Integrating Case-Based Reasoning and Meta-Learning for a Self-Improving Intelligent Tutoring System

Leen-Kiat Soh, Todd Blank, *National Center for Information Technology in Education, Computer Science and Engineering, University of Nebraska, 256 Avery Hall, Lincoln NE 68588-0115, USA*
lksoh@cse.unl.edu

Abstract. A framework integrating case-based reasoning (CBR) and meta-learning is proposed in this paper as the underlying methodology enabling self-improving intelligent tutoring systems (ITSs). Pedagogical strategies are stored in cases, each dictating, given a specific situation, which tutoring action to make next. Reinforcement learning is used to improve various aspects of the CBR module – cases are learned and retrieval and adaptation are improved, thus modifying the pedagogical strategies based on empirical feedback on each tutoring session. To minimize canceling out effects due to the multiple strategies used for meta-learning – for example, the learning result of one strategy undoes or reduces the impact of the learning result of another strategy, a principled design that is both cautious and prioritized is put in place. An ITS application, called Intelligent Learning Material Delivery Agent (ILMDA), has been implemented, powered by this framework, on introductory computer science topics, and deployed at the Computer Science and Engineering Department of the University of Nebraska. Studies show the feasibility of such a framework and impact analyses are reported on pedagogical strategies and outcomes.

Keywords. Intelligent tutoring systems, case-based reasoning, meta-learning, reinforcement learning

INTRODUCTION

Developing instructional material for an intelligent tutoring system (ITS) is very time consuming. For example, Woolf and Cunningham (1987) estimated that an hour of instructional material required more than 200 hours of ITS development time. Furthermore, an already constructed ITS for a specific domain can neither be re-constructed to function for a different domain, nor can it be altered without spending much time and effort (Virvou & Moundridou, 2001). In the past, there have been attempts (e.g., Kimball, 1982; O'Shea, 1982; MacMillan & Sleeman, 1987; Dillenbourg, 1989; Gutstein, 1992; Elorriaga & Fernández-Castro, 2000) to build self-improving ITSs. However, these attempts have focused on learning about the students that they encountered to improve the ITS's modeling of the students in order to better decide the next appropriate tutoring actions. Thus, these attempts have not emphasized self-evaluation or self-improvement of the fundamental reasoning component of the ITSs. For example, is a particular instructional strategy appropriate? Is a particular weight used in making a tutorial decision correct? Thus, these attempts have not significantly reduced the cognitive load needed to develop instructional material for ITSs. Hence, there is a need for self-improving ITSs that are introspective and meta-cognitive, having the capability to examine their own reasoning, such that the need for laborious and expert quality control of the material can be reduced.

Specifically, such an ITS will improve ITS development in the following aspects (Murray, 1999): (1) decrease the effort for making intelligent tutors; (2) decrease the skill threshold for building intelligent tutors; (3) help the designer/author articulate or organize his or her domain or pedagogical knowledge; (4) support good design principles; and (5) enable rapid prototyping of intelligent tutor designs. Note that the research described in this paper does not attempt to build authoring tools that support the development of ITSs; on the contrary, the framework is grounded on the hypothesis that an observant, intelligent ITS with machine learning capabilities is able to refine its knowledge base to address those needs automatically. Our approach embraces this hypothesis using a framework of case-based reasoning (CBR) with meta-learning to enable self-improving ITSs. This framework allows an ITS to be generative, able to model students, able to model expert performance, able to change pedagogical strategies, and "self-improving" in which the ITS "has the capacity to monitor, evaluate, and improve its own teaching performance as a function of experience" (Woolf et al., 2002).

A framework integrating CBR and meta-learning called CBRMETAL is proposed in this paper. Meta-learning is defined as a learning mechanism of a system that learns about the system itself and how to improve the system's performance over time. Pedagogical strategies are stored in cases, each dictating how, given a specific situation, to decide which tutoring action to make next. Reinforcement learning is used to improve various aspects of the CBR module: cases are learned and retrieval and adaptation are improved. To minimize canceling out effects due to the use of these multiple machine learning strategies to conduct meta-learning - for example, it is possible for the refinements in adaptation to conflict with the refinements in retrieval, a principled design that is both cautious and prioritized is also put in place. An ITS application, called Intelligent Learning Material Delivery Agent (ILMDA), has been implemented in Java, based on this framework, to deliver instructional material on introductory computer science (CS1) topics, and deployed at the Computer Science and Engineering Department of the University of Nebraska. Two studies were conducted. The first study focused on how the learning-enabled ITS impacted the pedagogical strategies and outcomes. The second study focused on the use of case-based reasoning and meta-learning in terms of effectiveness and efficiency.

This paper is a comprehensive extension to previous conference papers published about the project (Soh & Blank, 2005; Soh et al., 2005; Blank et al., 2004).

In the following, related work in CBR and meta-learning and the use of CBR in ITSs are presented first. Then, the CBRMETAL framework is discussed, followed by a section on implementation. Finally, results of two deployments of ILMDA are reported before the paper is concluded.

RELATED WORK

Self-Improving CBR

Integrating meta-learning to self-improve CBR has been proposed and described in the literature. Emphasis has been mostly on parameter or feature weighting for similarity-based case retrieval. For example, Wettschereck and Aha (1995) proposed weighting features automatically for case retrieval using a hill-climbing algorithm; Cardie (1999) used cognitive biases to modify feature set selection (changing, deleting, and weighting features appropriately); Bonzano et al. (1997) used a decay policy together with a push-pull perspective to adjust the term weights; Avesani et al. (1998) used

reinforcement learning to reward nearest neighbors that can be used correctly to solve input problems to adapt the local weights to the input space; Jarmulak et al. (2000) used genetic algorithms to determine the relevance/importance of case features and to find optimal retrieval parameters; Zhang and Yang (2001) used quantitative introspective learning resembling back-propagation neural networks to learn feature weights of cases; Park and Han (2002) used an analogical reasoning structure for feature weighting using a new framework called the analytic hierarchy process; and Patterson et al. (2002) proposed a hybrid approach based on the k-nearest neighbor algorithm and regression analysis. While case learning has also been a staple of CBR (e.g., Watson & Marir, 1994), learning about case adaptation has not received as much attention. Leake et al. (1995) formulated the task of acquiring case adaptation knowledge as learning the transformation and memory search knowledge, by learning successful adaptation cases for future use. In Stahl's (2005) formal view of a generalized CBR model, the author considered learning similarity measures while assuming that the adaptation and output function remain static during the lifetime of the CBR system. *However, none of the above examples proposed an integrated, introspective learning framework for CBR as proposed in this paper.*

ITS with CBR or Machine Learning

There are ITSs that use CBR or machine learning. For example, Weber and Brusilovsky (2001) described the *ELM Adaptive Remote Tutor* (ELM-ART), an ITS that supports learning programming in LISP. ELM-ART models individual learners as a collection of episodes that are descriptions of how exercise problems have been solved by a particular student. These descriptions are explanation structures of how a programming task has been solved by the student; i.e. stored episodes contain all the information about which concepts and rules were needed to produce the program code the students offered as solutions to programming tasks. Each episode is stored as *cases*, with each case describing a concept and a rule that was used to solve a plan or sub-plan of the programming task. Using a combination of an overlay model and the above episodic student model, ELM-ART provides adaptive navigation support, course sequencing, individualized diagnosis of student solutions, and example-based problem-solving support. ELM-ART also selects the best next step for a particular user. Starting from the current learning goal, the system recursively computes all prerequisites that are necessary to fulfill the goal. The first concept belonging to the set of prerequisites that is not learned or solved already will be selected and offered to the learner. The learner completes the course successfully when all prerequisites to the current goal are fulfilled and no further goal can be selected. The system also provides feedback by providing a sequence of help messages with increasingly detailed explanation of the error or suboptimal solution. The sequence starts with a very vague hint on what is wrong and ends with a code-level suggestion of how to correct the error or how to complete the solution. The system also provides an ordered list of relevant examples. Basically, using case-based retrieval, ELM-ART selects the episodes with the highest similarity values to the current frame and presents a list of links to examples and reminders. However, unlike CBRMETAL, ELM-ART used neither adaptation nor learning. Thus, significant effort must be invested to ensure the quality of the cases; while in the CBRMETAL, only minimal initial effort is necessary to develop cases.

The CBRMETAL framework described in this paper shares the same underpinnings to its design as the *Case-Based Instructional Planner* (CBIP). In the CBIP project, Elorriaga and Fernández-Castro (2000) integrated a case-based instructional planner with existing ITSs to enhance the pedagogical component with learning capabilities, transforming ITSs into self-improving systems that learn from memorization and learn from their own experiences, where instructional planning is the process of

mapping out a global sequence of instructional goals and actions that provides consistency, coherence, and continuity throughout an instructional session. In CBIP, the instructional plan memory (IPM) is the repository of the past teaching/learning experiences of the case-based system. A case defines a piece of a previously used instructional plan and contains: the context in which it was applied, the instructional plan itself or a part of it (subplan) if the plan is layered, and the results that it achieved. Similar to the design described in this paper, the CBIP application context consists of a sequence of student-related features, a sequence of session-related features, and a sequence of domain-related features. Arruarte, Fernández-Castro, Ferrero and Greer applied the framework to build *Maisu* that tutors the derivatives topic (Arruarte et al., 1997). For adaptation, they used a critic-based approach: a set of rules that identify the specific adaptation needs. In CBRMETAL, reinforcement learning is used to refine both similarity and adaptation heuristics.

Mayo and Mitrovic (2001) proposed a methodology for building tractable normative ITSs using a Bayesian network for long-term student modeling and decision theory to select the next tutorial action. They built the *Capitalization and Punctuation Intelligent Tutor* (CAPIT), a normative constraint-based tutor for English capitalization and punctuation and showed that, through evaluation results, a class using the full normative version of CAPIT learned the domain rules at a faster rate than the class that used a non-normative version of the same system. CAPIT has a five-step methodology for designing decision-theoretic pedagogical action selection (PAS) strategies: (1) randomized data collection, (2) model generation, (3) decision-theoretic strategic implementation, (4) online adaptation, and (5) evaluation. One area of concern with the above approach is scalability, both to larger domains and different domains. In a larger domain, the space of <state, action, outcome> triples may be so large as to effectively render network induction impossible. Thus, we see CBR with its ability to adapt a viable solution to address this concern. Further, the ability to learn and store new cases allows the system to evolve. Scaling to different domains hinges on handling ambiguity. CBR allows for partial matching and thus can help address this particular concern.

Mayo and Mitrovic (2001) also categorized ITSs into three models: expert-centric, efficiency-centric, and data-centric. In expert-centric models, student models are unrestricted products of domain analysis. That is, an expert specifies either directly or indirectly the complete structure and conditional probabilities of the Bayesian student model, in a manner similar to that with which expert systems are produced. Our ILMDA can be considered as following such a model, as the initial casebase – together with the adaptation and retrieval heuristics – contains domain expertise on what tutorial actions to take based on certain student models, though without the corresponding conditional probabilities. In data-centric models, the structure and conditional probabilities of the network are learned primarily from data. Thus, our ILMDA can also be seen as following the data-centric model, as it learns from its interactive sessions and refines its knowledge base. Benefits of the data-centric model include the following. First, because the model is inducted from actual data, its predictive performance can easily be evaluated by testing the network (or casebase) on data that was not used to train it. Second, data-centric models can be expected to be much smaller than the typical expert-centric model because the latter represents both observed and hidden variables, while the former models only observable variables.

Melis et al. (2001) provided a comprehensive account of *ActiveMath*, a generic web-based learning system that dynamically generates interactive (mathematical) courses adapted to student goals, preferences, capabilities, and knowledge. When the user has chosen her goal concepts and scenario, the session manager sends this request to the course generator. The course generator is responsible for choosing and arranging the content to be learned. It checks the user model to find out

the user's prior knowledge and preferences, and uses pedagogical rules to select, annotate, and arrange the content – including examples and exercises. However, unlike CBRMETAL, the ActiveMath unit of course module is the entire tutorial together with the examples and exercises and does not consider the real-time interactivity between the user and the ITS. While ActiveMath uses rules, cases are used to allow for partial matching in CBRMETAL. For user modeling, ActiveMath incorporates persistent information about the user as well as a representation of the user's learning progress. There are 'static' properties such as field, scenario, goal concepts, and preferences as well as the 'dynamic' properties such as the knowledge mastery values for concepts and the user's actual behavior, that have to be stored in the user model. ActiveMath also keeps track of a history about the actions the user performed. The history elements contain information such as the IDs of the content of a read page or the ID of an exercise, the reading time, the success rate of the exercise, which is similar to CBRMETAL.

In terms of potentially useful applications, the CBRMETAL framework can be used as the following:

(1) A testbed to collect empirical data to, for example, automatically register the transitions in terms of the path taken by students to achieve a tutorial goal defined in *MetaMuse* (Cook, 2001). Because of the reasoning and self-learning power of CBRMETAL, it is possible to separate the reasoning module from the instructional expertise (as in the cases) and the content set (as separate databases). This modularity enables experimentation: cases or content sets can be substituted without having to modify the reasoning module at all.

