Unsupervised MDP Value Selection for Automating ITS Capabilities

John Stamper[1] and Tiffany Barnes[2]
[1]john@stamper.org, [2]Tiffany.Barnes@gmail.com
Department of Computer Science, University of North Carolina at Charlotte

Abstract. We seek to simplify the creation of intelligent tutors by using student data acquired from standard computer aided instruction (CAI) in conjunction with educational data mining methods to automatically generate adaptive hints. In our previous work, we have automatically generated hints for logic tutoring by constructing a Markov Decision Process (MDP) that holds and rates historical student work for automatic selection of the best prior cases for hint generation. This method has promise for domain-independent use, but requires that correct solutions be assigned high positive values by the CAI or an expert. In this research we propose a novel method for assigning prior values to student work that depends only on frequency of occurrence for the component steps, and compare how these values impact automatic hint generation when compared to our MDP approach. Our results show that the utility metric outperforms a classic MDP solution in selecting hints in logic. We believe this method will be particularly useful for automatic hint generation for ill-defined domains.

1 Introduction

Our goal is to simplify the creation of intelligent tutoring systems (ITSs) by augmenting existing computer aided instruction (CAI) with intelligent behaviors, such as adaptive feedback and help, derived using educational data mining on CAI data. In our previous work, we have shown that we can successfully generate appropriate context-specific hints for a logic tutor using a Markov decision process (MDP) built from historical student data [2]. The core element of this work that makes automated hint generation possible is the assignment of relative values or "rewards" to each step in a problem solution. We have proposed that alternate assignment values may allow for hints tailored to specific student needs or readiness to learn. As in a recommender system that makes purchase suggestion based on frequent behaviors, we believe that hints generated based on the frequency of a particular step represent those that the majority of students would understand and be able to apply. Vygotsky's theory of the zone of proximal development [19] states that students are able to learn new things that are closest to what they already know. Presumably, frequent actions could be those that more students feel fluent using. Therefore, paths based on typical student behavior may be more helpful than optimal or expert solutions, which may be above a student's current ability to understand.

Based on this idea, and the observation that our MDP method sometimes generates a hint that a typical student would not do, or one that is technically correct but was not necessary to the problem solution, we hypothesized that we may be able to generate hints based on "usefulness" and frequency for a particular step in a student's attempt. Currently, when we construct our MDP, we connect all correct student attempts to a synthetic goal state, assign this state a high value and errors negative values, and rely on value iteration to assign high values to states that are close to the goal, and lower values

to those further away. Using this approach results in high values for expert-like solutions, which are short and have few errors. However, in a few instances our tutor gives hints that suggest a less popular path with additional, unnecessary steps. Close inspection of the MDP showed that the states derived from a single, error-free student's solution could get higher values than a more popular route, but that had a significant number of errors. A metric that more heavily emphasizes frequency can mitigate this issue.

The overall goal of our work is to derive domain-independent ways to add intelligence to tutors. However, our prior approach requires that we can label all data as correct or incorrect. In the logic proofs domain, this is simple but in other domains, especially ill-defined domains, hand grading of all student solutions might be required. For example, it is often difficult to determine if a computer program is complete and correct, but it is possible to extract features that many attempts contain, such as variables or loop structures. It seems reasonable to propose that the more student attempts that contain a particular feature, the more likely it is that this feature is a necessary part of a correct program. To lay the foundation for hint generation in such ill-defined domains, we performed an experiment to verify that we could use an unsupervised utility metric to label and value states in an MDP for logic. We hypothesized that this metric would result in similar hints in the logic domain to those we derive using our MDP method.

2 Background and Related Work

The most successful intelligent tutors require the construction of complex models (of knowledge or constraints) that are applicable only to a specific tutorial in a specific field, requiring the time of experts to create and test. It takes between 100-1000 work hours to create 1 hour of content for an intelligent tutor [16]. In order to bring the benefits of intelligent tutors to a wider audience, we must find a way to simplify their creation. One approach is to use generalized authoring tools to simplify the creation of intelligent tutoring systems. Two of the most widely-known authoring tools, including CTAT [11] for building cognitive tutors and ASPIRE [15] for building constraint based tutors, have been used to successfully create and deploy new tutors. Yet, both of these authoring tools discount the tremendous amount of CAI that already exists and require the construction of new tutors. In our work, we have shown that it is possible to provide intelligent, context specific-hints through educational data mining, that allows us to augment existing CAI with the intelligent behaviors found in other tutoring systems.

