I learn from you, you learn from me: How to make iList learn from students [1,2]

Davide FOSSATI [a,3], Barbara DI EUGENIO [a], Stellan OHLSSON [b], Christopher BROWN [c], Lin CHEN [a] and David COSEJO [b]

[a] *Department of Computer Science, University of Illinois at Chicago*
[b] *Department of Psychology, University of Illinois at Chicago*
[c] *Department of Computer Science, United States Naval Academy*

Abstract. We developed a new model for iList, our system that helps students learn linked list. The model is automatically extracted from past student data, and allows iList to track students' problem-solving behavior in order to provide targeted feedback. We evaluated the new model both intrinsically and extrinsically. We show that the model can match most student actions after a relatively small sequence of observations, and that iList can effectively use the new student tracker to provide feedback and help students learn.

Keywords. Knowledge Representation, Student Modeling, Feedback Generation

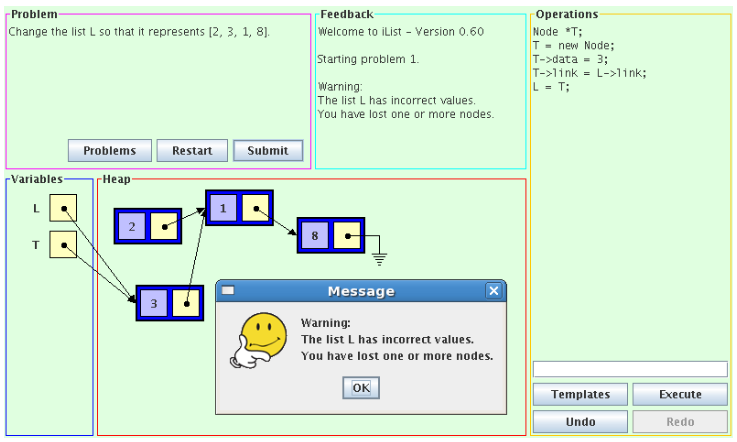
Introduction

In this paper, we address the problem of automatic generation of models that Intelligent Tutoring Systems (ITSs) can use to provide feedback to students. An important characteristic of ITSs is their ability to adapt their instruction to the needs of individual students [15]. In order to do so, the system needs to incorporate computational models of domain knowledge and student knowledge. A traditional and successful approach is model tracing, based on the ACT cognitive theory [1]. A model tracing system explicitly incorporates "expert rules," that encode the possible correct steps to solve a problem; "buggy rules," that model the most common student errors; and a mechanism to trace student actions during the solution of a problem according to the mentioned sets of rules. Effective feedback can be generated from these rules, because the rules are closely related to relevant cognitive processes and important features of the subject domain. An alternative approach to model tracing is constraint-based modeling [11]. In this paradigm, domain knowledge is encoded as a set of constraints that, depending on the context of the student solution, can be irrelevant, satisfied, or violated. Constraint-based tutors can give effective feedback in response to student mistakes.

[1]The iList system is available free of charge. Please contact the authors for further information.

[2]This work is supported by the Office of Naval Research (N000140010640), the Graduate College of the University of Illinois at Chicago (2008/2009 Dean's Scholar Award), and in part by the National Science Foundation (ALT–0536968).

[3]Contact information: davide@fossati.us (http://www.fossati.us/), bdieugen@cs.uic.edu, stellan@uic.edu, wcbrown@usna.edu.edu, linchen04@gmail.com, dcosej1@uic.edu.



Change the list L so that it represents [2, 3, 1, 8].

Welcome to iList – Version 0.60

Starting problem 1.

Warning:
The list L has incorrect values.
You have lost one or more nodes.

```
Node *T;
T = new Node;
T->data = 3;
T->link = L->link;
L = T;
```

Warning:
The list L has incorrect values.
You have lost one or more nodes.