(2) An intelligent authoring tool – similar to *Disciple*, a learning agent shell by (Tecuci & Keeling, 1999), or *WEAR*, an instructor modeler in terms of level of expertise, interests and activities, and preferences in teaching strategies, by (Virvou & Moundridou, 2001) – that quality-tags domain expertise for the instructors designing the content, especially in the categories of tutoring strategies and multiple knowledge types (Murray, 1999). Because of its self-improving capabilities, CBRMETAL can evaluate its content sets and cases. Content sets that have been applied with consistent results could be tagged as such, for example. Cases that have been used regularly could be tagged as such, for example. An intelligent authoring tool could solicit content sets or cases from a developer, run them through its CBRMETAL process using the data or experience that it has accumulated, and then make suggestions to the developer regarding the projected quality of the content sets or cases. Also, this framework enables the learning of specific cases and heuristics for specific combinations of contents and students. That is, it is possible to deploy a CBRMETAL-powered ITS for a trial period to collect automatically customized cases and heuristics.

UNDERLYING REASONING AND META-LEARNING FRAMEWORK

The underlying methodology of the framework integrates case-based reasoning (CBR) with meta-learning. The traditional CBR framework is shown in Figure 1(a), in which the module receives a new situation, searches its casebase to locate the most similar case (or best case) matching the new situation, and adapts the solution from the best case to fit the new situation. The extended framework, shown in Figure 1(b) incorporates meta-learning to self-adjust various components of the traditional CBR framework: the casebase, the heuristics used to compute case similarity for retrieval and storage, and the heuristics used to adapt case solutions. This framework is named CBR and Meta-Learning or CBRMETAL.

The *case learning module* learns new, different cases to improve the coverage of the casebase. The module designates a new situation and its adapted solution as a potential new case. It compares the potential new case with all the existing cases in the casebase. If the potential new case is found to be distinct enough, then the module adds it to the casebase for future use.

The *similarity heuristics adjuster* module uses reinforcement learning to adjust the heuristics used in computing case similarity. Note that each case's situation description consists of a set of attribute-value pairs. These similarity heuristics determine the relative weights of the attributes. Learning to refine these heuristics allows the CBR system to improve its case retrieval mechanism – retrieving best cases that are more suitable because of the different weights. The reinforcement learning is based on the following assumption. For a retrieved, most similar case, C_{best}, it has a situation description and a solution, $p_{C_{best}}$ and $s_{C_{best}}$, respectively. If $s_{C_{best}}$ has been observed to be successful in the past, and when it is adapted to a new situation, p_{new}, deemed similar to $p_{C_{best}}$, then $s_{C_{best}}$ should be successful as well. If that is not observed, then p_{new} may be not as similar to $p_{C_{best}}$ as initially computed. As a result, similarity heuristics that contributed positively to the similarity are penalized. Similarly, similarity heuristics that contributed negatively to the similarity are rewarded.

The *adaptation heuristics adjuster* module learns how to adapt solutions to new situations by adjusting the weights of the adaptation heuristics. The underlying strategy is reinforcement learning, similar to that used in the above module. For each adapted best case solution, $s'_{C_{best}}$, the module keeps track of the adaptation heuristics that have contributed to the changes. If the $s'_{C_{best}}$ is observed to be successful (or not successful), then all contributing adaptation heuristics are rewarded (or penalized). The level of reward or penalty is also based on the amount of contribution of each heuristic, allowing the system to selectively adjust individual heuristics.

This approach presents several advantages and disadvantages over a traditional CBR system. The main advantage is that no one component is entirely responsible for improving the performance of the system. By adjusting all three aspects of the CBR system at once, the problem of biased learning can be avoided. For instance, if only one aspect was adjusted until it was believed to be functioning correctly, and only then another aspect was adjusted; then it is possible that, after the second aspect was adjusted, the first aspect may no longer be optimal or appropriate. However, this also exposes a weakness of our approach. With so many variables changing and adapting, the system may never converge to a steady state, and one component "improving" may undo "improvements" made in another component of the system. For example, adjusting the similarity weights affects how the system decides which new cases to learn and which best case to retrieve and adapt. Learning new cases reduces the need for the adaptation module to make radical, far reaching heuristics, and allows for more choices when the similarity module attempts to find a case to use. Adjusting the adaptation heuristics will reduce the need for the casebase to cover *every* situation. It will also give some leeway to the similarity heuristics, requiring them to only find a close match in the situation space, and not have to find a good situational match in order to provide a good solution for the new situation.

As pointed out by Cox and Ram (2001), through their extensive empirical evaluations of a system that implemented a meta-learner that uses multiple machine learning strategies, "explicit representation and sequencing of learning goals is necessary for avoiding negative interactions between learning algorithms that can lead to less effective learning." Based on our experience over the years in developing and testing the CBRMETAL framework, six principles have been devised to minimize such canceling each other out effects.

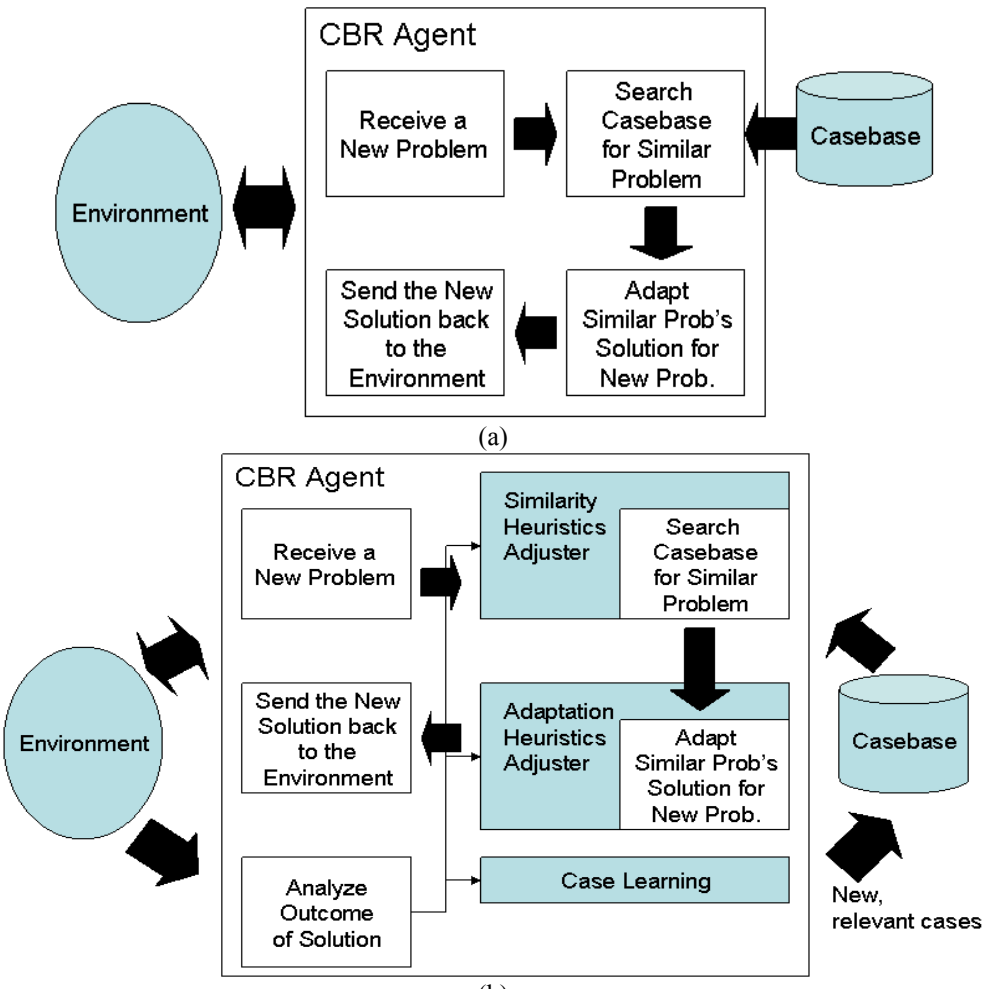


Fig.1. (a) Traditional or classic case-based reasoning framework; (b) Extended case-based reasoning framework with machine learning capabilities, CBRMETAL.

Principle 1. When CBRMETAL learns a new case based on diversity, its objective is to expand the situation coverage but not the solution coverage of the casebase. With this arrangement, though it is still possible for the solution coverage of the casebase to expand, it is the adaptation heuristics that have to shoulder the task of generating new solutions. Thus, it is less likely for the new case to be learned to significantly impact the coverage of solutions.

Principle 2. The CBRMETAL framework may also learn new cases with failed solutions as the CBR system also performs failure-driven adaptations. This means if the same situation arises in the future, it is possible for the system to retrieve a newly-learned case with a failed solution, and perform a failure-driven adaptation, to incrementally and eventually obtain a successful solution without straining the adaptation heuristics. Failure-driven adaptation heuristics are those that adjust the solution parameters in an opposite manner to what usual different-driven adaptation heuristics would do.

Principle 3. Each case is tagged with a utility or competence vector that records how successful the case has been used, how often the case has been retrieved, and how many new cases have been spawned as a result of the retrieval of this case (Soh & Luo, 2004). This allows CBRMETAL to react with the appropriate degree of "zeal" to the reinforcement. Cases that have been more successful will carry more weight, for example, in determining the similarity heuristics. Cases that have spawned more new cases will carry more weight in determining the adaptation heuristics.

Principle 4. CBRMETAL puts in place parameters indicating the degree of aggressiveness of each learning module (e.g., t in the similarity heuristics learning module and h in the adaptation heuristics learning module). This allows the developers of a CBR system to incorporate confidence into these two sets of domain expertise. For example, if the similarity heuristics are highly regarded as correct, then t can be set to be smaller than h. This in turn could speed up the overall rate of coherence or convergence of the various learning modules. Also, if the initial casebase is good – with good coverage of the situation space and good situation-solution mapping, then it is less likely for the system to learn new cases or have to adapt to new situations, and thus it is more important to learn good similarity heuristics than to learn good adaptation heuristics. On the other hand, if the initial casebase is average or not good, then the burden lies on case learning and adaptation. For this, learning good adaptation heuristics is thus more important, and thus h should be greater than t. Of course, if the developer has no prior knowledge on the quality of the cases, then setting $t = h$ would allow the framework to evolve the casebase and the heuristics in an unbiased manner. Note that the CBRMETAL framework does not attempt to tune these degrees of aggressiveness, as they are in place only for speeding up the convergence in learning.

Principle 5. CBRMETAL learns conservatively – changing only the most influential similarity and adaptation heuristics for each learning episode. That is, after the blame or credit assignment, similarity (or adaptation) heuristics are ranked in terms of the blame or credit. Only the top contributor to a failure or a success is penalized or rewarded accordingly. This has the effect of reducing the impact of a single outcome on the heuristics, allowing the system to gradually adjust the impact of each heuristic.

Principle 6. CBRMETAL staggers its learning activities with different activation frequencies. For example, case learning can be activated every time a new case is evaluated while heuristics learning can be activated after every n cases, or only when the system has seen too many failures for some period of time. Once again, as discussed in Principle 4, confidence in the qualities of the casebase and heuristics could help determine the activation frequencies. Usually, case learning can be conditioned upon the utility of the new case to be learned: if the new case adds to the coverage or diversity of the solution space, then it is learned; otherwise, it is not. However, weights for the similarity and adaptation heuristics, in general, should not be modified after, for example, every single ITS session as that could cause the learning process to oscillate and never converge.

Principles 1 and 2 have been adopted in the case learning module; Principles 3, 4, and 5 in the two heuristics learning modules; and Principle 6 in the overall learning management.

AN APPLICATION TO INTELLIGENT TUTORING SYSTEMS: ILMDA

This section describes one application of the CBRMETAL framework to the area of intelligent tutoring systems (ITSs). In this application, called Intelligent Learning Material Delivery Agent (ILMDA), each instructional content set consists of three parts: (1) a tutorial, (2) a set of related examples, and (3) a set of exercise problems to assess the student's understanding of the topic. Based

on how a student progresses through the content set and based on his or her profile, ILMDA chooses the appropriate examples and exercise problems for the student. ILMDA makes its choice using case-based reasoning. Each case contains an instructional strategy, mapping a particular problem to a specific solution, where the problem consists of parameters describing a student's progress and his or her profile, and where the solution consists of parameters prescribing the characteristics of the next example or exercise problem appropriate for that student. The case that best matches the current situation is retrieved from the casebase and its solution is then adapted to the current situation. After the modified solution has been applied – i.e. an example or an exercise problem given to the student, its outcome is recorded. This outcome in turn facilitates the meta-learning of the system.

Cases

ILMDA has a casebase of cases. Each case is composed of four parts: situation, solution, outcome, and performance parameters, as shown in Figure 2.