There are several ways that researchers have proposed to simplify the creation and improvement of intelligent tutors. CTAT has used demonstrated examples to learn production rules that are problem-solving models for cognitive tutors [1]. For CTAT example-based tutors, teachers work problems in what they predict to be frequent correct and incorrect approaches, and then annotate the learned rules with appropriate hints and feedback. This system has also been used with data to build initial models for an ITS, in an approach called Bootstrapping Novice Data (BND) [13]. However, in both of these approaches, considerable time must still be spent in identifying student approaches and creating appropriate hints. Machine learning has also been used not only to build but also to improve tutoring systems. In the ADVISOR tutor, machine learning was used to build student models that could predict the amount of time students took to solve arithmetic

problems, and to adapt instruction to minimize this time while meeting teacher-set instructional goals [5]. In the Logic-ITA tutor, student data was mined to create hints to warn students when they were likely to make mistakes using their current approach [14].

Our research uses past student data to generate Markov Decision Processes (MDPs) that assign numerical values to every state reached by past students solving a problem in an existing CAI. Using these values, we can estimate for any problem state what the "best" next step that any student has taken from the current problem state in the past. Our method of automatic hint generation using previous student data reduces the expert knowledge needed to generate intelligent, context-dependent hints [2], and allows for visualization of student approaches to problem solving [8]. The system is capable of continued refinement as new data is provided. In [2] we performed a feasibility study for hint generation using historical student data, and found that we could have made hints available for 71% of past student steps using one semester of past data. Our results indicated valuable tradeoffs between hint specificity and the amount of data used to create an MDP. In [3], we discussed how we added the method to existing CAI used to teach logic and reported the results of our initial pilot study.

Ill-defined domains, such as medical diagnosis, computer programming and legal reasoning, pose particular problems for ITS developers [12]. In particular, it is difficult to generate feedback for environments where there are many possible ways to solve a problem. Sequential Pattern Matching (SPM) [17] is a data-mining method used in ill-defined domains to extract frequent actions into plans. SPM has been used in a tutor to teach astronauts to use a robotic arm, where the tutor suggested a plan based on their current location in the problem. Like our approach, this method only uses good solutions and takes into account how often different actions occur, but this is specific to the robotic arm control domain.

3 Method

A Markov decision process (MDP) is defined by its state set S, action set A, transition probabilities P, and a reward function R [18]. For a particular point in a student attempt, our method takes the current problem features as the state, and the student's input as the action. Therefore, each student problem attempt can be seen as a graph, or Markov chain, with a sequence of states (each describing the solution up to the current point), connected by actions. We combine all student solution graphs into a single graph, by taking the union of all states and actions, and mapping identical states to one another. Once this graph is constructed, it represents all of the paths students have taken in working a particular problem. Typically, at this step value iteration is used to find an optimal solution to the MDP. A large initial value is set for the goal state, penalties for incorrect states, and a transition cost for taking each action. Setting a non-zero cost on actions causes the MDP to penalize longer solutions. We apply value iteration to assign reward values to all states in the MDP [18]. Once this is complete, the optimal solution corresponds to taking a greedy traversal approach in the MDP [4]. The reward values for each state then indicate how close to the goal a state is, while probabilities of each transition reveal the frequency of taking a certain action in a certain state.

In our original MDP method, all paths which solved the problem were directed to a goal state which was given a high reward value. The use of a single goal state works well when we know whether each student attempt is correct. Our new utility metric determines the "goodness" of a state by based on the frequency of each component step in the state. Unlike our original method where the goal state was known, the utility method has no known goal states so all terminal states are treated as possible goals. Terminal states are defined as those that are not errors, where no subsequent student actions were taken.

We derive our utility metric using techniques related to Latent Semantic Indexing (LSI), which are used to search large databases of text documents [11]. In LSI, terms refer to words, while for logic proofs, we define a term, or feature, as the statement a student derives in a single problem-solving step. Therefore, each attempt is composed of a sequence of statements. As in LSI, we use a term-document matrix, as shown in Table 2, to show the occurrence of each statement or term in each student attempt, marking a 1 for terms that occur and 0 that do not occur. We then compute the frequency by summing the columns. We set a percentage frequency threshold such that all state features above the threshold had a good potential of being a part of the solution. Setting this threshold can be done automatically or with the help of a domain expert. We discuss selection of the threshold in this experiment in section 4.2.

