Model Tracing – A Diagnostic Technique in Intelligent Tutoring Systems

Ani Aižič, Slavomir Stankov, Marko Rosić
Faculty of Natural Sciences, Mathematics and Education
Nikole Tesle 12, 21000 Split, Croatia
Phone: (385) 21-38 51 33-105, Fax: (385) 21-38 54 31
E-mail: ani.amizic {slavomir.stankov, marko.rosic}@pmfst.hr

Abstract – An educational system is a community where students and teachers are involved in a process of learning and teaching. Intelligent tutoring systems (ITS) are computer systems designed for support and improvement of learning and teaching process in freely chosen domain knowledge. Tutor–Expert System (TEx-Sys) is a hypermedial authoring shell for building intelligent tutoring systems. While testing a student, human tutor observes sequence of student actions, remembers a number and type of student mistakes and number of additional questions or other type of help used to push student forward. Human tutor makes evaluation of student knowledge that is based on gathered information. Those is why ITS that considers only final results, that is, student answer, cannot justly and completely evaluate student knowledge. For that reason we have decided to introduce a model tracing – a diagnostic technique to be used in student modelling in TEx-Sys. The TEx-Sys provides help and remediation for future work that is based on consideration of number of misconceptions.

1. Introduction

Student module is a component of intelligent tutoring system which actions can be easily explained by thinking of it as a tutor in a real meaning of that word. Tutor would have to know what student knows, what is the level of his capabilities, his foreknowledge. Besides that, tutor must choose if it would disregard mistake that student had made, point it out, rectify that mistake or guide student towards recognition and correction of that mistake.

Tutor–Expert System (TEx-Sys) is a hypermedial authoring shell for building intelligent tutoring systems. During his process of learning and teaching in system TEx-Sys, student uses programming module Learning and Teaching. In a process of evaluating his knowledge by overlay method, student uses modules Testing and Evaluating ([1]). Our attention in this paper is directed to a certain improvements that have to be made in a process of evaluating knowledge by overlay method. Underlying assumption for better understanding of the facts presented in this paper is acquaintance with a structure of the system TEx-Sys and presentation of knowledge in the system (more details in [1] and [6]).

We start from the statement that we need more than just final states for getting a final mark – we also need to observe intermediate states. We can find out more about student knowledge from his actions such as: (i) adding and deleting certain types of nodes and links; (ii) usage of help (INFO) and finally (iii) number of calls and a way of using a module NodeHelp. Observation of these situations can make us aware of student’s insecurity, maybe even an ignorance, which the system TEx-Sys has not so far taken into consideration while evaluating student knowledge. That is the reason why we have decided to supplement evaluation system with tracing of situations like those mentioned before.
2. Student modeling

Many intelligent tutoring systems share the same characteristics: they infer a model of student's current understanding of the subject matter and use this individualized model to adapt the instruction to the student's needs. Inferring a student model is called diagnosis. An ITS diagnosis system uncovers a hidden cognitive state from observable behaviour. It must make a conclusion about what was student thinking and doing during his learning process. The component of an ITS that represents the student's current state of knowledge is called the student model. The student model is a data structure, and diagnosis is a process that manipulates it.

The design of these two components is called the student modelling problem ([2]). This approach uses: (i) the three dimensions of student models each with three distinguished values (Table I), as well as (ii) nine diagnostic techniques (Table II) employed in the corresponding student modelling systems.

After analysing all mentioned diagnosis techniques and the structure of the system TEx-Sys, we have come to conclusion that solution of the student modelling problem of this system would be gained easily if we would use diagnosis technique called model tracing.

| TABLE I |
| The three dimensions of student models |

1) **Bandwidth** – How much of the student’s activity is available to the diagnostic program?
   a) *Mental states* – All the activity, both physical and mental, is available
   b) *Intermediate states* – All the observable, physical activity is available
   c) *Final states* – Only the final state (the answer) is available
2) **Knowledge type** – What is the type of the subject matter knowledge?
   a) *Flat procedural* – Procedural knowledge without subgoaling
   b) *Hierarchical procedural* – Procedural knowledge with subgoals
   c) *Declarative*
3) **Student-Expert Difference** – How does the student model differ from the expert model?
   a) *Overlay* – Some items in the expert model are missing
   b) *Bug library* – In addition to missing knowledge, the student model may have incorrect “buggy” knowledge. The bugs come from a predefined library
   c) *Bug part library* – Bugs are assembled dynamically to fit the student’s behaviour