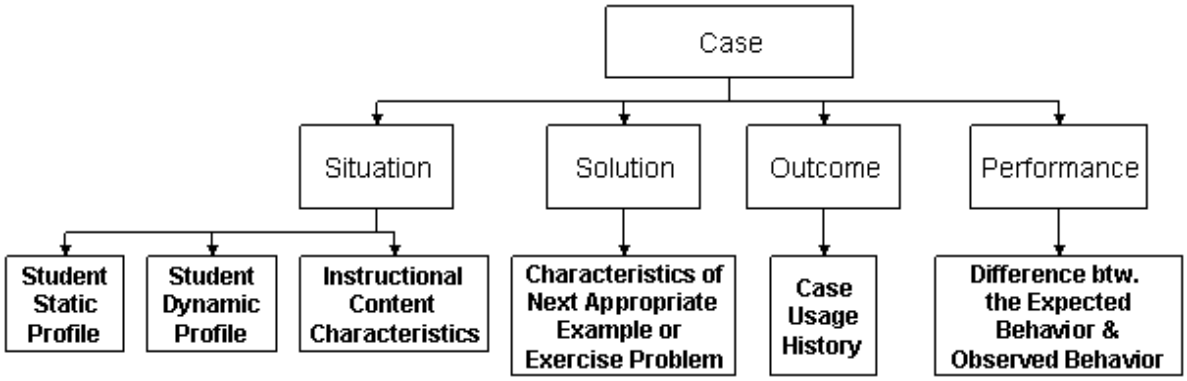


Fig.2. Components of a case in our CBRMETAL-powered ITS application, ILMDA.

The *situation parameters* include the student static and dynamic profiles and the instructional content's characteristics. The student static and dynamic profiles form the basis for ILMDA's learner modeling. This is achieved by profiling a learner/student along two dimensions: student static background and dynamic student activity. The background of a student stays relatively static and consists of the student's last name, first name, major, GPA, goals, affiliations, aptitudes, and competencies. It also includes self-reported self-efficacy and motivation, based on a survey taken before a student processes a content set. The dynamic student profile captures the student's real-time behavior and patterns. It consists of the student's online interactions with the GUI module of ILMDA including the number of attempts on the same item, number of different modules taken so far, average number of mouse clicks during the tutorial, average number of mouse clicks viewing the examples, average length of time spent during the tutorial, number of quits after tutorial, number of successes, and so on.

The *solution parameters* specify the characteristics of the example or exercise problem to be delivered to the student. Each example or exercise problem is meta-tagged with a set of attributes: length, interest, Bloom's taxonomy (Bloom et al., 1964), level of difficulty, amount of scaffolding, the number of times viewed, the average time per use, average number of clicks per use, and so on. Instructional scaffolding (Vygotsky, 1978) is support for learning, and the level of scaffolding varies

for different students and scenarios. In ILMDA, scaffolding similar to those proposed in (Hartman, 2002) is used: cues, hints, references, and elaborations. Cues are highlighted phrases. A hint poses "what if " and "think about this" statements. A reference points the student to a particular item in the content set. An elaboration explains the steps or partial solutions. We use highlighted phrases, diagrams, and figures as cues; "What if " and "Think about this" questions as hints and prompts; and stepwise, spelled-out solutions as partial solutions. In addition, references are used to point students back to certain pages of a tutorial or an example. For example, "If you are not clear about how a try-catch block works, please refer to the Tutorial page, under the 'Throwing Exceptions' section." Elaborations are guidelines such as "trying to divide a number by 0 or trying to convert a string that contains letters into an integer are both unchecked exceptions".

The *outcome parameters* are based on the usage history of a case, which includes the number of times the case has been used, the number of times the case has been used successfully, whether the student quit the example or exercise problem, and whether the student answers the exercise problem correctly. These parameters are accumulated from what is observed each time the case is used.

The *performance parameters* document the difference between the expected and the observed behavior, also accumulated from what is observed each time the case is used. For example, how much time the student spent on an exercise problem or an example with respect to the average time all students spent on the same exercise problem or example, how many times the student went back-and-forth between an exercise problem and a tutorial page with respect to the average recorded for all students when given the same exercise problem, and so on. This allows the CBR module to evaluate how each session performs with respect to the norm.

Tables 1 and 2 document the situation and solution parameters of a case, respectively.

How CBR is Used

CBR is used in our system to retrieve the best matching case to the current situation, adapt the solution of the best matching case to the current situation, and retrieve the next appropriate example or exercise problem based on the characteristics prescribed by the solution.

When a student clicks "Next Example" or "Next Exercise Problem", ILMDA records what it has observed so far of the student (student static and dynamic profiles, see Table 1) and denotes these as the current situation parameters.

ILMDA then computes the similarity of each case C_i in the casebase to this current situation:

$$Curr: \ sim(Curr, C_i) = \sum_{k=1}^{N_p} sw_k \cdot |p_{Curr,k} - p_{C_i,k}|$$

where N_p is the number of situation parameters, p_{C_i} is the situation description of C_i, $p_{C_i,k}$ is the *k*th situation parameter of C_i, and sw_k is the similarity weight for the *k*th situation parameter. These sw_k weights are considered the similarity heuristics; each indicates the importance of a particular parameter in determining the similarity between two sets of situation parameters.

Table 1
Situation parameters of a case used in the intelligent tutoring system application of the CBRMETAL framework

Parameters	Description
GPA	The student's self-reported grade point average
AVE_TTRL_TIME	The average time spent (in milliseconds) per tutorial
AVE_TTRL_CLICKS	The average number of times the student clicks the mouse in the tutorials he or she has seen
AVE_EXMP_CLICKS	The average number of times the student clicks the mouse in the examples he or she has seen
AVE_EXMP_TIME	The average time spent (in milliseconds) per example
AVE_EXMP_TO_TTRL	The average number of times the student goes back to the tutorial from the example
AVE_GRADE	The student's average grade on the exercise problems
AVE_PROB_CLICKS	The average number of times the student clicks the mouse in the exercise problems he or she has seen
AVE_PROB_TIME	The average time spent (in milliseconds) per exercise problem
AVE_PROB_TO_EXMP	The average number of times the student goes back to the example from the exercise problem
AVE_PROB_TO_TTRL	The average number of times the student goes back to the Tutorial from the exercise problem
AVE_SES_TIME	The student's average total time spent in the interface during a session
EXMP_QUITS	The number of times the student has quit at the example stage
MAX_SES_TIME	The length of the student's longest session, in milliseconds
MIN_SES_TIME	The length of the student's shortest session, in milliseconds
NUM_EXMP	The number of examples the student has seen
NUM_PROB	The number of exercise problems the student has seen
NUM_SESSIONS	The total number of sessions the student has had
PROB_QUITS	The number of times the student has quit at the exercise problem stage
SUCCESSES	The number of successful sessions the student has had
TTRL_CLICKS	The number of times the user has clicked the mouse during the tutorial
TTRL_QUITS	The number of times the user has clicked the mouse during the tutorial
TTRL_TIME	The length, in milliseconds, the student spent in the tutorial
SELF_EFFICACY	The student's self efficacy, measured by a pre-topic quiz
MOTIVATION	The student's motivation, measured by a pre-topic quiz

Table 2
Solution parameters of a case used in the intelligent tutoring system application of the CBRMETAL framework

Parameters	Description
DIFF_LEVEL	The difficulty level of the exercise problem or example
MIN_USE_TIME	The shortest anyone has looked at the exercise problem or example
MAX_USE_TIME	The longest anyone has looked at the exercise problem or example
AVE_USE_TIME`	The average time students view the exercise problem or example
AVE_CLICK	The average # of times students click the mouse in the exercise problem or example
LENGTH	The # of characters in the example or exercise problem
BLOOM	The Bloom's taxonomy value for the exercise problem or example
SCAFFOLDING	The amount of scaffolding to give the material

After computing the similarity values of all cases in the casebase with respect to the current situation $Curr$, ILMDA selects the case with the highest similarity as the best case, C_{best}. The solution of C_{best}, s_{Cbest}, has to be adapted to $Curr$. The adaptation is based on the differences between $Curr$ and p_{Cbest}. For each solution parameter, there is one adaptation heuristic: $ah_l = \langle aw_{l,1}, aw_{l,2}, \mathrm{K}, aw_{l,N_p} \rangle$ where N_p is the number of situation parameters, and $hw_{l,n}$ is the weight in heuristic ah_l that influences the lth solution parameter. Each lth solution parameter is adapted in the following manner:

$$s'_{C_{best},l} = \sum_{k=1}^{N_p} \left[aw_{l,k} \cdot \left(p_{Curr,k} - p_{C_{best},k} \right) \right] \cdot s_{C_{best},l}$$

Thus, the difference between the kth situation parameter in the current situation and the best case situation parameters are moderated by a heuristic weight is computed. These differences are then summed and multiplied with the original value of the lth solution parameter of the best case. This process is repeated for all solution parameters of the best case. When it is completed, the best case solution, s_{Cbest}, is considered to have been adapted, and the modified best case solution is s'_{Cbest}.

Note that for solution parameters that are continuous $s'_{Cbest,l}$ is just the sum of weighted differences. For discrete solution parameters, however, the difference is less straightforward. For example, as shown in Table 2, one of the solution parameters is BLOOM, which indicates Bloom's taxonomy value for an exercise problem or an example. For this parameter, we denote the six Bloom levels as 1-6 (corresponding to knowledge, comprehension, analysis, application, evaluation, and synthesis). Thus, instead of using the actual real value of $s'_{Cbest,BLOOM}$, it is normalized between 1 and 6, and rounded down to an integer. Another solution parameter is SCAFFOLD, which indicates the amount of scaffolding to be displayed with the example or exercise problem. The four levels of scaffolding are denoted in binary codes: 0001 for a cue, 0010 for a hint, 0100 for a reference, and 1000 for an elaboration. Given these levels, a cue has 1 point, a hint has 2 points, a reference has 4 points, and an elaboration has 8 points, indicating the different amounts of scaffolding these items offer. Likewise, it is possible to have a SCAFFOLD of 9 points or 1001, which means to use both elaborations and cues. As a result, the amount of scaffolding is between 0 and 15. Following the same approach to BLOOM, instead of using the actual real value of $s'_{Cbest,SCAFFOLD}$, it is normalized between 0 and 15, and then rounded down to an integer.

Equipped with $s'_{C_{best}}$, ILMDA now has a prescription for the next appropriate example or exercise problem. All examples and exercise problems of a tutorial are stored in a database, each meta-tagged with a set of parameters exactly the same as those found in Table 2. ILMDA finds the best matching example or exercise problem to $s'_{C_{best}}$ and duly displays it to the student.

How Meta-Learning is Used

Meta-learning is used in our system to adjust the adaptation heuristics, similarity heuristics, and flesh out the casebase used by the CBR system. After a student has completed the session, ILMDA informs its CBR module the result of the student's session. The CBR module uses this information to analyze the results and improve its performance, and as a result, the performance of the whole system.

ILMDA learns new cases in order to allow the system to more easily handle a wider range of situations that it will inevitably encounter when faced with different types of students. The initial casebase is designed by experts in order to cover a wide range of students, but may not actually cover the possibilities or may be faulty. For instance, a new case which has been learned may have a similar situation space with a different solution, providing another way of approaching the situation that it describes. Or an entirely new type of situation space may be encountered, which was unplanned for. The ability to learn new cases allows the system to overcome this. In the application, the design focuses on expanding the situation space of the casebase, in order to cover as many situations as possible in the following manner. Given the potential new case, C_{new}:

if $sim_{\max}(C_{new}) = \max_i sim(C_{new}, C_i)$ is less than a threshold, t_{learn},

then C_{new} is added to the casebase, with $sim(C_{new}, C_i) = \sum_{k=1}^{r} sw_k \cdot |p_{C_{new},k} - p_{C_i,k}|$

where N_p is the number of situation parameters, p_{C_i} is the situation description of C_i, $p_{C_i,k}$ is the kth situation parameter of C_i, and sw_k is the similarity weight for the kth situation parameter. (Note that this similarity measure is similar to the one used in CBR to retrieve the best matching case to a current situation.)