Figure 1. Screenshot of iList

With both approaches, a significant amount of manual knowledge engineering work is required. There are authoring tools for both model tracing and constraint-based tutoring [4,10], but these tools do not yet fully automatize the knowledge acquisition process. Promising steps towards that goal have been taken [16], but we are still far from a practical, completely automatic knowledge acquisition. Researchers in the new field of Educational Data Mining are extracting information from past student data [9,12,14]. Particularly relevant to our work is the work of Barnes and Stamper [2,3], who automatically extracted Markov Decision Processes from past student interactions with their logic proof tutor. Among the differences between our work and theirs, we mention the absence of an explicit reward function in our model; the representation of states, which are "virtual machine snapshots" in our system and sequences of steps in Barnes and Stamper's; and the way we use our model to generate feedback.

This work takes place in the context of the iList project, a system that helps students learn linked lists, a difficult topic in computer science undergraduate curricula [8,7,6]. In previous work, we proved that our system is successful in helping students learn, and we showed that more sophisticated feedback can help improve the effectiveness of the system. Here, we introduce a model that is automatically extracted from past student interactions with iList; we describe how iList uses the model to generate additional feedback; and we evaluate the system in two ways, analyzing the learning curve of the model, and assessing the learning outcomes of students working with our system.

1. Description of iList

The iList system provides the student with a simulated environment where linked lists can be seen and manipulated. Lists are represented graphically, and can be manipulated with programming language commands. Students are asked by the system to solve problems in this environment, such as insert new nodes in a given linked list, remove nodes, or perform other more complicated operations. As a student is working towards a solution, the system provides feedback to help the student make progress.

The graphical interface of iList is divided in four main parts: an area containing the description of the problem to be solved; an area reporting the history of the feedback messages given to the student; an area representing the current state of the linked list virtual machine; and an area in which students can enter commands and see a history of the previously executed operations (Figure 1). Using this interface, students can interactively manipulate the data structures using C++ or Java commands.

In previous work [8,7], we reported on two versions of iList, capable of delivering feedback of increasing complexity, in three main circumstances.

1. The student entered a command that iList could not understand (*syntax feedback*).
2. The student entered a command and iList understood it, but the command could not be executed because of the contingent state of the virtual machine (*execution feedback*). For example, the student might have tried to access a variable that had never been declared, or tried to reference a node that did not exist.
3. The student explicitly asked for his/her solution to be evaluated by pressing the "submit" button on the user interface (*final feedback*).

The main difference between the first version (iList-1) and the second (iList-2) is the sophistication of syntax feedback and execution feedback. The provided messages are different in the two versions, but they are given exactly in the same circumstances.

A limitation of iList-1 and iList-2 is that the systems are not able to evaluate the "semantic" qualities of a student solution before the end. In the new version of iList (iList-3) we wanted to provide semantic feedback while the student is working on a problem. Inspired by cognitive theories and by our empirical study of human tutoring [8,6], we envision two important strategies: *reactive feedback* and *proactive feedback*. We address the generation of reactive feedback in the following two sections, and we briefly mention our plan for generating proactive feedback in the "current work" section.

2. Reactive Feedback Generation

In a tutoring context, reactive feedback is given in response to student actions that were not explicitly prompted by the tutor. In an exploratory environment like iList, this is the dominant case, as students are working on solving problems on their own. With our new strategy, iList evaluates a student's move on two main factors: the *goodness* of a student move and the level of *uncertainty* of the student in making that move. Both factors can be quantitatively estimated from our model, as we will explain in the following section. The goodness of a move is directly related to the probability that a student will eventually reach a correct solution, starting from the current state of his/her solution. Student's uncertainty is estimated by monitoring the time taken by the student to make the move, and the student's "undo behavior." If the time taken by the student is more than a standard deviation greater than the average time spent by past students at that same point, plus a correction factor based on a student's personal history, then the student is considered uncertain. Also, if the student performed an "undo," "redo," or "restart" operation at that point in the past, he/she is considered uncertain there.

More specifically, the feedback generation algorithm works as follows. If the student just got into a "hopeless" state, i.e., a state from which the estimated probability of success is zero, then a *negative feedback* message is generated, to help the student get unstuck. If the student has made a good move, i.e., has improved his/her probability of reaching a correct solution, *and* the student showed uncertainty, then a *positive feedback* message is provided. The rationale is that a student could have performed a correct but tentative move. In this situation, positive feedback can help consolidate correct knowledge that the student has not fully acquired yet. Moreover, it has been recently shown that human tutors may regulate their feedback according to student uncertainty [5].

