User Model User-Adap Inter (2012) 22:39–72
DOI 10.1007/s11257-011-9105-9

ORIGINAL PAPER

Fifteen years of constraint-based tutors: what we have achieved and where we are going

Antonija Mitrovic

Received: 16 August 2010 / Accepted in revised form: 31 December 2010 /
Published online: 8 October 2011

Abstract Fifteen years ago, research started on SQL-Tutor, the first constraint-based tutor. The initial efforts were focused on evaluating Constraint-Based Modeling (CBM), its effectiveness and applicability to various instructional domains. Since then, we extended CBM in a number of ways, and developed many constraint-based tutors. Our tutors teach both well- and ill-defined domains and tasks, and deal with domain- and meta-level skills. We have supported mainly individual learning, but also the acquisition of collaborative skills. Authoring support for constraint-based tutors is now available, as well as mature, well-tested deployment environments. Our current research focuses on building affect-sensitive and motivational tutors. Over the period of fifteen years, CBM has progressed from a theoretical idea to a mature, reliable and effective methodology for developing effective tutors.

Keywords Constraint-based modeling · Constraint-based tutors · Authoring · Affective modeling · Metacognitive skills · Collaborative learning

1 Introduction

Stellan Ohlsson proposed Constraint-Based Modeling (CBM) in 1992, as a way to overcome problems with student modeling. In 1995, we started working on SQL-Tutor, the first constraint-based tutor. At the time, CBM was a theoretical idea, a way to represent the domain knowledge as a set of constraints, which can be used to analyze students' solutions in order to provide feedback on errors. Our early work

A. Mitrovic (✉)
Intelligent Computer Tutoring Group, Department of Computer Science and Software Engineering, University of Canterbury, Private Bag 4800, Christchurch, New Zealand
e-mail: tanja.mitrovic@canterbury.ac.nz

within the Intelligent Computer Tutoring Group (ICTG) focused on showing that CBM was an effective way of modeling domains and student knowledge for Intelligent Tutoring Systems (ITS). In order to develop SQL-Tutor, we proposed a number of extensions to CBM, including a way to develop long-term student models, the distinction between syntax and semantic constraints, and the use of ideal solutions in order to deal with complex domains, in which there are potentially many correct solutions. Many other constraint-based tutors followed after SQL-Tutor, and the research focused expanded from investigating the applicability of CBM to various instructional domains, to representing and supporting the acquisition of cognitive and meta-cognitive skills, collaborative learning, open student models, and affective modeling. CBM is now a thoroughly tested and widely used methodology and we 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. It is not used solely by ICTG, but also by various groups of researchers worldwide—see e.g. (Rosatelli and Self 2004; Riccucci et al. 2005; Petry and Rosatelli 2006; Mills and Dalgarno 2007; Siddappa and Manjunath 2008; Menzel 2006; Oh et al. 2009; Galvez et al. 2009a,b; Le 2006; Le et al. 2009; Roll et al. 2010).

We start from the basic idea (Sect. 2) and the psychological foundation of CBM (Sect. 3), and present various extensions we have made over the years. Section 4 discusses syntax and semantic constraints, and presents example constraints from SQL-Tutor. The following section discusses the kinds of instructional domains and tasks that CBM is applicable to. Section 6 discusses student models in constraint-based tutors. An important part of our research is devoted to implications of CBM for pedagogical decision making. In Sect. 7, we discuss the types of pedagogical actions constraint-based tutors are capable of: presenting feedback, selecting problems, supporting higher-order skills, collaborative learning and affect. Section 8 presents the deployment and authoring support that exists for constraint-based tutors, as well as some tutors developed in ASPIRE.

CBM is a flexible approach which can be used in a wide variety of instructional domains, to support many teaching strategies. We conclude with a discussion of current and future projects.

2 CBM: modeling domain knowledge and diagnosing student solutions

When CBM was proposed in 1992 (Ohlsson 1992), the model/knowledge tracing approach championed by the CMU researchers (Anderson et al. 1990, 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 requiring runnable models of the expert and the student. There are several problems with developing such runnable models, expressed as sets of production rules. Ohlsson noted the complexity of developing the production set that can produce the solution to a given problem, which is then 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 (i.e. provide appropriate and useful

feedback), 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. In the absence of a buggy rule that corresponds to an incorrect student action, the only feedback that the system can generate is to flag the action as incorrect, without being able to provide feedback on it.

Identifying student errors is a time-consuming process, which requires extensive studies of students behaviour while problem solving; furthermore, the process is 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 as in model tracing, it is enough to represent the correct space alone by capturing domain principles. Any solution or an individual student 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.

Therefore, 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. An equivalence class thus represents all solutions that warrant the same instructional action. The advantage of the CBM approach is in its modularity; rather than looking for a specific way of solving the problem (correct or incorrect), each constraint focuses on one small part of the domain, which needs to be satisfied by the student's solution for it to be correct. An important assumption is that the actual sequence of actions the student performed is not crucial for being able to diagnose mistakes: it is enough to observe the current state of the solution. The actual problem-solving approach is irrelevant; two different procedures resulting in the same solution state require the same feedback, as they violate the same domain principles. The student model does not represent the student's actions, but the effects of his/her actions instead.

Each constraint consists of an ordered pair (C_r, C_s), where C_r is the relevance condition and C_s is the satisfaction condition. The relevance condition checks whether the constraint is applicable to the student solution by testing the features of the solution. For example, the constraint might be applicable to situations when the student has added two fractions with the common denominator. The satisfaction condition specifies additional test(s) that must be met by correct solutions; 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; the tutoring system does not need to be able to reproduce the error, or understand how

it was generated in order to react intelligently.[1] If a constraint is violated, the ITS can provide feedback to the student about the domain principle that was violated, without knowing exactly what kind of incorrect knowledge caused the mistake. In the case above, if the student has added two fractions with the same denominator (e.g. a/b +c/b) but the result is (a+c)/2b (i.e. the student believes that fractions can be added by independently adding denominators and nominators), the constraint will be violated. The ITS would inform the student that the resulting fraction must have the same denominator as the two starting fractions. Constraint-based tutors normally attach feedback messages directly to constraints, and use them to generate feedback, as discussed later (please see Sect. 7).

At this point, it is necessary to point out the difference between constraints and production rules. Although the (English) statement above seems similar to an IF-THEN production rule, it is of a very different nature. The IF part of a production rule specifies the characteristics of a situation to which the rule is applicable, and the goal to be reached, while the THEN part specifies the action to be taken if both the goal and the situation 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.

The set of constraints for a given domain explicitly represents the features of all correct solutions. Any solution violating one or more constraints is incorrect; therefore the constraint set models errors indirectly, without enumerating them. Therefore, 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.

3 Back to the basics: the underlying learning theory

CBM comes from Ohlsson's theory of learning from performance errors (Ohlsson 1996). This theory assumes the existence of procedural and declarative knowledge, as is 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

[1] On the other hand, if the origin of the error is known, such information may be useful for providing feedback to the student—this issue is still open for debate.

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 support the corrections of procedural knowledge, please see (Ohlsson 1996).

4 Syntax and semantic constraints

Originally, Ohlsson envisioned constraints to represent syntax knowledge only, i.e. problem-independent domain principles. An example constraint presented in the 1992 paper comes 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 (the satisfaction condition). 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 and/or incomplete. 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[2] of syntax constraints from SQL-Tutor are given in Fig. 1. Constraint 2 is the simplest constraint in this system: the relevance condition is always

[2] We present the constraints in their English form. The constraints as specified in the constraint language used in SQL-Tutor are given in (Mitrovic 2003).

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. 1 Some syntax constraints from SQL-tutor

true, and therefore the constraint is relevant to all solutions. Its satisfaction condition requires that the student specified the SELECT clause. In other words, every query must list some expression(s) to be returned from the database. There is a similar constraint that makes sure that the student has specified the FROM clause, by listing the necessary tables.

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 Fig. 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 specifies 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

Constraint 207:
C_r: the WHERE clause is empty in both the student's and ideal solutions,
and 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 and 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. 2 Semantic constraints from SQL-tutor

$s2$), 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 constraints (Mitrovic 1998a,b,c; Mitrovic and Ohlsson 1999). A semantic constraint compares the student solution to 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 in the light of the relevant domain principles. 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 (as in many other domains) there are often several correct solutions for the same problem, as the language contains a lot of redundancy. Therefore, semantic constraints cannot simply check whether the student's solution is identical to the ideal one; they need to check for alternative ways of solving the problem.

Let us discuss the semantic constraints in terms of several examples. Figure 2 illustrates some semantic constraints from SQL-Tutor. For example, constraint 207 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 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 FROM (because the WHERE clause is empty). 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 specifying 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.

Semantic constraints therefore need to check for all possible alternative ways of solving problems correctly; this is a complex task, and is in principle similar to the requirement for model-tracing tutors to be able to generate all correct solutions for a problem.

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 a single domain principle. 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.

