decade on recognizing the student's affective state and responding to it actively. We developed an animated pedagogical agent which is capable of identifying the trend in which the student's affective state is changing. The system analyzes student's facial features and identifies changes from a positive to negative state or vice versa. This information is then combined with the information about the cognitive state, and used to modify the feedback provided to the student and also the agent's behaviour (Zakharov et al. 2008). Although there is a lot of research questions still to be answered, our initial experiences have been very positive. The students appreciated getting feedback from an affect-sensitive agent.

4.7 Conclusions

In the last fifteen years, CBM has grown from a theoretical proposal to a fully developed, mature methodology for building ITSs. We have used CBM successfully in many instructional domains, with students of different ages and backgrounds, in real classrooms at universities and in schools. Additionally, three of our database tutors have been available to students worldwide via the Addison-Wesley's DatabasePlace[3] Web portal since 2003, and have been used by more than 10,000 students worldwide. We have performed more than 30 evaluation studies, which proved the effectiveness of this modeling approach.

Constraint-based tutors, as discussed above, do not require runnable expert models in order to diagnose student solutions; this feature enables CBM to be applicable in ill-defined tasks/domains, where model-tracing tutors cannot be applied. Constraints can capture whatever is known about the ill-defined domain and the problem specification, thus begin able to evaluate the mandatory parts of the solution. Such a tutor can provide feedback to student, while still allowing for multiple solutions differing in non-essential elements, such as aesthetical and personal preferences.

CBM also does not require bug libraries and consequently constraint-based tutors require less effort than model-tracing ones. CBM has a strong theoretical foundation, which provides guidelines for generating feedback content.

CBM is a flexible approach, but it is not the only possible way of developing ITSs. Many questions have been answered, but many more are still open. Our current work focuses on improving authoring support, modeling affect and student engagement, as well as meta-cognitive skills. We do not believe that only a single representation (constraint or production rules) is superior in all respects and situations. Hybrid systems, using combinations of different representations have a much higher probability of supporting different kinds of instructions and different learners.

[3] www.databaseplace.com

Acknowledgments. The work reported here could not have been done without the wonderful bunch of students and colleagues at the Intelligent Computer Tutoring Group (ICTG). Thank you all for all the discussions and friendship over the years – I have been privileged to work with you.

References

Anderson, J.R., Boyle, C.F., Corbett, A.T., Lewis, M.W.: Cognitive Modeling and Intelligent Tutoring. Artificial Intelligence 42, 7–49 (1990)

Anderson, J.R., Corbett, A.T., Koedinger, K.R., Pelletier, R.: Cognitive Tutors: Lessons Learned. Learning Sciences 4(2), 167–207 (1995)

Baghaei, N., Mitrovic, A., Irvin, W.: Problem-Solving Support in a Constraint-based Intelligent Tutoring System for UML. Technology, Instruction, Cognition and Learning 4(2), 113–137 (2006)

Baghaei, N., Mitrovic, A., Irwin, W.: Supporting collaborative learning and problem-solving in a constraint-based CSCL environment for UML class diagrams. Computer-Supported Collaborative Learning 2(2-3), 159–190 (2007)

Barrow, D., Mitrovic, A., Ohlsson, S., Grimley, M.: Assessing the Impact of Positive Feedback in Constraint-based ITSs. In: Woolf, B., et al. (eds.) ITS 2008. LNCS, vol. 5091, pp. 250–259. Springer, Heidelberg (2008)

Chi, M.T.H.: Self-explaining Expository Texts: The dual processes of generating inferences and repairing mental models. Advances in Instructional Psychology, 161–238 (2000)

Galvez, J., Guzman, E., Conejo, R., Millan, E.: Student Knowledge Diagnosis using Item Response Theory and Constraint-based Modeling. In: Dimitrova, V., Mizoguchi, R., du Boulay, B., Graesser, A. (eds.) Proc. 14th Int. Conf. Artificial Intelligence in Education, pp. 291–298 (2009)

Koedinger, K.R., Anderson, J.R., Hadley, W.H., Mark, M.A.: Intelligent Tutoring Goes to the Big City. Artificial Intelligence in Education 8, 30–43 (1997)

Le, N.-T., Menzel, W., Pinkwart, N.: Evaluation of a Constraint-Based Homework Assistance System for Logic Programming. In: Kong, S.C., Ogata, H., Arnseth, H.C., Chan, C.K.K., Hirashima, T., Klett, F., Lee, J.H.M., Liu, C.C., Looi, C.K., Milrad, M., Mitrovic, A., Nakabayashi, K., Wong, S.L., Yang, S.J.H. (eds.) Proc. 17th Int. Conf. Computers in Education, APSCE (2009)

Martin, B., Mitrovic, A.: Domain Modeling: Art or Science? In: Hoppe, U., Verdejo, F., Kay, J. (eds.) Proc. 11th Int. Conference on Artificial Intelligence in Education AIED. IOS Press, Amsterdam (2003)

