# Prevention of Off-Task Gaming Behavior in Intelligent Tutoring Systems

Jason A. Walonoski and Neil T. Heffernan

Worcester Polytechnic Institute, Computer Science Department, 100 Institute Rd, Worcester, MA 01601 USA
{jwalon, nth}@wpi.edu

**Abstract.** A major issue in Intelligent Tutoring Systems is off-task student behavior, especially performance-based gaming, where students systematically exploit tutor behavior in order to advance through a curriculum quickly and easily, with as little active thought directed at the educational content as possible. This research developed both active interventions to combat gaming and passive interventions to prevent gaming. Our passive graphical intervention has been well received by teachers, and our experimental results suggest that using a combination of intervention types is effective at reducing off-task gaming behavior.

## 1 Introduction

Intelligent Tutoring Systems (ITS) have been shown to have a positive effect on student learning [1], however these effects may be negated by a lack of student motivation or student misuse, particularly "gaming" of the system [2]. Gaming is the systematic use of tutor feedback and help methods as a means to obtain correct answers with little or no work, in order to advance through a curriculum as fast or as easily as possible.

Within ITS there have been a variety of approaches towards remediation of gaming behavior in students [2], which are mostly active interventions focused on combating student gaming, with few approaches focused on prevention. This research aimed at exploring a more comprehensive approach using active interventions to combat gaming along with a passive method to prevent gaming within the *Assistments* mathematics ITS [3].

## 2 Prevention of Gaming

We developed three gaming interventions, two traditional active interventions, and one passive deterrent or prevention mechanism. The interventions were deployed and evaluated experimentally. Two active interventions were used to respond separately to the two types of hallmark gaming behavior: rapid-fire guessing-and-checking and hint/help abuse. These interventions were triggered by simple gaming detection algorithms, which marked a student as guessing-and-checking or abusing-hints *prima facie* of the appropriate surface-level characteristics [4]. When triggered a message was displayed to the offending student encouraging them to try harder, ask a teacher for help, or pursue other suitable actions.

M. Ikeda, K. Ashley, and T.-W. Chan (Eds.): ITS 2006, LNCS 4053, pp. 722 – 724, 2006.

The passive intervention sought to prevent gaming by providing visual feedback on student actions and progress. It had no triggering mechanism, and was continuously featured prominently on-screen for easy viewing by the student and teachers. Theoretically, gaming would then be prevented through Panopticon-like paranoia (when a fear of being watched, without knowing whether one is being watched at any given moment, causes self-corrective behavior) [5].

Our passive intervention (not shown) graphically plots all recorded student actions (such as problem attempts, hint requests, bottom-out hints) in a horizontal timeline. Each action has associated summary text that identifies and provides relevant details and results of the action on mouse-over. The horizontal distance between points reflects the amount of time between the actions. The vertical height of actions is based on their outcome (correct actions are higher than incorrect actions). Throughout the design, the ubiquitous traffic-light color conventions of modern society are used, where green is implicitly "good" or "correct," yellow is "caution," and red is therefore "bad" or "incorrect." The graphical plot was designed to (1) allow teachers and students to easily identify gaming behavior via emergent visual patterns, (2) thereby preventing gaming behavior in the students by the students themselves, and (3) providing a launching point for teacher intervention where gaming behavior or student misunderstandings are identified.

Once all three interventions were designed and implemented, we conducted an experiment to test their effectiveness within the *Assistments* system. One group of students (70 students) received both the active and passive interventions (group 1); while a second group (57 students) received no interventions (group 2). Both groups of students used the tutoring system for an average of 3 class periods (approximately 45 minutes each period), each session having their rate of gaming measured by our *prima facie* gaming recognition algorithm [4]. Then we swapped the conditions, so that group 1 no longer received interventions, while the group 2 began to encounter them. The students used the tutoring system for another class period, and the rate of gaming was compared before and after the swapping of conditions.

Before switching conditions, group 1 had an average rate of gaming that was almost half the rate of group 2 (an average of 3.62 occurrences of gaming per session compared to 6.235), suggesting that the interventions were perhaps having some sort of effect. However, in order to show that those differences were not the result of some sort of selection effect in the groups, the conditions were swapped. After the swap, both groups had decreased amounts of gaming. Group 1 reduced gaming on average by 2.8 occurrences per session, while group 2 decreased their gaming by an average of 4.4 occurrences per session. One-side t-tests were performed on both groups, to see if the resulting change was significantly different from zero, and in both cases the answer was yes ($p < 0.0001$, in both tests).