5 What type of instructional tasks can CBM be used for?

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. In this ITS, the student needs to design a high-level database schema using the Enhanced Entity-Relationship (EER) data model (Elmasri and Navathe 2006) 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 also needs to use their general knowledge in order to design the EER diagram. There is no algorithm to use to identify

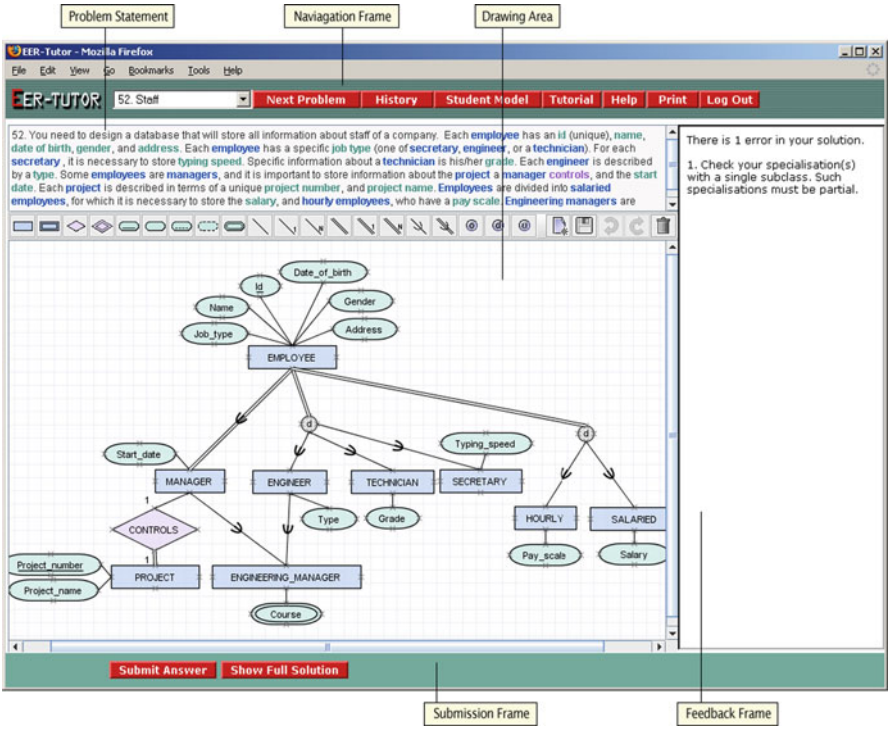

Fig. 3 A screenshot of EER-tutor

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. A screenshot of EER-Tutor is given in Fig. 3, showing the various parts of the system's interface.

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). Please see Sect. 7.5 for more information about this tutor. We also developed J-LATTE, a constraint-based tutor for novices learning Java (Holland et al. 2009). When learning programming, the student faces a lot of complexity: it is necessary to learn the syntax of the language, develop the design of the overall program as well as think about the low-level detail. J-LATTE supports two modes: *concept* mode, in which the student designs the program without having to specify contents of statements, and *coding* mode, in which the student completes the code of statements. In the concept mode, the student designs

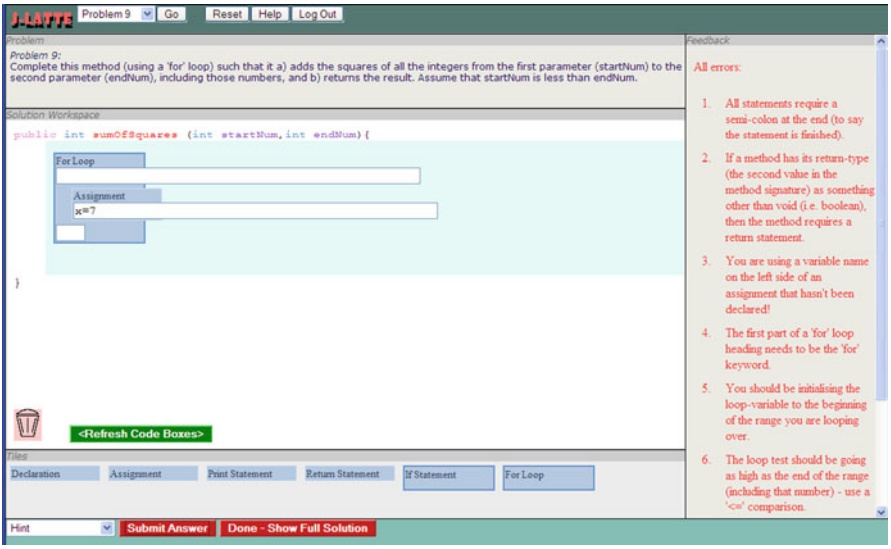

Fig. 4 The interface of J-LATTE

the overall structure of the code by selecting the tiles corresponding to the needed statements and placing them on the workspace. At any time, the student can switch to the coding mode, by completing the code within the tiles. The screenshot of the system shown in Fig. 4 shows a state where the student placed an assignment tile within the FOR loop tile, specified the code for the assignment. J-LATTE analyzes the student's solution and provides feedback both on the design and the actual code specified by the student.

In all of the discussed 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 (Mitrovic and Weerasinghe 2009). 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. Figure 5 shows a screenshot from the system, showing the step in which the student is asked to identify all candidate keys for the given table. The decision to force the student to complete a step before attempting the next step is deliberate, 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

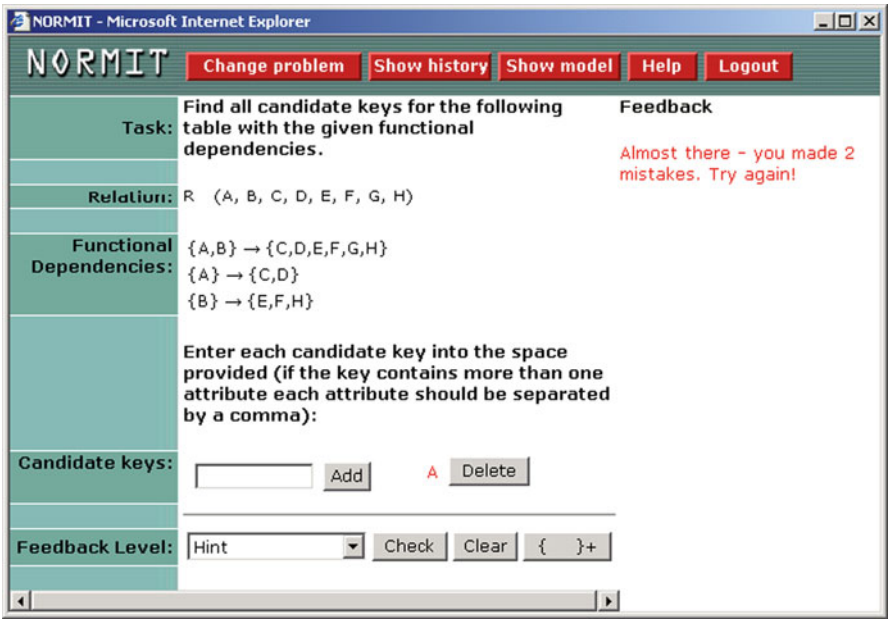

Fig. 5 A screenshot from NORMIT

previous steps. We refer to such constraints as the path constraints.[3] 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.

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, J-LATTE) 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 ill-defined tasks, such as essay writing or artistic tasks. Please note that model tracing effectively requires a problem solver (i.e. a set of production rules to solve the problem). CBM does not require a problem solver, as solution diagnosis is done by comparing the student's solution to the ideal solution using constraints. In general it is hard (or even impossible) to develop problem solvers for ill-defined tasks, exactly because the algorithms for solving problems of that type do not exist. The difference between the two methodologies basically comes from the nature of production rules (generative) and constraint (evaluative). Please see (Mitrovic and Weerasinghe 2009)

[3] Please note that path constraints allow for similar interventions as in model-tracing tutors.

for a fuller discussion of various types of tasks and the comparison between model tracing and CBM.

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 many 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, such as visualizing the goal structure, reducing the working memory load, providing declarative knowledge in the form of dictionaries or on-demand help etc. (Anderson et al. 1995; Woolf 2009), can also be used in ill-defined domains. 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.

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 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 Fig. 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? We discuss this question in Sect. 7.

6 Student modeling in constraint-based tutors

The very first paper on CBM (Ohlsson 1992) focused on short-term student modeling, or the diagnosis of the current solution state. 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. Please note that this process can be applied incrementally, to isolated actions as the student is performing them, or to the whole solution. This process corresponds to what VanLehn (2006) calls the *inner loop*, which executes once for each step the student has made; the system diagnosis the step, provides feedback and help on it, and updates the student model. Some constraint-based tutors do track every step the student makes (e.g. NORMIT), while others only analyze the student's solution on demand, when the student requests feedback, like SQL-Tutor.

Therefore the short-term student model in constraint-based tutors 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. We discuss feedback generation further in Sect. 7.1.

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. This kind of pedagogical planning is what VanLehn refers to as the *outer loop* (VanLehn 2006). We therefore extended CBM by proposing several different ways of representing the long-term model of the student's knowledge. Section 7.2 discusses how the long-term student model is used for problem selection.

In our early work (Mitrovic 1998a,b,c), the long-term model of the student's knowledge was represented in terms of the overlay model (Holt et al. 1994). 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 is done in many existing tutoring systems. Bayesian networks are based on a solid theoretical foundation, and provide ways of reasoning from observable actions (such as student answers) to non-observable ones, such as student skills and knowledge of domain concepts. Please see (Desmarais and Baker 2012) in this issue for an excellent overview of research in this area.

We developed a probabilistic model for SQL-Tutor (Mayo and Mitrovic 2000), which is used to make decisions about the next problem to be given to the student. For each problem, there is a simple Bayesian network which makes predictions about student performance on constraints relevant for that problem. These multiple predictions are then combined heuristically to give an overall measure of the value of the problem for a particular student. The value of a problem depends on the predicted number of violated constraints. If the system poses a problem that is too difficult, there will be many feedback messages coming from various violated constraints, and it is unlikely that the student will be able to cope with them all. If the problem is too easy, there will be no feedback messages, as all constraints will be satisfied. A problem of appropriate complexity will generate an optimal number of feedback messages. The results of an evaluation study showed that such adaptive, probabilistic problem selection was superior to the default problem-selection strategy (discussed in Sect. 7.2). The experimental group completed problems selected by the system on the basis of the probabilistic model in significantly lower number of attempts compared to the control group (Mayo and Mitrovic 2000, 2001).

Another constraint-based tutor that uses probabilistic models and decision theory is CAPIT, which teaches elementary school children about punctuation and capitalization rules in English (Mayo and Mitrovic 2001). We developed a data-centric methodology, in which the structure of the Bayesian network is induced from the actual student data. The Bayesian model predicts student's performance on a particular problem. We then defined utility functions to select the most effective feedback to be given to the student, and the best next problem for the student to work on. The classroom evaluation showed that the decision-theoretic pedagogical strategies result in students acquiring constraints at a faster rate than an equivalent tutor with random, rather than decision-theoretic, strategies. Both groups also performed better on a post-test than another class that did not use the tutor at all (Mayo and Mitrovic 2001).

