Bootstrapping Novice Data: Semi-Automated Tutor Authoring Using Student Log Files

Bruce M. McLaren, Kenneth R. Koedinger, Mike Schneider
Human-Computer Interaction Institute
Carnegie Mellon University
Pittsburgh, PA USA
bmclaren@cs.cmu.edu, koedinger@cmu.edu
mike.schneider@cs.cmu.edu

Andreas Harrer, Lars Bollen

Collide Research Group
University Duisburg-Essen
Duisburg, Germany
harrer@collide.info, bollen@collide.info

A potentially powerful way to aid in the authoring of intelligent tutoring systems is to directly leverage student interaction log data. While problem-solving data has been used in the past to guide the development of tutors, such data has not typically been used as a means to directly construct an initial tutoring system model. We propose an approach called *bootstrapping novice data* (BND) in which a problem-solving tool is integrated with tutor development software through log files and that integration is then used to create the beginnings of a tutor for the tool. We describe an initial implementation of the BND approach in which Cool Modes, a collaborative software tool, is integrated with the Behavior Recorder, tutor-authoring software that supports development by demonstration. A key to this implementation is a component-based approach in which complementary pieces of software are integrated with little or no change to either software component. We argue that more tutors could be built, and with substantial time savings, using this approach. We discuss some of the lessons learned from this initial effort and from applying the component-based approach, as well as some data analyses that could eventually be performed using the data collected during BND.

Keywords: Intelligent Tutors, Authoring Tools, Log files, Collaborative Learning

Introduction

Intelligent Tutoring Systems (ITS) have often been developed by programmers having expertise in cognitive modelling and/or Artificial Intelligence together with domain experts having experience and a depth of knowledge with a particular task. This approach is subject to problems, such as the so called "expert blind spot" in which those who are accomplished in a domain fail to recognize which aspects of that domain might prove difficult to novices (Nathan *et al.*, 2001). An expert's input is certainly important to the development of an intelligent tutor, but there is also much to be gained by capturing and analyzing the behavior of novices (cf, Lovett, 1998).

As an alternative to the traditional approach to tutor development, we propose an approach that leverages actual problem solving data not only to guide tutor design, as has been done before (cf, Koedinger and Terao, 2002), but also to contribute *directly* to tutor implementation. While others have used student data to, for instance, tune knowledge tracing parameters (Corbett *et al.*, 2000), we intend to use student (as well as expert) data as the fundamental basis for model development. In our approach, called *bootstrapping novice data (BND)*, we provide students with a computer-based tool, let them attempt to solve problems with the tool, and record that problem-solving activity in a tutor-specific representation. This integrated record of student activity helps in two

ways: (1) we learn a great deal about students' problem-solving approaches, both good and bad, and (2) it provides the initial representational structure for tutor implementation. In this research we are particularly interested in how we can develop tutors by leveraging the successes, errors, and inefficiencies of a group of learners. Our initial step has been to develop a prototype integration between a system for authoring collaborative modeling software, Cool Modes (COllaborative Open Learning and MODEling System) (Pinkwart, 2003), and a tutor authoring environment, the Cognitive Tutor Authoring Tools (CTAT) (Koedinger *et al.*, 2004). The Cool Modes software generates computer log files of student activities that are, in turn, used by CTAT as an initial representation of the tutor. In this paper, we discuss how we have effected this integration and how we could use this model as a way of collecting student data to create an initial, skeletal tutoring system.

An important underpinning of this work is the notion of component-based development. Our approach takes an existing software application and integrates it, with little or no modification, with a tutor or tutor agent. Using off-the-shelf or pre-existing software as the basis for building tutoring systems could result in substantial time savings and thus more tutor development, as compared to the traditional approach of building tutors "from scratch" (Ritter and Koedinger, 1996; McArthur *et al.*, 1996). With the component-based approach, tutor developers are able to leverage complementary pieces of software and focus attention on pedagogical techniques and technology of the tutor piece independent of domain-specific user interfaces, representations, and functionality of the tool piece.

The Pittsburgh Advanced Cognitive Tutor (PACT) lab at CMU has been working since the mid-1990s to fully realize the component-based integration model. The project has proven more challenging than originally anticipated. More generally, the educational object economy (http://www.eoe.org/) and other similar efforts showed great promise initially, but have not caught on. Part of the challenge is to get the component-based approach right. In particular, defining open, general, and well-structured software interfaces is a non-trivial task, and systems not originally designed as "components" are hard to integrate. Even if systems are designed as components, it is not always possible to anticipate in advance the input and output requirements of a complementary component. Another challenge is accumulating a critical mass of component producers and consumers so that the approach can pay off. Nevertheless, we believe the component-based approach will ultimately be the primary means of deploying computer-based tutoring, and thus we continue to push this as a critical agenda item in the learning technology community.

