Using Enhanced Concept Map for Student Modeling in Programming Tutors

Amruth N. Kumar

Ramapo College of New Jersey
505 Ramapo Valley Road
Mahwah, NJ 07430
amruth@ramapo.edu

Abstract

We have been using the concept map of the domain, enhanced with pedagogical concepts called learning objectives, as the overlay student model in our intelligent tutors for programming. The resulting student model is fine-grained, and has several advantages: (1) it facilitates better adaptation during problem generation; (2) it makes it possible for the tutor to automatically vary the level of locality during problem generation to meet the needs of the learner; (3) it clarifies to the learner the relationship among domain concepts when opened to scrutiny; (4) the tutor can estimate the level of understanding of a student in any higher-level concept, not just the concepts for which it presents problems; and (5) two tutors in a domain can affect each other's adaptation of problems. Prior evaluations have shown that tutors that use enhanced concept maps help improve learning.

Student Modeling

Traditionally, student models in tutoring systems have consisted of cognitive, affective and inferential components. The cognitive student model has been popularly built as an overlay of the domain model. Researchers have used various organizations for the domain model and the resulting overlay cognitive student model. These representations include conceptual graphs [5], Bayesian networks [22], directed acyclic graphs [10], tables [3] and Prolog clauses [19].

Conceptual graphs have been used because they are graphically inspectable, and facilitate interaction planning and student diagnosis [6]. Bayesian networks have been used to model cause and effect relationships among concepts (e.g., if you know 'for' and 'while' statements, you know loops). Tables and Prolog clauses provide a mechanism to aggregate concepts but do not explicitly represent any inherent relationships among them.

Other organizations used for domain model include networks and trees. In WADEIn [2] used to teach students expression evaluation in C, the student model and domain model are a network of knowledge elements. In ELM-ART [21], overlay of a hierarchical network of concepts is used for declarative knowledge and distributed episodic model is used for procedural knowledge. In KERMIT [8] used to teach conceptual database design, constraints are grouped into a hierarchy where the leaf nodes are individual constraints and intermediate nodes are pedagogically important domain categories. In SAM [10] used to teach the use of a text editor, the student model is a directed acyclic graph, with the leaf nodes being the commands in the editor. The intermediate nodes are partial models introduced explicitly to build a hierarchical structure. They have no pre-defined semantics.

In this paper, we will discuss using the concept map of a domain as the domain model, and an overlay of it as the student model. We will discuss enhancing the concept map with pedagogic concepts called learning objectives in order to help build a finer-grained student model. There are several advantages in organizing the domain model and the overlay student model as a concept map - we will examine these in the context of programming tutors.

Tutors for Program Analysis

We have been developing tutors to help students learn C++/Java by solving problems. Our tutors provide problems on program analysis as opposed to program synthesis, which has been the traditional focus of intelligent tutors (e.g., LISP Tutor [20], ELM-ART [21]). The tutors generate problems, grade the student's answer, provide feedback to the student and log the student's grade. The tutors generate problems as instances of parameterized templates. Each tutor contains 200+ templates. Since each template can be instantiated to generate a large number of syntactically different, but semantically similar programs, each tutor is capable of generating problems ad-infinitum. The types of problems generated by the tutors include: predicting the output of a program, debugging a program, and evaluating an expression. The tutors adapt the problem sequence to the

learning needs of the user. The tutors provide a reified interface for the user to enter the answer – the user enters the output of a program one line at a time, identifies bugs in a program by line number and the program object to which the bug applies, and lists the steps in the evaluation of an expression one operator at a time [13]. For each problem, the tutors log whether the student attempted the problem, the number of steps that the student solved correctly/incorrectly, the bugs the student missed, etc. The tutors use a model-based domain model [11,14,15], i.e., a custom-built interpreter to solve the problems and generate feedback. As part of the feedback, the tutors provide explanation of the step-by-step execution of the program [11]. To date, we have developed and evaluated tutors on arithmetic and relational expressions, selection statements, pretest and counter-controlled loops, C++ pointers [11,12,16], scope and its implementation and parameter passing mechanisms. Intelligent tutors have been categorized as pedagogy-oriented, wherein, the focus is on sequencing and teaching canned content, and performance-oriented, wherein, the focus is on interaction and feedback [18]. Our tutors fall under the performance-oriented category.