Once a list of frequent statements is determined, we calculate initial utility values for all terminal states (leaves) in the MDP, which are potential goal states. This replaces our original approach of creating a goal state with a single positive value. The utility value of a terminal state is the sum of the value for each statement (or feature) in the student attempt. The value of each step is positive if it was frequent and negative otherwise. Error states receive a high negative start value, and all other states start at zero. After the initial values are set, value iteration is applied until the state values become stable.

4 Experiment

We applied the utility method on a known dataset from CAI used to teach logic. This dataset had been used in previous research and had state values calculated using the MDP method. We compared the results of both methods paying special attention to states that had different best values.

4.1 Data

We have used the NCSU-Proof1 dataset in [2] and [4]. The data comes from four fall semesters of 2003-2006, where an average of 220 students take the discrete math course each year. Students attend several lectures on logic and then use the Proofs Tutorial to solve 10 proofs. Sixty percent of students used direct proof when solving proof 1. We extracted 537 of students' first attempts at direct solutions to proof 1. An example attempt of proof 1 is shown in Figure 1.

Statement	Line	Reason
1. a → b		Given
2. c → d		Given
3. ¬ (a → d)		Given
¬ a v d	**3**	**rule IM (error)**
4. a ^ ¬ d	3	rule IM implication
5. a	4	rule S simplification
b	**4**	**rule MP (error)**
b	**1**	**rule MP (error)**
6. b	1,5	rule MP modus ponens
7. ¬ d	4	rule S simplification
8. ¬c	2,7	rule MT modus tollens
9. b ^ ¬c	6,8	rule CJ conjunction

Figure 1. Sample student attempt to NCSU Proof 1

The data were validated by hand, by extracting all statements generated by students, and removing those that 1) were false or unjustifiable, or 2) were of improper format. We also remove all student steps using axioms Conjunction, Double Negation, and Commutative, since students are allowed to skip these steps in the tutorial. After cleaning the data, there were 523 attempts at proof 1. Of these, 381 (73%) were complete and 142 (27%) were partial proofs, indicating that most students completed the proof. The average lengths, including errors, were 13 and 10 steps, respectively, for completed and partial proofs. When excluding errors and removed steps, the average number of lines in each student proof is 6.3 steps. The validation process took about 2 hours for an experienced instructor, and could be automated using the existing truth and syntax-checking program in our tutorial. We realized that on rare occasions, errors are not properly detected in the tutorial (less than 10 premises were removed).

Table 1. Sample states derived from example student attempt in Figure 1

State	State Description	Error	Action	Result State
1	a → b, c → d, -(~(a → d)		IM	2
2	a → b, c → d, -(~(a → d), -~a v d	Yes		1
1	a → b, c → d, -(~(a → d)		IM	3
3	a → b, c → d, -(~(a → d), a ^ -~d		S	4
4	a → b, c → d, -(~(a → d), a ^ -~d, a		MP	5
5	a → b, c → d, -(~(a → d), a ^ -~d, a, b	Yes		4
4	a → b, c → d, -(~(a → d), a ^ -~d, a		MP	6
6	a → b, c → d, -(~(a → d), a ^ -~d, a, b		S	7
7	a → b, c → d, -(~(a → d), a ^ -~d, a, b, -~d		MT	8
8	a → b, c → d, -(~(a → d), a ^ -~d, a, b, -~d, -~c		CJ	9
9	a → b, c → d, -(~(a → d), a ^ -~d, a, b, -~d, -~c, b ^ -~c			

An MDP was created from this data using our MDP method resulting in 821 unique states. Table 1 shows the states created in our MDP for the student attempt shown in Figure 1. In the logic proofs domain, a step in the solution is considered to be a new

statement added to the previous state. For example, in state 2, the statement ~a v d is the next "step" in the problem, however, since it is an error detected by the software, this statement is deleted and the problem is returned to state 1.

4.2 Utility Process

If our data are labeled, we simply connect all valid solutions to a synthetic goal state. However, when goal states are unknown, we need a way to label or measure correct attempts. Our proposed utility metric is one way that assumes that frequent features are important in the problem solution. From our 523 attempts, we extracted 50 unique statements (including 3 given statements) and calculated their frequencies. A partial sample of the statement-attempts matrix is shown in Table 2. Note that only the first three attempts and only those statements appearing in those three attempts are shown. The complete statements-attempts matrix would contain all 50 statements in rows and all 523 attempts in the columns. To determine statement frequency, we sum each column.

Table 2. Sample matrix showing the occurrence of elements in student solution attempts.

