Data driven automatic feedback generation in the iList intelligent tutoring system

Davide Fossati
Computer Science
Carnegie Mellon University
Education City, PO Box 24866
Doha, Qatar 24866
dfossati@cmu.edu

Barbara Di Eugenio
Computer Science
University of Illinois at Chicago
851 S. Morgan St.
Chicago, IL 60607
bdieugen@uic.edu

Stellan Ohlsson
Psychology
University of Illinois at Chicago
1007 W. Harrison St.
Chicago, IL 60607
stellan@uic.edu

Christopher Brown
Computer Science
United States Naval Academy
572M Holloway Rd. Stop 9F
Annapolis, MD 21402
wcbrown@usna.edu

Lin Chen
Computer Science
University of Illinois at Chicago
851 S. Morgan St.
Chicago, IL 60607
linchen04@gmail.com

Abstract

Based on our empirical studies of effective human tutoring, we developed an Intelligent Tutoring System, iList, that helps students learn linked lists, a challenging topic in Computer Science education. The iList system can provide several forms of feedback to students. Feedback is automatically generated thanks to a Procedural Knowledge Model extracted from the history of interaction of students with the system. This model allows iList to provide effective reactive and proactive procedural feedback while a student is solving a problem. We tested five different versions of iList, differing in the level of feedback they can provide, in multiple classrooms, with a total of more than 200 students. The evaluation study showed that iList is effective in helping students learn; students liked working with the system; and the feedback generated by the most sophisticated versions of the system is helpful in keeping students on the right path.

Keywords: intelligent tutoring systems, interactive learning environments, educational data mining, feedback generation, procedural knowledge modeling, system evaluation, computer science education.

1 Introduction

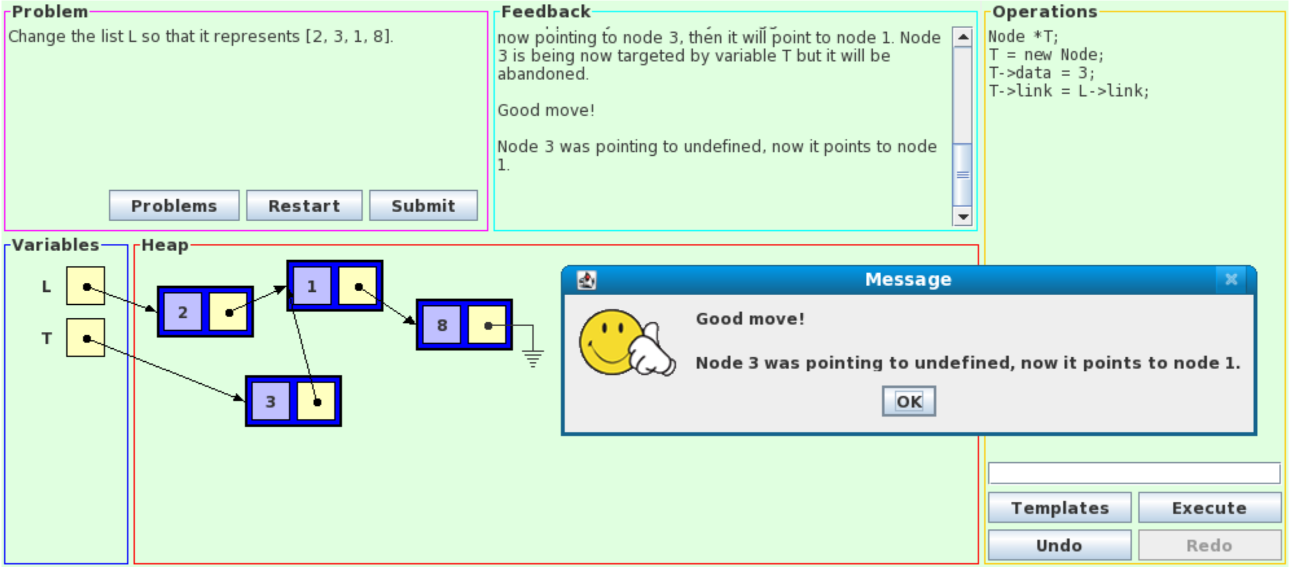
One of the most effective strategies to address weak learning, in any discipline, is the interaction of students with tutors, skilled or even not as skilled [Bloom, 1984, Chi et al., 2001, Lu et al., 2007, Rothman and Henderson, 2011, VanLehn, 2011]. Abundant empirical evidence shows that an essential component of learning from a tutor is the interactive dialogue that occurs, and the pedagogical strategies human tutors employ in such exchanges [Fox, 1993, Jeong and Chi, 2007, Chi, 2009, Di Eugenio et al., 2009, Chi et al., 2011, Lehman et al., 2012, Ezen-Can and Boyer, 2013]. In turn, many researchers have attempted to incorporate human tutoring strategies in Intelligent Tutoring Systems (ITSs), computer systems designed to interact with students and help them learn. Many of these systems have been shown to be effective, although not yet at the level of human tutors [Person et al., 2001, Mitrović et al., 2004, Evens and Michael, 2006, VanLehn et al., 2007, Di Eugenio et al., 2008, VanLehn, 2011].

Many strategies used by human tutors can be directly related to cognitive mechanisms, for example learning from errors [Ohlsson, 1996b, Ohlsson, 1996a, Ohlsson, 2010]. This mode of learning makes use of error signals: environmental events that tell the learner that a tentative problem solving step was inappropriate, incorrect or unproductive. We often broadly refer to such events as *negative feedback*, to include the interlocutor (the tutor) in the learner's environment. A second mode of learning can be named, *uncertainty reduction* [Neves and Anderson, 1981, VanLehn, 1999, Sun et al., 2001]. This mode of learning is driven by information indicating that a problem solving step was appropriate, correct or useful, what we commonly call *positive feedback* [Barrow et al., 2008, Di Eugenio et al., 2009]. We hypothesize that to learn from this type of information, the learner must either create a knowledge structure that recommends the action performed under the relevant conditions; or alternatively, increase the priority associated with the relevant knowledge structure to make its recommendation less tentative.

Not surprisingly then, human tutors provide both negative and positive feedback, and ITSs have attempted at replicating this behavior, although mostly as concerns negative feedback [Anderson et al., 1995]: it is easier to recognize an error than to formulate the specific circumstances under which positive feedback should be provided. Our work is situated in this tradition. We have mined human-human tutoring that we collected in the domain of introductory Computer Science (CS) in order to understand when human tutors provide different types of feedback. That analysis has been presented elsewhere [Ohlsson et al., 2007, Di Eugenio et al., 2009, Chen et al., 2011].