The similarity heuristics adjuster module in turn learns about the set of sw_k in the following manner**.** Each case has a distribution, $dist_{C_i}$, of how it has been used, based on its performance parameters. In this particular implementation, the distribution is assumed to be uniform and thus only the average is computed. For each pair of similar cases, C_i and C_j – determined by using the set of sw_k, compute the performance similarity

$$sim_{per}(C_i, C_j) = \sum_{k=1}^{N_{pf}} |dist_{C_i,k} - dist_{C_j,k}|$$

where N_{pf} is the number of performance parameters. C_i and C_j are considered to be truly similar if both $sim(C_i, C_j)$ and $sim_{per}(C_i, C_j)$ are high ($> t_{learn}$ and $> t_{similar}$, respectively). For each such occurrence, the sw_{sel} value (such that $sel = \arg\max_k sw_k$) is increased by a percentage α_{sim}. If $sim(C_i, C_j)$ is high and $sim_{per}(C_i, C_j)$ is low, then the two cases are not considered to be truly similar and the sw_{sel} value is decreased by α_{sim}. α_{sim} is essentially the learning rate of this reinforcement learning design. If $sim(C_i, C_j)$ is low, then sw_{sel} remains unchanged. With this design, if the system deemed that situation parameter p_k was the most influential in determining that C_i and C_j were very similar, but the cases had very different performances, then the learning module

will adjust the similarity weights to make p_k less influential, thereby making C_i and C_j less similar. However, if the similarity measurement did not determine that the cases were similar, then it would not be expected that the cases would perform similarly. With this learning process, the system would, for example, eventually appoint the amount of time a student spends in a particular session as a more important parameter than the number of times a student views a tutorial.

The adaptation heuristics in the ILMDA modify a solution provided by the case-based reasoning in order to account for differences in the situation descriptions. Once again, the reinforcement is driven by the performance of a session. For instance, if the student took an excessive amount of time and was unsuccessful in answering the question they were presented with, then the performance of that session would be low. Similarly, if the student spent much less time than usual on a question, and answered it correctly, then the performance of the session would also be low, reflecting that the solution was too easy to present a worthwhile experience to the student. This performance information is used to adjust the adaptation heuristics of the system by determining if an adaptation was successful or not. If an adaptation is deemed successful, then the heuristics that were influential in making that adaptation will be reinforced. And, similarly, if an adaptation is deemed unsuccessful, then the heuristic most responsible for that adaptation will be weakened. The system determines which heuristic is most responsible in the following manner. Given the solution of a best case, $s_{C_{best}}$ and the adapted solution $s'_{C_{best}}$, and a set of adaptation heuristics such that each heuristic $ah_l = \langle aw_{l,1}, aw_{l,2}, \mathsf{K}, aw_{l,N_p} \rangle$ where N_p is the number of situation parameters, and $hw_{l,n}$ is the weight in heuristic ah_l that influences the *l*th solution parameter, as described in previous subsection. Designate $ah_{sel}(n)$ such that:

$$sel = \arg\max_{l} \left(aw_n \cdot \left(s_{C_{best},n} - s'_{C_{best},n} \right) \right),$$

which identifies the adaptation heuristic that contributes the most to the difference in the solutions for the *n*th solution parameter. If the performance of the session is deemed successful (observed behavior matches closely the norm, > t_{adapt}), then the $ah_{sel}(n)$ is increased by a percentage of α_{ada}. Likewise, if the performance is deemed unsuccessful, then $ah_{sel}(n)$ is decreased by a percentage of α_{ada}. A high α_{ada} indicates an aggressive learning system, and vice versa. This adjustment of the heuristics allows ILMDA to more accurately adapt to the situations that ILMDA encounters. This adaptation behavior also allows ILMDA to expand its coverage of the situation space, without explicitly growing the casebase' coverage of the situation space.

Following the principles of meta-learning outlined earlier, we minimize possible conflicts in learning in several ways. Two of the modules, the similarity and adaptation heuristics adjustor modules, are run offline after several sessions. This minimizes the noise by reducing the influence that a single outlying result will have on the learning. It also allows a significant amount of sessions to be present before each adaptation, to provide accurate statistics on the cases. The case learning module, on the other hand, is run online with the system. ILMDA decides whether or not to store a case as soon as the session is finished. This allows for new cases to be used immediately. The similarity and adaptation heuristics also act conservatively, modifying only one heuristic at a time, to prevent drastic changes. This increases the stability of our learning while retaining the benefits of meta-learning.

IMPLEMENTATION

The ILMDA system powered by CBRMETAL has been implemented in Java. It has a front end graphical user interface (GUI), a mySQL database backend, and a reasoning module connecting the two. Here we discuss how the content sets were developed, how cases were developed and initial similarity and adaptation heuristics were created, the architecture of ILMDA, the flow of a student session with ILMDA, and how ILMDA performs its meta-learning. Details of the implementation can be found in (Blank, 2005).

Content Set

A total of five content sets were developed on CS1: (1) File I/O, (2) Event-Driven Programming, (3) Exceptions, (4) Inheritance and Polymorphism, and (5) Recursion. For each topic, the content set had a tutorial, a set of 3-4 examples, and a set of 20-25 exercise problems. Two computer science graduate students developed these content sets based on the textbook by Wu (2005), and then revised and refined by Soh, one of the authors of this paper. Soh has taught CS1 at the University of Nebraska for three semesters, and has also been involved heavily in the Computer Science and Engineering Department's Reinventing CS Curriculum Project (http://cse.unl.edu/reinventCS). The content set was then reviewed by a handful of computer science undergraduate students. When developing the content set, scaffolding was embedded and tagged with in-line HTML-like tags. This allows ILMDA to dynamically display them when the solution of a retrieved case calls for a certain level of scaffolding to be in place. The following shows how a reference is tagged.

> … This concludes the discussion on the Tower of Hanoi example.
> <REFERENCE>Please refer to Section 1.1 to review the components of a recursion.</REFERENCE> …

Each example or exercise problem was then tagged with the appropriate Bloom's level (BLOOM: 1-6 for knowledge, comprehension, application, analysis, evaluation, and synthesis) and difficulty level (DIFF_LEVEL: between 1-10).

Cases and CBR Heuristics

The casebase and the similarity and adaptation heuristics essentially constitute the knowledge base of ILMDA. The cases were developed in the following manner. Given the set of situation parameters as shown in Table 1, a range of values was estimated for each *k*th parameter. For example, for the parameter AVE_EXMP_CLICKS, the range was estimated at 1-10. Once the bounds were estimated, nine scenarios (or situations) that roughly approximate the behavior of a particular type of student behavior were identified. For example, "a situation where a student has spent, on average, little time on his/her previous sessions, has tried to go back-and-forth between exercise problems and the tutorial frequently, has seen only a few examples, and has quit previous sessions early (after seeing only a few exercise problems)" is one that might be representative of an impatient student with less motivation to learn but more motivation to get through the content set as soon as possible. "A situation where, with everything else relatively about average, the student's self-efficacy is low and the student spent a lot of time in the tutorial" is one that might be representative of a student with low confidence but with determination to try to understand the tutorial. After selecting a set of interesting situations, their

solutions were identified. For example, for the "impatient student" situation, the solution – i.e. the prescription of the next appropriate example or exercise problem – was to have an example or exercise problem of average difficulty level, short length, with less than average scaffolding, and of low average use time (AVE_USE_TIME). The strategy here was to provide a sufficiently challenging example or exercise problem that would not take too much time to complete. For the "motivated but low-confident student" situation, the solution was to have an example or exercise problem of low difficulty level, with high scaffolding, and of about average use time. The strategy here was to help raise the student's confidence by giving him or her a relatively easy item with a lot of help, in the hope of getting the student to understand the example successfully or answer the exercise problem correctly. Note that this strategy also assumed the following. If the student successfully completed the item, then his or her profile would go up for the next example or exercise problem. Then ILMDA would give a more difficult item (via the adaptation heuristics) and lower the amount of scaffolding. Gradually, if the student continued to do well, ILMDA would select a more difficult item with lower scaffolding.

The similarity heuristics were all initialized to the weight of 1.0. That is, all situation parameters were considered to have the same importance in determining the similarity between two situations. This initialization was possible because of the CBRMETAL framework: the similarity heuristics would be revised automatically by the system as it interacted with the users.

The adaptation heuristics were developed for each solution parameter. Take the amount of scaffolding, SCAFFOLDING, for example. Out of the 16 situation parameters listed in Table 1, 9 played a role in the initial adaptation heuristic: GPA (-10), AVE_GRADE (-10), AVE_TTRL_TIME (5), AVE_EXMP_TIME (5), AVE_PROB_TIME (5), AVE_EXMP_TO_TTRL (5), AVE_PROB_TO_TTRL (5), MOTIVATION (-5), and SELF_EFFICACY (-5). A negative weight means an inverse proportionality: if the difference between two situations for that parameter is positive, then the amount of scaffolding is reduced; and vice versa. Thus, in the above, it is seen if the GPA, AVE_GRADE, MOTIVATION, or SELF_EFFICACY in the current situation is higher than that of the best case' situation, then the best case' SCAFFOLDING is lowered accordingly; and vice versa. On the other hand, if AVE_TTRL_TIME, AVE_EXMP_TIME, AVE_PROB_TIME, AVE_EXMP_TO_TTRL, or AVE_PROB_TO_TTRL of the current situation is higher than that of the best case' situation, then the best case' SCAFFOLDING is increased accordingly. This thus gives rise to the adapted solution. Yet another example is the level of difficulty, DIFF_LEVEL. Nine situation parameters played a role in the initial adaptation heuristic: GPA (6), SUCCESS (8), AVE_GRADE (10), AVE_TTRL_TIME (1), AVE_EXMP_TIME (1), AVE_PROB_TIME (1), AVE_EXMP_TO_TTRL (-1), AVE_PROB_TO_TTRL (-1), and AVE_PROB_TO_TTRL (-1). Here, GPA, SUCCESS, and AVE_GRADE were considered to be more important in deciding the difficulty level than other parameters. In general, if the student has a better GPA, has had more successes with ILMDA sessions, or done better on the exercise problems than the best case' situation, then DIFF_LEVEL is increased; and vice versa.

Once developed, the initial cases and the adaptation heuristics were then reviewed by an educational expert in instruction. The expert was told to review whether the cases and heuristics were "more or less" correct and not to dwell on the actual values too closely. For example, whether GPA should be weighted 6 and SUCCESS weighted 8 in the adaptation heuristic in DIFF_LEVEL was not as important as whether SUCCESS should be weighted more than GPA. It took the expert relatively little time (within an hour) to review the cases and the adaptation heuristics. Note that this was

possible because of the CBRMETAL framework: the cases and the heuristics would be modified by ILMDA over time, and thus the actual values did not have to be accurately initialized in the first place.

Architecture

Figure 3 shows the architecture of the CBRMETAL-powered ITS application. It has a front end GUI interacting with the user (student). This GUI tracks every interaction between the student and the GUI, and sends the information to the reasoning core. The reasoning core denotes each set of information as the current situation, and retrieves the best case from the casebase. The core module then adapts the solution to fit the current situation. As a result of this, a prescription for the next appropriate example (or exercise problem) is obtained, and then used to retrieve the actual item from the content set database. Meanwhile, the information or situation is updated with the student and session databases. The retrieved item is then displayed to the student. The process repeats. When outcomes are obtained and transmitted to the reasoning core, depending on the schedules, different learning mechanisms are triggered. More on this will be discussed later.

In addition to ILMDA, an authoring suite for developers, and a report-and-review suite for instructor and students, and a simulation (Soh and Miller, 2005) have also been implemented as *support* tools for the ILMDA system. The authoring suite is used to author, edit, and upload content sets – i.e. tutorials, examples, and exercise problems. The report-and-review suite is used by students to view their own usage and performance statistics, and by the instructor to get a sense of how the individual and average student performances compare. The simulator allowed us to test the machine learning components of the CBRMETAL framework and how they worked together.

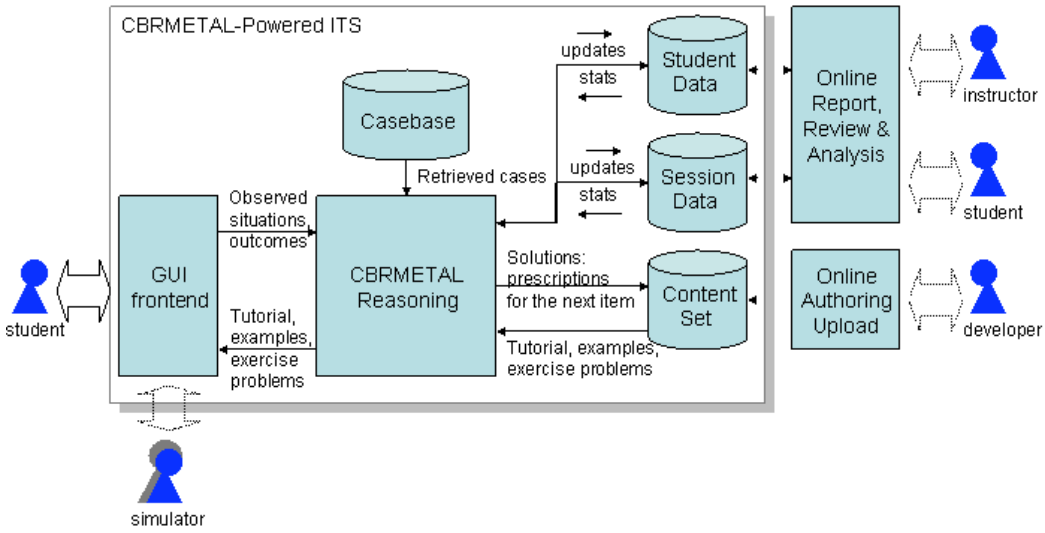


Fig.3. Architecture of our CBRMETAL-powered ITS application, ILMDA.