The new reactive feedback has two main components. The first part is a content-free sentence expressing the goodness of the student's move, such as “Mmmhh... Probably you can't go very far from here” (negative feedback) and “Good move!” (positive feedback). This is followed by a summarization of the effects of that move on the problem state space, for example “Node 2 was pointing to node 1, now it points to node 3. Node 1 was being targeted by node 2 but now it is abandoned.” This explanation is dynamically generated comparing the previous state with the current state, then reporting the differences between those states. The facts to be communicated are chosen using a set of rules. Finally, the surface realization is performed using the SimpleNLG library [13].

3. The New Student Tracking Model

In order to generate the feedback explained in the previous section, we need a model that is able to assess the goodness of a state and that can support the determination of student uncertainty. Traditional model tracing techniques would allow us to do that. However, we wanted to avoid the expensive, time consuming, and rigid process of manually encode procedural models for each problem in our system. A more fundamental reason is that problems in the linked list domain allow a great degree of flexibility, and many different paths can lead to a successful solution. Anticipating all the possible correct and incorrect paths and manually encoding them into the system would be almost impossible. So, we decided to use a machine learning approach to automatically generate a useful model from the past interactions of students with iList.

The core of our model is a probabilistic graph equivalent to a Markov Chain. Its main components are *states* and *actions*. A state is a snapshot of iList's virtual machine, which includes the simulated linked lists. Linked lists in iList are internally represented with graphs. This representation allows the flexibility of modeling unusual or inconsistent linked list configurations, which can happen as a result of student exploratory actions. Actions in iList are first-class objects, which are created by the students from C++ or Java-like commands. Actions can modify a state into a different one. The model is represented with a *simple directed graph* with two types of vertices, *state vertices* and *action vertices*, and the constraint that a state vertex can point only to action vertices, and that an action vertex must point to exactly one state vertex. The set of actions in the graph is associated with a probability mass function. So, each action is associated with the probability that a student will take that action. Figure 2 shows an example of graph for problem 1, generated from only one student session for reading clarity. “Undo,” “redo,” and “restart” operations are not represented in this graph.

The algorithm to build the graph works as follows. First, iList scans and executes the student actions recorded in past log files. For each action, a new state is generated. If the new state can be matched to a state already present in the graph, the frequency of the pre-existing state is updated; otherwise, the new state is added to the graph. A similar procedure is performed for actions. Then, iList checks if a newly added state is a correct solution for the current problem. In that case, the state is connected to a special “success”