Concept Map and Learning Objectives

We use an epistemological map of the programming domain as the concept map. It is a hierarchical tree, with programming concepts as nodes, and is-a and part-of links as arcs. Part-of links may be either and-links or or-links. The concept map differs from a generic conceptual graph (e.g., as used in [6]) or directed acyclic graphs in that it also encodes relationships in the domain (concept map = conceptual graph + domain relationships).

Traditionally, the nodes in a domain model have been correct concepts in the domain and/or errors. So, an overlay student model is tightly coupled with the domain ontology. Each problem might correspond to a correct concept or error, and solving it correctly earns the learner credit for that concept or error. This is the scheme used in ELM-ART [21] for declarative knowledge.

We have enhanced the concept map for our tutors by introducing additional nodes corresponding to learning objectives for each correct concept in the domain. These are pedagogic concepts that must be learned in order to understand a domain concept. For instance, the learning objectives for the + operator are an understanding of correct evaluation, precedence, associativity and coercion. The learning objectives for a variable are an understanding of declaration, assignment and referencing. In addition, we treat each potential error associated with a concept as a separate learning objective, e.g., dangling pointers and memory leak are learning objectives for pointers. Learning objectives of a problem are also the expected outcomes of learning to correctly solve the problem. Learning objectives differ from domain concepts in the following ways:

- Learning objectives are at a level of granularity such that whether a learner has met those objectives can be directly measured. However, the same cannot always be said about the domain concepts themselves - traditionally, errors have been directly measurable, whereas correct concepts have not been. For instance, a tutor can clearly measure whether a learner understands overflow error by analyzing the learner's answer to a problem on overflow error. But, after analyzing the learner's answer to a programming problem, the tutor may at best only estimate how well the student has learned the concept of variables, especially if the learner's answer is only partially correct, or incorrect. In contrast, by probing the learner at the right points in the execution of the program, the tutor can directly measure whether the learner has understood some or all of the stages in the lifetime of a variable - declaration, assignment and referencing. Since these can be directly measured, they are considered learning objectives, and a student's understanding of these objectives can be aggregated to estimate the student's understanding of variables, a domain concept. Typically, multiple learning objectives contribute to each domain concept.
- Problems in a tutor are designed to test domain concepts, not learning objectives. Often, it is infeasible to devise problems for individual learning objectives. E.g., we cannot design a problem to evaluate the precedence of an operator without also evaluating the correct evaluation of the operator.

  Since each domain concept is usually associated with multiple learning objectives, each problem will affect multiple learning objectives. For instance, evaluation of the expression 3 + 4 * 5 will affect all the following learning objectives: correct evaluation of +, correct evaluation of *, precedence of + and precedence of *.

  A single problem may affect the same learning objective multiple times, and in contradictory ways. For instance, consider a problem involving divide-by-zero error on line 13 of a program. If the learner does not identify this error, but instead, identifies a non-existent divide-by-zero error on line 25, the learner is credited as having both missed and incorrectly identified a divide-by-zero error.
- Whereas the concepts in a domain are all unique, the learning objectives need not be. The same learning objective could be associated with more than one domain concept. For instance, precedence, associativity, and correct evaluation are learning objectives that apply to every one of the operators in a language. Similarly, condition is a learning objective that applies to if-statements as well as all the loops in a language.

  We distinguish between two learning objectives by "fully qualifying" them, i.e., appending them to the sequence of concepts that serve as their ancestors in

the concept map. For instance, Operator.Arithmetic.+.Precedence, and Operator.Relational.==.Precedence are two fully qualified learning objectives for expressions.

- Finally, the set of concepts in a domain is determined by the epistemology of the domain. Learning objectives on the other hand are determined by the pedagogic needs for the domain. There is no canonical set of learning objectives for a domain.

Learning objectives are not pre-requisites included in other student models. Whereas pre-requisites suggest the order in which domain concepts should be covered in a topic, learning objectives describe what should be covered about each topic. They are leaf nodes in the concept map.