In this paper, we present how a different type of data (rich traces of the steps that students took when solving problems) helped us operationalize the human data findings in order to automatically generate appropriate, modulated feedback in an ITS. These traces provided the basis to build what we call the *Procedural Knowledge Model (PKM)*, that contains information about global students' problem-solving behavior, specifically all possible solution paths that have been observed and their respective goodness towards reaching a solution. We also show that versions of the same ITS that are progressively more sophisticated as concerns tutorial feedback, engender progressively higher student learning (even if not significantly different, but importantly, for the most sophisticated version, indistinguishable from the human tutors). One additional dimension of feedback we will address is the distinction between *reactive* and *proactive*: tutors provide *reactive* feedback when they respond to an actual move (correct or incorrect) that the student has done; *proactive* feedback instead looks to the future, a preemptive action taken by the tutor in anticipation of future student moves.

We address these questions in the context of iList, an ITS that helps students learn linked list, a fundamental Computer Science concept taught in introductory undergraduate curricula [Fossati et al., 2008, Fossati et al., 2009a, Fossati et al., 2009b, Fossati et al., 2010]. We tested five different versions of iList, differing in type and level of feedback they can provide (see Table 1 in the next section), with more than 200 students in multiple classrooms. Versions 3-5 of iList provide feedback based on the PKM, which in turn was mined from the logs of the students interacting with iList versions 1 and 2. We conducted a detailed analysis of the interactive behavior of students working with the system, and correlated it with their learning. Our evaluation study showed that iList is effective in helping students learn; that students liked working with the system; that the feedback generated by the most sophisticated versions of the system is helpful in keeping the students on the right path; and that version 5 of the system generates both more *negative* and *positive* feedback messages than any of the other versions.



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

now pointing to node 3, then it will point to node 1. Node 3 is being now targeted by variable T but it will be abandoned.

Good move!

Node 3 was pointing to undefined, now it points to node 1.

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

Good move!

Node 3 was pointing to undefined, now it points to node 1.

Figure 1: A screenshot of iList

The rest of this paper is organized as follows. Section 2 illustrates iList, and summarizes the different kinds of feedback it can provide. Section 3 is devoted to the PKM and how it was automatically built; Section 4 shows how it is used by iList to generate feedback. Section 5 presents a thorough evaluation of five versions of iList that differ in the kind of feedback they provide. Section 6 discusses our results, and presents current work on ChiQat-Tutor, the successor to iList.

2 The iList system

We developed iList [Fossati et al., 2008, Fossati et al., 2009a, Fossati et al., 2009b, Fossati et al., 2010], an ITS in the domain of linked lists. Linked lists (lists for short) are perhaps the most fundamental *data structure* in CS (*data structures* is the generic CS term used to refer to information storage representations). The main idea behind linked lists is that different pieces of information can be "linked" one after each other and then accessed sequentially. A common graphical representation of lists makes use of boxes and arrows (like those in Figure 1). The iList system allows students to explore and learn about linked lists by operating on scenarios in which lists can be manipulated with programming language commands. The environment is interactive, and the visualization of linked lists is updated in real time according to the actions performed by the students. The system provides a set of problems that can be solved by providing sequences of operations that transform the original lists into the desired configurations.

The user interface of iList is displayed in Figure 1. The top-left corner contains the *problem view*, where a student can select a new problem, read its definition, and submit a solution to the system. The left and central parts of the screen show a visualization of the current *state space* of the simulated environment. The right part of the screen includes the *operation panel*, where students can enter commands to manipulate the state space. Occasionally, iList pops up a new window containing tutorial feedback messages. Those windows disappear when students close them. However, the history of feedback messages is available in the top-center part of the screen, the *feedback view*. One of the main advantages of this user interface is that it makes the abstract concept of linked lists much more concrete. Making a connection between a static data structure and the dynamic procedures necessary to manipulate it can be a challenge. With its interactive interface, iList makes such connections more

Table 1: Feedback types in iList

Feedback type	iList-1	iList-2	iList-3	iList-4	iList-5
Syntax	Minimal	Yes	Yes	Yes	Yes
Execution	Minimal	Yes	Yes	Yes	Yes
Final	Yes	Yes	Yes	Yes	Yes
Reactive procedural	No	No	Yes	Yes	Yes
Proactive procedural	No	No	No	Yes but infrequent	Yes

explicit and accessible.

In appropriate circumstances, iList provides feedback to students. There are five types of feedback: *syntax*, *execution*, *final*, *reactive procedural*, and *proactive procedural* feedback, provided in different combinations by the five different versions of iList we developed (see Table 1).

Syntax feedback is provided in reaction to genuine syntax errors, which occur when a student types in something that can not be interpreted by iList. The parser of iList is fairly resilient, and tolerates minor syntactical imperfections. When the mistake is more substantial, iList tries to match the student's input with a set of error rules that are then used to generate a descriptive error message. Example: "You're trying to write a pointer assignment statement, right? Did you mean '$->$' instead of '>' ?"

Execution feedback happens when a student enters a command that is syntactically correct, but the command cannot be executed because of the current configuration of the state space. A typical case would be a student trying to reference a variable that has not been declared. Example: "Sorry, I can't do that. Variable T does not exist."

Final feedback is delivered when a student believes he or she is done with the problem and clicks the "Submit" button. At this point, iList evaluates the correctness of the solution using a constraint-based approach [Ohlsson, 1992]. If the solution is incorrect, the information from the violated constraints is used to build an explanatory message. Example: "The list L has incorrect values. You have lost one or more nodes."

Reactive procedural feedback is given during the problem-solving process, in reaction to student moves. The system evaluates student moves using a Procedural Knowledge Model (PKM) that is automatically constructed from the interaction with previous students. Reactive procedural feedback is dynamically generated by comparing the state spaces before and after a student's move, and explaining the effects of that move to the student. Reactive procedural feedback can be used to correct a student's mistake (*negative* feedback) or to reinforce the understanding of correct moves (*positive* feedback). Example: "Mmmhh... Probably you can't go very far from here... 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."

Proactive procedural feedback is also given during the problem-solving process, but instead of reactively responding to students' past actions, iList tries to anticipate the *future* moves of a student, and possibly initiates a tutorial interaction with the student. An example of this rather sophisticated interaction is reported later.

Reactive and *Proactive* feedback are clearly the more sophisticated kinds of feedback iList can provide. In part, they are motivated by our analysis of human-human tutorial interactions in the introductory CS domain. For example, whereas many ITSs provide reactive negative feedback when a student makes a mistake, far fewer provide positive feedback; in our study of the human-human tutoring dialogues, we found that positive feedback did correlate with students' learning [Fossati et al., 2009b, Chen et al., 2011]. However, the pedagogical strategies we uncovered do not specify the conditions that trigger this kind of feedback to the level of detail required by a computational implementation. Specifically, to generate this kind of feedback, we need to be able to computationally evaluate the

Table 2: Size of the Procedural Knowledge Models for different problems

Training dataset	Problem	Dataset size		Model size	
		Sessions	Actions	State nodes	Action nodes
iList-1 + iList-2	Problem 1	149	1360	272	353
	Problem 2	155	2688	742	960
	Problem 3	136	1165	269	356
	Problem 4	89	1078	356	467
	Problem 5	65	758	240	290

goodness of a student move and the level of *uncertainty* of the student in making that move. Both factors can be quantitatively estimated from the PKM which we developed by mining the interactions of the students with iList itself.