Session Flow

Figure 4 shows a session flow and pages displayed by ILMDA. Each student is required to create a profile in order to log into the system. This password-protected profile holds the student's static information, such as their GPA, name, major, and number of credit hours taken.

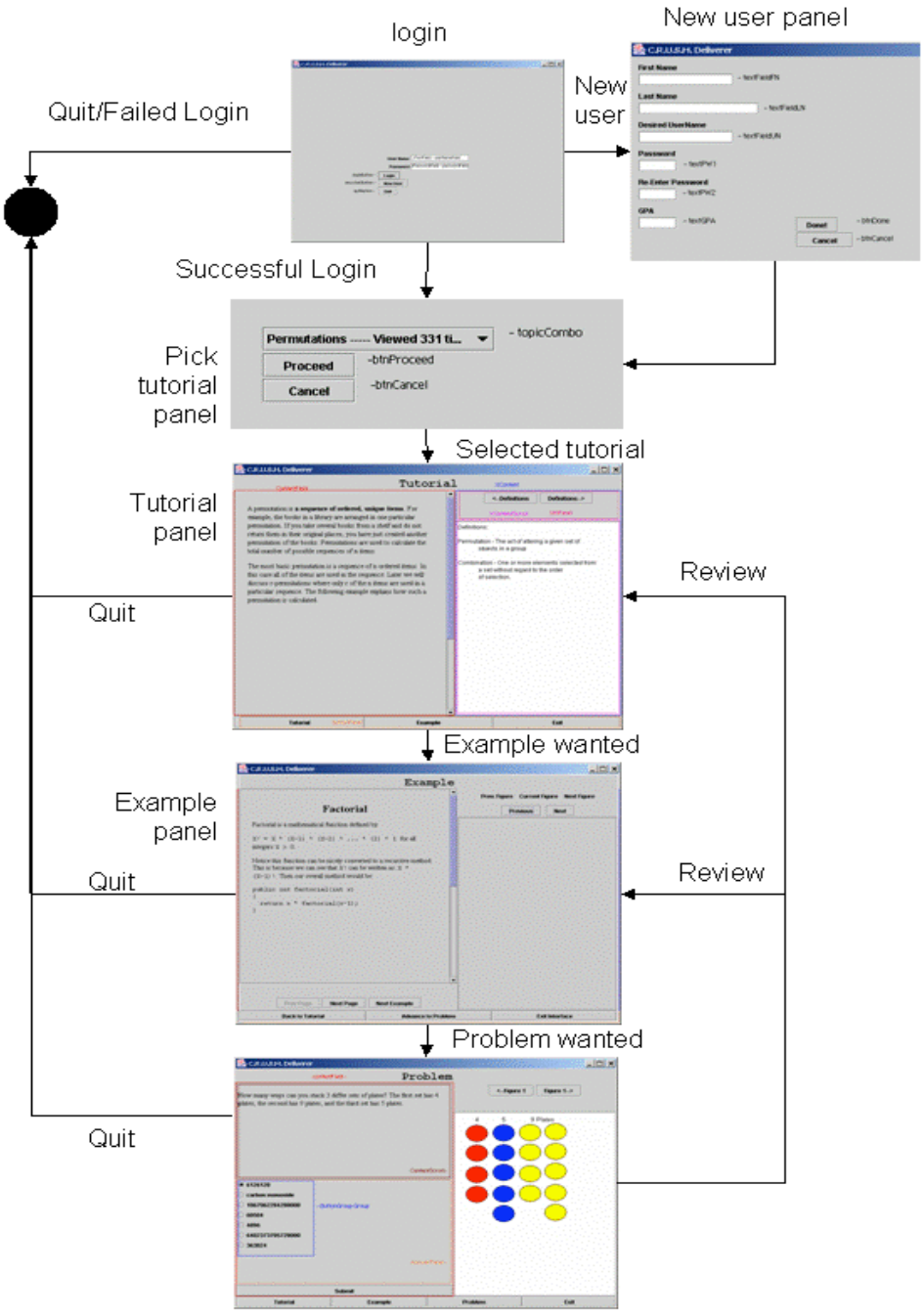


Fig.4. The session flow of the ILMDA application.

These screens display the learning material on the left side which can be several pages long. On the right side of the screens is a panel to display support material such as figures or key terms and definitions. The exercise problem panel also has an answer panel, which displays the multiple choice answers for the exercise problem, and then the explanation of that answer after the student has answered the question.

Once a student is given one example or an exercise problem, they have the opportunity to ask the system for another example or another exercise problem. When a student chooses to view another example, the system records the session as unsuccessful (since the student either did not understand the example, or the example did not adequately explain the topic) and uses its CBR module to retrieve another example, updating the student's usage history and disallowing examples that the student has already seen. The system, however, automatically feeds exercise problems to the student as long as the system thinks the student needs one. When a student chooses to quit an exercise problem or fails to answer an exercise problem correctly, the system records the session as unsuccessful as well. Support material for each tutorial, example, and exercise problem can be either text or an image. Screenshots of the tutorial panel, the example panel, and the exercise problem panel are shown in Figures 5-7, respectively.

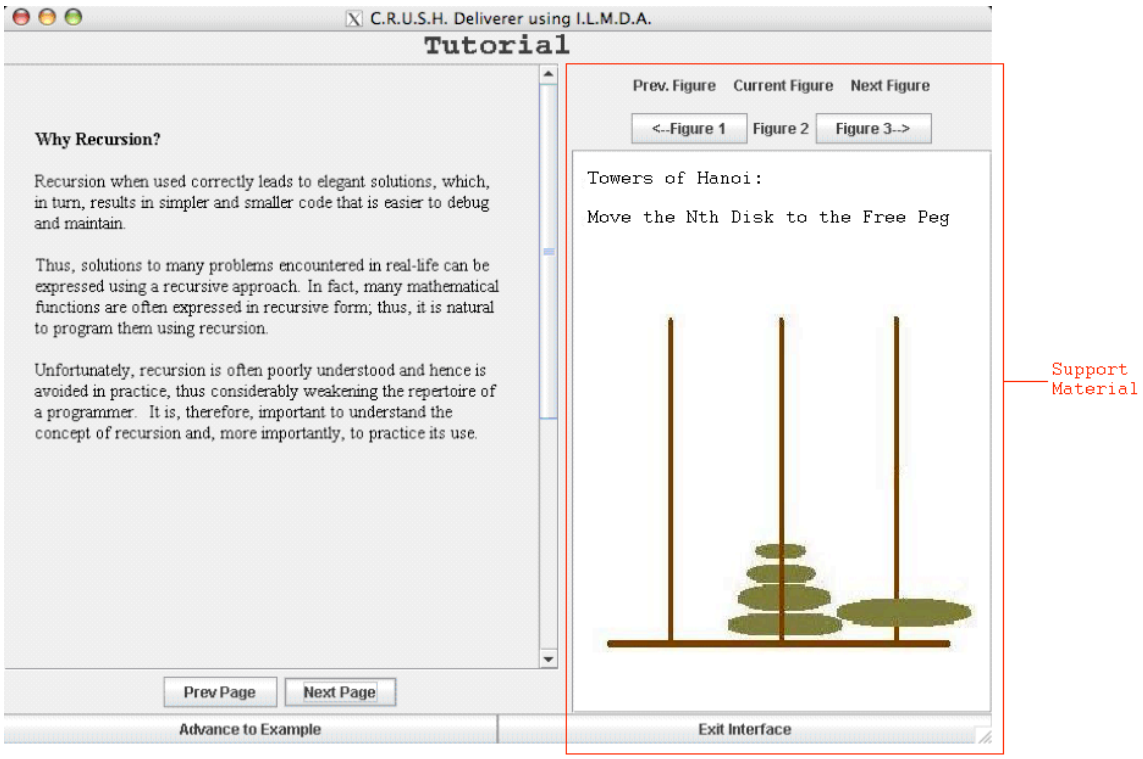


Fig.5. The tutorial panel of the ILMDA application, with support material.

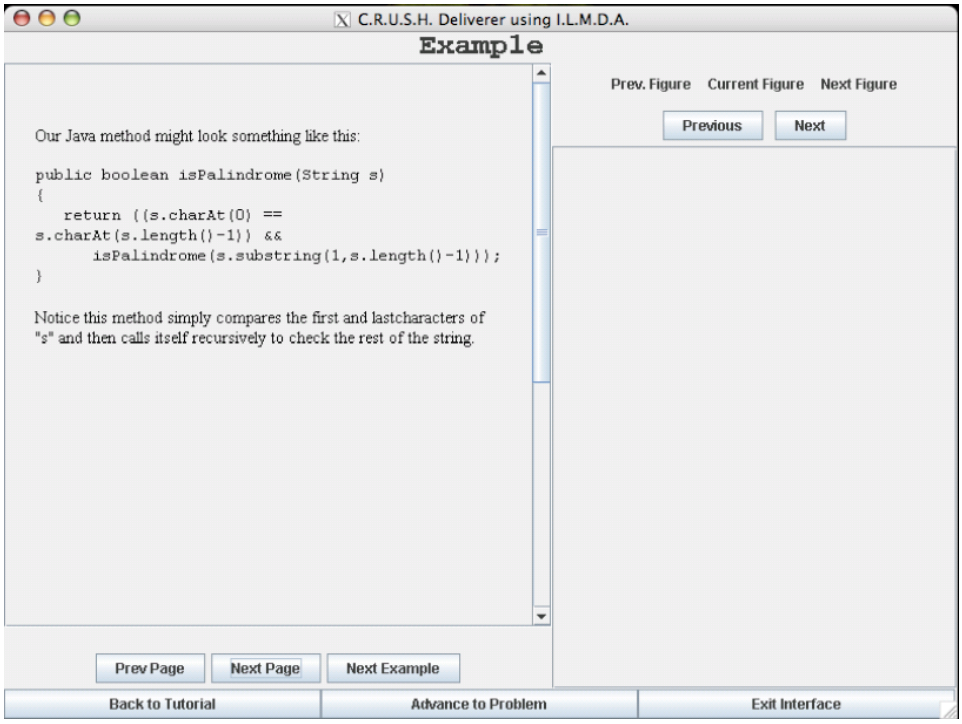


Fig.6. The example panel of the ILMDA application.

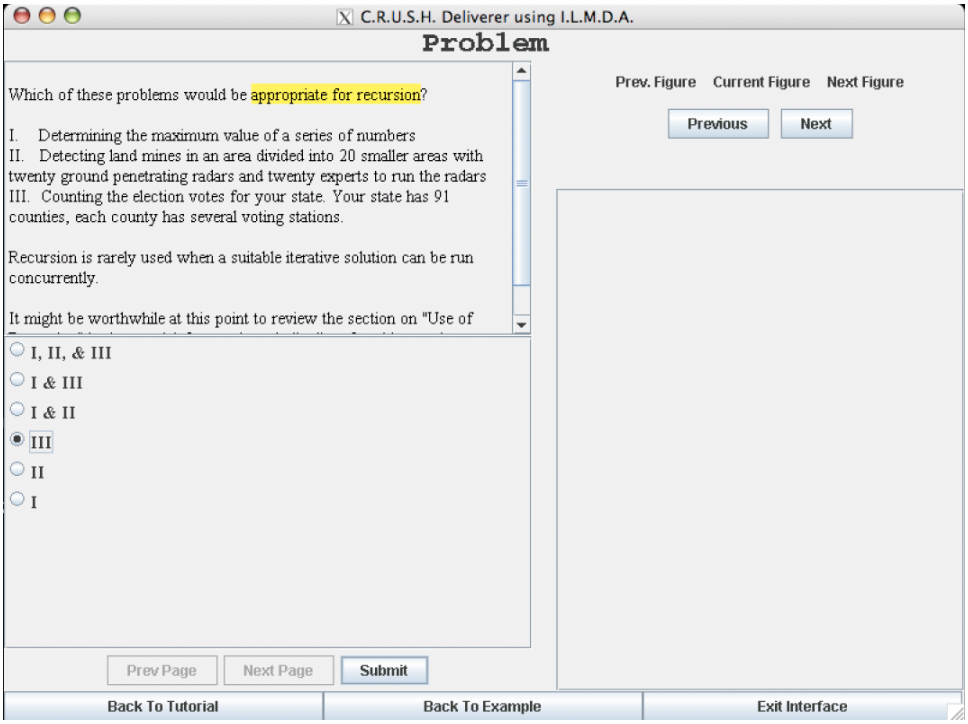


Fig.7. The exercise problem panel of the ILMDA application.