Bootstrapping Novice Data: Creating the Initial Representation of a Tutor

The process that we describe here, which we call Bootstrapping Novice Data or BND, creates an initial, skeletal model of a tutor from log data collected from actual tool use by students. BND begins with the integration of two software components, each with complementary features.

Cool Modes is a collaborative software tool designed to support "conversations" and shared graphical modelling facilities between collaborative learners (Pinkwart, 2003). It is a domain independent tool that supports a variety of modelling and learning tasks and provides users with various plug-in objects, such as Petri nets, a turtle programming environment, text widgets, and a "chat" area, each of which has its own semantics and underlying representation. All of the Cool Modes objects are available on a palette from which students may drag and drop objects into workspaces to build models. Cool Modes is extensible; new objects adhering to a well-defined API may be added to the palette as required. Translation between different object types is achieved through reference frames, a set of entities and rules that facilitate semantic object mapping. Cool Modes also provides operational facilities, such as execution of simulations and automated calculations.

Each Cool Modes user runs a client program that contains a private workspace in which objects can be privately created and updated. In addition, all users have access to a shared workspace, which is rendered in all of the collaborating clients and may be updated by any of the participants. The Cool Modes client program communicates with a server, called MatchMaker (Jansen 2003) that maintains the shared workspace and handles all communication between the collaborating clients. An example of a Cool Modes client is shown in Figure 1.

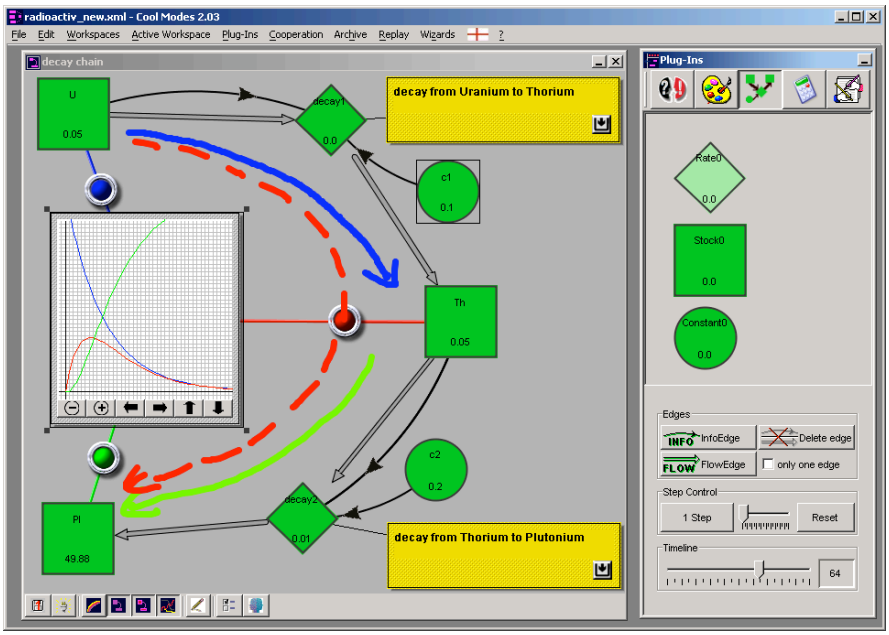

Figure 1: An example Cool Modes problem space. Here students are collaborating on a nuclear decay problem in a shared workspace.

Cool Modes does not assess or critique students' solutions apart from helping the students create syntactically correct models and allowing them to execute and observe the models. Thus, the tutoring component of the integration is provided by CTAT (Koedinger *et al*., 2004), an authoring tool for intelligent tutors (Murray *et al*, 2003). CTAT fits into the "Domain Expert System" category of authoring systems (Murray, 1999). It supports authors in building Cognitive Tutors, a form of "model-tracing" tutor based on cognitive psychology theory. As of the spring of 2004, Cognitive Tutors have been deployed in over 1700 schools in the United States. A Cognitive Tutor is composed of a problem representation and a set of production rules that model both desired and buggy behavior and is able to tutor students on a range of problems within a particular domain (e.g., geometry, algebra). Model tracing (a) matches actual student behavior during problem solving with the desired behavior represented in the production rules and (b) identifies deviations from that behavior. Cognitive Tutors are difficult to develop, typically requiring AI programming expertise. On the other hand, a specialized type of Cognitive Tutor also supported within CTAT is a *Pseudo Tutor*, a tutor that behaves much like a regular Cognitive Tutor, except that it provides instruction for only a single problem instance and is much easier to develop. Pseudo Tutors are developed using a "programming by demonstration" approach (Lieberman, 2001) that allows authors with no programming skills to build tutors.