Martin, B., Mitrović, A.: The Effect of Adapting Feedback Generality in ITSs. In: Wade, V., Ashman, H., Smyth, B. (eds.) AH 2006. LNCS, vol. 4018, pp. 192–202. Springer, Heidelberg (2006)

Martin, B., Kirkbride, T., Mitrovic, A., Holland, J., Zakharov, K.: An Intelligent Tutoring System for Medical Imaging. In: Bastiaens, T., Dron, J., Xin, C. (eds.) Proc. World Conf. E-Learning in Corporate, Government, Healthcare, and Higher Education. AACE, Vancouver, CA (2009)

Mathews, M., Mitrovic, A.: Investigating the Effectiveness of Problem Templates on Learning in ITSs. In: Luckin, R., Koedinger, K., Greer, J. (eds.) Proc. Artificial Intelligence in Education, pp. 611–613 (2007)

Mayo, M., Mitrovic, A.: Using a Probabilistic Student Model to Control Problem Difficulty. In: Gauthier, G., VanLehn, K., Frasson, C. (eds.) ITS 2000. LNCS, vol. 1839, p. 524. Springer, Heidelberg (2000)

Mayo, M., Mitrovic, A.: Optimising ITS Behaviour with Bayesian Networks and Decision Theory. Artificial Intelligence in Education 12(2), 124–153 (2001)

Menzel, W.: Constraint-based Modeling and Ambiguity. Artificial Intelligence in Education 16(1), 29–63 (2006)

Milik, N., Marshall, M., Mitrović, A.: Teaching Logical Database Design in ERM-Tutor M. In: Ikeda, M., Ashley, K., Chan, T.-W. (eds.) ITS 2006. LNCS, vol. 4053, pp. 707–709. Springer, Heidelberg (2006)

Mitrovic, A.: Learning SQL with a Computerized Tutor. In: 29th ACM SIGCSE Technical Symposium, pp. 307–311 (1998a)

Mitrovic, A.: A Knowledge-Based Teaching System for SQL. In: Ottmann, T., Tomek, I. (eds.) Proc. ED-MEDIA 1998, AACE, pp. 1027–1032 (1998b)

Mitrovic, A.: Experiences in Implementing Constraint-Based Modeling in SQL-Tutor. In: Goettl, B., Halff, H., Redfield, C., Shute, V. (eds.) ITS 1998. LNCS, vol. 1452, pp. 414–423. Springer, Heidelberg (1998c)

Mitrovic, A.: An Intelligent SQL Tutor on the Web. Artificial Intelligence in Education 13(2), 173–197 (2003)

Mitrovic, A.: The Effect of Explaining on Learning: a Case Study with a Data Normalization Tutor. In: Looi, C.-K., McCalla, G., Bredeweg, B., Breuker, J. (eds.) Proc. Conf. Artificial Intelligence in Education, pp. 499–506 (2005)

Mitrović, A., Martin, B.: Evaluating Adaptive Problem Selection. In: De Bra, P., Nejdl, W. (eds.) AH 2004. LNCS, vol. 3137, pp. 185–194. Springer, Heidelberg (2004)

Mitrovic, A., Martin, B.: Evaluating the Effect of Open Student Models on Self-Assessment. Artificial Intelligence in Education 17(2), 121–144 (2007)

Mitrovic, A., Koedinger, K., Martin, B.: A Comparative Analysis of Cognitive Tutoring and Constraint-based Modeling. In: Brusilovsky, P., Corbett, A., de Rosis, F. (eds.) UM 2003. LNCS (LNAI), vol. 2702, pp. 313–322. Springer, Heidelberg (2003)

Mitrovic, A., Martin, B., Suraweera, P.: Intelligent Tutors for all: Constraint-based Modeling Methodology, Systems and Authoring. IEEE Intelligent Systems 22(4), 38–45 (2007)

Mitrovic, A., Martin, B., Suraweera, P., Zakharov, K., Milik, N., Holland, J., McGuigan, N.: ASPIRE: an Authoring System and Deployment Environment for Constraint-based Tutors. Artificial Intelligence in Education 19(2), 155–188 (2009)

Mitrovic, A., McGuigan, N., Martin, B., Suraweera, P., Milik, N., Holland, J.: Authoring Constraint-based Systems in ASPIRE: A Case Study of a Capital Investment Tutor. In: Proc. ED-MEDIA 2008, pp. 4607–4616 (2008)

Mitrovic, A., Ohlsson, S.: Evaluation of a Constraint-Based Tutor for a Database Language. Artificial Intelligence in Education 10(3-4), 238–256 (1999)

Mitrovic, A., Suraweera, P., Martin, B., Weerasinghe, A.: DB-suite: Experiences with Three Intelligent, Web-based Database Tutors. Journal of Interactive Learning Research 15(4), 409–432 (2004)