	Terms								
	a → b	c → d	-(a → d)	a ^ -d	a	b	-d	-c	b ^ -c
Attempt 1	1	1	1	1	1	1	1	1	1
Attempt 2	1	1	1	0	0	1	1	1	1
Attempt 3	1	1	1	1	0	0	1	0	0

We then graphed the frequency of each statement, and the frequencies of statements (number 1-47) with more than 1 usage are shown in Figure 2. Statements 1-22 occurred only once in the data, while statements 43-47 occur in over 370 unique student attempts. Since there is variation in correct solutions, we set a low threshold frequency of 8 attempts for statements we might consider "useful" in a proof, and this is true for statements 29-47 and higher. A logic instructor verified that all the statements 29-47could be expected to occur in correct student solutions, while those with fewer were not as useful. The threshold value could be chosen automatically using the frequency profile.

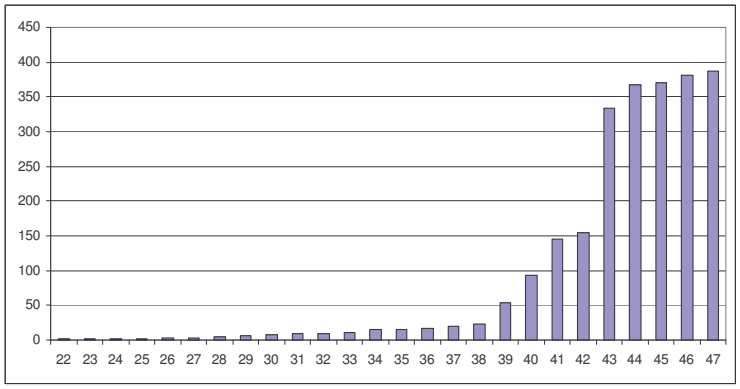

Figure 2. Frequency of Statements in Proof 1

Next we calculate the initial values for MDP states. For the possible goal states (valid terminal states), the initial value was a sum of the individual scores given to the component statements. Each statement score was +5 if its frequency was above the threshold and was -1 for those below. Error states received a value of -2, and all other states started at zero. Finally, after the initial values were set we ran a value iteration algorithm until the state values stabilized. Note that during value iteration, a -1 transaction cost was associated with each action taken.

4.3 Comparing Utility Method to MDP Method

We use an MDP along with its state values to generate hints that provide students with details of the best next state reachable from their current state [3]. To compare the utility method to our traditional MDP method we compared the effects of state values on the choice of the "best" next state. Both methods create the same 821 states, of which 384 were valid, non-error states. From the valid states, 180 states had more than one action resulting in new state. These 180 states are the ones that we focused on since these are the only states that could lead to different hints between the two methods. Comparing the two methods, they agree on the next best state in 163 states out of 180 (90.56%). For the remaining 17 states where the two methods disagreed, experts identified 4 states where the MDP method identified the better choice, 9 states where the utility method identified the better choice, and 4 states where the methods were essentially equivalent. These 17 states can be seen in Table 3, with the highlighted cells marking the expert choice.

Table 3. States where the methods disagree (17 total states)

State	State Description	# of Possible Actions	MDP next State	MDP added Statement	MDP Value	Utility Next State	Utility added Stmt	Utility Value
1	a>b,c>d,-(a>d)	14	53	-d>-c	49.91	2	-(-a+d)	10.57
2	a>b,c>d,-(a>d),-(-a+d)	9	238	b	98.00	579	(a*-d)	14.00
3	a>b,c>d,-(a>d),-(-a+d),a*-d	8	310	-(a*-b)	93.00	310	-(a*-b)	29.00
4	a>b,c>d,-(a>d),-(-a+d),a*-d,b	4	5	-c	87.72	119	-d>-c	38.74
7	a>b,c>d,-(a>d),-a+b	6	780	-d>-c	29.00	780	-d>-c	18.00
8	a>b,c>d,-(a>d),-a+b,-c+d	2	599	b+-c	99.00	10	-(-a+d)	18.02
19	a>b,c>d,-(a>d),-(-d>-a)	2	20	-(d+-a)	27.13	274	a*-d	7.67
36	a>b,c>d,-(a>d),-c+d,-(-a+d),a*-d	2	170	-c	24.33	186	b	6.04
53	a>b,c>d,-(a>d),-d>-c	5	460	-(-a+d)	96.00	684	-b>-a	21.00
82	a>b,c>d,-(a>d),(a*-d),-c	3	84	b	99.00	320	-(a*-b)	14.00
91	a>b,c>d,-(a>d),-(-a+d),a*-d,-d>-c	3	92	(a*-d)>(b*-c)	99.00	473	b	19.33
119	a>b,c>d,-(a>d),-(-a+d),a*-d,b,-d>-c	3	773	-c+d	98.00	120	-c	42.71
156	a>b,c>d,-(a>d),-(-a+d),a*-d,-a+b	2	208	-d>-c	98.00	423	b	29.60
228	a>b,c>d,-(a>d),a*-d,-d>-c	2	288	-c	76.20	619	b	14.00
333	a>b,c>d,-(a>d),a*-d,-c+d	2	334	-a+b	99.00	785	-c	19.00
337	a>b,c>d,-(a>d),-a+b,-(-a+d),a*-d,b	2	646	-c+d	61.67	339	-c	20.20
522	a>b,c>d,-(a>d),-(-a+d),a*-d,b,-c+d,-d>-c	2	766	-c	99.00	523	-c+d	30.00