An example Pseudo Tutor for fraction addition is shown in Figure 2. A Pseudo Tutor is developed as follows. First, the author builds a graphical user interface (GUI) with a specialized set of CTAT widgets. The GUI for the fraction addition tutor is shown on the right side of Figure 2. Second, the author demonstrates sequences of correct, alternative correct, and incorrect actions on these widgets. A CTAT tool known as the Behavior Recorder records all of these actions and builds a structure known as a *behavior graph*, shown on the left side of Figure 2. Each edge of the graph represents an action taken by the student on a particular widget of the GUI. The thicker edges represent the preferred or primary action taken from a particular node. The student's action is represented as a triple (selection, action, input): the Selection identifies the GUI widget selected, such as TextArea1; the action is the type of action taken, such as "Update Text"; and the input is the value provided by the student, such as "20." Each node of the graph represents a state of the interface after a path of edges from the root to that node has been traversed[1]. Third, after the behavior graph has been created by problem demonstration, the author can annotate the graph by labelling buggy edges (e.g., the incorrect path represented by "2, F13num" in Figure 2), inserting hints and feedback messages, and associating skills with edges. Finally, the author can test the model by "executing" the Pseudo Tutor, acting like a student or observing actual student use: in this case, the Behavior Recorder no longer builds the graph but instead only tries to trace student actions on it. The whole process typically is iterated several times, to refine the graph and account for more sequences of actions.

[1] There can be multiple paths to a node and thus a node can represent multiple states.

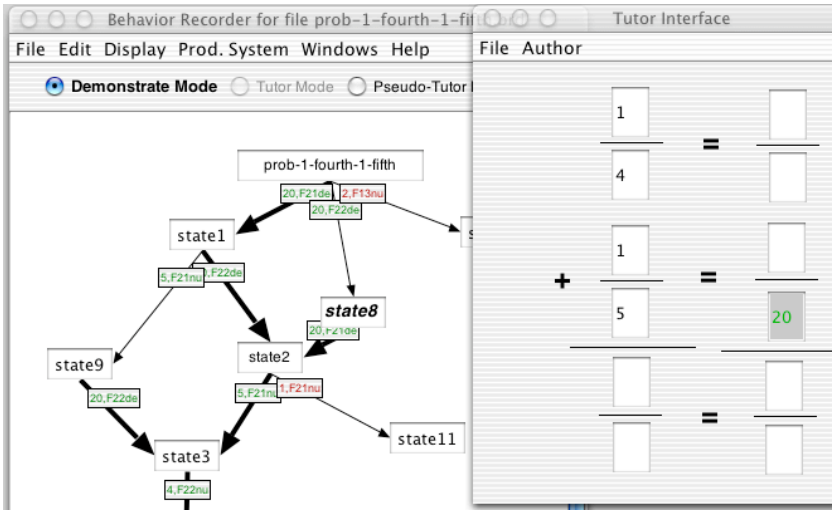

Figure 2. The Behavior Recorder records authors' actions in any interface which use CTAT's specialized GUI widgets. The author demonstrates alternative correct and incorrect paths. From the start state (labeled "prob-1-fourth-1-fifth") there are two correct paths ("20, F21den" and "20, F22den"), in which a common denominator is entered in either of the converted fractions, and one incorrect path ("2, F13num"). Although not visible in this black and white figure, the incorrect path has a red edge label and the two correct paths have green labels. One of the edges emanating from each node is thicker than the others; this indicates that it is the "preferred" path from that node. Since state8 is selected in the Behavior Recorder – you can see this because it is boldfaced and italicized – the Tutor Interface displays that state, namely with the 20 entered in the second converted fraction.

The fundamental idea of Bootstrapping Novice Data, depicted in Figure 3, is to use Cool Modes to generate a log of actual student problem solving actions and then send those actions to the Behavior Recorder, so that it can build a behavior graph as the beginnings of a tutor. In other words, the raw data from Cool Modes – with no information about the correctness of solutions – is passed to and translated by the Behavior Recorder into a behavior graph, which is then updated and annotated by the author. Because the Behavior Recorder is a component with a well-defined message interface (i.e., it accepts XML messages we call "Dormin Messages"), this integration was easy to develop. As per our earlier work (Ritter and Koedinger, 1996), all that was required was a Translator, developed in XSLT, to accept the Cool Modes XML log files and convert them to XML Dormin messages. Currently, the XSLT Log Translator is a standalone process that is manually run to translate the Cool Modes XML log files into files of XML Dormin Message. A file reader routine in the Behavior Recorder, invoked by a menu selection, reads in the XML Dormin Messages. In our next integration step, we will automate this process by creating socket connections directly from Cool Modes to the Translator and from the Translator to the Behavior Recorder.