3 Student tracking

3.1 Procedural knowledge model

Traditional model tracing techniques would allow us to determine goodness of a state and assess the student's uncertainty. However, the process of manually encoding procedural models for each problem is expensive, time consuming, and rigid. Additionally, problems in the linked list domain allow a great degree of flexibility, as 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. Following [Merceron and Yacef, 2005, Barnes and Stamper, 2008, Carlson et al., 2013, Kinnebrew et al., 2013], we applied 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. 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. In reality, the model is generated from the interactions of the students with the first two versions of iList, and the size of training data and resulting models are as shown in Table 2.

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" node. Each state gets also annotated with statistics about the time students took to exit from that state.

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 to disambiguate and choose the appropriate one.

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 log-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. This computation can be done efficiently with a variation of the traditional Bellman-Ford algorithm, a label correcting algorithm that computes shortest paths in weighted directed graphs. The algorithm is implemented in the JGraphT library [Naveh and Sichi, 2003]. Additionally, iList computes and stores the *criticality* (c) of each node. The criticality of a node is the probability that a student will get into a "hopeless" state (a state with $g = 0$) at the next step from the current node. It is calculated by counting the number of direct transitions from the current state to those states with $g = 0$, then dividing it by the total number of transitions from the current state.

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 strategies we discuss in section 4.

3.2 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 to 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 students' 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 all the problems. In these experiments the maximum standard error of the mean is 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 3 training sessions; the 85% match level after 9 session; and more than 90% of the states can be matched after 30 sessions. For the other problems the learning rate is slower, with problem 4 being the slowest. For problem 4, the 80% match level reached after 64 training sessions; the 85% match level after 127 session; and the 90% level is never reached with the available data. Notice that the lines of problem 4 and 5 are shorter, because fewer students worked on these problems in our past iList trials (see Figure 2). We believe that the steepness of the learning curve depends on the complexity of a problem. On easier problems, students make fewer "creative" moves than they do on more difficult ones. This leads to more predictable actions and consequently a faster learning of the model.

4 Reactive and proactive feedback generation

4.1 Reactive procedural feedback

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. iList evaluates a student's move on two main factors: the *goodness* of the move, that we just discussed, and the level of *uncertainty* of the student in making that move.

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 shown that human tutors may regulate their feedback according to student uncertainty [Forbes-Riley and Litman, 2008].

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 [Reiter, 2007].

4.2 Proactive procedural feedback

Sometimes, providing feedback after a student made a move is too late. As we saw in our study of human tutoring [Di Eugenio et al., 2009, Chen et al., 2011], tutors often proactively anticipate the next student move and intervene with guidance to steer them in the right direction. Sometimes, tutors explicitly tell students what to do (direct procedural instruction), other times they try to elicit the right move from the student using more subtle strategies, such as a hint "hidden" within a question or prompt.

In the strategy implemented in iList, we decided to combine elements of direct procedural instruction hidden in a tutor-student interaction which includes the following elements:

1. A question from the tutor. It is composed of three parts:

   (a) a statement of the goal to be achieved by the following move;

   (b) the explicit question about how to accomplish that goal;

   (c) a set of up to four choices, which includes the correct answer and some of the most frequent incorrect answers given by students.

   Example: "Let's see what we can do now... Pointer T is pointing to node 5, we want it to point to null. How would you do that? (1) T = NULL; (2) delete T;"

2. An answer from the student, given by clicking one of the given choices.

3. Feedback from the tutor. If the answer was right, the message is a positive statement such as "That sounds right! I suggest you try it now." If the answer was incorrect, the message points out the mistake and illustrates the consequences of that choice. Example: "Uhmm... This is probably not a good idea. Here is what will happen if you do what you suggested. You will

delete the node that is pointed by pointer T and that contains 2. Variable T is now pointing to node 2, then it will point to garbage."

Once the interaction has started, the student must complete the three steps described above before he/she can continue working on the problem. To decide when to start this type of interaction, iList monitors the student's activity. If the situation is considered *critical* and enough time has elapsed since the last move, iList initiates the proactive interaction.

Remember that at each state iList estimates two useful quantities: g (goodness), the probability that the student will eventually reach a correct solution from the current state; and c (criticality), the probability that the student will make a fatal mistake at the following step. A necessary condition for a proactive interaction to be initiated is that $g > 0$ and $c > 0$, which means that the current state is not hopeless but somewhat critical. The threshold value $c = 0$ for criticality was determined experimentally, by simulating the effects of different threshold values on our training data.

The exact time to initiate a proactive interaction is also determined by the context. The amount of time is determined by four variables: mean and standard deviation of the time that previous students spent into the current state; the criticality of the current state, as defined above; and a measure of the correctness of the current student's behavior B, defined as a smoothed function of how often the student visits "good" or "bad" states (n is the current step, $n-1$ is the previous step, g is the goodness of the current state):

$$B_n = \alpha x + (1-\alpha)B_{n-1} \qquad \alpha = 0.5 \qquad x = \begin{cases} 0 & \text{if } g = 0 \\ 1 & \text{if } g > 0 \end{cases}$$

The delay T after which the proactive interaction will start is determined by the following formula, where μ_T, σ_T are the mean and standard deviation of students' think time in the current state (i.e., the time students spent in that state before moving on to the next); B is the behavior of the current student; c is the criticality of the current state:

$$T = T_{min} + (T_{max} - T_{min})B(1-c) \qquad \begin{aligned} T_{min} &= \max(5, \mu_T - \sigma_T) \\ T_{max} &= \max(5, \mu_T + \sigma_T) \end{aligned}$$

Essentially, the above formula chooses a delay T between plus and minus a standard deviation from the mean student think time in the current state, modulated by the current student's behavior value B. If the student has a generally good behavior (B closer to 1), T will be longer; conversely T will be shorter if the student has a generally worse behavior (B closer to 0).

5 Evaluation

In this section, we evaluate five versions of iList (Table 1). The first two versions (iList-1, iList-2) provide syntax, execution, and final feedback. iList-3 adds reactive procedural feedback. The two most advanced versions (iList-4 and iList-5) can also generate proactive procedural feedback. The only difference between iList-4 and iList-5 is a threshold value that caused iList-4 to generate very few proactive feedback episodes. We evaluate the systems on three student-centered dimensions: *learning*, *satisfaction*, and *problem-solving behavior*.