How ILMDA Performs Meta-Learning

As discussed earlier, the design of meta-learning of CBRMETAL is based on a set of cautious and prioritized principles. When a student quits a session, the session's performance data (e.g., number of seconds used, number of mouse clicks in tutorial, number of mouse clicks in examples, number of examples viewed, and so on) is captured and recorded. Also recorded is also the outcome data (e.g., the number of exercise problems that the student answered correctly, the number of exercise problems given, whether the student has answered one of the most difficult exercise problems, and so on). Every time a student clicks "Next Example", or "Next Problem" on the GUI, the reasoning core module composes a new case. This new case is based on the situation prior to that click, the applied solution, the performance, and the outcome. Then, this new case is evaluated against the casebase to determine whether it is worthwhile to store this new case in the casebase following the algorithm outlined in the "How Meta-Learning Is Used" subsection earlier. Thus, case learning impacts the subsequent ILMDA sessions immediately.

Two of the modules, the similarity and adaptation heuristics adjustor modules, are run offline. That is, the weights were adjusted not immediately after each session, but after a number of sessions or a period of time. In our implementation, the five topics assigned to the students over a period of five weeks. The students had about one week to go online and go through a topic. Thus, heuristics adjuster modules were invoked a week after a topic was assigned, or after about 40-50 sessions. Basically, each module would go through the database, combing through each session and propagating the learning affects onto its heuristics.

RESULTS

Fall 2004

In Fall 2004, ILMDA was first deployed at CSCE155 (CS1) at the Department of Computer Science and Engineering at the University of Nebraska. CSCE155 is the first core course for the Computer Science majors. Typically, it has about 150 students per year, with a diverse group of students from majors such as CS, Math, Electrical Engineering, Industrial Engineering, and so on. Further, the programming background of these students is highly diverse. Some incoming freshmen have had some programming in their high schools; some have had none. Thus, it is important for the course to be able to adapt to the different student aptitude levels and motivations. This course is thus well-suited for evaluating the ILMDA application.

For this study, two versions of ILMDA were used: learning and non-learning. The learning ILMDA used the full meta-learning capabilities described in the above sections. It learned new cases and adjusted its similarity and adaptation heuristics. On the other hand, the non-learning ILMDA did not use any of the meta-learning capabilities. The casebase, similarity heuristics, and adaptation heuristics remained unchanged as initialized. Basically, all its learning capabilities were disabled. However, case-based reasoning was still used in *both* versions. For the study, ILMDA was set up such that it automatically toggled on and off the learning mechanisms alternatively for each student session so that both designs handled almost exactly the same number of sessions.

Here we report a case study on the Recursion topic:

- The average numbers of examples and exercise problems that a student read when interacting with the learning ILMDA were 2.216, and 10.946, respectively and that with the non-learning ILMDA were 3.047 and 15.116, respectively. This indicates that the learning ILMDA was able to deliver fewer examples and exercise problems.
- The average percentage of exercise problems answered correctly by a student indicates how well a student did as part of the assessment. When interacting with the learning ILMDA, the percentage was 66.2%. When interacting with the non-learning ILMDA, it was 60.2%. This is encouraging. This hints that the learning ILMDA was able to deliver more appropriate examples and exercise problems. Combining these two observations, the learning ILMDA seems to be effective (delivering more appropriate examples and exercise problems) and efficient (delivering fewer examples and exercise problems).
- Table 3 shows the average number of seconds spent in each section. Students interacting with the learning ILMDA spent more time than students interacting with the non-learning ILMDA. This could be due to the learning ILMDA providing more interesting and appropriate examples and exercise problems, engaging the students to invest more time and effort into reading the materials.

Table 3
Average time spent on each section in seconds for the learning and non-learning ILMDA systems

Section	Tutorial	Per Example	Per Exercise problem
Learning	457	64	74
Non-Learning	276	61	30

To further measure the efficiency and effectiveness of ILMDA's meta-learning capabilities, another aspect of the students session is analysed as follows. First, a specific level of topical comprehension for students to attain is identified: answering one of the most difficult exercise problems correctly. To be effective, an ITS should learn to guide students to reach that level. To be efficient, the same ITS should learn to guide students to reach that level with a small number of exercise problems. Tables 4 and 5 show the results. From Table 4, for File I/O, Exceptions, and Inheritance, the learning ILMDA consistently used fewer exercise problems to bring a student to the level of topical understanding. However, the learning ILMDA did more poorly for the Recursion topic – requiring more than 2 additional exercise problems on average to achieve a level of topical understanding.

It is suspected that the reason for this learning failure is due to the Recursion topic. Unlike the other topics which are closely related to programming, recursion is more closely related to problem solving. It is realized that the learning ILMDA gave students difficult questions more often than easy questions in the Recursion topic, with a correlation of 0.11 between the number of times a question is given and its difficulty level, whereas the same system gave students easy questions more often in other topics (correlations = 0.2–0.56).

Table 5 shows the percentages of failed sessions. A failed session is one in which a student quits before reaching one of the most difficult exercise problems. The results are not conclusive. There is no evidence that learning ILMDA is more effective than the non-learning ILMDA in this regard. It is realized that the cases in the casebase were initially tuned to minimize the number of exercise problems given to the students, thus prompting the ITS to be more aggressive in giving more difficult

exercise problems but more conservative in giving easier exercise problems. Further, in a student profile, a student usually answers questions correctly more often than not since there are more easy questions. Thus, an incorrect answer might not cause ILMDA to change its perception of the student drastically.

Combining the results of Table 5 with that of the case study on recursion, it is concluded that: Even though the learning system was able to increase the percentage of correct answers from the students, it was not able to engage students consistently longer for them to stay on through the set of exercise problems.

Table 4

Average number of exercise problems given by ILMDA to a student to eventually correctly answer an exercise problem with a certain degree of difficulty. Empty cells are due to the fact that there were no exercise problems with that degree of difficulty. Also, we did not include the data for exercise problems with degree of difficulty smaller than 5.5. Results for the Event-Driven Programming topic were not available due to a disk storage problem.

	File I/O		Exceptions		Inheritance		Recursion	
Deg. Diff.	No-L	L	No-L	L	No-L	L	No-L	L
5.5	8.78	6.50	8.75	4.10	3.57	2.50	4.34	6.67
6.0	8.78	6.50	8.75	4.10	3.57	2.50	4.34	6.67
6.5	12.00	11.36	15.28	15.80	4.00	2.50	7.44	10.33
7.0	12.00	11.36	25.66	14.50	6.60	2.75	7.44	10.33
7.5					8.2	7.25	8.00	11.28
8.0					8.2	7.25	8.08	11.28
8.5							13.13	15.10
9.0							13.13	15.10

Table 5

Percentage of failed sessions at various degrees of difficulty for the four topics. For the Inheritance-Learning column, students who quit all quit at level 5.5. Results for the Event-Driven Programming topic were not available due to a disk storage problem.

	File I/O		Exceptions		Inheritance		Recursion	
Deg. Diff.	No-L	L	No-L	L	No-L	L	No-L	L
5.5	41.94%	31.43%	66.67%	40.00%	53.33%	69.23%	40.91%	51.35%
6.0	41.94%	31.43%	66.67%	40.00%	53.33%	69.23%	40.91%	51.35%
6.5	45.16%	37.14%	70.83%	66.67%	60.00%	69.23%	43.18%	59.46%
7.0	45.16%	37.14%	75.00%	66.67%	66.67%	69.23%	43.18%	59.46%
7.5					66.67%	69.23%	45.45%	62.16%
8.0					66.67%	69.23%	45.45%	62.16%
8.5							50.00%	72.97%
9.0							50.00%	72.97%

Initially, at the beginning of the semester, the similarity weights on the situation parameters were set to 1.0; that is, all situation parameters were considered equally important. At the end of the semester, however, the learning ILMDA modified these weights based on its observation of how many times a retrieved case was useful, as shown in Figure 8. It was found that GPA was not that important (dropping from 1.0 to 0.46), quitting an exercise problem before answering it was important

(increasing from 1.0 to 2.54), the number of times a student went back-and-forth between an example and the tutorial was quite important (0.80) but not as important as the back-and-forth between an exercise problem and the tutorial (0.99), and so on. These results are empirical data, key to the self-evaluation capabilities of the learning ILMDA to meta-learn about when to use which pedagogical strategies.

Note that GPA's importance as a situation parameter was reduced by ILMDA. This might be due to the nature of the GPA values. This study was conducted on CS1, the first CS course for our students. Most of these students were first-semester freshmen college students, and thus the GPAs that they reported were most likely their high school GPAs, and thus GPA turned out to be a lesser parameter in this study. It is possible that if we had conducted another study on juniors or seniors in a different course, this GPA parameter could have had a more important role. Here the learning results for the adaptation heuristics are discussed. There are 9 sets of heuristics, and each heuristic is triggered by a subset of about 20 situation parameters. Figure 9 shows the example of how two sets of heuristics changed after learning.

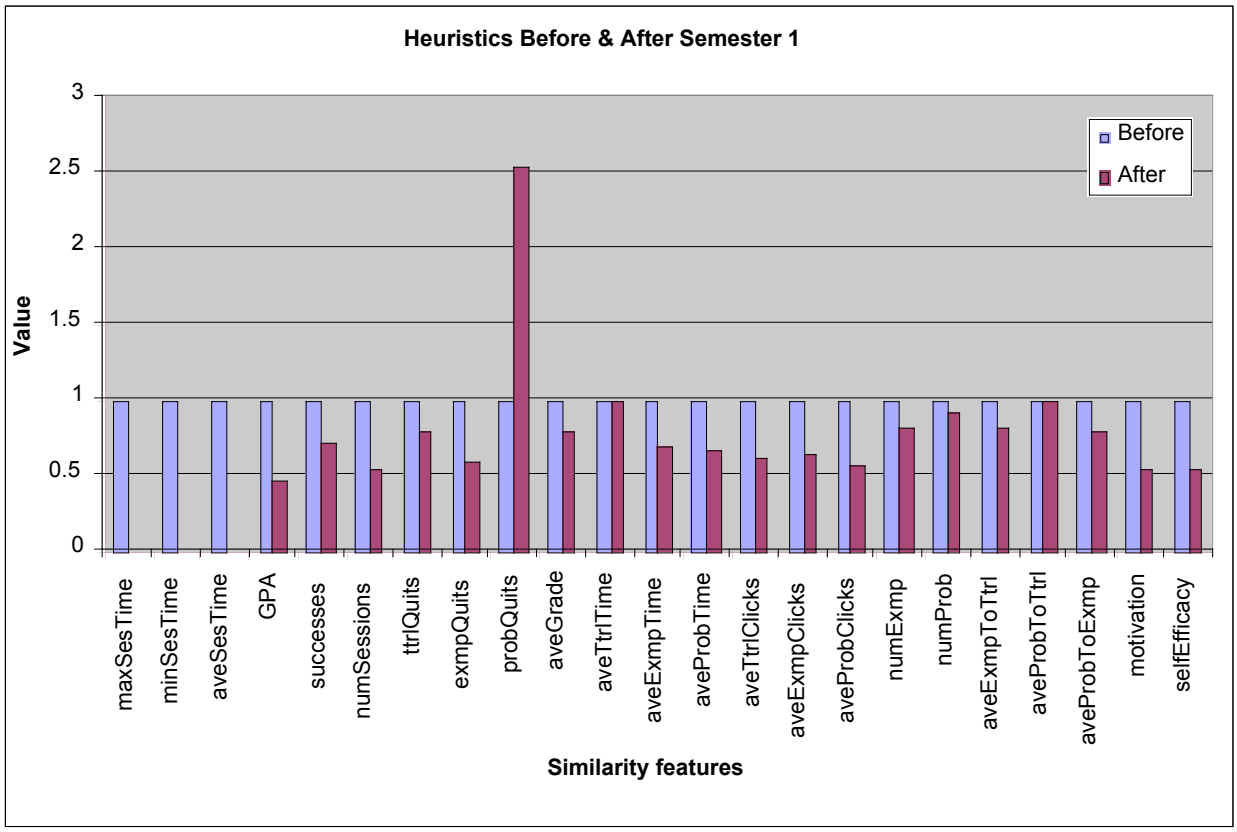
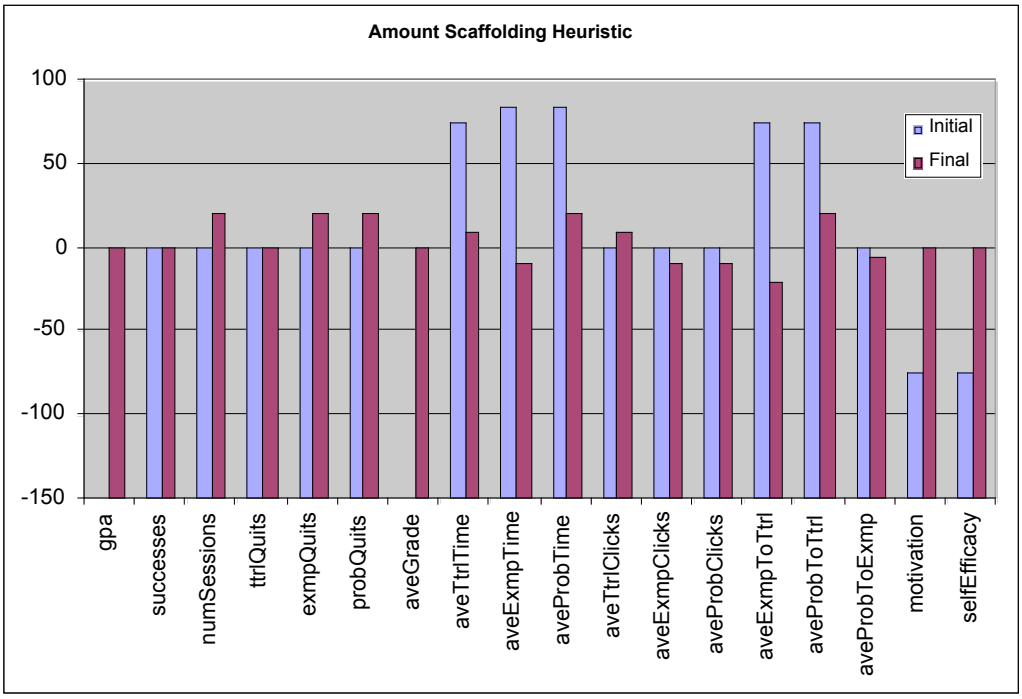
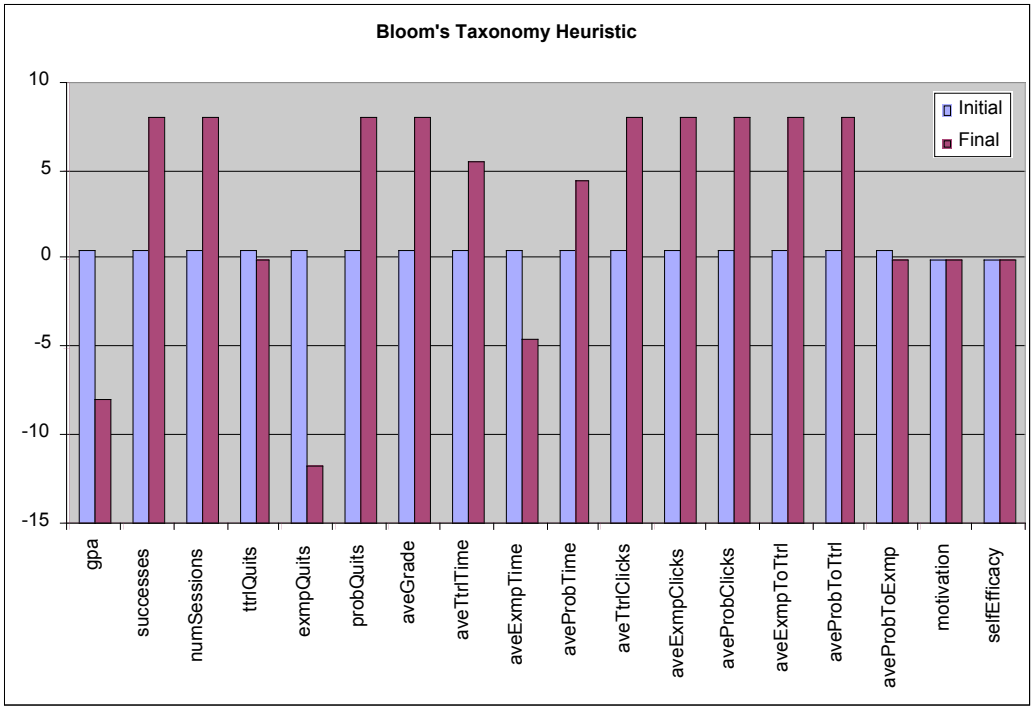