To determine whether there really was a bigger impact with group 2 – turning the intervention mechanisms on versus off – we conducted an analysis of variance (ANOVA) and the resulting p-value of .08 suggests that turning the interventions on (group 2) makes a bigger impact on *prima facie* gaming than turning them off (group 1). One possible interpretation and explanation of these results would be that when interventions are turned on students learn not to game, and once interventions are turned off, they simply continue not to game. Further analysis might reveal whether actual invocation or receiving of the active interventions is correlated with this

decrease in gaming, as opposed to simply the *possibility* of receiving them (a student might have never seen the active interventions when they were turned on if they were never gaming). Otherwise, we might be able to conclude that the decrease in gaming was due more to the passive intervention, or perhaps other factors. We leave the identification of the particular effects each factor for future work.

## 3 Conclusions

The goal of this research was to explore the intervention and prevention of off-task gaming behavior within the *Assistments* system. Three dynamic mechanisms were designed: two active interventions for hint-abuse and guessing-and-checking, and one passive intervention. Our experimental results suggest that the combined application of active and passive interventions successfully reduces off-task gaming behavior more effectively than no intervention mechanisms.

## References

1. Koedinger, K. R., Anderson, J. R., Hadley, W. H., & Mark, M. A. (1997). *Intelligent tutoring goes to school in the big city*. International Journal of Artificial Intelligence in Education, 8, 30-43.
2. Baker, R.S., Corbett, A.T., Koedinger, K.R., Wagner, A.Z. (2004) *Off-Task Behavior in the Cognitive Tutor Classroom: When Students "Game The System"*. Proceedings of ACM CHI 2004: Computer-Human Interaction, 383-390.
3. Razzaq, L, Feng, M., Nuzzo-Jones, G., Heffernan, N.T. et. al (2005). *The Assistment Project: Blending Assessment and Assisting*. Proceedings of the 12th Annual Conference on Artificial Intelligence in Education 2005, 555-562.
4. Walonoski, J.A., Heffernan, N.T. (2006). *Detection and Analysis of Off-Task Gaming Behavior in Intelligent Tutoring Systems*. Proceedings of the 8th International Conference on Intelligent Tutoring Systems.
5. Bentham, Jeremy. *The Panopticon Writings*. Ed. Miran Bozovic (London: Verso, 1995).

# Learning Linear Equation Solving Algorithm and Its Steps in Intelligent Learning Environment

Marina Issakova*

University of Tartu, Institute of Computer Science, J. Liivi 2, 50409 Tartu, Estonia
marina.issakova@ut.ee

**Abstract.** T-algebra is an interactive learning environment for step-by-step solving algebra problems, including linear equations. A step in T-algebra combines conversion by rules with entering of the result. The rules, which correspond to the steps of school algorithms, give the program information about the student's intentions and enable the program to check the knowledge of the student, to understand the mistakes and give feedback. T-algebra is intended to help learning solution algorithms and their steps with designed rules. This article describes the rules designed for linear equation solving in T-algebra.

## 1 Introduction

At school most of algebra problems (including linear equations) are solved using some algorithms. To solve linear equations, the student should know step-by-step solution algorithm and know how to perform each algorithm step: choose a transformation rule corresponding to a certain operation in the algorithm, select the operands for this rule, and replace them with the result of the operation.

There are two different kinds of interactive learning environments available, which allow building step-by-step solutions. In the first kind of environments (such as MathXpert [2], AlgeBrain[1]), the student solves a problem working in terms of rules: selects a part of the expression and the rule. In such environments the student learns solution algorithm, but the learning of performing algorithm steps is passive, because the transformation itself is made by the computer. In the second kind of environments (for example, Aplusix [4]), the student can produce step-by-step solution themselves, because a solution step consists simply of entering the next line. Yet the program does not handle the solution algorithms of different types of problems.

T-algebra is an environment enabling to solve algebra problems step-by-step, including solving of linear equations. In the design of the T-algebra environment, we have been guided by the principle that all the necessary decisions and calculations at each step should be made by the student. For that T-algebra combines the two approaches described above: selection by rules is supplemented by entering the result. This gives the student the possibility to learn the algorithms and their steps. And it enables the program to check the knowledge and skills of the student, to monitor, whether the student works according to the algorithm and to diagnose errors.