5.1 Student learning

To measure student learning, all students took a test before and after their interaction with the system. We compared the 5 versions of iList with a control group of students that did not receive any form

Table 3: Learning gain of students in seven conditions

Tutor	N	Pre-test		Post-test		Gain	
		μ	σ	μ	σ	μ	σ
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
iList-4	53	.53	.24	.63	.22	.10	.16
iList-5	30	.37	.24	.51	.26	.14	.17
Human	54	.40	.26	.54	.26	.14	.25

Table 4: Survey, scaled questions (1=No to 5=Yes)

Question	iList-1		iList-2		iList-3		iList-4		iList-5	
	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ
1. Do you feel that iList helped you learn about linked lists?	2.9	1.1	2.9	1.1	3.2	1.4	3.3	1.4	3.9	1.2
2. Do you feel that working with iList was interesting?	4.0	1.1	3.8	1.1	3.3	1.3	3.8	1.2	4.0	1.2
3. Did you read the verbal feedback the system provided?	4.1	1.1	4.1	1.0	3.5	1.5	4.3	1.1	4.6	0.8
4. Did you have any difficulty understanding the feedback?	2.8	1.5	2.9	1.2	3.5	1.2	2.9	1.4	3.4	1.3
5. Did you find the feedback useful?	2.6	1.2	3.0	1.0	2.8	1.4	3.4	1.1	3.4	1.2
6. Did you ever find the feedback misleading?	2.3	1.3	2.4	1.1	3.2	1.4	2.8	1.4	3.2	1.3
7. Did you find the feedback repetitive?	3.8	1.2	3.1	1.1	4.2	1.0	3.2	1.4	3.2	1.3

of instruction between the tests, and another group of students that interacted with a human tutor. The learning gain of the seven groups of students is reported in Table 3.

ANOVA revealed an overall significant difference among the seven groups ($F(6, 319) = 3.04$, $P = .007$). Tukey post-hoc tests revealed significant differences only between the control group and the human tutored group ($P = .004$), and between the control group and iList-5 ($P = 0.021$). The point-to-point differences between the multiple versions of iList is not significant, but the progression of effect sizes indicates an overall positive performance trend. The performance of iList is comparable to human tutors.

5.2 Student satisfaction

After working with iList, students filled in a survey to report their satisfaction with the system. The survey included seven 5-point Likert scaled questions, plus an open ended question asking for general comments on the system. Each individual student saw only one version of the system. Mean and standard deviation of the numeric answers are reported in Table 4.

Overall, students liked the system, especially the latest versions, which they considered more helpful (question 1, ANOVA: $F(4, 209) = 4.44$, $P = .002$; Tukey: iList-5 > iList1, $P = .003$; iList-5 > iList-2, $P = .001$). Interestingly, they found the feedback of the newer versions of iList more useful (question 5, ANOVA: $F(4, 209) = 4.46$, $P = .002$; Tukey: iList-4 > iList-1, $P = .004$; iList-5 > iList-1, $P = .01$) but sometimes misleading (question 6, ANOVA: $F(4, 209) = 4.27$, $P = .002$; Tukey: iList-3 > iList-1, $P = .097$, marginally significant; iList-5 > iList-1, $P = .01$; iList-5 > iList-2, $P = .02$). It is not clear why the feedback may be considered misleading, as iList does not communicate incorrect information. A confusing factor may be that the meaning of the scale is reversed (lower values are better) for that question. Retrospectively we should have rephrased the survey to make the scale consistent. From question 7 (feedback repetitiveness, ANOVA: $F(4, 209) = 5.33$, $P = .0004$), the feedback of the old iList-1 is considered more repetitive that that of iList-2 (Tukey: $P = .01$) and iList-4 ($P = .04$); the feedback of iList-3 is more repetitive than that of iList-2 ($P = .008$), iList-4 ($P = .02$), and iList-5 ($P = .06$, marginally significant). From question 3 (ANOVA: $F(4, 209) = 3.56$, $P = .008$), students reported to have read the feedback of iList-3 less than that of iList-4 (Tukey: $P = .04$) and iList-5 ($P = .006$). The user satisfaction scores dropped in iList-3, most likely due to a severe crash of the lab's network that forced them to repeat the experiment on a different day. This accident shows how the acceptance of a tutoring system can be affected by external factors, which is a concern when migrating from a controlled experimental setting to the real world.

Linear regression showed significant correlations between some of the questions and learning gain. In particular, there is a positive correlation between question 1 (system helpfulness) and learning ($R^2 = .08$, $\beta = .29$, $F(1, 210) = 18.7$, $P < .001$); a positive correlation between question 5 (feedback usefulness) and learning ($R^2 = .02$, $\beta = .15$, $F(1, 210) = 4.94$, $P = .027$); and a negative correlation between question 7 (feedback repetitiveness) and learning ($R^2 = .05$, $\beta = -.23$, $F(1, 210) = 11.3$, $P < .001$).

5.3 Student behavior

We wanted to understand in more detail the features of the student-system interaction that mostly correlate with learning. The interaction of iList and the students was comprehensively logged. From the log files, we extracted several features and compared them using ANOVA and linear regression. We highlight the impact of *problem solving*, *path goodness*, and *positive versus negative feedback*.

Problem solving. The problems included in iList's curriculum are of increasing difficulty, as can be seen from the success rate for each problem (Figure 4). ANOVA revealed overall significant differences among the five groups on the number of problems successfully solved by the students ($F(4, 214) = 19.5$, $P < .001$). Tukey post-hoc test pointed out that all the pairs are significantly different ($P < .05$), except iList-1/iList-4 and iList-3/iList-5. The three groups that worked with a version of iList enhanced with procedural feedback (iList-3, iList-4, and iList-5) generally solved more problems. Linear regression showed a positive correlation between the number of problems solved and learning: generally, students that solved more problems also learned more ($N = 219$, $\beta = .31$, $t = 4.73$, $P < .001$, $R^2 = .09$).

Path goodness. Using the PKM, we can assess the goodness of entire student paths, defined as the average goodness of the states visited by a student while solving a problem. If procedural feedback has an effect, we would see an increase in the average path goodness from the first two versions of iList, which did not provide procedural feedback, and the latest three, which provided procedural feedback of varying degree. Indeed, we observed such an increase (for iList-1 and iList-2 combined, $\mu = .19, \sigma = .13$; iList-3+4+5, $\mu = .31, \sigma = .11$). ANOVA found overall significant differences ($F(4, 214) = 19.1$, $P < .001$). Tukey confirmed all the pairwise differences ($P < .05$), except iList-3/iList-5 and iList-4/iList-5 which are not significantly different, and iList-1/iList-4 which are marginally different ($P = .089$). The difference between iList-1 and iList-2 was unexpected. Linear regression found significant positive correlation between path goodness and learning ($N = 219$, $\beta = .28$, $t = 4.30$, $P < .001$, $R^2 = .08$). This finding supports the validity of the PKM and the strategy of iList, which guides the students towards productive paths.