Fig.8. Changes in the similarity heuristics after meta-learning.

As shown in Figure 9(a), an example is to adapt the solution parameter SCAFFOLDING. Initially, it had been expected that if a student is observed to have spent more time on a tutorial, example, or exercise problem, or if a student is observed to go back-and-forth between example and



(a)



(b)

Fig.9. Changes in the adaptation heuristics after meta-learning: (a) BLOOM, and (b) SCAFFOLDING.

tutorial, or between exercise problem and example, then the ITS should provide a larger amount of scaffolding. However, after a semester of using ILMDA, this expectation was not met for "aveExmpTime" (i.e. AVE_EXMP_TIME, average time spent on an example), "expToTtrl" (number of times going from example to tutorial, and "probToExmp" (number of times going from exercise problem to example). It was realized that when students were found to spend more time on examples, or were found to refer back to the examples, they were more likely to quit ILMDA before answering questions correctly. As a result, the learning ILMDA adjusted its adaptation heuristics to reduce the amount of scaffolding, thinking that this would improve its performance.

As shown in Figure 9(b), initially, for the heuristic that adapted the solution parameter BLOOM, at the beginning of the semester, the weight of each situation parameter of the heuristic was set at 1.0. At the end of the semester, this number increased to 9.0. A significant change in the weight of "exmpQuits" (i.e. EXMP_QUITS, the number of quits during an example), going from 1.0 to -11.76, was also observed. This indicates that the more often a student has quit during the viewing of an example, the lower the Bloom's taxonomy level is for the exercise problem to be selected.

Details of the study can be found in (Blank, 2005). Interested readers are also referred to (Soh & Miller, 2005) for an analysis on how student motivation and self-efficacy impact student use of ILMDA.

Spring 2005

In Spring 2005, ILMDA was once again deployed at CSCE155 (CS1) at the Department of Computer Science and Engineering at the University of Nebraska. However, for this study, three versions of ILMDA were used: static, non-learning, and learning. The learning and non-learning ILMDA versions were the same as those used later in the Fall 2006 study. The static version simply used a single tutoring strategy without any case-based reasoning, chosen from one of the original nine cases used in the Fall 2006 study. In the static version, this case was always retrieved and its solution was applied directly without any adaptation. And no new cases were learned.

As alluded to earlier in the discussion of Fall 2004 results, one metric to measure the impact of meta-learning is the average number of exercise problems delivered between a wrong answer and solving an exercise problem correctly. This metric indicates how effective and efficient the system is in adjusting to a failed session (in which a wrong answer was given by the student) to ultimately model the student correctly (in which the student was able to solve the exercise problem correctly). A safe strategy that immediately gave the student the easiest exercise problem to solve after a wrong answer would be the most effective. However, that is not efficient as more exercise problems would have to be delivered in order to bring a student to solve one of the most difficult exercise problems correctly. Our static agent remedies this by always selecting the next easier exercise problem after a wrong answer. Figure 10 shows the comparison.

It is observed that the static ILMDA outperformed the non-learning ILMDA. This indicates that the initial casebase of the non-learning ILMDA was not as effective as the static ILMDA's one single heuristic. Likely, the non-learning ILMDA used CBR to adapt to the different situations poorly, resulting in delivering more exercise problems to the students. However, though given the same poor casebase to begin with, the learning ILMDA outperformed both the static and non-learning ILMDA designs. This is because the learning ILMDA was able to adjust its own adaptation heuristics and learned new cases, thereby improving its performance over time, reducing the number of exercise problems needed before re-capturing a correct profile of the student.

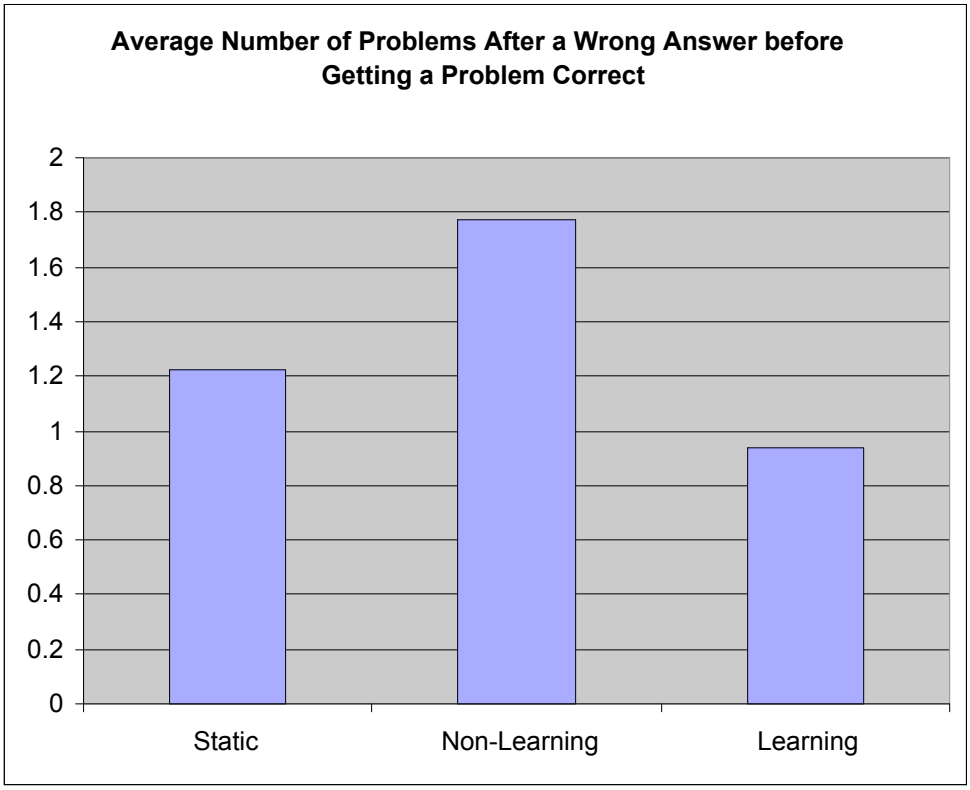


Fig.10. Average number of exercise problems after a wrong answer before getting an exercise problem correct.

To investigate further the role of the meta-learning in ILMDA, the utility of the system's knowledge set is looked at. A knowledge set in ILMDA is defined as the set of adaptation heuristics, the set of similarity heuristics, and the casebase that the system has to work with. For each set, its utility is measured. The utility of a similarity heuristic is its success rate in retrieving a successfully applied case; the utility of an adaptation heuristic is its success rate in contributing significantly to an adaptation of the old solution to the new situation that led to the successful application of the adapted solution; and the utility of a case is the ratio of the number of successful applications of the case over the number of times the case has been retrieved.

Figure 11 shows the comparison between the learning and the non-learning ILMDA systems. First, for the learning ILMDA, as discussed in earlier sections, the adaptation and similarity heuristics are adapted after the conclusion of each topic (offline), which means they were updated five times because of the five topics used in the deployment. The casebase was updated online, immediately after each session. There was a significant improvement from the first knowledge set to the second set, and then there was a slow decline. However, even with the slight decline in outcome, the final knowledge set still outperformed the initial knowledge set significantly.

Worth noticing is the much narrower gap between the performances of the two systems at the fifth topic, only about 0.03. After studying the modified knowledge sets more closely, it was found that the heuristics were modified quite significantly at the fourth topic. The fourth topic was Inheritance & Polymorphism, and upon retrospective analysis, it was found that this topic had the worst-labeled exercise problems: the level of difficulty of each exercise problem was not labeled

accurately. This caused ILMDA to penalize its similarity and adaptation heuristics incorrectly. As a result, when the fifth topic came around, ILMDA had to deal with heuristics that were not as good as before. This is an encouraging design problem: (1) the learning ILMDA is certainly adaptive, and (b) the learning ILMDA can be further refined to quality-tag its learning materials to assign blame for failed sessions.

Details of the study can be found in (Blank, 2005).

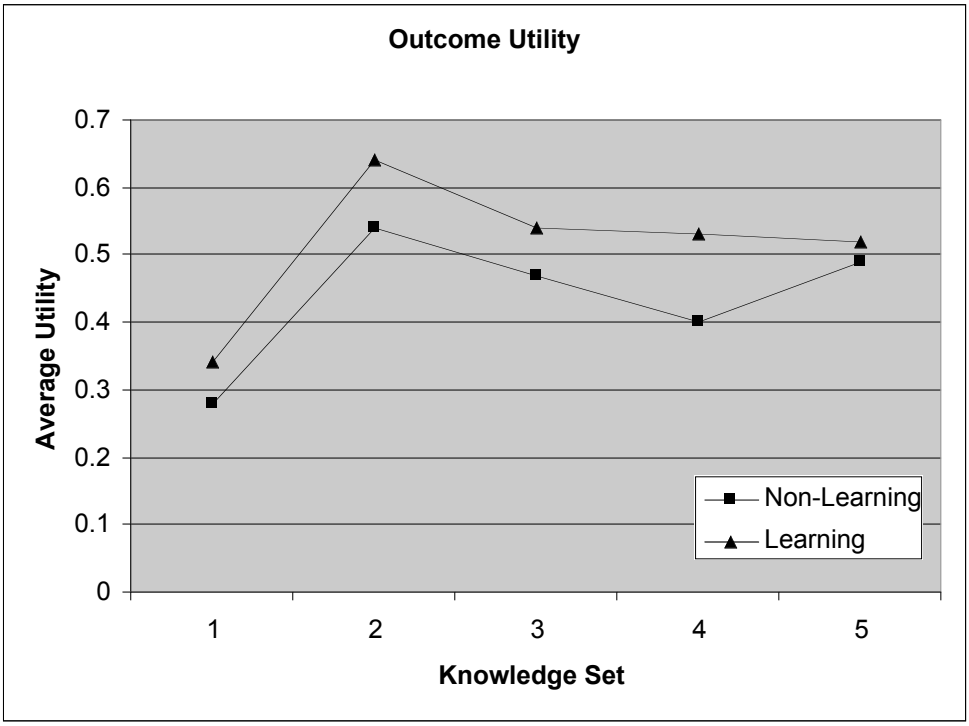


Fig.11. Outcome utilities over the course of five time periods for non-learning and learning ILMDA systems.