These results show that the unsupervised utility metric does at least as good a job as the traditional MDP method in determining state values even when it is not known if the student attempt was successful. In all cases, the hints that would be delivered with either

method would be helpful and appropriate. We believe that the utility metric provides a strong way to bias our hint selection toward statements derived by a majority of students, which may give students hints at a more appropriate level.

Before we derived the utility metric presented here, we considered modifying MDP values by combining them in a weighted sum with a utility factor after value iteration had been completed. In our first attempt to integrate frequency and usefulness into a single metric, we analyzed all of our attempts to find derived statements that were necessary to complete the proof, by doing a recursive search for reference lines starting from the conclusion back through a student's proof. For each attempt, this "used again" value was set to 1 if a derived statement could be reached backward from the goal, and zero otherwise. We summed the total times a statement was used again, and compared this with the total times a statement occurred in attempts. Table 4 shows the comparison of the frequency and used again values for all statements where used again was more than 1. The values have no real correlation, but most items that were used again had high (>7) frequencies, so we decided that frequency was a relatively good indictor of usefulness in the logic proof domain. The "used again" calculation is possible in the logic domain because students must provide a justification for the current statement using rules and references to prior statements. In other domains, this may not be possible but we believe that frequency of occurrence in student solutions indicates that a step is either needed, or is a very common step that will only skew state values in a consistent way.

Table 4. Comparison of frequency and used again

Statement Number	Statement	Frequency	Used Again
30	(a+c)>(b+d)	8	2
31	-(a*c)+(b*d)	9	2
32	-(d+-a)	9	7
33	(a*-d)>(b*-c)	10	10
34	-(-d>-a)	15	7
35	-b>-a	16	5
36	-(c*-d)	17	6
37	(a*c)>(b*d)	20	4
38	-(a*-b)	23	8
39	(a*-d)	53	44
40	-d>-c	93	71
41	-a+b	145	69
42	-c+d	155	80
43	-(-a+d)	334	300
44	-c	367	344

5 Conclusion and Future Work

The most important feature of the MDP method is the ability to assign a "value" to the states. This allows the tutor to identify the action that will lead to the next state with the highest value. In this research we have shown that the utility metric that assigns values to terminal states based on the component steps in the state can be used to achieve hint-source decisions as one that assigns a single value to all goal states.

The main contribution of this paper is to show how this new utility metric can be used to generate MDP values based on features of student solution attempts. Our results show that the utility metric could be used to achieve equivalent or better hints than our prior single-goal MDP approach. This is significant because the utility metric does not require a known goal state, so it can be applied in domains where the correctness of the student attempts is unknown, or difficult or costly to compute. We believe that this utility metric combined with our MDP method can be used to generate hints for a computer programming tutor. In this domain, it is difficult to say that a program is complete, but it is possible to say whether specific features are represented. The method of using a term-document matrix to determine utility could also be extended into using more complicated LSI techniques which would be a natural fit for tutors using textual answers such as essay response questions. Text based answers are prevalent in legal reasoning and medical diagnosis tutors.

In our future work, we plan to construct and compare traditional and utility-based MDPs for other proofs and for student work in other domains. We also plan to analyze our logic tutor hint data to see if the utility method would result in different hints. This will give an indication of how much the utility technique is needed for our logic tutor. We also plan to analyze log data compiled from a C++ programming course to determine what kind of features we might extract and how well we can calculate the utility of those features.