Mitrovic, A., Suraweera, P., Martin, B., Zakharov, K., Milik, N., Holland, J.: Authoring Constraint-based Tutors in ASPIRE. In: Ikeda, M., Ashley, K., Chan, T.-W. (eds.) ITS 2006. LNCS, vol. 4053, pp. 41–50. Springer, Heidelberg (2006)

Mitrovic, A., Weerasinghe, A.: Revisiting the Ill-Definedness and Consequences for ITSs. In: Dimitrova, V., Mizoguchi, R., du Boulay, B., Graesser, A. (eds.) Proc. 14th Int. Conf. Artificial Intelligence in Education (2009)

Oh, Y., Gross, M.D., Ishizaki, S., Do, Y.-L.: Constraint-based Design Critic for Flat-pack Furniture Design. In: Kong, S.C., Ogata, H., Arnseth, H.C., Chan, C.K.K., Hirashima, T., Klett, F., Lee, J.H.M., Liu, C.C., Looi, C.K., Milrad, M., Mitrovic, A., Nakabayashi, K., Wong, S.L., Yang, S.J.H. (eds.) Proc. 17th Int. Conf. Computers in Education, APSCE (2009)

Ohlsson, S.: Constraint-based Student Modeling. Artificial Intelligence in Education 3(4), 429–447 (1992)

Ohlsson, S.: Learning from performance errors. Psychological Review 103, 241–262 (1996)

Ohlsson, S., Bee, N.: Strategy Variability: A Challenge to Models of Procedural Learning. In: Birnbaum, L. (ed.) Proc. Int. Conf. of the Learning Sciences, AACE, pp. 351–356 (1991)

Ohlsson, S., Mitrovic, A.: Fidelity and Efficiency of Knowledge representations for intelligent tutoring systems. Technology, Instruction, Cognition and Learning 5(2), 101–132 (2007)

Petry, P.G., Rosatelli, M.: AlgoLC: A Learning Companion System for Teaching and Learning Algorithms. In: Ikeda, M., Ashley, K.D., Chan, T.-W. (eds.) ITS 2006. LNCS, vol. 4053, pp. 775–777. Springer, Heidelberg (2006)

Riccucci, S., Carbonaro, A., Casadei, G.: An Architecture for Knowledge Management in Intelligent Tutoring Systems. In: Proc. IADIS Int. Cong. Cognition and Exploratory Learning in Digital Age (2005)

Rosatelli, M., Self, J.: A Collaborative Case Study System for Distance Learning. Artificial Intelligence in Education 14(1), 1–29 (2004)

Siddappa, M., Manjunath, A.S.: Intelligent Tutor Generator for Intelligent Tutoring Systems. In: Proc. World Congress on Engineering and Computer Science, pp. 578–583 (2008)

Suraweera, P., Mitrovic, A.: KERMIT: A Constraint-Based Tutor for Database Modelling. In: Cerri, S.A., Gouardéres, G., Paraguaçu, F. (eds.) ITS 2002. LNCS, vol. 2363, pp. 376–387. Springer, Heidelberg (2002)

Suraweera, P., Mitrovic, A.: An Intelligent Tutoring System for Entity Relationship Modelling. Artificial Intelligence in Education 14(3-4), 375–417 (2004)

Wang, T., Mitrovic, A.: Using neural networks to predict student's behaviour. In: Kinshuk, R., Lewis, K., Akahori, R., Kemp, T., Okamoto, L., Henderson, C.-H. (eds.) Proc. Int. Conf. Computers in Education, pp. 969–973 (2002)

Weerasinghe, A., Mitrovic, A.: Facilitating Deep Learning through Self-Explanation in an Open-ended Domain. Int. J. of Knowledge-based and Intelligent Engineering Systems 10(1), 3–19 (2006)

Weerasinghe, A., Mitrovic, A.: A Preliminary Study of a General Model for Supporting Tutorial Dialogues. In: Proc. Int. Conf. Computers in Education, pp. 125–132 (2008)

Weerasinghe, A., Mitrovic, A., Martin, B.: Towards individualized dialogue support for ill-defined domains. Artificial Intelligence in Education 14 (2009) (in print)

Zakharov, K., Mitrovic, A., Johnston, L.: Towards Emotionally-Intelligent Pedagogical Agents. In: Woolf, B.P., Aïmeur, E., Nkambou, R., Lajoie, S. (eds.) ITS 2008. LNCS, vol. 5091, pp. 19–28. Springer, Heidelberg (2008)

Zakharov, K., Mitrovic, A., Ohlsson, S.: Feedback Micro-Engineering in EER-Tutor. In: Looi, C.-K., McCalla, G., Bredeweg, B., Breuker, J. (eds.) Proc. 12th Int. Conf. Artificial Intelligence in Education, pp. 718–725. IOS Press, Amsterdam (2005)