Our programming tutors automatically build the cognitive student model as an overlay of the domain's enhanced concept map. Each tutor enumerates all the learning objectives in the student model that have not yet been met, and picks the next problem such that the problem addresses one or more of these learning objectives. The problems in the tutors are indexed by fully qualified learning objectives rather than domain concepts, which helps in the retrieval of problems appropriate for the learner. After the student solves each problem, the tutor updates credit for all the affected learning objectives in the student model, and re-calculates the set of learning objectives that remain to be met. Therefore, the tutor continually adapts to the needs of the learner. This adaptation is dynamically determined and not scripted, unlike the sequencing that is implemented using pre-requisites. Therefore, it is more flexible and has a non-deterministic behavior.

This overlay student model has enabled us to integrate the student modeling, credit assignment and learner adaptation of all our programming tutors, including the tutors on operators, selection statements, loops and pointers, even though they are very different in the types of problems they generate, the feedback they provide and the activities they support. For instance, the learner is expected to evaluate expressions in our tutors on arithmetic and relational operators; predict the output of the program in the tutor on while loop; and debug the program in the tutor on pointers. Each topic has its own set of learning objectives, and their presence ensures that the student sees a well-rounded set of problems on the topic. However, the student models that serve these tutors are sub-trees of the domain model for programming. Therefore, the tutors update different parts of the same overlay student model. A typical programming problem may involve learning objectives from different sub-trees in this model, e.g., a problem involving a selection statement inside a loop will affect learning objectives on selection statements, loops, variables and relational operators. When two tutors share learning objectives, e.g., tutors on loops and selection statements both share learning objectives on relational operators, they will influence each other's adaptation of problems.

Recording Student Progress

In the cognitive student model, the progress of a student has been recorded using mechanisms such as slots and values, fuzzy distributions (e.g., [9]) and Bayesian networks (e.g., [22]). Slot and value representations have recorded the percentage of a concept the student knows (e.g., ELM-ART [21]), percentage of a concept the student has covered (e.g., SQL Tutor [14]), and percentage of a concept the student has misconceptions about (e.g., C-POLMILE [4]). Both Bayes networks and fuzzy distributions are good at representing uncertainty, but fuzzy distributions have to be validated empirically and therefore, have not been used as extensively.

In our cognitive student model, we record information about each learning objective as a five-tuple: (G, A, C, I, M), G being the number of problems generated, A attempted, C correctly solved, I incorrectly solved and M missed for that objective. Correct, incorrect and missed are the cognitive counterparts of correct, erroneous and incomplete beliefs in STyLE-OLM [6]. Whenever the learner attempts a problem, we update the five-tuples of all the learning objectives affected by the problem.

Maintaining student progress in this raw form enables us to be flexible about how we interpret it. A few models of interpretation of the data include:

- $A >= M_1$ - ensures that the student has attempted a minimum number of problems for the learning objective.
- $(C / A) >= M_2$ - ensures that the student has solved a minimum number of problems correctly for the learning objective.
- $(C – I) / A >= M_3$ - provides for negative grading.
- $(C – M) / A >= M_4$ - takes consistency of student performance into account.

The constants M_1 … M_4 can be selected appropriately for each topic. Currently, we use a combination of the first two models in our programming tutors, with $M_1 = 2$ and $M_2 = 60\%$.

Advantages of Using the Concept Map

There are several advantages to using an enhanced concept map as the domain and student model. We will briefly discuss some of these advantages in this section.

Most student models are based on independent skill assessments that do not influence each other, or probabilistic models that adequately capture inter-dependencies among concepts, but are intractable for large domains. Concept maps embody the best of both these approaches – they permit individual assessments to influence each other, and are a scalable solution.

A tutor may want to focus on a particular topic in a session, the topic being a subset of the domain model. In order to do so, the tutor has to simply consider the node

corresponding to that topic as the root of the domain model and the overlay student model for that session. This is similar to making a point query on a term in a digraph representing an ontology map of the domain in order to extract the relevant student model [1] - by focusing on only the relevant concepts in the student model, the efficiency of accessing and communicating the student model can be significantly improved.