As an example of how the Translator operates, consider a student specifying the number of tokens to use in a Petri Net in Cool Modes. After the student takes this action, Cool Modes produces the following XML log file entry:

```
<SyncAction action="actionExecuted"
        internalName="19"
        number="0"
        objectType="class info.collide.xml.helpers.XMLInteger"
        time="1083687234064"
        typeOfAction="setTokens"
        user="bollen">
    <XMLString value="2"/> </SyncAction>
```

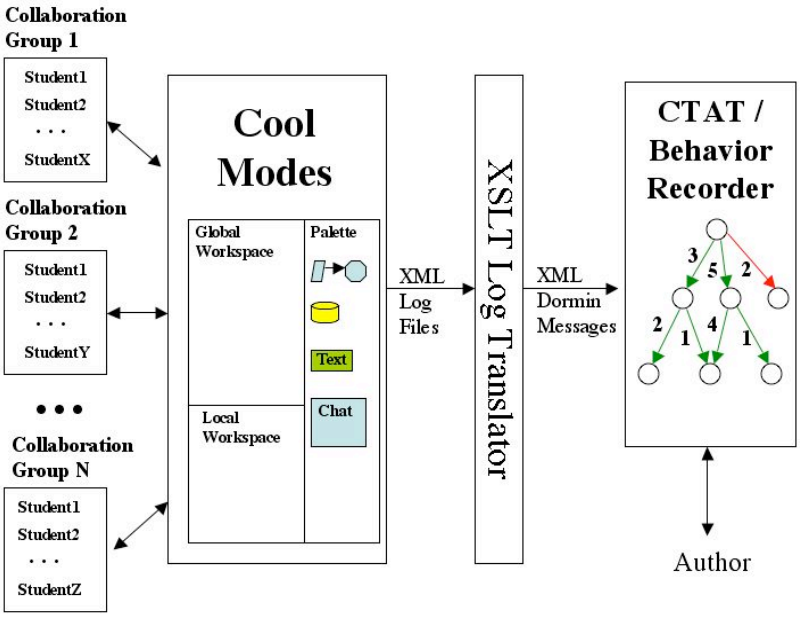

Figure 3: An implementation of the Bootstrapping Novice Data (BND) approach: Cool Modes / CTAT integration. Cool Modes provides the user interface for student activities and logs student actions; the Behavior Recorder builds a behavior graph – and the beginnings of a tutor – from those logs.

Cool Modes records an "actionExecuted" log entry with type of action "setTokens" and the number of tokens ("XMLString value") set to 2. Other information is also recorded, such as the user is "bollen" and the time the action took place. The XSLT Log Translator converts this record into a Dormin-style selection-action-input message that the Behavior Recorder understands:

```
<message>
    <verb>NotePropertySet</verb>
    <properties>
        <MessageType>InterfaceAction</MessageType>
        <Selection>
            <value>19</value>
        </Selection>
        <Action>
            <value>setTokens</value>
        </Action>
        <Input>
            <value>2</value>
        </Input>
    </properties>
</message>
```

The Behavior Recorder accepts XML Dormin messages, such as this one, and uses the data to create and augment the behavior graph, either by adding a new edge and node or by incrementing the traversal count over an existing edge.

The full BND vision is to have different students (in the case of Cool Modes, groups of collaborating students) generate (possibly different) correct and faulty solutions to the same problem. The logs of the different student actions are translated into a single behavior graph in the Behavior Recorder. As each group of students tackles the problem, the behavior graph is refocused on the "start node" (i.e., the root node which depicts the initial state of the GUI, prior to any problem solving), so that each solution path has the same starting point in the behavior graph. For each action in a log file (represented as a selection-action-input triple when translated to Dormin), if that action has not yet been recorded in the behavior graph, a new edge and state node are created. If the selection-action-input has already been recorded, an "edge traversal count" is incremented, representing the number of times that particular selection-action-input has occurred. These counts are shown by the numbers along the edges of the behavior graph on the right side of Figure 3.

After the behavior graph is generated, a domain expert author manually updates it by adding hints and bug messages, annotating buggy paths, and adding skills to edges. Not only does the BND approach provide the author with examples of *actual* correct and buggy paths taken by students, it also presents the author with another important piece of information: traversal frequencies of those paths. The edge traversal counts are good indicators of which of the correct solution paths might be considered primary, which secondary, and the counts along incorrect paths provide real data to show which errors occur frequently enough to merit writing specific buggy messages. The traversal counts can also help authors identify slips and careless errors (e.g., accidental item selections): an edge with a traversal count of 1, as compared to much higher counts on alternative edges, may indicate that an accidental action was taken by a student and thus can be deleted from the graph.