5.4 Positive and negative feedback

The PKM allows iList to provide more feedback, in particular positive feedback, in addition to the relatively more primitive syntax, execution, and final feedback (Table 1). This makes the latest three versions of iList able to cover more learning opportunities than the first two versions of the system. The different groups of students received significantly different amounts of feedback of every type (all the ANOVAs have overall $P < .001$, pairwise differences vary). Figure 5 illustrates the average number of different types of messages that are provided by the five versions of iList. Recall that iList-1 and iList-2 cannot provide reactive or proactive feedback of any sort, they only provide positive feedback as concerns having solved the problem (*final*), and negative feedback as concerns not having solved the problem (*final*), syntax and execution. This is why in Figure 5, negative feedback is grouped into two types: *syntax + execution*, and *final + reactive + proactive*. As concerns negative feedback on syntax and execution, it progressively decreases from iList-1 to iList-4, but it increases back in iList-5. We do not have a convincing explanation for this finding: apparently the last group of students was more prone to syntax errors. As concerns the second type of negative feedback, we can see that it progressively increases from iList-1 to iList-5. The increase is due to increasingly generating more reactive and proactive negative feedback messages – recall that iList-3 does not generate proactive feedback, but it still generates fewer reactive negative feedback messages than iList-4 and iList-5 (please refer to [Fossati, 2009], specifically Table XIII, for further details). As concerns positive feedback, notice that the number of positive final feedback messages is basically constant across the five versions of iList, since it is almost equal to the number of problems solved by the students: usually a student receives only a single “good job, you have solved the problem” message before moving to the next problem. The amount of positive procedural (reactive + proactive) feedback increases remarkably in the most recent versions of the system. In iList-5, the ratio of positive procedural feedback over negative procedural feedback is approximately 2.5 to 1.

Hence, we have demonstrated that the last versions of iList (4 and 5) generate progressively more reactive and more proactive messages of both types. However, linear regression did not find any significant correlation between the number of feedback messages of various types and learning gain (excluding the positive final messages, because they are equivalent to the number of problems solved).

6 Discussion and Conclusions

In this paper, we presented a study with several important contributions. Our iList tutoring system is among the first systems that tutor Computer Science data structures. The graphical interface of iList is designed to help bring an abstract and difficult concept like linked lists to a more concrete level. Moreover, iList brings together the static representations of lists to the procedures used to manipulate them. An innovative feature of iList is its PKM automatically extracted from previous interactions of students with the system. To date, iList is among the few systems that can build such a statistical model automatically (another relevant example is the logic proof tutor developed by Barnes and Stamper [Barnes and Stamper, 2007, Barnes and Stamper, 2008]) and is the first one that uses it to provide the kind of reactive and proactive procedural feedback described in this paper.

The evaluation of iList was conducted in a “real world” setting, where more than 200 students worked with the system in multiple classrooms. The results showed that iList-5 is as effective as human tutors in helping students learn; students liked working with the system and found it useful; and the procedural feedback automatically generated by the most advanced versions of iList effectively guides students towards the right solution paths. We also showed that these same versions of iList (4 and 5) generate progressively more reactive and more proactive messages, both positive and negative. A limitation of the evaluation study we conducted is its incremental nature. The study was conducted over several semesters across different institutions. Thus, it is difficult to factor out numerous confounding variables such as differences in student population. Future evaluations will be run in a way more conducive to controlling for such differences.

From an instructional perspective, our system initially relied mostly on students’ exploration of its simulated environment. Such exploration was supposed to encourage knowledge construction, according to constructivist learning theories [Piaget, 1954]. However, more recent evidence suggested that minimally guided instruction does not work as well as expected [Kirschner et al., 2006], and students do benefit from some direct guidance from instructors or more experienced peers. In ITSs it is difficult to determine a good balance between exploration and direct guidance. For example, it has been shown that providing on-demand help can also have detrimental effects because it can encourage students to cheat [Baker et al., 2004]. In our experiments with iList we progressively increased the amount of guidance provided by the system, but we always kept the system in control of when to provide hints: no on-demand help was ever implemented. Our results show that this approach is effective. Of course more studies are needed to extend and generalize these findings to other systems and other domains.

The iList project is now completed, but our work on intelligent learning environment to support CS education is not over. iList is the starting point for ChiQat-Tutor, a new system that we recently started developing. ChiQat-Tutor will extend iList in several ways. First of all, ChiQat-Tutor will support additional data structures such as binary search trees and stacks, and algorithmic strategies such as recursion. ChiQat-Tutor will also use additional tutoring strategies such as worked-out examples and analogies. Similarly to iList, ChiQat-Tutor is inspired by findings from our corpus of human tutoring in Computer Science [Di Eugenio et al., 2013].

At this point, iList is a mature system to support learning linked lists. Since the time of the original study presented in this paper, iList has been used by hundreds more students in many different institutions around the world. The system is freely available on the web at *http://www.digitaltutor.net*. We encourage Computer Science instructors to try iList, use it with their students, and give us comments and advice on how to improve the system.

Acknowledgments.

This work was supported by the Office of Naval Research (Awards N00014–07–1–0040 and N00014–00–1–0640); the Graduate College of the University of Illinois at Chicago (Dean's Scholar Award 2008/2009); the National Science Foundation (Awards ALT–0536968 and IIS–0133123); and the Computing Research Association (Computing Innovation Fellowship CIF–186). Current work on ChiQat-Tutor is supported by the Qatar National Research Fund (Award NPRP 5–939–1–115).

References

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

[Baker et al., 2004] Baker, R. S., Corbett, A. T., Koedinger, K. R., and Wagner, A. Z. (2004). Off-task behavior in the cognitive tutor classroom: When students "game the system". In *ACM CHI 2004: Computer-Human Interaction*, pages 383–390.

[Barnes and Stamper, 2007] Barnes, T. and Stamper, J. C. (2007). 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.

[Barnes and Stamper, 2008] Barnes, T. and Stamper, J. C. (2008). 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.

[Barrow et al., 2008] Barrow, D., Mitrović, A., Ohlsson, S., and Grimley, M. (2008). Assessing the impact of positive feedback in constraint-based tutors. In *ITS 2008, The 9th International Conference on Intelligent Tutoring Systems*, Montreal, Canada.

[Bloom, 1984] Bloom, B. S. (1984). The 2 sigma problem: The search for methods of group instruction as effective as one-to-one tutoring. *Educational Researcher*, 13:4–16.

[Carlson et al., 2013] Carlson, R., Genin, K., Rau, M., and Scheines, R. (2013). Student Profiling from Tutoring System Log Data: When do Multiple Graphical Representations Matter? In *The 6th International Conference on Educational Data Mining*, Memphis, TN.

[Chen et al., 2011] Chen, L., Di Eugenio, B., Fossati, D., Ohlsson, S., and Cosejo, D. (2011). Exploring effective dialogue act sequences in one-on-one computer science tutoring dialogues. In *ACL-HLT 2011, 49th Annual Meeting of the Association for Computational Linguistics, Workshop on Innovative Use of NLP for Building Educational Applications*, Portland, OR.

[Chi et al., 2011] Chi, M., VanLehn, K., Litman, D., and Jordan, P. (2011). Empirically evaluating the application of reinforcement learning to the induction of effective and adaptive pedagogical strategies. *User Modeling and User-Adapted Interaction*, 21(1-2):137–180.