* The author was supported by ICT graduate school.

M. Ikeda, K. Ashley, and T.-W. Chan (Eds.): ITS 2006, LNCS 4053, pp. 725 – 727, 2006.

## 2 Designed Rules for Linear Equation Solving in T-algebra

Problem solving in T-algebra takes place step-by-step. To make the program more intelligent, our own rule dialogue was designed. Each solution step consists of three stages: selection of the transformation rule, marking the parts of expression, entering the result of the operation. The student can make mistakes, receive feedback and ask for help at all three stages. The problem solution window is shown on Figure 1.

At the first stage of each solution step, the student has to choose the rule that he is going to apply. The information on which rule is applied enables the program to estimate, whether the student knows the algorithm for solving this problem and check more efficiently, whether the student's actions on the next stages are correct. The textbook algorithms were followed as closely as possible in the design of the rules, which correspond to the steps of school algorithms. We have tried to make the student's approach to solving the problems within the program parallel to the approach the student would take solving on paper. We hope that such work in the program will help the student to develop skills, which will carry over to the work on paper.

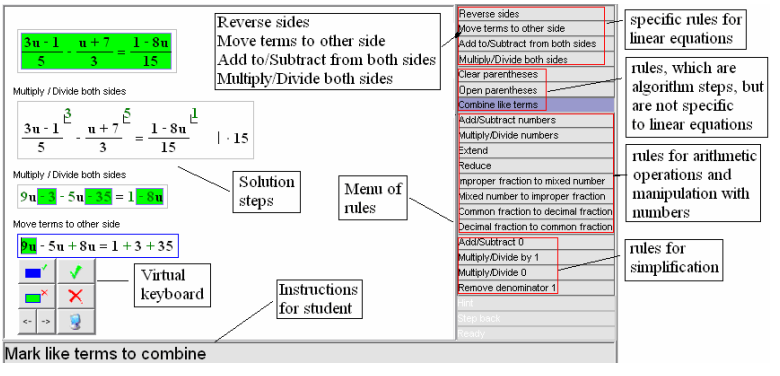


**Fig. 1.** The problem-solution window of the T-algebra program

The designed set of rules is complete, i.e., all exercises in this field are solvable with these rules. The set of rules consists of the new rules for the algorithm, which is being learned, and of the simplification and computation rules learned before (Fig. 1 shows the set of rules for linear equations). This set of rules is small enough to be displayed in the menu at all times, and it gives the possibility to diagnose whether the student knows which step to perform at the moment. In other environments the set of rules, which allows solving the same equations as in T-algebra, is much larger. For example, MathXpert has 11 specific rules instead of our 4 rules. With so many rules in MathXpert the student cannot see all the rules and cannot select an unsuitable rule, because "…only correct rules are offered as menu choices for you to choose" [2].

Designing the rules, we have taken into account results of research on students' mistakes made on paper [3], and have attempted to leave an opportunity for the student to make the same mistakes in T-algebra. We have also tried to make the rules interface as transparent as possible to be sure that mistakes made by the student are caused by misconceptions not by poor interface design.

Application of the rules *Multiply/Divide both sides* and *Move terms to other side* is displayed on Figure 1. Let us take a closer look at the last rule (Fig. 2). In order to apply this rule, the student has to mark the terms that he wants to move. The program checks, whether the selected parts are appropriate. If the marking was correct, the equation with boxes is written onto the next line. The boxes appear on other side of selected parts. The student can make most common errors [3], which he would do on paper: forget some moved term and not change the sign of a moved term. The program can diagnose these mistakes, can give appropriate error message and show the exact position (box) of the incorrect part. During the input, the student can ask for help (button with computer) and the program will put the right answers to the boxes.

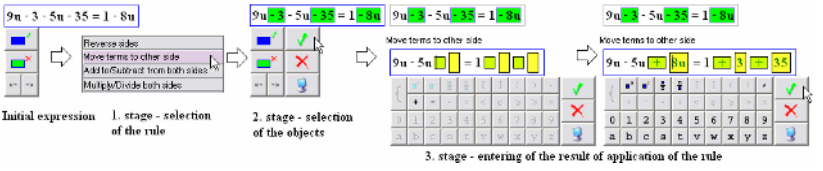


**Fig. 2.** Applying the rule *Move terms to other side*

## 3 Conclusion

In existing systems the student either can learn algorithm steps and the program does not handle the solution algorithms (Aplusix) or the student can learn only algorithm and the learning of performing algorithm steps is passive (MathXpert, AlgeBrain). We have succeeded to create such rule dialogue in T-algebra that gives the student the possibility to learn both solution algorithms and their steps, to make the same mistakes as on paper and enables the program to give understandable feedback about mistakes. Designing the set of rules, we followed school solution algorithms. As we have seen the designed set of rules is small enough to be displayed in the menu and this gives the student the possibility to choose. The design of rules and of rule dialogue is most important part in T-algebra that distinguishes it from other environments.