7 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 suitable for supporting problem solving. In this section, we discuss how constraint models can be used to provide feedback to students, select problems, support students' meta-cognitive skills and collaborative learning.

7.1 Feedback in constraint-based tutors

Constraints augment the student's declarative knowledge. If the student cannot detect errors on their own, 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), on demand (when the student requests feedback), or in a delayed fashion, after the student is done with a problem or at the end of the session. The choice of the timing of feedback is therefore flexible.

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.

7.1.1 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.

7.1.2 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. On the following submission, the feedback is provided on the next level provided (*error flag*) and 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 a single clause that was not right in the student's solution (as identified by the error flag level), 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. This default sequence of feedback level progression can easily

be modified. The student, however, can ask for higher-level feedback whenever he/she desires.

It is important to note that this is the typical behaviour of constraint-based tutors: the amount of feedback can be easily changed by implementing a particular pedagogical strategy. For example, in some studies we have restricted students to only the initial three levels of feedback. In other studies, we prevented students from seeing the full solution unless they made at least three attempts at solving the current problem. CBM is completely flexible in this respect.

7.1.3 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 from which CBM was derived. The theory says that effective feedback should tell the student where 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.

To investigate whether such *theory-based* feedback is better than intuitive feedback, 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 theory-based feedback is more effective in supporting learning than intuitive one, as it results in an increased learning rate (Zakharov et al. 2005).

7.1.4 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 (e.g., Brown and Burton 1978; Sleeman et al. 1989; Spohrer et al. 1985; Anderson et al. 1990, 1995; Koedinger et al. 1997; VanLehn 2006). However, human teachers very often provide *positive feedback*, i.e. feedback on correct actions. Several recent studies of human tutoring show that human tutors use positive feedback more often than negative feedback (Ohlsson et al. 2007; Boyer et al. 2008; Cade et al. 2008; Di Eugenio et al. 2009).

Positive feedback is useful as it confirms tentative actions and supports students in strengthening their knowledge. This type of feedback also helps the student to inte-

grate 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 (Barrow et al. 2008).

7.1.5 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).

7.1.6 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).

7.2 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,b,c) 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[4] (based on the student model) which exercised the chosen constraints.

In a later version of SQL-Tutor, we used decision theory and the probabilistic student model to select the best problem for the student, discussed previously in Sect. 6. 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).

We experimented with computing the problem difficulty dynamically, in relation to the student model. The difficulty of a particular problem is computed as the weighted sum of probability of the student having already learned each constraint relevant for that problem. The weight of a constraint is number between 0 and 1, which represents the relative complexity of the constraint in relation to the whole constraint set (computed as the total number of tests the constraint contains divided by the number of tests in the most complex constraint in the constraint set). The dynamically computed problem difficulty performed well for a wide range of students, whereas static problem complexity performed well for students of intermediate ability, but fared badly for beginners and advanced students (Mitrovic and Martin 2004). Finally, we also introduced problem templates and presented them to students, during problem selection (Mathews and Mitrovic 2007).

7.3 Supporting self-assessment with open student models

Student models provide the basis for adaptive instruction, but students are very often not aware of their existence. By opening the student model, the system becomes more user-friendly. More importantly, the open student model (OSM) serves as a learning tool on its own; by showing the system's understanding of the student's knowledge, the student is encouraged to reflect on his/her knowledge. Czarkowski et al. (2005) show that students are capable of scrutinising their models in order to explore the adaptive nature of educational systems, and are able to understand and control the adaptation. Open student models engage students in thinking about their own knowledge, thus involving the student at the meta-cognitive level. Several studies have shown that OSMs raise students' awareness of their current knowledge levels and encourage them to reflect on the learning processes (Bull and Hghiem 2002; Bull 2004; Kay 1997; Mabbott and Bull 2007).

OSMs have been used in a variety of systems, ranging from text editors (Kay 1997) to ITSs (e.g. Aleven and Koedinger 2000). In some cases, developing the open student model is the primary activity the student is involved with (e.g. Dimitrova 2003). The actual representations used for OSM range from simple skill meters, represented as progress bars indicating the percentage of material a student has learned for a particular topic or concept (Aleven and Koedinger 2000; Bull 2004), to more complex representations. A concept/topic hierarchy is a tree structure built on the basis of conceptual relationships in a domain (Cook and Kay 1994; Mabbott and Bull 2006, 2007).

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

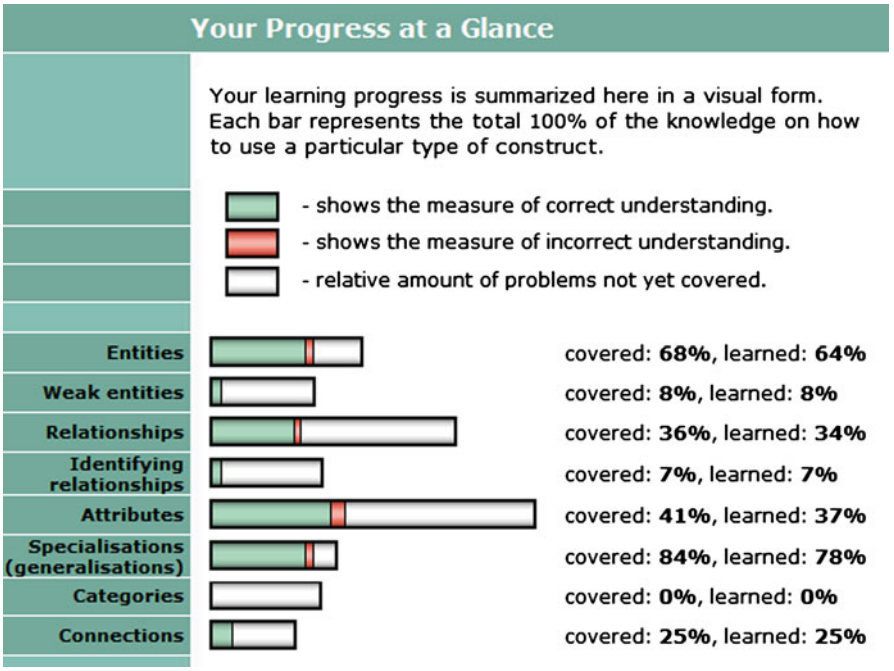

Fig. 6 The OSM from EER-tutor

Concept maps have also been used for OSM (Dimitrova 2003; Perez-Martin et al. 2007), but they are more complicated and may be difficult to design and understand (Mabbott and Bull 2006).

We explored the effect of open student models on students' 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 do improve their self-assessment skills when having access to relatively simple open student models (OSM), in the form of skill meters. Although the student model was presented in a highly abstract form, it helped students reflect on their knowledge and select more appropriate problems for further work.

In addition to the initial studies with SQL-Tutor, we conducted a number of studies involving open student models (Hartley and Mitrovic 2002) and negotiable student models (Thomson and Mitrovic 2010) in EER-Tutor, with similar results: students report being interested in examining their models, and believe that OSMs are helpful for reflecting on their knowledge. OSMs help students be better at self-assessment and also positively impact students' motivation to persist with learning. Figure 6 illustrates an open student model represented as a set of skill-meters from EER-Tutor. In a recent study, we experimented with multiple representations of OSM, ranging from simple skill-meters, over concept hierarchies to more complicated and detailed concept maps (Duan et al. 2010). The results of this initial study are encouraging, and show that different visualizations are preferred by groups of students based on their expertise in the domain. Novices preferred skill-meters, while more experienced

students preferred concept hierarchies, noting that such visualization helped understand the domain structure. More research is needed in this area, however, to explore the effectiveness of various visualization styles. We also plan to do more research on visualizing the progress of learning over time.

7.4 Tutorial dialogs and CBM

Tutorial dialogues are considered as one of the critical factors contributing to the effectiveness of human one-on-one tutoring. Dialogues provide opportunities for students to reflect on their existing knowledge and to construct new knowledge. Why2-Atlas and Auto Tutor (Graesser et al. 2001) use dialogues as the main activity to help students learn the domain knowledge. CIRCSIM-Tutor (Millis et al. 2004; Woo et al. 2005) is a natural-language tutor that helps students learn cardiovascular physiology related to regulation of blood pressure. The Geometry Explanation Tutor (Aleven et al. 2004) uses tutorial dialogs in addition to problem solving, and requires students to justify the problem-solving steps in their own words.

In a series of studies with our database design tutors, we evaluated the tutorial dialogs which elicitated explanations from students in situations when students make mistakes. Such explanations enable students to relate their problem-solving skills to declarative knowledge (Mitrovic 2005). Initial work concentrated on supporting dialogs via an error hierarchy, which classifies violated constraints into groups of similar situations (Weerasinghe and Mitrovic 2006). At the lowest level of the hierarchy, an error type is associated with one or more violated constraints, which form leaves of the hierarchy. The error types are then grouped into higher-level categories. Remediation is facilitated through tutorial dialogues, one of which is developed for each error type (Fig. 7). Initial studies have shown the effectiveness of such tutorial dialogs.

Recently we have extended the work to provide adaptive tutorial dialogs (Weerasinghe et al. 2008, 2009; Weerasinghe and Mitrovic 2010). The dialogs are selected adaptively, on the basis of the student model, by identifying the concepts that the student had most difficulties with, and then selecting the tutorial dialogs corresponding to those concepts. Additionally, there are adaptation rules which individualize the dialogs to suit the student's knowledge, in terms of the length of the dialog and the exact content of the dialog. In response to the generated dialog, learners are able to provide answers by selecting the correct option from a list provided by the tutor. In a study conducted in March 2010, the students who had adaptive dialogs outperformed their peers who only received non-adaptive dialogs, with the effect size[5] based of 0.69 after approximately 100 min of interaction with the system (Weerasinghe and Mitrovic 2010). The obtained effect size is remarkable because the only difference between the two groups was the adaptivity of the dialogues.