The power of the BND approach is that instead of authors building Pseudo Tutors from scratch, tapping only their own experience or incorporating student data "by hand" as in traditional ITS development, they can semi-automatically leverage the empirical data of a wide range of students engaged in actual problem-solving activity. This approach contrasts markedly with the usual ITS development method in which a domain expert author first creates "expert" problem solutions. In our approach, the student novices create initial solutions. The expert's judgement as to which novice solutions are correct is, of course, critical to creating a final version of the tutor. It may also be the case that an expert will have to augment the model by demonstrating a correct solution or solutions, if the student novices fail to generate any correct solution paths. But the critical aspect of BND is how it directly captures and encodes incorrect and inefficient novice solutions, information that will prove invaluable in building a full ITS.

An Example: Mutual Exclusion with Petri Nets

We tested our BND approach and integration of Cool Modes and CTAT with an example involving mutual exclusion, a class of problems in which specific objects or resources may be used by at most one actor at a time. When several actors attempt to use or claim an exclusive object, a mechanism for synchronization, referred to formally as a *semaphore*, acts as a gatekeeper. For example, in rural areas train rails are sometimes single-track. To avoid a collision of trains travelling in opposite directions, operators must guarantee that only one train uses this section of the rail at a time. In Figure 4 the teacher has prepared a Cool Modes worksheet in which each of two trains, Train A and Train B, are represented as Petri Nets (Petri, 1966)[2]. The trains can either be located in a station, which avoids the possibility of a collision, or on the one-track rail, which is potentially dangerous. The students' task is to extend the network to guarantee that the one-track rail is used by at most one train at a time.

The solution, shown in Figure 5, is to introduce an additional place node with exactly one token (call it the "mutual exclusion node") and several edges that connect it with the pre-existing sub-networks. Each time one train wants to enter the track, the token from the mutual exclusion node is taken (if available) and the other train is not able to enter the track.

[2] A Petri Net consists of *transition nodes*, which represent actions and are shown as rectangles in Figure 4, and *place nodes*, which are connected as inputs or outputs to transition nodes and are shown as circles in Figure 4. A transition is *activated* when each incoming place has at least the amount of tokens required by the connection's label. In Cool Modes, the user can interactively "fire" an activated transition by pressing the button of the transition (which is enabled, when activated and disabled otherwise). Firing the transition causes consumption of the respective number of tokens in the incoming places and the production of the number on the outgoing edges' labels in the outgoing places.

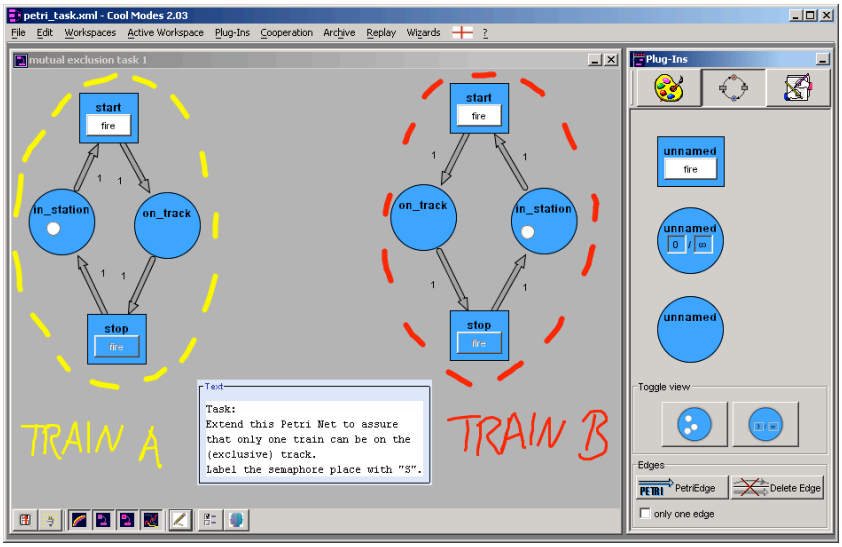

Figure 4: A simple representation of two trains using Petri Nets in Cool Modes. For each train, pressing the transition node labelled "start" moves a token (the small white circle) from the "in_station" to the "on_track" node. Conversely, pressing the button labelled "stop" moves the token in the opposite direction. Since the two subnets are independent, it is possible to have both trains on the track at the same time – the collision situation we wish to avoid in this problem. Notice that in the figure the "stop" transition is greyed out (unavailable). This is because the token is currently at the "in_station" node. When the token is at the "on_track" node, the "start" transition is greyed out.