We may want to use different criteria of proficiency for different concepts in the domain. For instance, we may want the learner to solve at least 2 problems and correctly solve at least 75% of the attempted problems on arithmetic operators, which are relatively easy. For logical operators on the other hand, since they are harder, we may want the learner to solve at least 4 problems and correctly solve at least 60% of the attempted problems. In order to implement differential proficiency criteria, a tutor would have to maintain a look-up table of proficiency criteria for all the concepts in the domain. The advantage of using a concept map as the domain model is that inheritance can be used to parsimoniously specify proficiency criteria. E.g., in the domain model of our tutors, we have specified different proficiency criteria for the nodes - Program, Operators, and Boolean operators. The criteria associated with the Program node are inherited by all its children, such as loops and selection statements. The criteria associated with Operators are inherited by arithmetic, relational and bitwise operators, but not boolean operators since different proficiency criteria have been specified for boolean operators.

A tutor can estimate the proficiency of the learner in any concept that is part of the concept map, not just the learning objectives that are the leaf nodes of the concept map. In order to estimate a learner's proficiency in a concept, the tutor can combine the evidence of the learner's proficiency in all the learning objectives that are leaf nodes in the sub-tree of which the concept is the root. The basis for such combining is the epistemic relationship (is-a/part-of) between the nodes in the concept map. Once again, many different models could be used by the tutor to combine the evidence from the learning objectives (or pedagogic concepts) to estimate a student's proficiency in a domain concept:

- The tutor could calculate the 5-tuple for the concept/root node as the matrix summation of the 5-tuples of all its leaf nodes, and use one of the models described earlier to interpret the resulting 5-tuple;
- The tutor could use a weighted matrix sum instead, the weights representing the relative significances of the various learning objectives to the understanding of the domain concept;
- The tutor could first interpret the data for the various learning objectives (using one of the models described earlier) and estimate the student's proficiency in the domain concept as either the minimum of these interpretations (pessimistic model), the maximum of these interpretations (optimistic model) or a weighted sum of the interpretations.
- The tutor could use fuzzy logic or Bayes theorem to combine evidences. In the latter case, the concept map would serve as the Bayesian network.

The concept map also helps efficiently integrate evidence from different tutors/topics into a single hierarchy and facilitates cross-referencing the evidence when necessary. However, in order to take advantage of this, the various tutors must refer to one persistent student model.

An open student model promotes reflection. It enables the learner to reason about his/her learning, and reconcile his/her idea of progress on a topic with that maintained by the tutor. Open student models are typically rendered graphically to facilitate inspection (e.g., STyLE-OLM [6]). When the open student model is an overlay of the concept map of the domain, and is presented graphically, it also helps the learner understand the relationships among the various concepts in the domain, and situate his/her learning. It could be seen as the next step in the evolution of the cognitive student model - from the student model that merely serves to record the student's performance; to a student model that, when opened up, promotes reflection and meta-cognition; to a student model that, as an overlay of the concept map, also provides instruction and explicates the learning context.

We use our tutors for both tutoring and testing. When used for testing, our tutors do not provide feedback. Typically, before a learner uses our tutors in tutoring mode to obtain adaptive instruction, s/he first uses them in test mode to initialize the student model necessary for adaptation. During tutoring, it is desirable for the tutor to stay focused on one topic at a time, and wait till the learner understands it before moving on to the next topic. So, a high degree of locality among domain concepts is desirable when generating problems for tutoring. On the other hand, during testing, it is desirable for the tutor to shuffle among topics to prevent the student from guessing answers to problems based on the predictability of the sequence in which the problems are generated. So, a low degree of locality among domain concepts is desirable when generating problems for testing. Using a concept map as the domain model makes it easy to vary the degree of locality during problem generation. Consider that the tutor picks one of the many leaf nodes in a tree as the basis for the next problem. For high degree of locality, the tutor uses a sub-tree that is at a greater depth in the concept map, and for a low degree of locality, it uses a sub-tree that is at a shallow depth in the concept map. Therefore, changing the degree of locality can be automated since it translates to selecting a depth in the concept map. Finally, since we use a concept map, at each level of locality, all the leaf nodes considered together are conceptually related, which maintains the coherence of the concepts that are covered during tutoring.

Advantages of Using Learning Objectives

Enhancing the concept map with learning objectives, i.e., pedagogic concepts, has several advantages.

Since the student model is maintained in terms of learning objectives rather than domain concepts or problems solved, it is more fine-grained. Since the student model is updated in terms of learning objectives, and multiple learning objectives may be updated by a single problem, it is loosely coupled with domain concepts and problems solved. The student model does not list the types of problems that have been solved correctly/incorrectly, but rather, the learning objectives that have been met correctly/incorrectly, which is in the spirit of modeling the student's knowledge rather than the student's progress on the presented problems.