[Chi et al., 2001] Chi, M. T., Siler, S. A., Jeong, H., Yamauchi, T., and Hausmann, R. G. (2001). Learning from human tutoring. *Cognitive Science*, 25:471–533.

[Chi, 2009] Chi, M. T. H. (2009). Active-constructive-interactive: A conceptual framework for differentiating learning activities. *Topics in Cognitive Science*, 1:73–105.

[Di Eugenio et al., 2013] Di Eugenio, B., Chen, L., Green, N., Fossati, D., and AlZoubi, O. (2013). Worked out examples in computer science tutoring. In *AIED 2013, 16th International Conference on Artificial Intelligence in Education*, Memphis, TN. Short paper.

[Di Eugenio et al., 2008] Di Eugenio, B., Fossati, D., Haller, S., Yu, D., and Glass, M. (2008). Be brief, and they shall learn: Generating concise language feedback for a computer tutor. *International Journal of AI in Education*, 18(4):317–345.

[Di Eugenio et al., 2009] Di Eugenio, B., Fossati, D., Ohlsson, S., and Cosejo, D. (2009). Towards explaining effective tutorial dialogues. In *CogSci 2009, The Annual Meeting of the Cognitive Science Society*, Amsterdam, The Netherlands.

[Evens and Michael, 2006] Evens, M. and Michael, J. (2006). *One-on-one Tutoring by Humans and Machines.* Mahwah, NJ: Lawrence Erlbaum Associates.

[Ezen-Can and Boyer, 2013] Ezen-Can, A. and Boyer, K. (2013). In-Context Evaluation of Unsupervised Dialogue Act Models for Tutorial Dialogue. In *Proceedings of the SIGDIAL 2013 Conference*, pages 324–328, Metz, France. Association for Computational Linguistics.

[Forbes-Riley and Litman, 2008] Forbes-Riley, K. and Litman, D. J. (2008). Analyzing dependencies between student certainness states and tutor responses in a spoken dialogue corpus. In Dybkjaer, L. and Minker, W., editors, *Recent Trends in Discourse and Dialogue*, chapter 11, pages 275–304. Springer.

[Fossati, 2009] Fossati, D. (2009). *Automatic Modeling of Procedural Knowledge and Feedback Generation in a Computer Science Tutoring System.* PhD thesis, University of Illinois - Chicago.

[Fossati et al., 2008] Fossati, D., Di Eugenio, B., Brown, C., and Ohlsson, S. (2008). Learning linked lists: Experiments with the iList system. In *ITS 2008, The 9th International Conference on Intelligent Tutoring Systems*, pages 80–89, Montreal, Canada.

[Fossati et al., 2009a] Fossati, D., Di Eugenio, B., Brown, C., Ohlsson, S., Cosejo, D., and Chen, L. (2009a). 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*, 2(2):107–120.

[Fossati et al., 2010] Fossati, D., Di Eugenio, B., Ohllson, S., Brown, C., and Chen, L. (2010). Generating proactive feedback to help students stay on track. In *ITS 2010, The 10th International Conference on Intelligent Tutoring Systems*, Pittsburgh, PA. Short paper.

[Fossati et al., 2009b] Fossati, D., Di Eugenio, B., Ohlsson, S., Brown, C., Chen, L., and Cosejo, D. (2009b). I learn from you, you learn from me: How to make ilist learn from students. In *AIED 2009, The 14th International Conference on Artificial Intelligence in Education*, Brighton, UK.

[Fox, 1993] Fox, B. A. (1993). *The Human Tutorial Dialogue Project: Issues in the design of instructional systems.* Lawrence Erlbaum Associates, Hillsdale, NJ.

[Jeong and Chi, 2007] Jeong, H. and Chi, M. T. H. (2007). Knowledge convergence and collaborative learning. *Instructional Science*, 35:287–315.

[Kinnebrew et al., 2013] Kinnebrew, J., Mack, D., and Biswas, G. (2013). Mining temporally-interesting learning behavior patterns. In *The 6th International Conference on Educational Data Mining*, Memphis, TN.

[Kirschner et al., 2006] Kirschner, P. A., Sweller, J., and Clark, R. E. (2006). Why minimal guidance during instruction does not work: an analysis of the failure of constructivist, discovery, problem-based, experiential, and inquiry-based teaching. *Educational Psychologist*, 41(2):75–86.

[Lehman et al., 2012] Lehman, B., D'Mello, S., Cade, W., and Person, N. (2012). How do they do it? investigating dialogue moves within dialogue modes in expert human tutoring. In *ITS 2012, the 11th International Conference on Intelligent Tutoring Systems*, pages 557–562.

[Lu et al., 2007] Lu, X., Di Eugenio, B., Kershaw, T. C., Ohlsson, S., and Corrigan-Halpern, A. (2007). Expert vs. non-expert tutoring: Dialogue moves, interaction patterns and multi-utterance turns. In *CICLing-2007, Eight International Conference on Computational Linguistics and Intelligent Text Processing*, pages 456–467, Mexico City. Best Student Paper Award.

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

[Mitrović et al., 2004] Mitrović, A., Suraweera, P., Martin, B., and Weerasinghe, A. (2004). DB-suite: Experiences with three intelligent, web-based database tutors. *Journal of Interactive Learning Research*, 15(4):409—-432.

[Naveh and Sichi, 2003] Naveh, B. and Sichi, J. V. (2003). JGraphT: A free Java graph library. http://jgrapht.sourceforge.net.

[Neves and Anderson, 1981] Neves, D. and Anderson, J. R. (1981). Knowledge compilation: Mechanisms for the automatization of cognitive skill. In Anderson, J. R., editor, *Cognitive skills and their acquisition*, pages 57–84. Hillsdale, NJ: Erlbaum.

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

[Ohlsson, 1996a] Ohlsson, S. (1996a). Learning from error and the design of task environments. *International Journal of Educational Research*, 25(5):419–448.

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

[Ohlsson, 2010] Ohlsson, S. (2010). *Deep Learning: How the Mind Overrides Past Experience.* Cambridge University Press.

[Ohlsson et al., 2007] Ohlsson, S., Di Eugenio, B., Chow, B., Fossati, D., Lu, X., and Kershaw, T. C. (2007). Beyond the code-and-count analysis of tutoring dialogues. In *AIED07, 13th International Conference on Artificial Intelligence in Education*, Marina Del Rey, CA.

[Person et al., 2001] Person, N. K., Graesser, A. C., Bautista, L., Mathews, E. C., and the Tutoring Research Group (2001). Evaluating student learning gains in two versions of AutoTutor. In Moore, J. D., Redfield, C. L., and Johnson, W. L., editors, *Artificial intelligence in education: AI-ED in the wired and wireless future*, pages 286–293. Amsterdam: IOS Press.

[Piaget, 1954] Piaget, J. (1954). *Construction of reality in the child.* Routledge & Kegan Paul, London.

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

[Rothman and Henderson, 2011] Rothman, T. and Henderson, M. (2011). Do school-based tutoring programs significantly improve student performance on standardized tests? *Research in Middle Level Education Online*, 34(6).