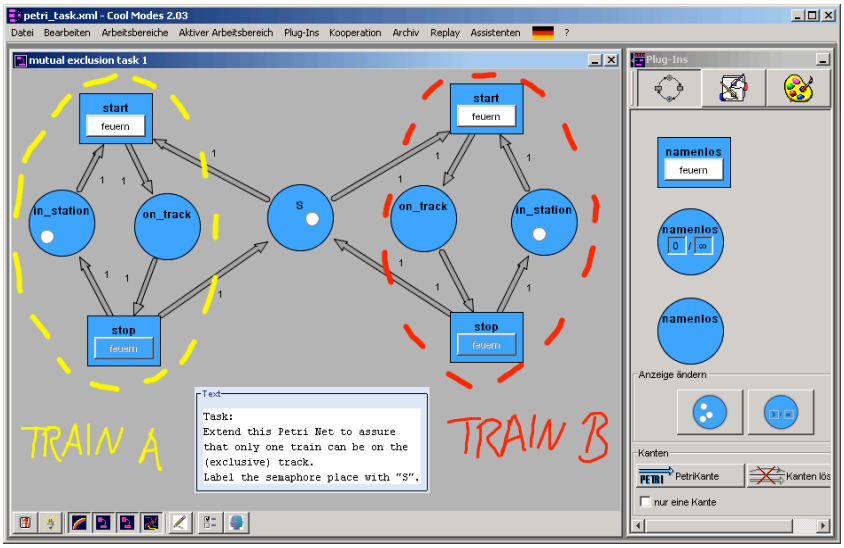

Figure 5: A solution to the problem of trains sharing the same track in Cool Modes. When the "start" transition is selected for one of the trains, both the "in_station" and "s" tokens are consumed, placing a single token in the "on_track" node of that train. This blocks the other train from moving "on_track" since the "start" transition for that train will be greyed out due to the unavailability of the "s" token. The second train cannot move "on_track" until the "stop" button of the first train is pressed, thus releasing the semaphore token back to the "s" node.

We obtained several student solutions, both correct and incorrect, by logging Cool Modes sessions, thus creating a set of XML log files. After transforming these logs to CTAT's XML-format, as per Figure 3 above, the files were fed into CTAT's Behavior Recorder and a behavior graph was created with weighted edges, showing the frequency of steps chosen by the students. (Note that there was no need to implement a graphical user interface (GUI) in CTAT, since Cool Modes provides all user input.) The resulting behavior graph is shown in Figure 6.

The numbers on the edges of the behavior graph in Figure 6 represent the number of times that edge was traversed during the 5 solution attempts. For instance, notice that all 5 of the students who attempted the mutual exclusion problem took the same first step, creating a place node to control the availability of the shared train track. This common initial step is represented by the edge coming from the start state and the "5" associated with that edge. After the first step, however, there were three different solution paths taken by the students: One was a buggy path, in which a student connected the new place node with a transition "start" in the wrong direction (from transition to place). The other two paths, each traversed by 2 students, were correct; the first pair of students connected the new place node with a transition "start" in the correct direction (from place to transition, which means that the track must be free before the train moves onto it), while the second pair set the number of tokens for the place to 1, which represents that the track is an exclusive resource. These are equally valid steps to take when solving the mutual exclusion problem.

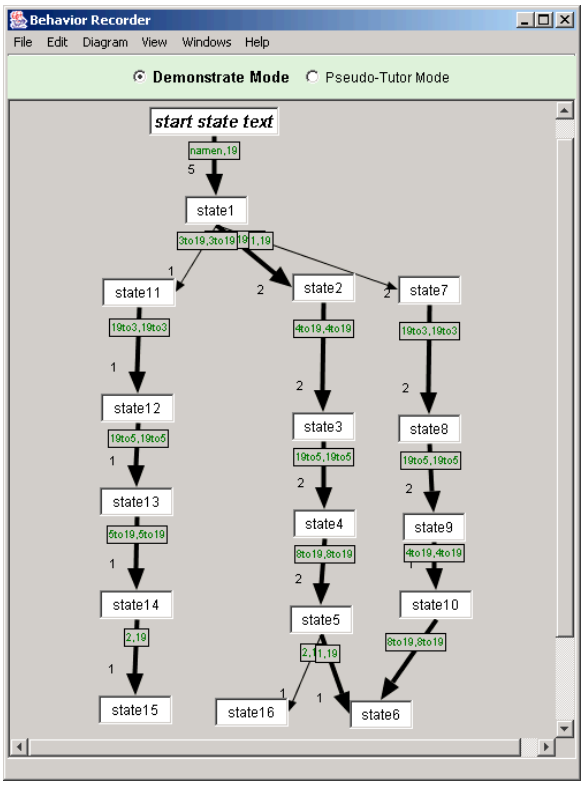

Figure 6: The Behavior Recorder after multiple student solutions of the mutual exclusion problem.