## References

1. Alpert, S.R., Singley, M.K., Fairweather, P.G.: Deploying Intelligent Tutors on the Web: An Architecture and an Example. International Journal of Artificial Intelligence in Education 10(2) (1999) 183-197
2. Beeson, M.: MathXpert : un logiciel pour aider les élèves à apprendre les mathématiques par l'action. Sciences et Techniques Educatives 9(1-2) (2002)
3. Issakova, M.: Possible Mistakes during Linear Equation Solving on Paper and In T-algebra Environment. In: Proceedings of the 7th International Conference on Technology in Mathematics Teaching, Vol. 1. Bristol, UK (2005) 250-258
4. Nicaud, J., Bouhineau, D., Chaachoua, H.: Mixing microworld and cas features in building computer systems that help students learn algebra. International Journal of Computers for Mathematical Learning 5(2) (2004) 169-211

# The Design of a Tutoring System with Learner-Initiating Instruction Strategy for Digital Logic Problems

Sheng-Cheng Hsu

Graduate School of Engineering Science and Technology,
National Yunlin University of Science and Technology, Yunlin, Taiwan
g9010802@dyuntech.edu.tw

**Abstract.** In this study, a tutoring system with a learner-initiating instruction strategy is proposed to help learners solve digital logic problems that they input to the system. After the system comprehends the problem inputted by the learner and determines the category of the problem, the system produces a solution plan for the learner. The learner solves the problem by following the plan step by step. After that, the system diagnoses the learner's answer when s/he executes a plan. If the learner's answer is incorrect, the system provides hints for the learner to revise the answer. Empirical evaluation results indicate that the system is effective in tutoring digital logic problem.

## 1 Introduction

Problem-solving oriented approaches have been widely applied to the instruction of mathematics, science, and engineering. Furthermore, many computer-based tutoring systems based upon problem-solving approaches are developed and applied to various fields successfully [1, 2, 3]. Most of these systems are based on an instructor-initiating instruction strategy and provide pre-designed problems for learners. When learners are asked to solve a problem, the system will instruct the learners what to do. This strategy is appropriate for beginners who develop their competence for problem solving from the very beginning. Nevertheless, such systems are generally not helpful if a learner encounters a problem that does not exist in the pre-designed database. Therefore, it is definitely not enough for an instruction system to be merely equipped with an instructor-initiating instruction strategy. In order to address this question, a tutoring system with a learner-initiating instruction strategy is proposed for helping more advanced learners. In this system, a learner becomes active in posing problems that he is interested in. After comprehending the problem posed by the learner, the system can provide him with relevant instruction materials and hints on problem solving. Currently, the system is applied to a course in digital logic, which is an important course in the department of electrical engineering in a university. In this course, students are taught techniques of digital circuit design including binary, combination and sequential logical circuit design, algorithmic state machine design, and VHDL modeling of digital circuits, etc. [4]. After accepting a problem inputted by a student, the system provides a solution plan and hints to help the student solve the problem.

M. Ikeda, K. Ashley, and T.-W. Chan (Eds.): ITS 2006, LNCS 4053, pp. 728 – 731, 2006.

## 2 System

The system assists a learner in the problem solving process described by Glass and Holyoak [5]. Three steps followed by the system are explained in the following.

*Step1: Comprehend a problem given by the learner*: In order to use the system, a learner enters the six components of a problem through the input interface (Fig. 1). If a learner has a problem, "Find the minimum product-of-sums expression for the function, F = BC'D'+BC'D +A'C'D'+BCD'+A'B'CD', using Karnaugh map." Firstly, the learner needs to select the problem topic "Boolean function minimization using Karnaugh map." Then the learner selects the problem goal "Find the minimum POS expression." Next the learner selects the problem operation "Karnaugh map". Then the system generates the problem constraints automatically according to the problem operation. Finally, the learner selects the problem object and inputs the content of the object. The problem object is "SOP expression" and the system provides a textbox for learner to input its content: "F = BC'D'+BC'D +A'C'D'+BCD'+A'B'CD' " (Fig. 1).