[5] Effect size(Cohen's d) indicates how much more the experimental group has learnt compared to the control group. It is calculated as the difference between the gains (improvement in scores between the pre- and post-test) of the experimental and control group divided by the standard deviation for all participants.

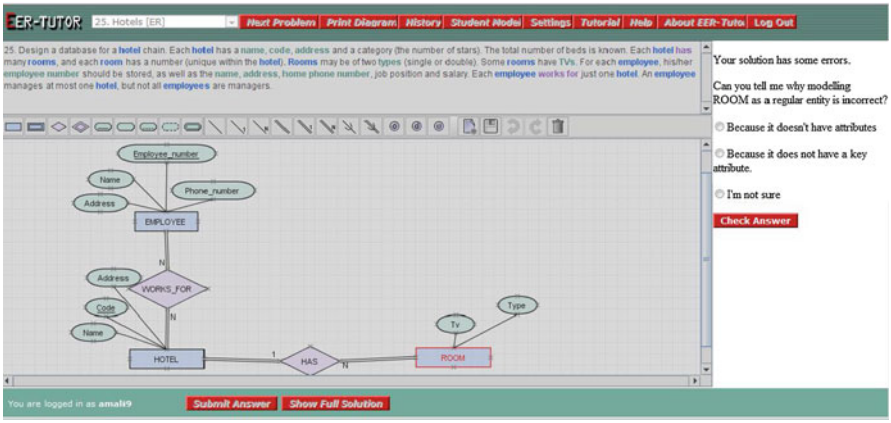

Fig. 7 Tutorial dialog within EER-tutor

7.5 Supporting collaborative learning

A lot of research has been done in the last fifteen years on computer-supported collaborative learning. Advantages of collaborative learning over individualised learning have been identified (Inaba and Mizoguchi 2004). Some particular benefits of collaborative problem-solving include: encouraging students to verbalise their thinking; encouraging students to work together, ask questions, explain and justify their opinions; increasing students' responsibility for their own learning; increasing the possibility of students solving or examining problems in a variety of ways; and encouraging them to elaborate and reflect upon their knowledge (Soller 2001; Webb et al. 1995). These benefits, however, are only achieved by active and well-functioning learning teams (Jarboe 1996). Different strategies for computationally supporting online collaborative learning have been proposed and used, but more studies are needed to examine the utility of these techniques (Jerman et al. 2001).

Several systems for collaborative learning have been developed (e.g. McManus and Aiken 1995; Barros and Verdejo 2000; Ogata et al. 2000; Rosatelli et al. 2000; Soller and Lesgold 2000; Constantino-Gonzalez et al. 2003), but much more research on supporting peer-to-peer interaction in CSCL systems is needed.

Constraints can also be used to represent collaborative skills. COLECT-UML supports teams of students developing UML class diagrams collaboratively (Baghaei et al. 2006). Figure 8 illustrates the system's interface, which allows the student to select problems, observe the individual or group models, ask for feedback and solutions to problems. Students construct their individual solutions in the private workspace (right), and use the shared workspace (left) to collaborate while communicating via the chat window (bottom). The system supports synchronous collaboration. The *Group Members* panel shows the team-mates already connected. The students start collaborating by introducing each other, and discussing the process. Whenever a new problem is attempted, each student is supposed to spend at least 10 min working on it individually (during this time, the group workspace is disabled). Only one student, the

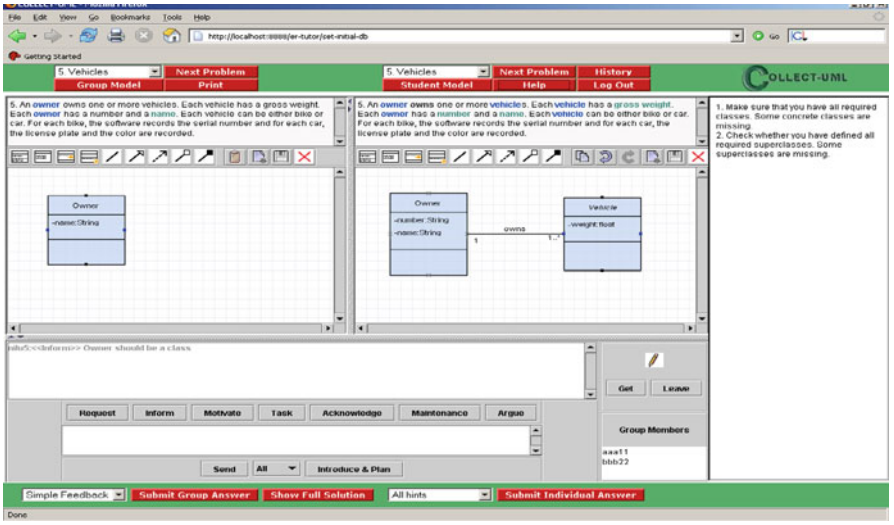

Fig. 8 Supporting collaboration in COLLECT-UML

one who has the pen, can update the shared workspace at a given time. The system allows students to copy and paste parts of their individual solutions to the group workspace. The chat area enables students to express their opinions using one of the communication categories. When a button is selected, the student has the option of annotating his/her selection with a justification. The contents of selected communication categories are displayed in the chat area along with any optional justifications. The students need to select one of the communication categories before being able to express their opinions. While all group members can contribute to the chat area and the group solution, only one member of the group (the group moderator) can submit the group solution (by clicking on the *Submit Group Answer* button). The system provides feedback on the individual solutions, as well as on group solutions and collaboration. All feedback messages appear in the frame located on the right-hand side of the interface.

Domain-level feedback is provided based on the analysis of individual and group solutions. In addition, the system also provides feedback on collaboration by analyzing each student's participation in group activities. The model of ideal collaboration is represented in the form of a set of meta-constraints. The collaboration-based feedback is given to individual students based on the initial planning of the problem, content of the chat area, the student's contributions to the shared diagram and the differences between student's individual solution and the group solution being constructed. The evaluation study performed with COLLECT-UML showed that in addition to improved problem-solving skills, the participants both acquired declarative knowledge about effective collaboration and did collaborate more effectively. The participants have enjoyed working with the system and found it a valuable asset to their learning (Baghaei et al. 2007).

7.6 Affect and CBM

Research shows that emotions play an important role in learning; stress, anxiety, and frustration experienced by a learner can severely degrade learning outcomes (Goleman 1995; Klein et al. 2002; Picard 1997, 2010; Forgas 2008). Human tutors are capable of identifying and responding to the affective states of their students; therefore, for ITSs to be truly affective, they should also be capable of tracking and appropriately responding to the emotional state of their users (Kort and Reilly 2002; Woolf 2009). Recognizing the student's affective state and responding to it actively has attracted the attention of many researchers in the last decade. For example, recent studies with AutoTutor explore strategies to address boredom, frustration, flow and confusion (D'Mello et al. 2007; D'Mello and Graesser 2010). AutoTutor detects affective states through conversational cues, posture and facial features. A study with Wayang Outpost (Arroyo et al. 2009) shows that students' emotional states can be inferred from physiological data collected via sensors, and those states are related to students' longer-term affective variables, such as self-concept. Data collected from sensors combined with the tutor data can also be used to predict pedagogically-important states, and thus allow the educational system to intervene (Muldner et al. 2009).

We developed an animated pedagogical agent for EER-Tutor (Fig. 9), with a mentor-like persona, which is capable of identifying and responding to the student's affective state (Zakharov et al. 2007). Using the dimensional approach to affective modeling, we track the users' affective state along the valence dimension. 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. The agent's response to the student's action depends on the student's cognitive state (as determined from the session history) as well as on the student's affective state.

The agent's persona is guided by a set of fifteen rules which implicitly encode the logic of session history appraisal. The rules assume that continuous lack of progress will be accompanied by a negative affective state, because the user will be dissatisfied with the progress of the current task; conversely, a satisfactory progress will result in a positive affective state. Each rule corresponds to a pedagogically-significant session state which requires the agent's response. For example there are rules requiring the agent to greet users when they sign on, submit a solution, ask for a new problem and so on. Each rule has a set of equivalent feedback messages determining the agent's verbal response; in addition, each rule includes a numeric value which triggers a change in the agent's affective appearance. For example, when the user reaches correct solution, along with a congratulatory message the agent responds with a cheerful smile. On the other hand, when the user is struggling with the solution resulting in multiple submissions with errors, the agent's verbal response consists of the list of errors, along with an affective facial expression—the agent's face looks sad as if the agent is empathizing with the user. The agent has its own affective module, which stores the current affective state; in the course of a session, the agent's affective state may be affected by the session events, but in the absence of the affect-triggering changes, the agent's affective state always gravitates towards the neutral state, as it is the case with human emotions. The experimental study of the agent (Zakharov et al. 2008) showed the

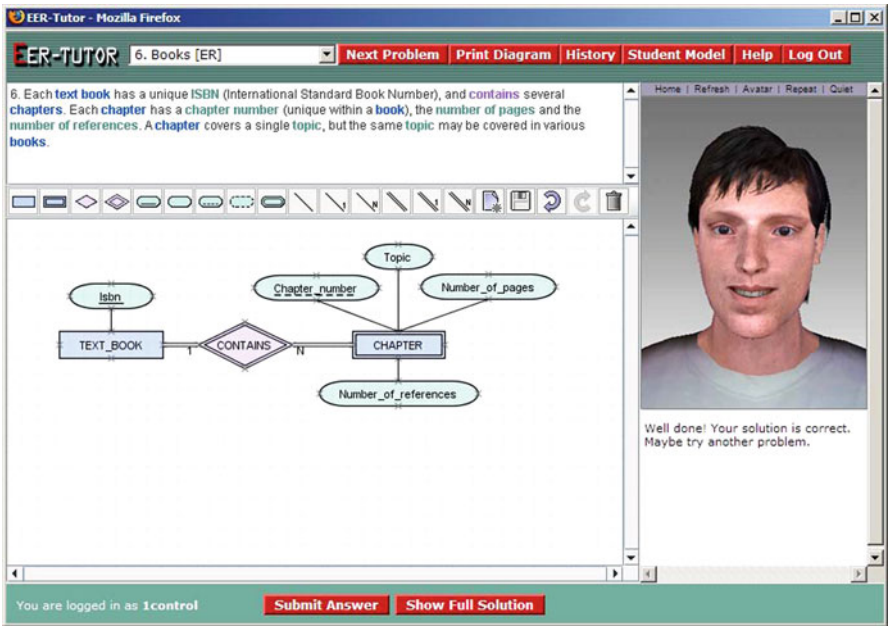

Fig. 9 The agent after a few consecutive incorrect submissions

general preference towards the affective agent over the non-affective agent. We are currently working on a different type of a persona for the agent, which will address the motivation of the student. Motivational states are influenced by cognitive, meta-cognitive and affective states of the student (du Boulay et al. 2010). Diagnosing the motivational state of the student, and being able to increase it is one of the important future research directions.