DISCUSSION

A comprehensive framework based on integrating case-based reasoning and meta-learning to introspectively refine an ITS's reasoning has been described. This framework, called CBRMETAL, specifically improves the casebase, the similarity heuristics and adaptation heuristics through reinforcement and adheres to a set of six principles to minimize interferences during meta-learning. It has been shown, through an ITS application called ILMDA, implementation, deployment, and two studies, that such a framework can be effective and efficient. A self-improving ITS based on CBRMETAL can modify its pedagogical strategies to improve its performance. Future work includes further deployments to obtain more significant data on student learning, the development of more CS1 topics, and a self-diagnostic component that allows an ITS to pinpoint faulty components in its reasoning.

There is an important issue in terms of the representativeness of the cases in the casebase with respect to the parameters: how many cases do we need to cover the situation and solution spaces properly? If the adaptation heuristics are powerful, then covering the solution space is not as important as covering the situation space, as the CBR system could still reach a viable solution derived from what it has in its casebase and thus would have no need to learn new cases to expand its solution space. Our initial casebase started out with 9 cases. After the semester was over in our first study, there were 103 cases. When we analyzed the standard deviation (or range) values of the situation and solution parameters in these cases, in the initial casebase and then the final casebase, the values changed insignificantly. This means that the system more or less "filled in the gaps" among the initial nine cases. Based on this observation, we would say that the initial casebase was rather sparse. The lack of growth in terms of the situation space could also be because of the narrowness in our content sets and the students. The five topics that we chose were all CS1 topics, and the students were mostly first-semester freshmen. It was true that these students had a diverse background in terms of proficiency in programming and CS knowledge. But still, the situations that the ITS encountered were restricted. From another aspect, we see that this enables the CBRMETAL framework to be used to prototype the knowledge base (i.e. casebase and heuristics) of an ITS for a particular content and for a particular group of students.

A key area that will be focused on to improve ILMDA and also the CBRMETAL framework is more accurate and precise student modeling. For example, in the current implementation, motivation and self-efficacy were self-reported through a survey when a student first registered with the system. These reports were inherently noisy. Further, a student's motivation and self-efficacy might change from one exercise problem to the next, or even after simply reading the tutorial. Right now, such a resolution is not in place in our framework and implementation. Also, in our current implementation, it is assumed that if a student asks for another example, or quit the tutorial, or quit an example, or quit the exercise problems before answering one of the most difficult problems, then the session is considered unsuccessful. However, it has been pointed out to us that it is perfectly possible for a genuinely interested student to ask for additional examples and thus such a request should not be viewed as a failure of the system. Indeed, new cases can be crafted to reflect "genuinely interested students": high motivation and self-efficacy, high number of examples viewed per session, long average time spent on the content, and high average grade on the exercise problems. In short, our cases do not cover a good spectrum of student types sufficiently. Thus, we see better student modeling as an important area in our continuing work to improve the CBRMETAL framework and the ILMDA application.

Our research focus has been to create a framework for an intelligent tutoring system that could improve its performance over time. In our studies, though poor cases were not purposely used in the non-learning or static versions, they were not intentionally optimized either. In retrospect, whether a system could improve itself when it was given a so-so initial knowledge base was our main concern. And from our two studies, there seemed to be indications that the system was able to do so. Whether the system could learn to improve on a good initial knowledge base is uncertain. However, since the case learning is conditioned upon diversity, and the reinforcement learning on heuristics is conditioned upon the session outcomes, conceptually the framework will stop learning – or will only fluctuate minimally – once it realizes that it is doing well. This sets up a safety measure: the system is unlikely to hurt its performance through learning unnecessarily. Actually, for our next steps in the future, we will be more concerned about adaptation across contents and student types. For example, let's suppose that, after using ILMDA for a while and perfecting the casebase and heuristics for students of

CS1, we want to apply ILMDA to elementary students in a geography course with the "perfect" casebase and heuristics and investigate whether the CBRMETAL framework facilitates ILMDA to adapt.

ACKNOWLEDGEMENTS

This work was supported in part by the Computer Science and Engineering Department, the Great Plains Software Technology Initiative, and an Enhancing Teaching and Learning Grant, all at the University of Nebraska. The authors would also like to thank L.D. Miller, Brandy Hauff, Ashok Thirunavukkarasu, and Suzette Person for their contributions to the project. The authors would also like to thank Dr. Gwen Nugent for her review of the initial casebase and the initial set of adaptation heuristics. The authors would also like to thank the anonymous reviewers for their comments that helped improve the paper.

REFERENCES

Arruarte, A., Fernandez-Castro, I., Ferrero, B., & Greer, J. (1997). The IRIS shell: how to build ITSs from pedagogical and design prerequisites. *International Journal of Artificial Intelligence in Education*, 8(3/4), 341-381.

Avesani, P., Perini, A., & Ricci, F. (1998). The twofold integration of CBR in decision support systems. *Technical Report AAAI WS-98-02*. Menlo Park, CA: AAAI Press.

Blank, T. (2005). ILMDA: An Intelligent Tutoring System with Integrated Learning. M.S. Thesis, Department of Computer Science and Engineering, University of Nebraska, Lincoln, NE.

Blank, T., Miller, L.D., Soh, L.-K., & Person, S. (2004). Case-based-learning mechanisms to deliver learning materials. In *Proceedings of the International Conference on. Machine Learning & Applications (ICMLA'2004)* (pp. 423-428). Louisville, KY, December 16-18.

Bloom, B. S., Mesia, B. B., & Krathwohl, D. R. (1964). *Taxonomy of Educational Objectives* (two vols: *The Affective Domain & The Cognitive Domain*). New York: David McKay.

Bonzano, A., Cunningham, P., & Smyth, B. (1997). Using Introspective Learning to Improve Retrieval in CBR: A Case Study in Air Traffic Control. In *Proceedings of the International Conference on Case-Based Reasoning (ICCBR'97)* (pp. 291-302).

Cardie, C. (1999). Integrating case-based learning and cognitive biases for machine learning of natural language. *Journal of Experimental & Theoretical Artificial Intelligence*, 11(3), 297-337.

Cook, J. (2001). Bridging the gap between empirical data on open-ended tutorial interactions and computational models. *International Journal of Artificial Intelligence in Education*, 12(1), 85-99.

Cox, M. T., & Ram, A. (2002). Introspective multistrategy learning: on the construction of learning strategies. *Artificial Intelligence*, 112, 1-55.

Dillenbourg, P. (1989). Designing a self-improving tutor: PROTO-TEG. *Instructional Science*, 18, 193-216.

Elorriaga, J. A., & Fernández-Castro, I. (2000). Using case-based reasoning in instructional planning. Towards a hybrid self-improving instructional planner. *International Journal of Artificial Intelligence in Education*, 11, 416-449.

Gutstein, E. (1992). Using expert tutor knowledge to design a self-improving intelligent tutoring system. In C. Frasson, C. Gauthier & G. I. McCalla (Eds.) *Intelligent Tutoring Systems* (pp. 625-632). Berlin: Springer-Verlag.

Hartman, H. (2002). Scaffolding and cooperative learning. *Human Learning and Instruction* (pp. 23-69). New York: City College of City University of New York.

Jarmulak, J., Craw, S., & Rowe, R. (2000). Self-optimising CBR retrieval. In *Proceedings of the International Conference on Tools with AI* (pp. 376-383).

Kimball, R. (1982). A self-improving tutor for symbolic integration. In D. Sleeman & J. S. Brown (Eds.) *Intelligent Tutoring Systems* (pp. 283-307). Academic Press.

Leake, D. B., Kinley, A., & Wilson, D. (1995). Learning to improve case adaptation by introspective reasoning and CBR. In *Proceedings of the International Conference on Case-Based Reasoning* (pp. 229-240).

MacMillan, S. A., & Sleeman, D. H. (1987). An architecture for a self-improving instructional planner for intelligent tutoring systems. *Computational Intelligence*, 3, 17-27.

Mayo, M., & A. Mitrovic (2001). Optimizing ITS behavior with Bayesian networks and decision theory. *International Journal of Artificial Intelligence in Education*, 12, 124-153.

Melis, E., Andrès, E., Büdenbender, J., Frischauf, A., Goguadze, G., Libbreht, P., Pollet, M., & Ullrich, C. (2001). ActiveMath: a generic and adaptive web-based learning environment. *International Journal of Artificial Intelligence in Education*, 12, 385-407.

Murray, T. (1999). Authoring intelligent tutoring systems: an analysis of the state of the art. *International Journal of Artificial Intelligence in Education*, 10, 98-129.

O'Shea, T. (1982). A self-improving quadratic tutor. In D. Sleeman & J. S. Brown (Eds.) *Intelligent Tutoring Systems* (pp. 309-336). Academic Press.

Park, C.-S., & Han, I. (2002). A case-based reasoning with the feature weights derived by analytic hierarchy process for bankruptcy prediction. *Expert Systems with Applications*, 23(3), 255-264.

Patterson, D., Rooney, N., & Galushka, M. (2002). A regression based adaptation strategy for case-based reasoning. In *Proceedings of AAAI'2002* (pp. 87-92).

Soh, L.-K. & Blank, T. (2005). An intelligent agent that learns how to tutor students: design and results. In *Proceedings of the International Conference on Computers in Education* (pp. 420-427).

Soh, L.-K. & Luo, J. (2004). Cautious cooperative learning for automated reasoning in a multiagent system. *Frontiers in Artificial Intelligence and Applications* (pp. 183-199). Amsterdam: IOS Press.

Soh, L.-K., & Miller, L. D. (2005). Analyzing student motivation and self-efficacy in using an intelligent tutoring system. In *Proceedings of the. International Conference on Computers in Education (ICCE'2005)* (pp. 896-899).

Soh, L.-K., Blank, T., & Miller, L. D. (2005). Intelligent agents that learn to deliver online materials to students better: agent design, simulation, and assumptions. In C. Chaoui, M. Jain, V. Bannore & L. C. Jain (Eds.) *Studies in Fuzziness and Soft Computing: Knowledge-Based Virtual Education*, Chapter 3 (pp. 49-80).

Stahl, A. (2005). Learning Similarity Measures: A Formal View Based on a Generalized CBR Model. In H. Muñoz-Avila & F. Ricci (Eds). *Case-Based Reasoning Research and Development* (pp. 507-521). Lecture Notes in Computer Science 3640. Berlin: Springer.

Tecuci, G., & Keeling, H. (1999). Developing an intelligent educational agent with disciple. *International Journal of Artificial Intelligence in Education*, 10, 221-237.

Virvou, M., & Moundridou, V. (2001). Adding an instructor modeling component to the architecture of ITS authoring tools. *International Journal of Artificial Intelligence in Education*, 12, 185-211.

Vygotsky, L. S. (1978). *Mind in Society*, Cambridge, MA: Harvard University Press.

Watson, I. & Marir, F. 1994. Case-based reasoning: a review. *The Knowledge Engineering Review*, 9(4), 327-354.

Weber, G., & P. Brusilovsky (2001). ELM-ART: an adaptive versatile system for web-based instruction. *International Journal of Artificial Intelligence in Education*, 12, 351-384.

Wettschereck, D., & Aha, D. W. (1995). Weighting features. In *Proceedings of ICCBR'1995* (pp. 347-358).

Woolf, B. P., & Cunningham, P. A. (1987). Multiple knowledge sources in intelligent teaching systems. *IEEE Expert*, 2(2), 41-54.

Woolf, B. P., Beck, J., Eliot, C., & Stern, M. (2002). Growth and maturity of intelligent tutoring systems: a status report. In K. D. Forbus & P. J. Feltovich (Eds.) *Smart Machines in Education* (pp. 99-144). Menlo Park, CA: AAAI Press.

Wu, T. C. (2005). An Introduction to Object-Oriented Programming with Java, 4th Edition. New York, NY: McGraw-Hill.

Zhang, Z., & Yang, Q. (2001). Feature weight maintenance in case bases using introspective learning. *Journal of Intelligent Information Systems*, 16(2), 95-116.