Another advantage of using learning objectives is a more fine-tuned adaptation, since adaptation is done based on learning objectives rather than problem types. Since leaming objectives may be shared among different domain concepts, correctly solving problems on one domain concept may affect how problems are generated on another concept. For instance, if the learner demonstrates proficiency in dangling pointers, our tutor on pointers will skip over the problems on correct dereferencing, because these two concepts share learning objectives.

As we had mentioned earlier, learning objectives need not be unique. A learning objective could be associated with more than one domain concept. This occurs when a learning objective is a cross-cutting issue. For instance, factoring out code is an issue that applies to if-statements, switch statements, and all the loops. Efficiency is another issue that applies to all the control structures in a programming language. Lately, the study of these cross-cutting issues has emerged as the new discipline of aspect-oriented programming [7]. In order to build tutors for aspect-oriented programming, we do not have to re-build our domain model. We can simply superimpose an alternative hierarchical tree on the shared learning objectives in our existing domain model. In order to tutor cross-cutting issues, our programming tutor would simply use the alternative tree instead of the domain model. The tutor could also go back and forth between regular programming topics and aspect-oriented programming issues in order to help the learner appreciate the cross-cutting issues in context and build a better-connected ontological map of the domain.

Discussion

We have proposed building the student model as an overlay of the concept map of the domain. We have also proposed enhancing the concept map with learning objectives when using it for student modeling. We listed several advantages of using the enhanced concept map as the domain model, and building the student model as an overlay of it. We illustrated our claims with examples from the programming domain. We have used such a student model to integrate all of our tutors, including the tutors on arithmetic expressions, relational expressions, selection statements, while loops, for loops and pointers. Prior evaluations of these tutors have shown that they help improve learning [11,12,16].

We use a declarative representation for the enhanced concept map that serves as the domain model, and therefore, the overlay student model in our tutors. In order to add new domain concepts or learning objectives to this map, we simply insert them in the appropriate place in the declarative representation - the tutors consult this representation to build the domain and student models that include the newly added concepts/learning objectives. In order to add new problems to a tutor, we simply add them to the template list consulted by the tutor. Since each template must specify the learning objective(s) for which it should be used, the tutor will automatically start using the new templates in the correct context. The addition of concepts and problems is an incremental operation that does not affect any of the existing concepts, problem templates or the adaptive operation of the tutor.

Enhanced concept maps can be used in any domain where the topics can be laid out in a hierarchy. Identifying the learning objectives for a topic is a subjective task, and is best performed by a pedagogic expert.

Currently, we are building a centralized persistent student model that will be used by all our tutors, so that cross-topic adaptation can be facilitated. We are working on making the overlay student model available to the student for inspection and interaction. We also plan to develop additional tutors on functions, boolean operators and arrays that will use the overlay student model based on the enhanced concept map.

Acknowledgments

Partial support for this work was provided by the National Science Foundation's EI Program under grant CNS-0426021.

References

[1] Apted, T., Kay, J., Lum, A. and Uther, J. Visualization of Learning Ontologies, In: U. Hoppe, F. Verdejo and J. Kay (eds.), Artificial Intelligence in Education IOS Press, 2003, 359-361.

[2] Brusilovsky, P., and Su, H. Adaptive Visualization Component of a Distributed Web-Based Adaptive Educational System, S. Cerri, G. Gouarderes, F. Paraguacu (eds.), Proceedings of ITS 2002, Springer (2002), 229-238.

[3] Bull, S. See Yourself Write: A Simple Student Model to make Students Think. In: A. Jameson, C. Paris and C. Tasso (eds.), Proceedings of UM 97, Springer (1997), 315-326.

[4] Bull, S. and McEvoy, A.T. An Intelligent Learning Environment with an Open Learner Model for the Desktop PC and Pocket PC. U. Hoppe, F. Verdejo and J. Kay (eds.), Artificial Intelligence in Education, Vol 97, 2003, 389-391.

[5] Dimitrova, V., Self, J., Brna, P. Applying Interactive Open Learner Models to Learning Technical Terminology. In: M. Bauer, P.J. Gmytrasiewics, J. Vassileva (eds.) Proceedings of UM 2001, Springer (2001) 148-157.