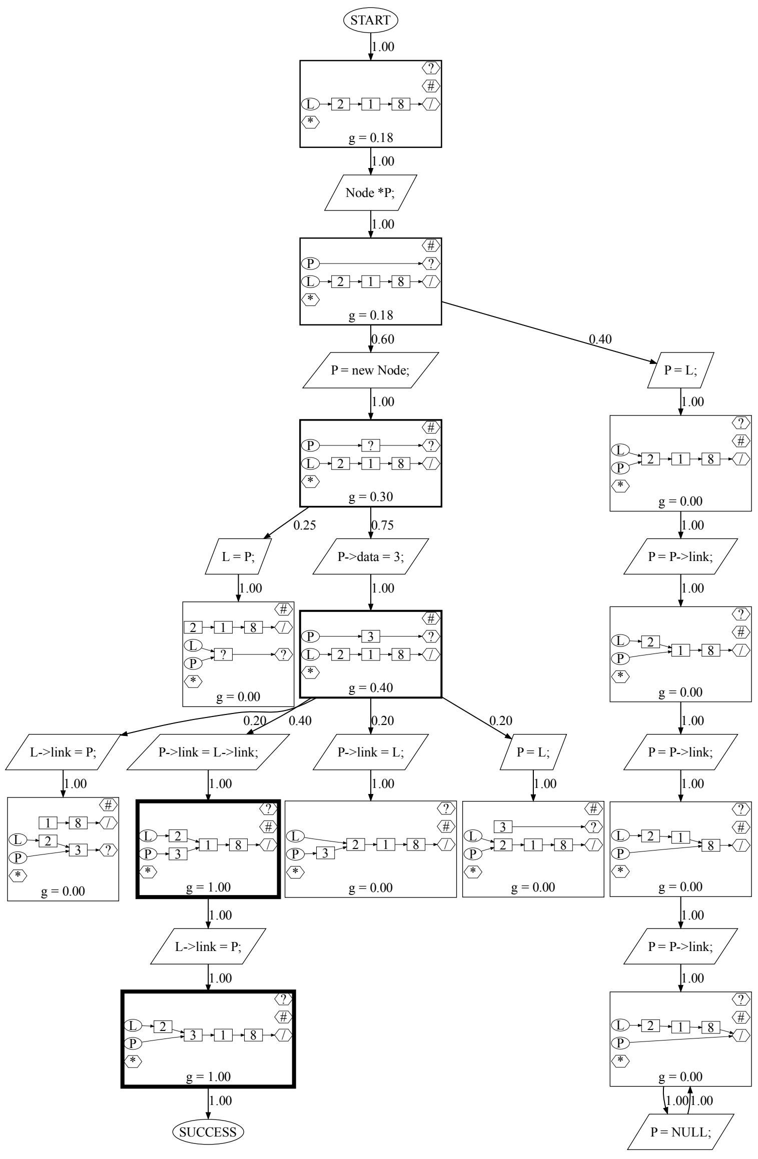


Figure 2. Example of generated graph. The thickness of the border is proportional to the g value

node. Each state gets also annotated with statistics about the time students took to exit from that state. This information is used by iList to assess student uncertainty. Individual students' histories are recorded into separate structures, together with sets of mappings that establish a correspondence between the "real" states in iList and the matched states in the graph, as well as "real" actions and matched actions. The matching process for states and actions is not trivial. We wanted iList to be able to match semantically equivalent state spaces. A state space in iList is also represented with a properly annotated graph. Matching is performed by looking for isomorphism relations between two states. If more than one isomorphism relation is found, iList looks back at the matching history

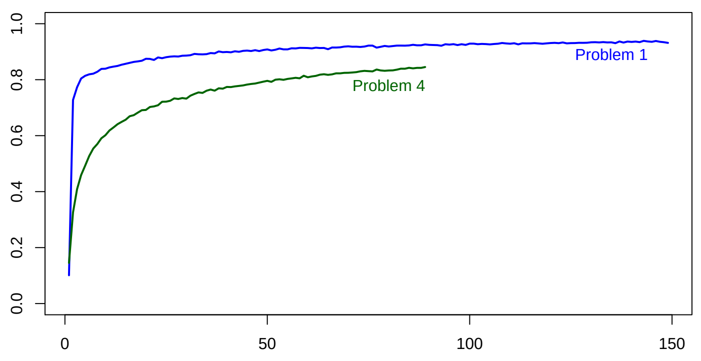


Figure 3. Learning curve of the student tracker model (X = session; Y = match rate)

to disambiguate and choose the appropriate one, which is used as mapping function. Finally, after the construction of the graph is completed, the entire graph is traversed and two important quantities are computed. The frequencies associated to states and actions are converted into probabilities using maximum likelihood estimation. These probabilities are stored in the edges of the graph. Then, iList computes a "goodness" value (g) for each state of the graph. The g value represents a lower bound on the probability that a student traversing that state will eventually reach a correct solution. It is calculated by summing the probabilities of the k most likely paths (with k empirically set to 10) from the current state to the special "success" node to which all the correct states are linked.

At run-time, when a new student comes in, his/her actions are matched against the graph. The comparison between the student's behavior and the model allows iList to make inferences and generate feedback, using the strategy explained in the previous section. Other strategies are possible, as we propose in the "current work" section.

4. Evaluation

We evaluated the new student tracker model in two ways: intrinsically, by assessing the learning curve of the model itself; and extrinsically, by evaluating the learning gain of students working with a version of iList that uses the new model to generate feedback.