| TABLE II |
| Diagnostic techniques |

<table>
<thead>
<tr>
<th>Knowledge type</th>
<th>Flat procedural</th>
<th>Hierarchical procedural</th>
<th>Declarative</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Bandwidth</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mental states</td>
<td></td>
<td>Model tracing</td>
<td></td>
</tr>
<tr>
<td>Intermediate States</td>
<td>Issue tracing</td>
<td>Plan recognition</td>
<td>Expert system</td>
</tr>
<tr>
<td>Final states</td>
<td>• Path finding</td>
<td>• Decision tree</td>
<td>Generate and test</td>
</tr>
<tr>
<td></td>
<td>• Condition Induction</td>
<td>• Generate and test</td>
<td>Interactive</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Generate and test</td>
</tr>
</tbody>
</table>
3. Model tracing

Model tracing assumes that all of the student’s significant mental states are available to the diagnostic program. The basic idea is to use an undetermined interpreter for modelling problem solving. At each step in problem solving, the underdetermined interpreter may suggest a whole set of rules to be applied next, whereas a deterministic interpreter can suggest only a single rule. The diagnostic algorithm activates all these suggested rules, obtaining a set of possible next states. One of these states should correspond to the state generated by the student. The name model tracing comes from the fact that the diagnostic program merely traces the (undetermined) execution of the model and compares it to the student’s activity.

Input information for this algorithm is:
1. expert knowledge and student’s task
2. set of productions
3. student’s answer that we want to trace

Output information is:
1. True or false depending on whereas the student’s answer has been traced.
2. If the student’s answer has been traced, output information is an array of chained production. If algorithm has not succeeded in finding an array of production that could generate student’s answer, we say that the student’s answer is not interpreted.

Figure 1. The traditional architecture of model tracing
There are some productions that help us find wrong student’s answers, that is, we make those answers understandable to the system. These “buggy” productions are used to make student’s answer meaningful even thou a student made a few wrong steps.

The system generates feedback that informs student about his actions. There are two types of feedback: (i) “buggy” feedback and (ii) hints or help. Every “buggy” rule generates massage about mistake that has been made as a feedback. Hints or help are given on student’s demand or when the system estimates that student needs them. Then the system generates an array of productions that represent a cognitive step a student should make to get himself to next step in problem solving process. That array of productions generates an array of hints that the system offers to student.

Model tracing tries to dynamically simulate student’s problem solving process and uses that simulation for interpretation of student’s behaviour. This diagnostic technique bases itself on previously given catalogue of productions. Tracing starts after each student action and is being used for supervising student during his problem solving process. Model tracing is based on a idea of analysing student cognitive process by reconstructing, step by step, process of making conclusions during a problem solving. To make this kind of tracing work, expert knowledge and catalogue of stereotype errors must be available.

VanLehn in [2] states several issues about value of this diagnostic technique:
1. What should the system do if the student’s state does not match any of the states produced by the rules in the model?
2. Suppose the student generates a next state by guessing or by mistake; the system will erroneously assume that the student knows the corresponding rule.
3. When should the system change its mind about its student model?

3.1. Objections to existing version of the system TEx-Sys

Module Testing of the system TEx-Sys is a module for inquiring student’s knowledge. On existing level of development it functions like follows:
1. A task is being generated after choosing a domain knowledge database and one of three types of problems.
2. Depending on a type of the problem, during a process of testing student needs to add and delete nodes and links.
3. Student may use INFO (Figure 2.) that gives information about current nodes and links status.
4. When student has done everything he should have done or knew, he ends testing.

After the testing student can choose module Evaluating that gives student information about his knowledge or ignorance.
1. Analytics—overview of nodes and links status
2. Diagnostic—overview of maximum score number and realised score number
3. Statistic—overview of numeric information about nodes and links status by type

Marks for nodes and links are given separately and brief description about what should have been done and what has been done.

Real teacher pays his attention to more than just student’s answer during testing a student.

Figure 2: INFO – gives information about current nodes and links status
Furthermore, he takes into consideration sequence of student actions (for example, student makes a mistake, then corrects himself and gives a correct answer), number and type of student’s mistakes and number of additional questions or other kind of help that has been used. On the bases of this information teacher evaluates student knowledge.

Intelligent tutoring system that takes into consideration only final result, that is, student’s answer, cannot justly and validly evaluate student’s knowledge. That is why we have to add some new functions to the module Testing of the system TEx-Sys, that is, programme module that will monitor student’s steps and give extra conclusions about student’s knowledge. Preciseness of evaluating student’s knowledge in module Evaluating will benefit from that extra information about student’s knowledge.