[6] Dimitrova, V., Brna, P. and Self, J. The Design and Implementation of a Graphical Communication Medium for Interactive Open Learner Modelling. S. Cerri, G. Gouarderes, F. Paraguacu (eds.), Proceedings of ITS 2002, Springer (2002), 433-441.

[7] Elrad, T., Filman, R. and Bader, A. Theme: Section on Aspect Oriented Programming, Communications of the ACM, Vol 44(10), 2001.

[8] Hartley, D. and Mitrovic, A. Supporting Learning by Opening the Student Model. S. Cerri, G. Gouarderes, F. Paraguacu (eds.), Proceedings of ITS 2002, Springer (2002), 453-462.

[9] Katz, S., Lesgold, A., Eggan, G., and Gordin, M. Modelling the student in Sherlock II. Journal of Artificial Intelligence in Education, Vol. 3(4), 1993, 495-518.

[10] Kay, J. The UM Toolkit for Cooperative Student Modeling. User Modeling and User Adapted Interaction, 4 (1995) 149-196

[11] Kumar, A.N., Explanation of step-by-step execution as feedback for problems on program analysis, and its generation in model-based problem-solving tutors, Technology, Instruction, Cognition and Learning (TICL) Journal, to appear.

[12] Kumar, A.N., Learning Programming by Solving Problems, in Informatics Curricula and Teaching Methods, L. Cassel and R.A. Reis ed., Kluwer Academic Publishers, Norwell, MA, 2003, 29-39.

[13] Kumar, A.N., A Reified Interface for a Tutor on Program Debugging, Proceedings of Third IEEE International Conference on Advanced Learning Technologies (ICALT 2003), Athens, Greece, 7/9-11/2003, 190-194.

[14] Kumar, A.N., Model-Based Reasoning for Domain Modeling in a Web-Based Intelligent Tutoring System to Help Students Learn to Debug C++ Programs, Intelligent Tutoring Systems (ITS 2002), Biarritz, France, June 5-8, 2002, 792-801.

[15] Kumar, A.N. Generation of Demand Feedback in Intelligent Tutors for Programming. Advances in Artificial Intelligence, Ahmed Tawfik and Scott Goodwin (eds.), Proceedings of The Seventeenth Canadian Conference on Artificial Intelligence (AI 04), London, Ontario, Canada, 5/17-19/2004, Lecture Notes in Artificial Intelligence 3060, Springer, 444-448.

[16] Kumar, A.N., Results from the Evaluation of the Effectiveness of an Online Tutor on Expression Evaluation. Proceedings of The 36th SIGCSE Technical Symposium on Computer Science Education, SIGCSE 2005, St. Louis, MO, February 2005, 216-220.

[17] Mitrovic, A. and Martin, B. Evaluating the Effects of Open Student Models On Learning. In P. DeBra, P. Brusilovsky and R. Conejo (eds.), Proceedings of Adaptive Hypermedia and Adaptive Web-Based Systems, Spring-Verlag, Berlin-Heidelberg, 2002, 296-305.

[18] Murray, T. An Overview of Intelligent Tutoring System Authoring Tools: Updated Analysis of the State of the Art. Chapter 17 in Murray, T., Blessing, S. and Ainsworth, S. (Eds.). Authoring Tools for Advanced Technology Learning Environments. Kluwer Academic Publishers, Dordrecht, 2003.

[19] Paiva, A., Self, J. TAGUS - A User and Learner Modeling Workbench. User Modeling and User-Adapted Interaction, 4 (1995), 197-226.

[20] Reiser, B., Anderson, J. and Farrell, R.: Dynamic student modeling in an intelligent tutor for LISP programming, in Proceedings of the Ninth International Joint Conference on Artificial Intelligence, A. Joshi (Ed.), Los Altos CA (1985).

[21] Weber, G. and Brusilovsky, P. ELM-ART: An Adaptive Versatile System for Web-Based Instruction. International Journal of Artificial Intelligence in Education, Vol 12, 351-384, 2001.

[22] Zapata-Rivera, J.D., Greer, J.E. Inspecting and Visualizing Distributed Bayesian Student Models. In: G. Gauthier, C. Frasson, K. VanLehn (eds.), Proceedings of ITS 2000, Springer (2000) 544-553.