Learning curve of the model. For the model to be useful in practice, it is important that a high percentage of the actions of new students can be matched with the model. The main hypothesis behind the model is that even though the state space of the graph is potentially infinite, only a relatively small number of states can represent most student actions. We calculated the learning curve of our models by training them incrementally and counting the percentage of matched states as each session is added to the model. We repeated the procedure 10000 times, randomly shuffling the dataset each time, and averaged the resulting curves. Figure 3 shows the learning curves for the fastest-learned problem (problem 1) and the slowest-learned one (problem 4). In these experiments, the maximum standard error of the mean is less than 0.003, which is too small to be plotted on the figure. Notice that the models can be learned quickly. For problem 1, the 80% match level is reached after just 4 training sessions; the 85% match level after 14 session; and more than 90% of the states can be matched after 40 sessions. For problem 4, the learning rate is slower, with the 80% match level reached after 52 training sessions.

Table 1. Test scores (range: 0 to 1)

Tutor	N	Pre-test score		Post-test score		Gain	
		Mean	Std	Mean	Std	Mean	Std
None	53	.34	.22	.35	.23	.01	.15
iList-1	61	.41	.23	.49	.27	.08	.14
iList-2	56	.31	.17	.41	.23	.10	.17
iList-3	19	.53	.29	.65	.26	.12	.24
Human	54	.40	.26	.54	.26	.14	.25

Notice that the line of problem 4 is shorter, because fewer students worked on problem 4 in our past iList trials. We believe that the steepness of the learning curve depends on the complexity of a problem. On easier problems, students make less "creative" moves than they do on more difficult ones. This leads to more predictable actions and consequently a faster learning of the model.

Learning gain of students. We ran a five-way comparison that included the three versions of iList, a group of students that worked with human tutors, and a control group of students that did an unrelated activity between pre and post-test. The new probabilistic model discussed in this paper is used in iList-3. For more details on testing procedures, human tutors, and control group, see [8,6]. Size of each group, pre-test scores, post-test scores, and learning gains (post-test minus pre-test) are reported in Table 1. ANOVA revealed an overall significant difference across the five groups ($F(4, 238) = 3.70$, $P < 0.01$). Tukey post-hoc test showed only a significant difference between the control group and the human group ($P < 0.01$), and a marginally significant difference between the control group and iList-2 ($P < 0.1$). The difference between iList-3 and the other two versions of iList is not significant, but the progression of P values indicates that the performance of iList-3 is even less distinguishable from human tutors than that of iList-1 and iList-2 (Human vs. iList-1: $P = 0.451$; Human vs. iList-2: $P = 0.738$; Human vs. iList-3: $P = 0.996$). Although this is not strong evidence that that iList-3 is better than iList-1 and iList-2, this result is encouraging. Overall, the performance of iList is very respectable compared to that of our human tutors. So far, we tested iList-3 in a relatively small class, and the group of students that worked with iList-3 is much smaller than the other groups. We are planning to run additional experiments with iList-3 in the future.

5. Conclusions and Current Work

We introduced an effective model that automatically learns a useful probabilistic knowledge representation to enhance the feedback capabilities of iList. Our evaluation showed that the model can be learned with a relatively small dataset, and that students can benefit from the additional feedback messages that can be generated.

Our current work has two main directions. On the one hand, we want to evaluate the performance of the system more thoroughly with more students. In addition to improving the statistical confidence of the results, more interactional data would allow us to discover features of the system that are responsible for the most learning, consistently with the broader goals of our project [8,6]. In particular, we would like to test the impact of positive feedback in iList, and iList-3 is indeed the first version of iList able to deliver a large amount of positive feedback. On the other hand, inspired by our study of human tutoring [8], we are implementing a new tutorial strategy which we call *proactive feedback*. In this kind of interactional episodes, iList will engage students with pedagogically motivated questions, with the goal of guiding them towards a good path. The students will answer the questions, then the tutor will deliver appropriate feedback. The student tracker model described in this paper will provide the context for this type of interaction.