References

[1] Aleven, V., McLaren, B. M., Sewall, J., & Koedinger, K. (2006). The Cognitive Tutor Authoring Tools (CTAT): Preliminary evaluation of efficiency gains. In M. Ikeda, K. D. Ashley, & T. W. Chan (Eds.), *Intelligent Tutoring Systems* (ITS 2006), (pp. 61-70). Berlin: Springer Verlag.

[2] Barnes, T., Stamper, J. (2008). Toward Automatic Hint Generation for Logic Proof Tutoring Using Historical Student Data. In E. Aimeur, & B. Woolf (Eds.) *Intelligent Tutoring Systems (ITS 2008)*, pp. 373-382. Berlin, Germany: Springer Verlag.

[3] Barnes, T., Stamper, J., Croy, M., Lehman, L. (2008). A Pilot Study on Logic Proof Tutoring Using Hints Generated from Historical Student Data. In R. Baker, T. Barnes, J. Beck (Eds.) *Educational Data Mining (EDM 2008)*, pp. 197-201. Montreal, Canada.

[4] Barnes, T. and Stamper, J. Toward the extraction of production rules for solving logic proofs, In *Artificial Intelligence in Education, Educational Data Mining Workshop (AIED2007)*, pp. 11-20. Los Angeles, CA.

[5] Beck, J., Woolf, B. P., and Beal, C. R. ADVISOR: A Machine Learning Architecture for Intelligent Tutor Construction. In: *7th National Conference on Artificial Intelligence*, pp. 552—557. AAAI Press / The MIT Press, 2000.

[6] Conati, C. Gertner, A. and VanLehn, K. (2002). Using Bayesian Networks to Manage Uncertainty in Student Modeling. In User Model. User-Adapt. Interact, vol. 12 (4).

[7] Croy, M. Graphic Interface Design and Deductive Proof Construction, *Journal of Computers in Mathematics and Science Teaching*, 1999, 18(4), 371-386.

[8] Croy, M., Barnes, T., and Stamper, J. Towards an Intelligent Tutoring System for propositional proof construction. In P. Brey, A. Briggle & K. Waelbers (eds.), *Proc. 2007 European Computing & Philosophy Conf.*, Amsterdam: IOS Publishers.

[9] Heffernan, N. and Koedinger, K.(2002). An Intelligent Tutoring System Incorporating a Model of an Experienced Human Tutor. In *Intelligent Tutoring Systems*, 596–608.

[10] Koedinger, K., Aleven, V., Heffernan. T., McLaren, B. & Hockenberry, M. Opening the door to non-programmers: Authoring intelligent tutor behavior by demonstration. *Intelligent Tutoring Systems 2004*.

[11] Landauer, T. K., Foltz, P. W., and Laham, D. (1998). Introduction to Latent Semantic Analysis. *Discourse Processes*, 25, 259-284.

[12] Lynch, C., Ashley, K., Aleven, V., & Pinkwart, N. (2006). Defining Ill-Defined Domains; A literature survey. In V. Aleven, K. Ashley, C. Lynch, & N. Pinkwart (Eds.), *Workshop on Intelligent Tutoring Systems for Ill-Defined Domains at the 8th International Conference on Intelligent Tutoring Systems* (p. 1-10).

[13] McLaren, B., Koedinger, K., Schneider, M., Harrer, A., & Bollen, L. Bootstrapping Novice Data: Semi-automated tutor authoring using student log files, In Workshop on Analyzing Student-Tutor Interaction Logs to Improve Educational Outcomes,. *Intelligent Tutoring Systems (ITS-2004),* Maceió, Brazil, August 30, 2004.

[14] Merceron, A. & Yacef, K.: Educational Data Mining: a Case Study. In *Artificial Intelligence in Education*, Amsterdam, Netherlands, IOS Press, 2005.

[16] Murray, Tom. Authoring intelligent tutoring systems: An analysis of the state of the art. *Intl. J. Artificial Intelligence in Education,* 1999, 10: 98-129.

[17] Nkambou, R., Mephu Nguifo, E., Fournier-Viger, P.: Using Knowledge Discovery Techniques to Support Tutoring in an Ill-Defined Domain. In *E. Aimeur, & B. Woolf (Eds.) Intelligent Tutoring Systems (ITS 2008)*, pp. 395-405. Berlin: Springer Verlag.

[18] Sutton, R. & A. Barto. Reinforcement Learning: An Introduction, 1998, The MIT Press, Cambridge, MA.

[19] Vygotsky, L. (1986). *Thought and language*. Cambridge, MA: MIT Press.