8 Authoring support for constraint-based tutors

Although ITSs promise to revolutionize education, due to their high effectiveness in supporting learning (e.g., Koedinger et al. 1997; Mitrovic and Ohlsson 1999; VanLehn et al. 2005), they are still not common in classrooms because their development requires extensive expertise, effort and time (Murray 1997, 2003). Murray (1999, 2003) classifies ITS authoring tools into two main groups: pedagogy-oriented and performance-oriented. Pedagogy-oriented systems focus on instructional planning and teaching strategies, assuming that the instructional content will be fairly simple (e.g., a set of instructional units containing canned text and graphics). Such systems provide support for curriculum sequencing and planning, authoring tutoring strategies, composing multiple knowledge-types (e.g., facts, concepts and procedures) and authoring adaptive hypermedia (Brusilovsky et al. 1996; Major et al. 1997; Shute 1998).

On the other hand, performance-oriented systems focus on providing rich learning environments where students learn by solving problems and receiving dynamic

feedback. The systems in this category include authoring systems for domain expert systems, simulation-based learning and some special purpose authoring systems focussing on performance. Authoring systems of this kind focus on generating the domain model. They typically use sophisticated machine learning techniques for acquiring domain rules with the assistance of a domain expert. Diligent (Angros et al. 2002) is an authoring system that acquires the knowledge required for a pedagogical agent in simulation-based learning environments. Disciple (Tecuci 1998) is a shell for developing intelligent educational agents. A domain expert teaches the agent to perform domain-specific tasks by providing examples and explanations. The expert is also required to supervise and correct the agent's behaviour. Disciple uses a collection of complementary learning methods, including inductive learning from examples, explanation-based learning, learning by analogy and learning by experimentation. The task of customising the Disciple agent requires extensive programming skills and considerable effort: the authors must develop a problem-solving interface for the domain and a problem solver. Demonstr8 (Blessing 1997) is an authoring system that assists in the development of model-tracing tutors for arithmetic domains. The system infers production rules using programming-by-demonstration techniques, coupled with methods to further abstract the generated productions. The Cognitive Tutor Authoring Tools (CTAT) (Aleven et al., 2006) also assist the creation and delivery of model-tracing tutors. CTAT allows authors to create two types of tutors: Cognitive tutors and Example-Tracing Tutors (previously called Pseudo Tutors). Cognitive tutors contain a model capable of tracing student actions while solving problems. In contrast, an example-tracing tutor contains a fixed trace from solving one particular problem. Matsuda (2007) report on SimStudent, a machine-learning approach to generating production rules within CTAT. In SimStudent, the author demonstrates how a particular problem can be solved by performing a sequence of problem-solving steps. Each step is represented in terms of the focus of attention (i.e., the interface components being modified), the value(s) entered by the author, and the skill demonstrated. SimStudent then generates a production rule for each problem-solving step. This approach has been used to generate rules for arithmetic, algebra and chemistry, resulting in fairly good domain models. However, this approach cannot be used for non-procedural tasks.

We have also devoted a lot of effort to providing authoring support. 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. WETAS provides the following functions: problem selection, answer evaluation, student modelling, feedback, and the user interface. In order to develop an ITS in WETAS, the author needs to provide the domain-dependent components, namely the structure of the domain, the domain model (in the form of constraints), the problem/solution set, the scaffolding information (if any), and possibly an input parser, if any specific preprocessing of the input is required. However, the development of a constraint set is still a demanding task.

We therefore investigated how the author can be supported in the process of developing constraints. Instead of asking the author to write constraints directly, we focused on the domain ontology. We proposed a three-phase approach: building a domain ontology, acquiring syntactic constraints automatically from the ontology, and engaging

the author in a dialog, in order to induce semantic constraints using machine learning techniques. A domain ontology contains a lot of knowledge about the domain, but is arguably easier to create than the final domain model. At the start, we performed a study which confirmed the hypothesis that domain ontologies are useful for reflecting on the instructional domain, so would be of great importance for building constraint sets manually (Suraweera et al. 2004).

We implemented the Constraint Acquisition System (CAS), which automatically acquires the required knowledge for ITSs by learning from examples (Suraweera et al. 2005, 2010). The knowledge acquisition process starts by the author describing the domain in terms of an ontology. Secondly, syntax constraints are automatically generated by analysing the ontology. Semantic constraints are generated in the third phase from problems and solutions provided by the author. Finally, the generated constraints are validated with the assistance of the author. The evaluation of CAS showed that the system was capable of generating most of the necessary constraints, and that the system was usable by novice authors, but that the ontology development is seen as a difficult task by the authors (Suraweera et al. 2007).

The constraint generation algorithms developed for CAS were later incorporated into ASPIRE, an authoring and deployment environment for constraint-based tutors. ASPIRE consists of the authoring server and the tutoring server (Mitrovic et al. 2006). The authoring process is similar to that in CAS, with the author developing a domain ontology, providing the structure for problems and solutions, and specifying examples of problems with their solutions, from which ASPIRE generates constraints. For certain more demanding domains, ASPIRE may not be able to generate all constraints, but the initial evaluations we performed shows that it is capable of generating the majority of necessary constraints (of the order of 90%) (Mitrovic et al. 2009). 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). ASPIRE generates text-based interfaces on the basis of the problem/solution structures and the domain ontology automatically; however, the default interface can be replaced with a custom-made applet. Figure 10, for example, shows the applet we developed for a thermodynamics tutor (Mitrovic et al. 2011). The tutoring server supports deployment of ITSs, creation and management of user accounts and other administrative services.

Many tutors have been developed in ASPIRE, both by members of ICTG and other researchers all over the world. The instructional tasks supported by those tutors include simple mathematical tasks (adding fractions, solving equations etc.), teaching data structures and computer science algorithms, solving thermodynamics and mechanics problems, and learning how to make decisions about managing oil palm plantations (Amalathas et al. 2010). Some of ITSs developed in ASPIRE have been evaluated in regular courses. An example is CIT, a constraint-based tutor that teaches students how to make decisions on capital investments (Mitrovic et al. 2008).

ASPIRE is a general, domain-independent system. Our experience shows that domain authors who do not have technical backgrounds find the ontology development very challenging (Mitrovic et al. 2009). In recent work, we developed VIPER (Martin et al. 2009), a new layer on top of ASPIRE, which further simplifies the authoring process by focusing on instructional domains with specific features, thus

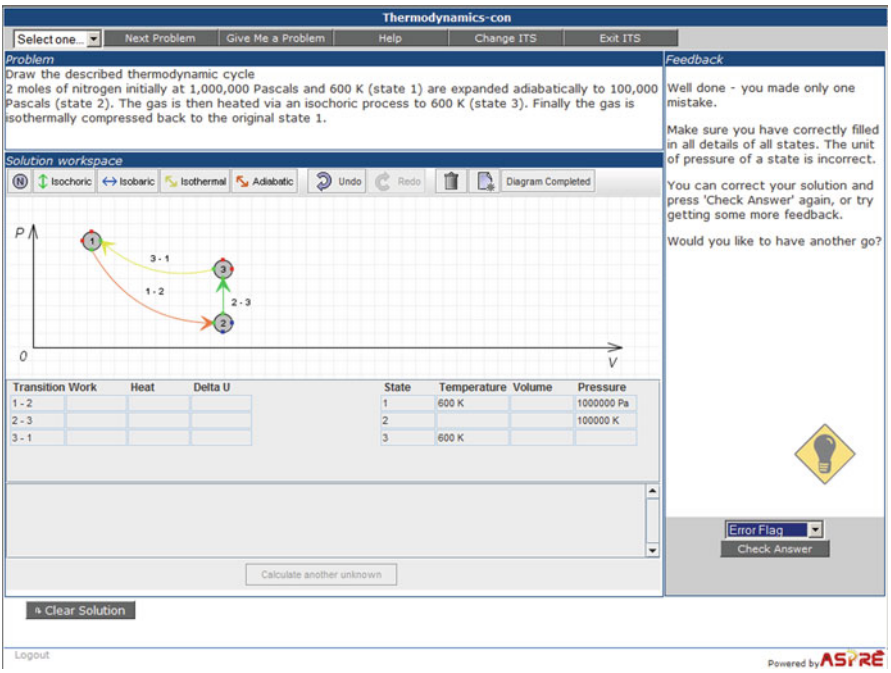

Fig. 10 A screenshot from the thermo-tutor developed in ASPIRE

making the author's task easier. VIPER supports authoring of domains in which the instructional tasks are closely related to images, and require the student to analyze an image, identify objects within the image, or critique/compare two or more images. For such domains, it is possible to specify domain templates within the authoring system. A domain template specifies the high-level domain ontology, where any domain of the given type shares the same basic domain model structure, diagnostic logic and interface type(s). For the medical imaging project we created three domain types, each of which may have one or more interface associated with them. The image analysis type allows comparing of images, or changing the parameters of images, and observing the effects. The image contents type supports tasks such as identifying features within an image, or labelling images. Finally, the question answering template supports multiple choice and fill-in-the-blanks questions.