The behavior graph in Figure 6 is shown prior to any manual updates by the author. For instance, the author would ultimately mark the path on the left as "buggy," which changes the selection-action-input of the first edge emanating from state1 to red (the color we use to indicate a "buggy" action). This labelling would lead to all states after state11 being deleted (i.e., the Behavior Recorder assumes that all buggy states are "dead ends").

Discussion

In taking preliminary steps toward our vision of Bootstrapping Novice Data, we have learned some important lessons. A number of these lessons relate to the component-based combination of Cool Modes and CTAT. In principle, doing plug-and-play integration should be straightforward, but in practice is quite challenging. Two tool characteristics identified in Ritter and Koedinger (1996) – *recordability* and *scriptability* – are essential. Being recordable means the tool is capable of capturing individual actions taken in its user interface (and not complete states). Cool Modes clearly meets the recordability criterion, as previously discussed. Being scriptable, on the other hand, means the tool is capable of playing back user actions in its user interface, typically through a scripting language. Scriptability is not necessary for the BND approach described in this paper, nor is it strictly necessary for real-time tutoring (as Ritter and Koedinger demonstrated with the Sketchpad tutor). However, it is *desirable* for real-time tutoring and necessary for convenient Behavior Recorder authoring. Selecting any state in the behavior graph should put the tool user interface into the corresponding configuration by replaying the actions along a path to that state. The MatchMaker server of Cool Modes, which provides

communication between the collaborating learners, can, in principle, be made scriptable. Non-Cool Modes applications could be connected to the MatchMaker server. To integrate the Behavior Recorder we will implement a special MatchMaker client that accepts messages from the Behaviour Recorder and which, in turn, relays them to the Cool Modes applications. Another plug-and-play criterion is that logging of student actions should occur in real-time. Currently, Cool Modes log files are not generated until the end of a session, thus behavior graph construction cannot proceed interactively with tool use. We will address this issue by modifying Cool Modes to log each action immediately.

We also learned that the Behavior Recorder and Translator need to be extended or supplemented to support integration across a variety of tools. Since many tool environments involve dynamic instantiation of objects (e.g., new Petri net nodes in Cool Modes), the Behavior Recorder must also be capable of handling dynamic object definition and recognition. Since Cool Modes uses a consistent, internal naming scheme for objects, we can leverage this to identify common dynamic objects across sessions using mapping tables. Finally, since the BR was originally designed to support single-student tutoring, there is no recording of who performs each action. Cool Modes has this data, and it is a relatively minor augmentation of the BR to extend its selection-action-input representation to include and reason about an "actor" field.

Besides using the student log files for building an initial version of a tutor, we envision other uses of the data. For instance, we plan to do problem profile analysis (Mark, 1998) in which we first have an expert, or group of experts, solve a problem using Cool Modes, record the steps in a behavior graph, and then have students attempt to solve the same problem, also recording the steps as in the BND approach described above. Assuming the expert solution(s) is the correct one, we can then use the resulting behavior graph to calculate the percentage of novice steps that diverge from the experts. The greater the divergence from expert behavior, the harder this problem can be assumed to be. Such data is helpful in designing and ordering problems in a curriculum, e.g., situating more difficult problems later in the curriculum. This expert-vs.-novice data can also be used to do skill proficiency analysis. Divergent paths that occur with a high frequency, according to the traversal counts collected by BND, likely relate to skills that should be the focus of additional problems and tutoring. If the author associates skills to edges of the behavior graph after BND data collection, we will have the data necessary to identify skills that, in general, cause the students more difficulty. The author could then design new problems that focus on these skills.

In order to improve a tutor's behavior graph over time, we also envision doing Learning Factors Analysis (Koedinger and Junker, 1999) with the Cool Modes log data. Learning Factors Analysis is a process whereby a cognitive model is evaluated and modified based on how closely student performance matches expected learning curves. The labelling of edges with skills – essentially the "factors" – will allow us to subsequently check whether students gradually reduce their error rate. If the error rate for a particular skill does not yield a smooth, downward sloping learning curve, this is an indication that the skill has been mis-assigned to particular edges in the behavior graph. Eventually, we intend to have the CTAT software support authors in viewing and specifying alternative skill labellings, viewing the resulting learning curves, and making changes to the skill labels accordingly.

Conclusions

In summary, we have discussed the use of log files and log analysis as a starting point in developing a tutoring component for a pre-existing software tool. Our approach, called Bootstrapping Novice Data, involves the transformation of student log files from a software tool into a sequence of student-action messages useable by tutor authoring software. To implement an initial version of BND, we used a component-based approach in which we integrated an existing collaborative software tool, Cool Modes, with tutor-authoring software, realized in the Behavior Recorder software. The BND approach is potentially quite powerful, as it obviates the need to (a) build a tutor “from scratch” and (b) rely primarily on a domain expert to build a tutor.