3.2. Situations and production rules

There are four starting actions that are always available for student when he starts solving problem:
1. adding nodes
2. deleting nodes
3. adding links
4. deleting links

We define sequences of elementary events, that is, student actions (steps) and we call them situations (we will mark them with letter $s$). Every event is described with certain node or link status. Using syntax and semantics of node and link status ([1]), we have given codes to elementary events and combinations of those events to make their marking and distinguishing easier.

There are only three kinds of actions that are available for student: adding, deleting and INFO. Situations that are result of actions over nodes are considered separately from those over links because working with nodes involves different cognitive process than working with links. What is saved in verbal longterm memory is usually used in one of two main forms: as a statement or intellectual skill. Statement is a sentence that has one relevant term. Associations among terms connect statements into a net of statements. Intellectual skill consists of terms and rules in a form of plan of action (how to do something) ([3]). When a student adds or deletes nodes, he handles statements. Connecting nodes or deleting links is considered to be an intellectual skill. Better said, familiarity with nodes shows familiarity with objects of certain knowledge, that is, facts of that knowledge. Familiarity with links shows cognitive process of generalization of the facts of certain knowledge, that is, relation among objects of that knowledge.

We have decided to remember three last student’s steps and pay attention to their combinations, that is, we observe situations that are result of three events. Reason for that is a fact that there are only three kinds of actions that are available for student: adding, deleting or INFO.

We define subset $S$ of set of all situations $s$, whose elements are only those situations that affect, positively or negatively, the process of making conclusions about student’s competency and knowledge ([4]). We define partition of set $S$ on two disjunctive subsets $N$ and $L$ of $S$ ($N \cap L = \emptyset$), that is, $S = N \cup L$:

$N = \{s \in S \mid \text{elementary events of situation } s \text{ are student’s actions on nodes}\}$

$L = \{s \in S \mid \text{elementary events of situation } s \text{ are student’s actions on links}\}$

We have described situations in a form of a production rule (see Pseudocode 1.) to make explanation of situations easier. We have described situations in a set $C$ with production rules where are $<\text{EventM}>$ events on nodes (production rules for nodes), and situations in a set $V$ with production rules where are $<\text{EventM}>$ events on links (production rules for links). Every production rule gives explanation of student’s actions and direction for further work.
If <Event1> conjunction. 
   <Event2> conjunction. 
   . . . 
   <EventN> 
Then 
   <Indicator_Mark>=<Indicator> 
   Print: Text of production rule 
EndIf 

where is: 
   <EventM> one of elementary events or their combinations 
   conjunction “and” or “or” 
   <Indicator> <Knowledge> or <Ignorance> 
   Text of production rule is direction for further work. 

Pseudocode 1: Form of production rule for nodes and links

3.3. Input and output information for model tracing

Input information for this algorithm is:
1. <PROBLEM>-Database 
2. set of productions (production rules for nodes and links) 
3. student’s answer that we want to trace (situation) 
Output information is: 
1. If there is a production rule for that situation, then the answer is traced and output information is the text of that production rule. 
2. If there is not a production rule for that situation, that does not mean that that student’s answer cannot be interpreted. That means that that situation belongs to a set of situations that do not significantly affect an image of student’s knowledge and there is no reason for their detail description with production rules. 

This gives an answer to first question stated by VanLehn. Namely, student always follows one of the rules in a model because there isn’t a sequence of three events that wouldn’t be output information of this algorithm.

Most of the productions that describe student’s answers, that is, situations, are so-called “buggy” productions. The system generates feedback to inform student about his actions. There are two kinds of feedback:
1. “Buggy” feedback that the system gives at the end of testing (texts of production rules) 
2. Hints for adding links or help for adding nodes 

Every “buggy” rule generates a message about mistake that has been made as a feedback. Hints or help are given when student asks for them or when the system estimates that student needs them. Student can get help in a form of information about nodes and links in a <SOLUTION> database, in a form of module NodeHelp or in a form of generated sequence of hints for adding nodes.

If student has moved into a next state by guessing or making mistakes, the system does not imply that student is aware what he has done. The system uses additional algorithms, such as remembering deleted and added nodes and links, for recognising those actions as wrong or made by guessing. This gives an answer to second question stated by VanLehn. An answer to third question stated by VanLehn is: the system must change it’s opinion about student model after generating feedback that informs it what significant situations has student been going through.