To create a tutor in VIPER, the author selects a template, specifies the content of the domain type by using a form-based interface, in which s/he specifies concepts and the related subconcepts, features of concepts and their values, and feedback messages to be given to students when features are incorrectly used by students. VIPER then converts those domain models into constraints automatically. The author then needs to provide problems, accompanying images, and solutions, and the tutoring system can be served to students.

VIPER is easier to use in comparison to ASPIRE, but is restricted to instructional tasks dealing with images. Applicable domains include medical diagnosis, art critiquing and architecture and similar domains, in which a student is expected to identify

parts of something that can be presented pictorially, such as electronic equipment, vehicle repairs, paleontology (find the bones) and even less obvious domains such as software design.

9 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[6] 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. Constraints can capture whatever is known about the ill-defined domain and the problem specification, thus being 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 development effort than model-tracing ones.

We used constraints to represent domain knowledge, student knowledge, and knowledge about good collaboration. The constraint models support various pedagogical actions, such as providing feedback, selecting problems, support the improvement of meta-cognitive and collaboration skills. Most of our work has concentrated on problem solving, but in current projects we are expanding the constraint-based tutors to support multiple types of learning (Mathews and Mitrovic 2009), such as learning by debugging incorrect solutions, learning from reflection and direct instruction.

CBM is not used at ICTG only, but also by other researchers, in lots of different domains, such as utterance analysis in case-based collaborative learning (Rosatelli and Self 2004), electronics (Billingsley et al. 2004), discrete mathematics (Billingsley and Robinson 2005), frameworks for ITSs (Riccucci et al. 2005; Siddappa and Manjunath 2008), teaching algorithms (Petry and Rosatelli 2006), UML class diagrams (Le 2006), language learning (Menzel 2006), Newtonian physics (Mills and Dalgarno 2007), programming in Prolog (Le et al. 2009), architectural design (Oh et al. 2009), adaptive testing (Galvez et al. 2009a), object-oriented programming (Galvez et al. 2009b) and inquiry learning (Roll et al. 2010). In some of these projects, CBM was used in combination with other modeling approaches, such as model tracing (Roll et al. 2010) and item response theory (Galvez et al. 2009a).

CBM is a flexible and powerful approach for building ITSs. Many questions have been answered, but many more are still open. Our future plans include developing new constraint-based tutors for challenging tasks; e.g., a new project we are embarking on

[6] www.databaseplace.com.

involves using CBM to provide adaptive computer-based cognitive training for post-stroke rehabilitation. We also plan to perform further research on supporting multiple teaching strategies, collaboration, motivation and meta-cognitive skills, as well as further improving authoring support. We are looking forward to new challenges and exciting possibilities in the next 15 years of CBM!

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

References

Aleven, V., Koedinger, K.:Limitations of student control: do students know when they need help? In: Gauthier, G., Frasson, C., VanLehn, K. (eds.) Proceedings of ITS 2000, pp. 292–303 (2000)

Aleven, V., Ogan, A., Popescu, O., Torrey, C., Koedinger, K.: Evaluating the effectiveness of a tutorial dialogue system for self-explanation. In: Lester, J., Vicario, R.M., Paraguacu, F. (eds.) Proceedings of ITS2004, pp. 443–454 (2004)

Amalathas, S., Mitrovic, A., Saravanan, R., Evison, D. et al: Developing an intelligent tutoring system for palm oil in ASPIRE. In: Wong, S.L. (ed.) Proceedings of 18th International Conference on Computers in Education, pp. 101–103. APSCE, Putrajaya, Malaysia (2010)

Anderson, J.R., Boyle, C.F., Corbett, A.T., Lewis, M.W.: Cognitive modeling and intelligent tutoring. Artif. Intell. **42**, 7–49 (1990)

Anderson, J.R., Corbett, A.T., Koedinger, K.R., Pelletier, R.: Cognitive tutors: lessons learned. Learn. Sci. **4**(2), 167–207 (1995)

Angros, R., Johnson, W.L., Rickel, J., Scholer, A.: Learning domain knowledge for teaching procedural skills. In: Gini, M., Ishida, T., Castelfranchi, C., Johnson, W.L. (eds.) Autonomous Agents and Multi-agent Systems., pp. 1372–1378. ACM, New York (2002)

Arroyo, I., Cooper, D.G., Burleson, W., Woolf, B., Muldner, K., Christopherson, R.: Emotion sensors go to school. In: Dimitrova, V., Mizoguchi, R., du Boulay, B., Graesser, A. (eds.) Proceedings of 14th International Conference on Artificial Intelligence in Education, pp. 41–48 (2009)

Baghaei, N., Mitrovic, A., Irvin, W.: Problem-solving support in a constraint-based intelligent tutoring system for UML. Technol. Instr. Cogn. Learn. **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. Comput. Support. Collab. Learn. **2**(2–3), 159–190 (2007)

Barros, B., Verdejo, M.F.: Analysing student interaction processes in order to improve collaboration: the DEGREE approach. Artif. Intell. Educ. **11**, 221–241 (2000)

Barrow, D., Mitrovic, A., Ohlsson, S., Grimley, M. et al: Assessing the impact of positive feedback in constraint-based ITSs. In: Woolf, B. (ed.) Proceedings of 9th International Conference on ITS 2008, LCNS 5091, pp. 250–259. Springer-Verlag, Heidelberg (2008)

Billingsley, W., Robinson, P.: Towards an intelligent online book for discrete mathematics. In: Proceedings of International Conference Active Media Technology, pp. 291–296 (2005)

Billingsley, W., Robinson, P., Ashdown, M., Hanson, C.: Intelligent tutoring and supervised problem solving in the browser. In: Proceedings of International Conference on WWW/Internet, pp. 806–811 (2004)

Blessing, S.B.: Programming by demonstration authoring tool for model-tracing tutors. Artif. Intell. Educ. **8**, 233–261 (1997)

Boyer, K.E., Phillips, R., Wallis, M., Vouk, M., Lester, J.: Balancing cognitive and motivational scaffolding in tutorial dialogues. In: Woolf, B.P., Aïmeur, E., Nkambou, R., Lajoi, S. (eds.) Proceedings Intelligent Tutoring Systems, pp. 239–249. Springer-Verlag, Berlin (2008)

Brown, J.S., Burton, R.R.: Diagnostic models for procedural bugs in basic mathematical skills. Cogn. Sci. **2**, 155–192 (1978)

Brusilovsky, P., Schwartz, E., Weber, G.: A tool for developing adaptive electronic textbooks on WWW. In: Proceedings of WebNet-96, AACE, (1996)

Bull, S.: Supporting learning with open learner models. In: Proceedings of 4th Hellenic Conference in Information and Communication Technologies in Education, pp. 47–61 (2004)

Bull, S., Hghiem, T.: Helping learners to understand themselves with a learner model open to students, peers and instructions. In: Proceedings of International Conference on Intelligent Tutoring Systems, pp. 5–13 (2002)

Cade, W.L., Copeland, J.L., Person, N.K., D'Mello, S.: Dialogue modes in expert tutoring. In: Woolf, B.P., Aïmeur, E., Nkambou, R., Lajoie, S. (eds.) Proceedings on Intelligent Tutoring Systems, pp. 470–479. Springer-Verlag, New York, NY (2008)

Constantino-Gonzalez, M.A., Suthers, D., Escamilladelos Santos, J.: Coaching web-based collaborative learning based on problem solution differences and participation. Artif. Intell. Educ. **13**(2–4), 263–299 (2003)

Cook, R., Kay, J.: The Justified User Model. In: Proceedings on UM 1994, pp. 145–150 (1994)

Czarkowski, M., Kay, J., Potts, S.: Scrutability as a core interface element. In: Looi, C.-K., McCalla, G., Bredeweg, B., Breuker, J. (eds.) Proceedings of 12th International Conference on Artificial Intelligence in Education, pp. 783–785. IOS Press, Amsterdam (2005)

D'Mello, S., Graesser, A.: Multimodal semi-automated affect detection from conversational cues, gross body language, and facial features. User Model. User-Adapt. Interact. **20**(2), 147–187 (2010)

D'Mello, S., Picard, R., Graesser, A.: Toward an affect-sensitive autotutor. IEEE Intell. Syst. **22**, 53–61 (2007)

Desmarais, M., Baker, R.: Learner models' adapting to student skills and behavioral factors. User Model. User-Adapt. Interact. **22** (this issue) (2012)

Di Eugenio, B., Fossati, D., Ohlsson, S., Cosejo, D.: Towards explaining effective tutorial dialogues. In: Taatgen, N.A., van Rijn, H. (eds.) Proceedings of 31th Annual Conference of the Cognitive Science Society, pp. 1430–1435. Cognitive Science Society, Austin, TX (2009)

Dimitrova, V.: StyLE-OLM: interactive open learner modelling. Artif. Intell. Educ. **13**(1), 35–78 (2003)

du Boulay, B., Avramides, K., Luckin, R., Martinez-Miron, E., Rebolledo-Mendez, G., Carr, A.: Towards systems that care: a conceptual framework based on motivation, metacognition and affect. Artif. Intell. Educ., **20**(3), 197–229 (2010)

Duan, D., Mitrovic, A., Churcher, N. et al: Evaluating the effectiveness of multiple open student models in EER-tutor. In: Wong, S.L. (ed.) Proceedings of 18th International Conference on Computers in Education ICCE 2010, pp. 86–88. APSCE, Putrajaya (2010)

Elmasri, R., Navathe, S.B.: Fundamentals of Database Systems. Addison-Wesley, Reading (2006)

Forgas, J.P.: Affect and cognition. Perspect. Psychol. Sci. **3**, 94–101 (2008)

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.) Proceedings of 14th International Conference on Artificial Intelligence in Education, pp. 291–298 (2009a)