[Sun et al., 2001] Sun, R., Merrill, E., and Peterson, T. (2001). From implicit skills to explicit knowledge: A bottom-up model of skill learning. *Cognitive Science*, 25(2):203–244.

[VanLehn, 1999] VanLehn, K. (1999). Rule learning events in the acquisition of a complex skill: An evaluation of cascade. *Journal of the Learning Sciences*, 8(1):71–125.

[VanLehn, 2011] VanLehn, K. (2011). The relative effectiveness of human tutoring, intelligent tutoring systems, and other tutoring systems. *Educational Psychologist*, 46(4):197–221.

[VanLehn et al., 2007] VanLehn, K., Graesser, A. C., Jackson, G. T., Jordan, P. W., Olney, A., and Rosé, C. P. (2007). When are tutorial dialogues more effective than reading? *Cognitive Science*, 31(1):3–62.

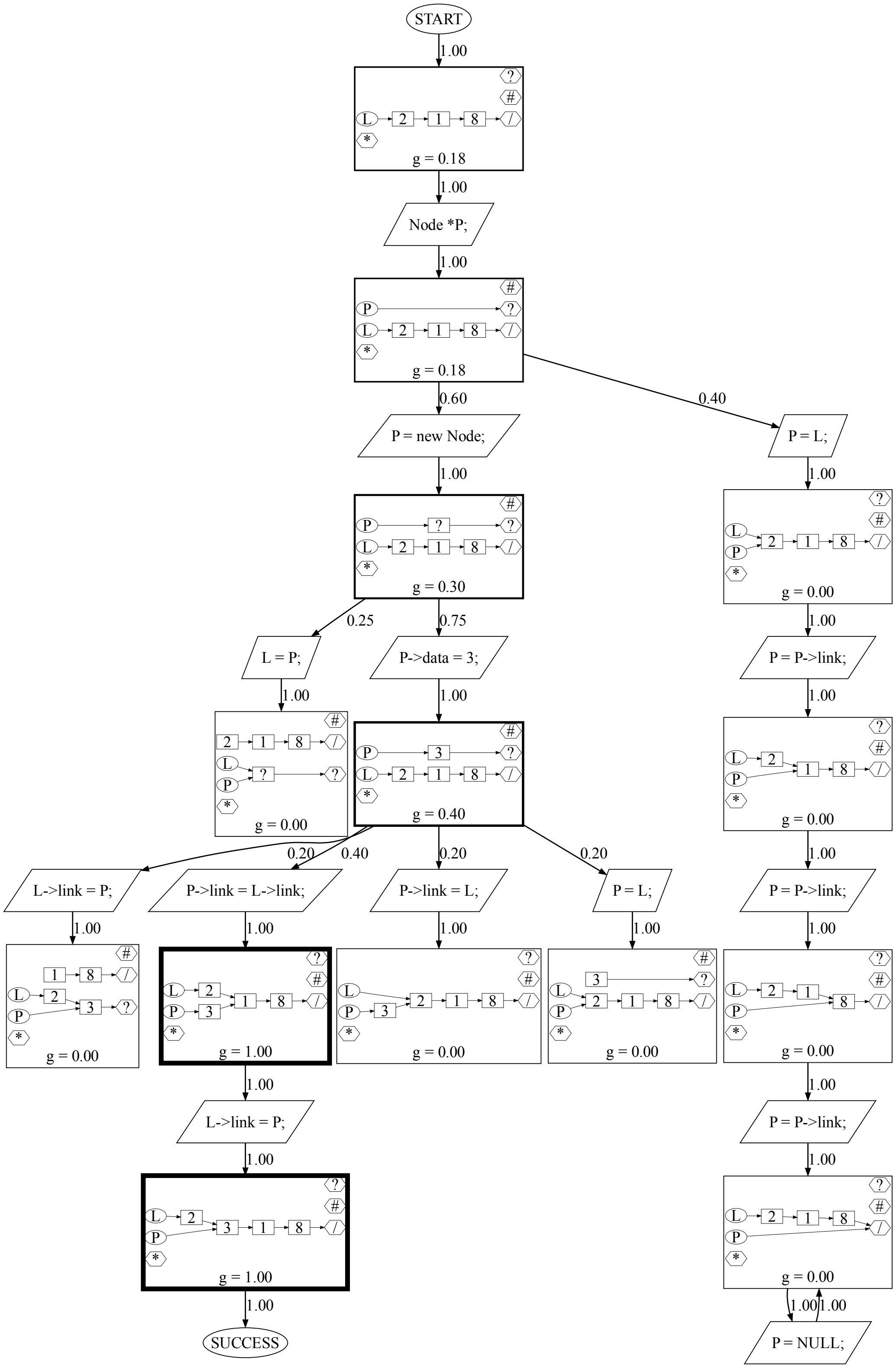


Figure 2: Example of generated graph. The thickness of the border is proportional to the g value