4. Helping with retrieval in the system TExSys

Problem that has shown up while using module Testing is with adding nodes. If student has to add node that has longer name and in foreign language, it is almost impossible to add it correctly. During learning process in module Learning and Teaching student learns correct names of simple nodes (simple in terms of length and syntax complexity), but when it comes to learning nodes such as “Using the Start button, you can accomplish almost any task.” student remembers just key term such as “Start button”, “any task”, that is, he appropriates essence meaning of the node. “Start button, any task” is not correct name of the node, and the system would conclude that
student doesn’t know that node. In a same situation only with real teacher, student would explain meaning of the node using terms “Start” and “task”.

Consequently, the system that demands from student literal memorizing of terms, imprecisely evaluates student knowledge because it cannot examine understanding of terms, but only literal reproduction of certain knowledge. Sometimes student gives correct answer, but he expresses it in a different way.

It is necessary to develop a help that would be offered to student only in certain situations certain number of times and that would enable adding nodes whose meaning student knows, but cannot reproduce it’s complete name. Namely, in practice of contemplative and verbal learning, information can be used in a same form it is received in or with a same meaning but in a form different of that it is received in.

Programming module \textit{NodeHelp} is based on input of key words, that is, terms that are part of node name. Namely, during retrieval from long-term memory, so-called retrieval symbols have great importance. They lead us in a wider area of memory where certain particle could be found, like key words and different kinds of indexes in librarianship help to find certain publication ([5]).

Programming module \textit{NodeHelp} generates offer that includes those nodes whose names contain some of that key word, and other nodes to make choice of node harder. In some situations recognition can be harder that retrieval. When an offer for a choice is big and rather similar, there can be more mistakes done during recognition than retrieval from a specific and limited area of memory ([3]).

Creating this kind of help would solve a problem of adding nodes because student, who understood a meaning of node, but has not memorized it’s syntax, would easily guess correct node name. There is a question why give students a possibility to add correct node on an easier way with recognition rather than retrieval?

When we cannot retrieve some information, but we recognize it, we conclude that a process of storage has been successful but we cannot find that information in a long-term memory. That does not mean that we haven’t appropriated that information. Namely, material that we have once appropriated and seemingly have completely forgotten, because we cannot retrieve it and recognize it, during anew learning process is learned much faster ([5]). That is the reason why the system directs student, who has used module \textit{NodeHelp} three times, to learn that material once again. After that anew learning student can test his knowledge again, but in this case can no longer use module \textit{NodeHelp}.

\subsection{4.1. Model tracing and help for adding nodes}

Start of module \textit{NodeHelp} is considered to be event that is result of certain situation ([4]). That situation is described with production rule RuleNode8 (see Pseudocode 2.).

\begin{verbatim}
If <Adding_Node>=<New> .And. 
<INFO>=True .And. 
<Deleting_Node>=<New> 
Then 
<Indicator_Mark>=<Ignorance> 
Start module NodeHelp 
EndIf
\end{verbatim}

\textbf{Pseudocode 2: Production rule for nodes RuleNode8}

However, to prevent misuse of help (analysing all NEW nodes can help us to determine if a student has problems with node name syntax or there are nodes from other domain knowledge), it has been decided that three situations described with production rule RuleNode8 must happen before offering student help for adding nodes. Student can use help for adding nodes only three times during testing. After student has used all his “coupons” for help, the system directs him to learn that knowledge once again.
5. Evaluating

Appearance of every situation (described with production rule) in module Testing along with module NodeHelp, is remembered and quantified (that quantifiers are determined after testing that took place in order to see the differences between previous and this version of the system and the benefits of model tracing). In that way appearance of situations that are described with production rules affect final mark.

Total score gained during testing by using diagnostic interpreter of student knowledge, is immediately affected with status of nodes and links in <SOLUTION> database (more details in [1]). We add RuleScore (score gained from the number of appearances of production rules) to total score. Consequently, total score depends on number of appearance of certain situations that describe student knowledge and student capabilities. Production rules also generate instruction for further work that can be of a great help for student because it describes situations that appeared during testing. That instruction enables student to restrict his learning on specific material that causes him problems or to correct his repetitive mistakes.

6. Conclusion

Problem of communicating with natural language within computer systems that support process of learning and teaching is still actual and for research world very intriguing. As a part of research and development of the system TEx-Sys we have approached to solving that problem and even have solved it in a significant matter with implementing diagnostic technique called model tracing. This diagnostic technique has enabled us to trace every student action and that has brought us to more precise evaluation of student knowledge. Tracing of student actions has enabled the system to estimate when student needs help and what kind of help, leaving a choice for using that help to student. This approach has made possible partial development of “thinking” component of the system TEx-Sys.

Acknowledgements

This work has been carried out within projects 177010 Independence of Student Using New Information Technology funded by the Ministry of Science and Technology of the Republic of Croatia.

References