Galvez, J., Guzman, E., Conejo, R.: A blended e-learning experience in a course of object oriented programming fundamentals. Knowl. Based Syst. **22**(4), 279–286 (2009b)

Graesser, A.C., VanLehn, K., Rose, C.P., Jordan, P.W., Harter, D.: Intelligent tutoring systems with conversational dialogue. AI Mag. **22**(4), 39–51 (2001)

Goleman, D.: Emotional Intelligence. Bantam Books, New York (1995)

Hartley, D., Mitrovic, A.: Supporting learning by opening the student model. In: Cerri, S., Gouarderes, G., Paraguacu, F. (eds.) Proceedings of 6th Internationa Conference on Intelligent Tutoring Systems ITS 2002. LCNS, vol. 2363, pp. 453–462. Biarritz, France (2002)

Holland, J., Mitrovic, A., Martin, B. : A constraint-based tutor for Java. 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. (eds.) Proceedings of 17th International Conference on Computers in Education ICCE 2009, pp. 142–146. Asia-Pacific Society for Computers in Education, Hong Kong (2009)

Holt, P., Dubs, S., Jones, M., Greer, J.: The state of student modeling. In: Student Modeling: The Key to Individualized Knowledge-Based Instruction, pp. 3–39. Springer-Verlag, Heidelberg (1994)

Inaba, A., Mizoguchi, R.: Learners' roles and predictable educational benefits in collaborative learning; an ontological approach to support design and analysis of CSCL. In: Lester, J., Vicari, R.M., Paraguacu, F. (eds.) Proceedings of ITS 2004, pp. 285–294 (2004)

Jarboe, S.Procedures for enhancing group decision making. In: Hirokawa, B., Poole, M. Communication and Group Decision Making, pp. 345–383. Sage Publications, Thousand Oaks, CA (1996)

Jerman, P., Soller, A., Muhlenbrock, M.: From mirroring to guiding: a review of state of the art technology for supporting collaborative learning. In: Dillenbourg, P., Eurelings, A., Hakkarainen, K. (eds.) European Perspectives on CSCL (CSCL 2001), pp. 324–331 (2001)

Kay, J.: Learner know thyself: student models to give learner control and responsibility. In: Halim, Z., Ottomann, T., Razak, Z. (eds) Proceedings of International Conference on Computers in Education, pp. 17–24 (1997)

Klein, J., Moon, Y., Picard, R.: This computer responds to user frustration: theory, design, and results. Interact. Comput. **14**, 119–140 (2002)

Koedinger, K.R., Anderson, J.R., Hadley, W.H., Mark, M.A.: Intelligent tutoring goes to the big city. Artif. Intell. Educ. **8**, 30–43 (1997)

Kort, B., Reilly, R.: An affective module for an intelligent tutoring system. In: Cerri, S.A., Gouardéres, G., Paraguaçu, F. (eds.) Proceedings of ITS 2002. LNCS, vol. 2363, pp. 955–962. Springer, Heidelberg (2002)

Le, N.-T.: A constraint-based assessment approach for free-form design of class diagrams using UML. In: Ashley, K., Pinkwart, N., Lynch, C. (eds.) Proceedings of Workshop on Intelligent Tutoring Systems for Ill-Defined Domains, 8th International Conference on ITS, pp. 11–19 (2006)

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. (eds.) Proceedings of 17th International Conference on Computers in Education, pp. 51–58. APSCE, Putrajaya (2009)

Mabbott, A., Bull, S.: Student preferences for editing, persuading and negotiating the open learner model. IN: Proceedings of ITS 2006, pp. 481–490 (2006)

Mabbott, A., Bull, S. et al: Comparing student-constructed open learner model presentations to the domain. In: Luckin, R. (ed.) Proceedings Artificial Intelligence in Education, pp. 281–288. IOS Press, Chris (2007)

Major, N., Ainsworth, S., Wood, D.: REDEEM: exploiting symbiosis between psychology and authoring environments. Artif. Intell. Educ. **8**(3–4), 317–340 (1997)

Martin, B., Mitrovic, A.: Domain modeling: art or science? In: Hoppe, U., Verdejo, F., Kay, J. (eds.) Proceeindgs of 11th International Conference on Artificial Intelligence in Education AIED 2003, pp. 183–190. IOS Press, Amsterdam (2003)

Martin, B., Mitrovic, A.: The effect of adapting feedback generality in ITSs. In: Wade, V., Ashman, H., Smyth, B. (eds.) Proceedings of 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.) Proceedings of World Conferences E-Learning in Corporate, Government, Healthcare, and Higher Education, pp. 502–509. 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.) Proceedings of Artificial Intelligence in Education, pp. 611–613 (2007)

Mathews, M., Mitrovic, A.: Does framing a problem-solving scenario influence learning? In: Kong, S.C., Ogata, H., Arnseth, H.C., Chan, C.K., Hirashima, T., Klett, F., Lee, J.H.M., Liu, C.C., Looi, C.K. (eds.) Proceedings of 17th International Conference on Computers in Education ICCE 2009, pp. 27–34. Asia-Pacific Society for Computers in Education, Hong Kong (2009)

Matsuda, N., Cohen, W., Sewall, J., Lacerda, G., Koedinger, K.: Evaluating a simulated student using real students data for training and testing. In: Conati, C., McCoy, K., Paliouras, G. (eds.) User Modelling 2007, pp. 61–70. Springer, Berlin (2007)

Mayo, M., Mitrovic, A.: Using a probabilistic student model to control problem difficulty. In: Gauthier, G., Frasson, C., VanLehn, K. (eds.) Proceedings of Intelligent Tutoring Systems, pp. 524–533. Springer, Heidelberg (2000)

Mayo, M., Mitrovic, A.: Optimising ITS behaviour with bayesian networks and decision theory. Artif. Intell. Educ. **12**(2), 124–153 (2001)

McManus, M., Aiken, R.: Monitoring computer-based problem solving. Artif. Intell. Educ. **6**(4), 307–336 (1995)

Menzel, W.: Constraint-based modeling and ambiguity. Artif. Intell. Educ. **16**(1), 29–63 (2006)

Milik, N., Marshall, M., Mitrovic, A.: Teaching logical database design. In: Ikeda, M., Ashley, K., Chan, T.-W. (eds.) Proceedings of ITS 2006 ERM-Tutor. LNCS, vol. 4053, pp. 707–709. Springer, Heidelberg (2006)

Mills, C., Dalgarno, B.: A conceptual model for game-based intelligent tutoring systems. In ICT: providing choices for learners and learning. In: Proceedings of ASCILITE, pp. 692–701. Singapore (2007)