References

[1] J. R. Anderson, A. T. Corbett, K. R. Koedinger, and R. Pelletier. Cognitive tutors: Lessons learned. *Journal of the Learning Sciences*, 4(2):167–207, 1995.

[2] Tiffany Barnes and John C. Stamper. Toward the extraction of production rules for solving logic proofs. In *AIED07, 13th International Conference on Artificial Intelligence in Education, Educational Data Mining Workshop*, pages 11–20, Marina Del Rey, CA, July 2007.

[3] Tiffany Barnes and John C. Stamper. Toward automatic hint generation for logic proof tutoring using historical student data. In *ITS 2008, The 9th International Conference on Intelligent Tutoring Systems*, pages 373–382, Montreal, Canada, June 2008.

[4] Stephen B. Blessing and Stephen Gilbert. Evaluating an authoring tool for model-tracing intelligent tutoring systems. In *ITS 2008, The 9th International Conference on Intelligent Tutoring Systems*, pages 204–215, Montreal, Canada, June 2008.

[5] Kate Forbes-Riley and Diane J. Litman. Analyzing dependencies between student certainness states and tutor responses in a spoken dialogue corpus. In Laila Dybkjaer and Wolfgang Minker, editors, *Recent Trends in Discourse and Dialogue*, chapter 11, pages 275–304. Springer, 2008.

[6] Davide Fossati. The role of positive feedback in intelligent tutoring systems. In *ACL 2008, The 46th Annual Meeting of the Association for Computational Linguistics, Student Research Workshop*, Columbus, OH, June 2008.

[7] Davide Fossati, Barbara Di Eugenio, Christopher Brown, and Stellan Ohlsson. Learning linked lists: Experiments with the iList system. In *ITS 2008, The 9th International Conference on Intelligent Tutoring Systems*, pages 80–89, Montreal, Canada, June 2008.

[8] Davide Fossati, Barbara Di Eugenio, Christopher Brown, Stellan Ohlsson, David Cosejo, and Lin Chen. Supporting computer science curriculum: Exploring and learning linked lists with iList. *IEEE Transactions on Learning Technologies, Special Issue on Real-World Applications of Intelligent Tutoring Systems*, 2009. In press.

[9] Agathe Merceron and Kalina Yacef. Educational data mining: A case study. In *AIED05, 12th International Conference of Artificial Intelligence in Education*, Amsterdam, The Netherlands, 2005. IOS Press.

[10] Antonija Mitrović, Pramuditha Suraweera, Brent Martin, Konstantin Zakharov, Nancy Milik, and Jay Holland. Authoring constraint-based tutors in ASPIRE. In *ITS 2006, The 8th International Conference on Intelligent Tutoring Systems*, pages 41–50, Jhongli, Taiwan, June 2006.

[11] Stellan Ohlsson. Constraint-based student modelling. *Journal of Artificial Intelligence in Education*, 3(4):429–447, 1992.

[12] Dilhan Perera, Judy Kay, Kalina Yacef, and Irena Koprinska. Mining learners' traces from an online collaboration tool. In *AIED07, 13th International Conference on Artificial Intelligence in Education, Educational Data Mining Workshop*, pages 60–69, Marina Del Rey, CA, July 2007.

[13] Ehud Reiter. An architecture for data-to-text systems. In *ENLG07, 11th European Workshop on Natural Language Generation*, Saarbruecken, Germany, June 2007.

[14] Benjamin Shih, Kenneth R. Koedinger, and Richard Scheines. A response time model for bottom-out hints as worked examples. In *EDM08, 1st International Conference on Educational Data Mining*, pages 117–126, Montreal, Canada, 2008.

[15] V. J. Shute and J. Psotka. Intelligent tutoring systems: Past, present and future. *Handbook of Research for Educational Communications and Technology*, pages 570–600, 1996. Macmillan Library Reference USA.

[16] Pramuditha Suraweera, Antonija Mitrović, and Brent Martin. A knowledge acquisition system for constraint-based intelligent tutoring systems. In *AIED05, Artificial Intelligence in Education*, pages 638–645, Amsterdam, The Netherlands, 2005.