A longer-term aim of our work is to explore how we can fully integrate cognitive tutoring techniques in a computer-mediated collaborative environment (Jermann *et al*., 2001; Muehlenbrock, 2001). In other words, we want to use the integration of Cool Modes and CTAT as a first step toward developing a fully integrated, pedagogical model to provide real-time tutoring in Cool Modes and other collaborative software environments.

Besides directly facilitating the construction of an intelligent tutor, the log files of Cool Modes and the BND approach promise to help us in other ways. For instance, we can use actual student data to identify the skills that are most likely to require additional practice and in what order problems should be presented in a curriculum.

We also intend to use this work as a means of leveraging log files to help improve the cognitive model (i.e., the behavior graph) through Learning Factors Analysis.

References

Corbett, A., McLaughlin, M., and Scarpinatto, K.C. (2000). Modeling Student Knowledge: Cognitive Tutors in High School and College. *User Modeling and User-Adapted Interaction*, 10, 81-108.

Jansen, M (2003), MatchMaker - A Framework to Support Collaborative Java Applications, In U. Hoppe, F. Verdejo & J. Kay (eds.): Shaping the Future of Learning through Intelligent Technologies. *Proceedings of the 11th Conference on Artificial Intelligence in Education*. pp 529-530, IOS Press, Amsterdam.

Jermann, P., Soller, A., and Muehlenbrock, M. (2001). From Mirroring to Guiding: A Review of State of the Art Technology for Supporting Collaborative Learning. Proceedings of the First European Conference on Computer-Supported Collaborative Learning, Maastricht, The Netherlands, 324-331.

Koedinger, K. R., Aleven, V., Heffernan, N., McLaren, B. M., and Hockenberry, M. (2004). Opening the Door to Non-Programmers: Authoring Intelligent Tutor Behavior by Demonstration. Accepted for presentation at the *Seventh International Conference on Intelligent Tutoring Systems*, Maceio, Brazil, September 2004.

Koedinger, K. R. and Junker, B. (1999). Learning Factors Analysis: Mining student-tutor interactions to optimize instruction. Presented at *Social Science Data Infrastructure Conference*. New York University. November, 12-13, 1999.

Koedinger, K. R. and Terao, A. (2002). A cognitive task analysis of using pictures to support pre-algebraic reasoning. In C. D. Schunn & W. Gray (Eds.), *Proceedings of the Twenty-Fourth Annual Conference of the Cognitive Science Society*, 542-547.

Lieberman, H. (ed) (2001). *Your Wish is My Command: Programming by Example*. Morgan Kauffman Publishers.

Lovett, M. C. (1998). Cognitive task analysis in service of intelligent tutoring system design: a case study in statistics. In the *Proceedings of the Fourth International Conference of Intelligent Tutoring Systems*. (pp. 234-243).

Mark, M. (1998) *Analysis of Protocol Files: PACT Center User's Manual*. Carnegie Mellon University.

McArthur, D., Lewis, M. W., and Bishay, M. (1996). ESSCOTS for learning: Transforming commercial software into powerful educational tools. In the *Journal of Artificial Intelligence in Education*, 6 (1), 3-33.

Muehlenbrock, M. (2001). *Action-based Collaboration Analysis for Group Learning*. IOS Press, Amsterdam.

Murray, T., Ainsworth, S., and Blessing, S. (eds.) (2003). *Authoring Tools for Advanced Technology Learning Environments: Toward Cost-Effective, Adaptive, Interactive, and Intelligent Educational Software.* Kluwer Academic Publishers. Printed in the Netherlands.

Murray, T. (1999). Authoring Intelligent Tutoring Systems: An analysis of the state of the art. *International Journal of Artificial Intelligence in Education*, Vol. 10, pp. 98-129.

Nathan, M., Koedinger, K., and Alibali, M. (2001). Expert blind spot: When content knowledge eclipses pedagogical content knowledge. Paper presented at the *Annual Meeting of the American Educational Research Association*, Seattle.

Petri, C.A., (1966). Kommunikation mit Automaten. Bonn: Institut für Instrumentelle Mathematik, Schriften des IIM Nr. 2, 1962, Second Edition:, New York: Griffiss Air Force Base, Technical Report RADC-TR-65--377, Vol.1, Pages: Suppl. 1, English translation.

Pinkwart, N. (2003). A Plug-In Architecture for Graph Based Collaborative Modeling Systems. In U. Hoppe, F. Verdejo & J. Kay (eds.): Shaping the Future of Learning through Intelligent Technologies. *Proceedings of the 11th Conference on Artificial Intelligence in Education*, 535-536.

Ritter, S. and Koedinger, K. R. (1996). An Architecture For Plug-In Tutor Agents. In the *Journal of Artificial Intelligence in Education*, 7 (3 / 4), 315-347.