Millis, B., Evens, M., Freedman, R.: Implementing directed lines of reasoning in an intelligent tutoring system using the atlas planning environment. In: International Conference on Information Technology, pp. 729–733 (2004)
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. Proceedings of ED-MEDIA'98, pp. 1027–1032. AACE, Vancouver, CA (1998b)
Mitrovic, A.: Experiences in implementing constraint-based modeling in SQL-Tutor. In: Goettl, B., Halff, H., Redfield, C., Shute, V. (eds.), Proceedings of ITS'98, pp. 414–423 (1998c)
Mitrovic, A.: An intelligent SQL tutor on the web. Artif. Intell. Educ. **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.) Proceedings of Conference on Artificial Intelligence in Education, pp. 499–506 (2005)
Mitrovic, A., Martin, B.: Evaluating adaptive problem selection. In: De Bra, P., Nejdl, W. (eds.) Proceedings of Adaptive Hypermedia and Adaptive Web-Based Systems. LNCS, vol. 3137, pp. 185–194. Springer-Verlag, Heidelberg (2004)
Mitrovic, A., Martin, B.: Evaluating the effect of open student models on self-assessment. Artif. Intell. Educ. **17**(2), 121–144 (2007)
Mitrovic, A., Ohlsson, S.: Evaluation of a constraint-based tutor for a database language. Artif. Intell. Educ. **10**(3–4), 238–256 (1999)
Mitrovic, A., Weerasinghe, A.: Revisiting the ill-definedness and consequences for ITSs. In: Dimitrova, V., Mizoguchi, R., du Boulay, B., Graesser, A. (eds) Proceedings of 14th International Conference on Artificial Intelligence in Education, pp. 375–382 (2009)
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.) Proceedings of Conference on User Modeling, LNAI 2702 , pp. 313–322. Springer-Verlag, Heidelberg (2003)
Mitrovic, A., Suraweera, P., Martin, B., Weerasinghe, A.: DB-suite: experiences with three intelligent, web-based database tutors. J. Interact. Learn. Res. **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.) Proceedings of ITS 2006. LNCS, vol. 4053, pp. 41–50 (2006)
Mitrovic, A., Martin, B., Suraweera, P.: Intelligent tutors for all: constraint-based modeling methodology, systems and authoring. IEEE Intell. Syst. **22**(4), 38–45 (2007)
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: Proceedings of ED-MEDIA 2008, pp. 4607–4616 (2008)
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. Artif. Intell. Educ. **19**(2), 155–188 (2009)
Mitrovic, A., Williamson, C., Bebbington, A., Mathews, M., Suraweera, P., Martin, B., Thomson, D., Holland, J.: An Intelligent Tutoring System for Thermodynamics. EDUCON 2011 Amman, Jordan, pp. 378–385 (2011)
Muldner, K., Burleson, W., VanLehn, K: Yes!: using tutor and sensor data to predict moments of delight during instructional activities. In: DeBra, P., Kobsa, A., Chin, D. User modeling, adaptation and personalization. LCNS, vol. 6075, pp. 170–195. Springer, Heidelberg (2009)
Murray, T.: Expanding the knowledge acquisition bottleneck for intelligent tutoring systems. Artif. Intell. Educ. **8**(3), 222–232 (1997)
Murray, T.: Authoring intelligent tutoring systems: an analysis of the state of the art. Artif. Intell. Educ. **10**(1), 98–129 (1999)
Murray, T: An overview of intelligent tutoring system authoring tools: updated analysis of the state of the art. In: Murray, T., Blessing, S., Ainsworth, S. (eds.) Authoring tools for advanced technology learning environments, pp. 491–545. Kluwer Academic Publishers, Norwell, MA (2003)
Ogata, H., Matsuura, K., Yano, Y.: Active knowledge awareness map: visualizing learners activities in a web based CSCL environment. In: Procedings of International Workshop on New Technologies in Collaborative Learning, pp. 89–97 (2000)
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. (eds.) Proceedings of 17th International Conference on Computers in Education, pp. 19–26. APSCE, Chris (2009)
Ohlsson, S.: Constraint-based student modeling. Artif. Intell. Educ. **3**(4), 429–447 (1992)
Ohlsson, S.: Learning from performance errors. Psychol. Rev. **103**, 241–262 (1996)
Ohlsson, S., Bee, N.: Strategy variability: a challenge to models of procedural learning. In: Birnbaum, L. (ed.) Proceedings of International Confernce of the Learning Sciences, pp. 351–356. AACE, Vancouver, CA (1991)
Ohlsson, S., Mitrovic, A.: Fidelity and efficiency of knowledge representations for intelligent tutoring systems. Technol. Instr. Cogn. Learn. **5**(2), 101–132 (2007)
Ohlsson, S., Di Eugenio, B., Chow, B., Fossati, D., Lu, X., Kershaw, T.: Beyond the code-and-count analysis of tutoring dialogues. In: Luckin, R., Koedinger, K., Greer, J. (eds.) Artificial Intelligence in Education: Building Technology Rich Learning Contexts that Work, pp. 349–356. IOS Press, Amsterdam (2007)
Perez-Martin, D., Alfonseca, E., Rodriguez, P., Pascual-Nieto, I.: Automatic generation of students conceptual models from answers in plain text. In: Procedings of UM 2007, pp. 329–333. Springer-Verlag, Heidelberg (2007)
Petry, P.G., Rosatelli, M.: AlgoLC: a learning companion system for teaching and learning algorithms. In: Ikeda, M., Ashley, K., Chan, T.-W. (eds.) Proceedings of ITS 2006. LNCS, vol. 4053, pp. 775–777 (2006)
Picard, R.W.: Affective Computing. MIT Press, Cambridge, MA (1997)
Picard, R.W.: Toward computers that recognize and respond to user emotion. IBM Syst. J. **39**(3–4), 705–719 (2010)
Riccucci, S., Carbonaro, A., Casadei, G.: An architecture for knowledge management in intelligent tutoring systems. In: Proceedings of IADIS International Congress on Cognition and Exploratory Learning in Digital Age, pp. 473–476 (2005)
Roll, I., Aleven, V., Koedinger, K.: The invention lab: using a hybrid of model tracing and constraint-based modeling to offer intelligent support in inquiry environments. In: Aleven, V., Kay, J., Mostow, J. (eds.) ITS 2010, Part I. LNCS, vol. 6094, pp. 115–124. Springer-Verlag, Berlin, Heidelberg (2010)
Rosatelli, M., Self, J., Thirty, M.: LeCS: a collaborative case study system. In: Proceedings of 5th International Conference on Intelligent Tutoring Systems, pp. 242–251 (2000)
Rosatelli, M., Self, J.: A collaborative case study system for distance learning. Artif. Intell. Educ. **14**(1), 1–29 (2004)
Siddappa, M., Manjunath A.S.: Intelligent tutor generator for intelligent tutoring systems. In: Proceedings of World Congress on Engineering and Computer Science, pp. 578–583 (2008)
Shute, V.J.: DNA—Uncorking the bottleneck in knowledge elicitation and organization. In: Proceedings of ITS-98, pp. 146–155 (1998)
Sleeman, D., Kelly, A.E., Martinak, R., Ward, R.D., Moore, J.L.: Studies of diagnosis and remediation with high school algebra students. Cogn. Sci. **13**, 551–568 (1989)
Soller, A.: Supporting social interaction in an intelligent collaborative learning system. Artif. Intell. Educ. **12**, 40–62 (2001)
Soller, A., Lesgold, A.: Knowledge acquisition for adaptive collaborative learning environments. In: Proceedings of AAAI Fall Symposium: Learning How to Do Things, Cape Cod, MA (2000)
Spohrer, J.C., Soloway, E., Pope, E.: A goal/plan analysis of buggy pascal programs. Hum.-Comput. Interact. **1**, 163–205 (1985)
Suraweera, P., Mitrovic, A.: KERMIT: a constraint-based tutor for database modelling. In: Cerri, S.A., Gouarderes, G., Paraguacu, F. (eds.) Proceedings of 6th International Conference on Intelligent Tutoring Systems, pp. 376–387 (2002)
Suraweera, P., Mitrovic, A.: An intelligent tutoring system for entity relationship modelling. Artif. Intell. Educ. **14**(3–4), 375–417 (2004)
Suraweera, P., Mitrovic, A., Martin, B.: The role of domain ontology in knowledge acquisition for ITSs. In: Lester, J., Vicari, R.M., Paraguacu, F. (eds.) Proceedings of 7th Interenational Conference on Intelligent Tutoring Systems ITS 2004. LNCS, vol. 3220, pp. 207–216. Springer-Verlag, Maceio (2004)
Suraweera, P., Mitrovic, A., Martin, B.: A knowledge acquisition system for constraint-based intelligent tutoring systems. In: Looi, C.-K., McCalla, G., Bredeweg, B., Breuker, J. (eds.) Proceedings Artificial Intelligence in Education AIED 2005, pp. 638–645. IOS Press, Amsterdam (2005)
Suraweera, P., Mitrovic, A., Martin, B.: Constraint authoring system: an empirical evaluation. In: Luckin, R., Koedinger, K., Greer, J. (eds.) Proceedings 13th International Conference Artificial Intelligence in Education AIED 2007, Los Angeles, pp. 451–458 (2007)

Suraweera, P., Mitrovic, A., Martin, B.: Widening the knowledge acquisition bottleneck for constraint-based tutors. Artif. Intell. Educ. **20**(2), 137–173 (2010)
Tecuci, G.: Building Intelligent Agents: An Apprenticeship Multistrategy Learning Theory, Methodology, Tool and Case Studies. Morgan Kaufmann, San Francisco (1998)
Thomson, D., Mitrovic, A.: Preliminary evaluation of a negotiable student model in a constraint-based ITS. Res. Prac. Technol. Enhanc. Learn. **5**(1), 19–33 (2010)
VanLehn, K.: The behaviour of tutoring systems. Artif. Intell. Educ. **16**(3), 227–265 (2006)
VanLehn, K., Lynch, C., Schulze, K., Shapiro, J.A., Shelby, R., Taylor, L., Treacy, D., Weinstein, A., Wintersgill, M.: The andes physics tutoring system: lessons learned. Artif. Intell. Educ. **15**, 147–204 (2005)
Wang, T., Mitrovic, A.: Using neural networks to predict students' behaviour. In: Kinshuk, Lewis, R., Akahori, K., Kemp, R., Okamoto, T., Henderson, L., Lee, C.-H. (eds.) Proceedings of International Conference on Computers in Education, pp. 969–973 (2002)
Webb, N.M., Troper, J.D., Fall, R.: Constructive activity and learning in collaborative small groups. J. Educ. Psychol. **87**, 406–423 (1995)
Weerasinghe, A., Mitrovic, A.: Facilitating deep learning through self-explanation in an open-ended domain. Int. J. Knowl.-Based Intell. Eng. Syst. **10**(1), 3–19 (2006)
Weerasinghe, A., Mitrovic, A., Martin, B.: A preliminary study of a general model for supporting tutorial dialogues. In: Proceedings of International Conference on Computers in Education, pp. 125–132 (2008)
Weerasinghe, A., Mitrovic, A., Martin, B.: Towards individualized dialogue support for ill-defined domains. Artif. Intell. Educ. **19**(4), 357–379 (2009)
Weerasinghe, A., Mitrovic, A. et al: Evaluating the effectiveness of adaptive tutorial dialogues in database design. In: Wong, S.L. (ed.) Proceedings of 18th International Conference on Computers in Education, pp. 33–40. APSCE, Malaysia (2010)
Woo, C.W., Evens, M., Freedman, R., Glass, M., Shim, L.S., Zhang, Y., Zhou, Y., Michael, J.: An intelligent tutoring system that generates a natural language dialogue using dynamic multi-level planning. Artif. Intell. Med. **38**(1), 25–46 (2005)
Woolf, B.P.: Building intelligent interactive tutors: student-centered strategies for revolutionizing E-learning. Morgan Kaufmann, Massachusetts (2009)
Zakharov, K., Mitrovic, A., Ohlsson, S. : Feedback micro-engineering in EER-tutor. In: Looi, C.-K., McCalla, G., Bredeweg, B., Breuker, J. (eds.) Proceedings of 12th International Conference on Artificial Intelligence in Education, pp. 718–725. IOS Press, Amsterdam (2005)
Zakharov, K., Mitrovic, A., Johnston, L.: Pedagogical agents trying on a caring mentor role. In: Luckin, R., Koedinger, K., Greer, J. (eds.) Proceedings of 13th International Conference on Artificial Intelligence in Education AIED 2007, Los Angeles, pp. 59–66 (2007)
Zakharov, K., Mitrovic, A., Johnston, L. et al: Towards emotionally-intelligent pedagogical agents. In: Woolf, B. (ed.) Proceedings of 9th International Conference on ITS 2008. LCNS, vol. 5091, pp. 19–28. Springer-Verlag, Heidelberg (2008)

Author Biography

Antonija Mitrovic is Professor of Computer Science at the University of Canterbury, founder and director of the Intelligent Computer Tutoring Group. Her research interests are in student modeling for Intelligent Tutoring Systems. She holds a PhD in Computer Science from the University of Nis, Yugoslavia. She is a member of AIED, AAAI, ACM, APSCE and IEEE. She has authored over a hundred technical papers and has edited several books.