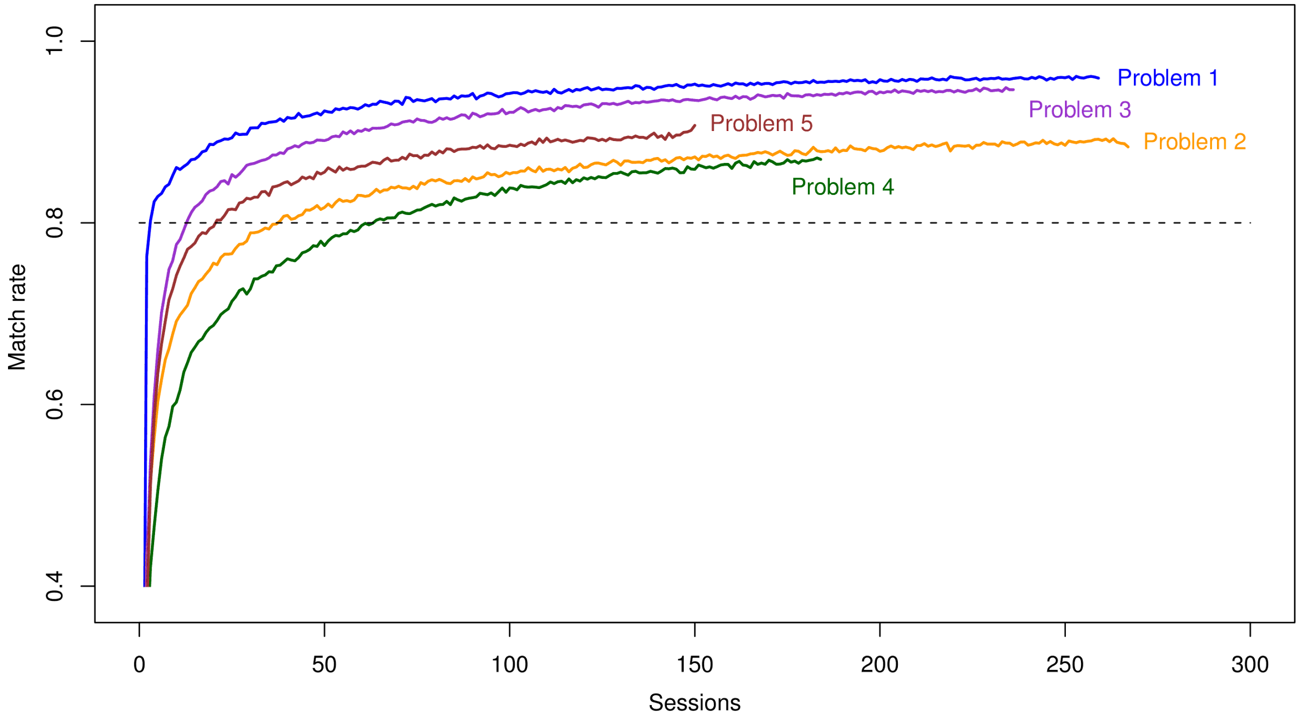


Figure 3: Learning curve of the procedural knowledge model

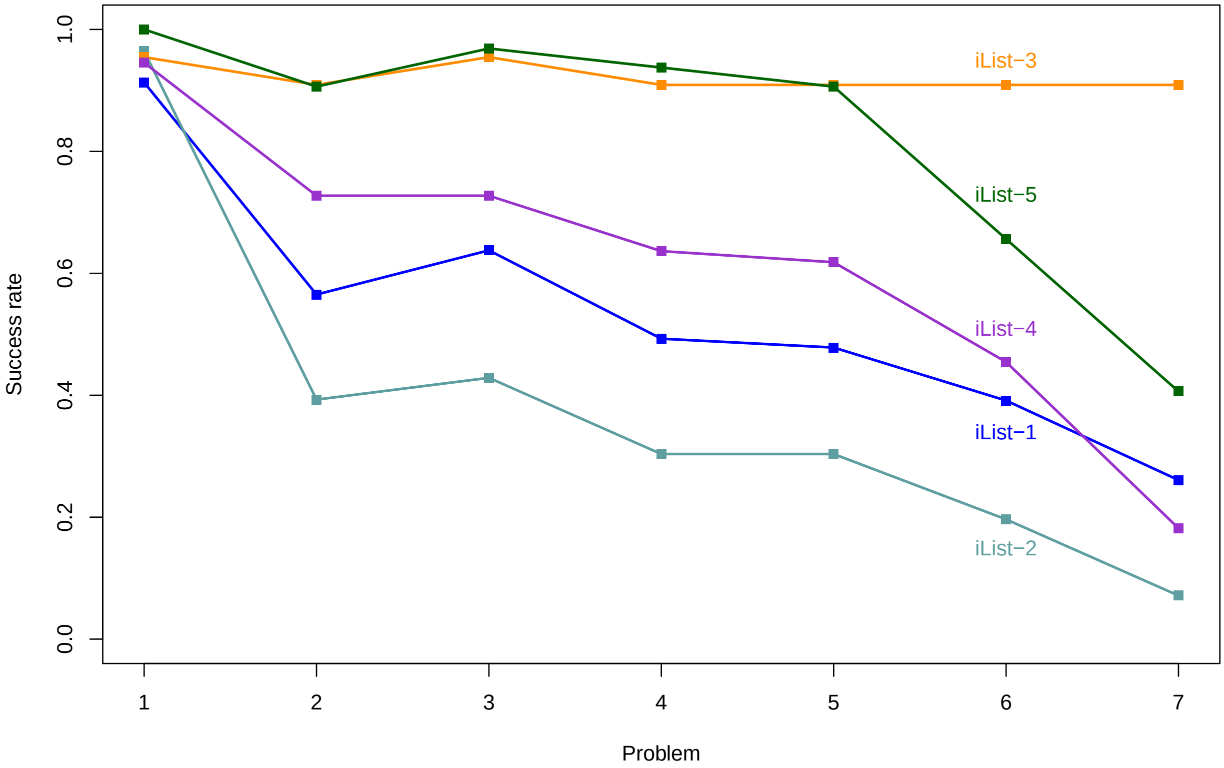


Figure 4: Success rates per problem, per system

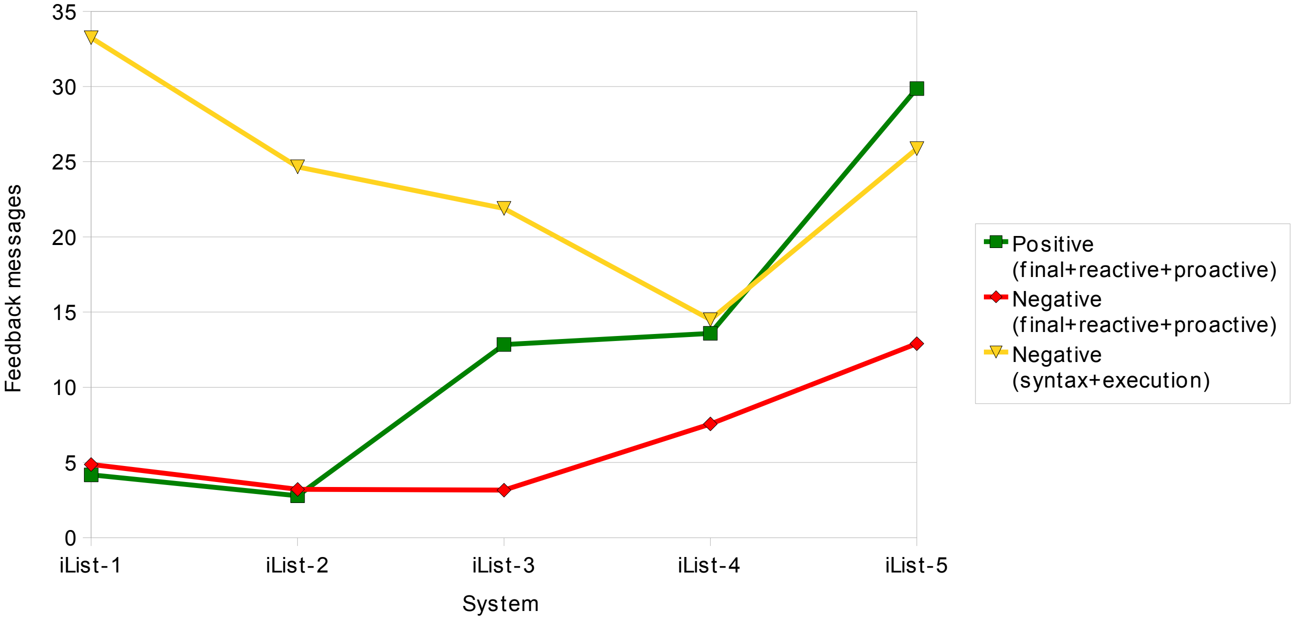


Figure 5: Number of feedback messages, grouped by type, in the 5 versions of the system