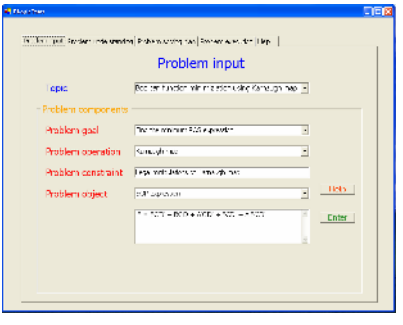


**Fig. 1.** The problem input interface

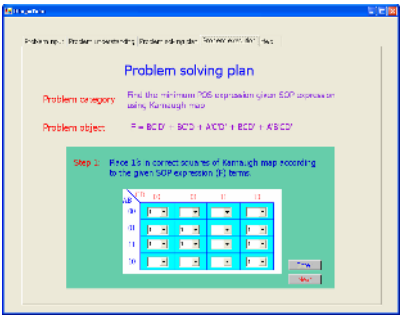


**Fig. 2.** The plan execution interface

*Step2: Generate candidate solution plans for the learner to choose:* After accepting the problem components from a learner, the system understands the problem with the help of a knowledge base. If the learner's input is correct, the system can identify the category of the problem and provides the relevant information (lexicon, definition, theory, and method) to help the learner understand the problem. For example, if the learner does not understand the term "Minimization product of sums", s/he can select the term, and then the system will provide its definition. After the learner understands the problem, the system retrieves the problem solving plans according to the category of the problem. All these plans can produce correct answers. The learner can choose any plan and the system will provide explanations and examples to assist the learner to comprehend the plan.

*Step3: Diagnose the execution of the learner's chosen plan*: After the learner understands the plan, he solves the problem step by step according to the chosen plan (Fig. 2). The system diagnoses the learner's answer when s/he executes a plan. If the learner's answer is incorrect, the system provides a hint for him to revise the answer.

## 3 Experiment

We evaluated the effectiveness of this system in training students to solve digital logic problem. Ninety freshmen in two classes participated in this experiment. One class with 47 students was assigned to the experimental group while the other class with 43 students was assigned to the control group. The following experiment steps were taken: a pre-test, a problem solving experiment, and a post-test. Firstly, the pre-test including thirty questions was used to assess the students' problem solving skills. The results showed no significant difference between the two classes in their pre-test scores ($t$ = 0.108, $p$ = 0.914 > 0.05). This implied that the two groups had no difference in their initial problem solving skills. The problem solving experiment lasted for three weeks. The instructor asked the students to solve twenty problems each week. The experimental group solved the problems with the help of the system, whereas the control group solved the problem without the system. After the experiment, a post-test with thirty questions was given and analyzed with an independent sample T-test shown in Table 1. The results showed the scores of the experimental group were significantly higher than those of the control group. This shows that system is useful in training students to solve digital logic problems.

**Table 1.** The result of independent samples *T*-test

| Group | N | Average score | SD | *t* | *p* |
|---|---|---|---|---|---|
| Experiment | 47 | 79.7234 | 9.10495 | 4.927* | 0.00 |
| Control | 43 | 68.1628 | 12.97237 | | |

*$P < 0.01$

## 4 Conclusion

In this study, a computer-assisted system with a learner-initiating instruction strategy is proposed to help freshman solve digital logic problems. A learner can input the four components of a digital logic problem and the system comprehends the problem with a knowledge base. Next, the system constructs the problem component content and generates the problem solving plans. The learner chooses a plan and solves the problem step by step. The system can diagnose the learner's answer when s/he executes the plan. In the experiment to evaluate the tutoring effectiveness of the system, 90 freshman participated as the experimental and control groups. By comparing the two groups' performance in the pre-test and the post-test, we conclude that the system is effective in tutoring.

## References

1. Chang, J. Chang M., Lin J.-L., Heh, J.-S.: Implements a diagnostic intelligent agent for problem solving in instructional systems. Proceeding of IWALT. (2000)
2. Looi, C. K., Tan, B. T.: A cognitive-apprenticeship-based environment for learning word problem solving. Journal for Research in Mathematical Education. 17, (1998) 339–354

3. Reisslein, J. Atkinson, R. K., Seeling, P., Reisslein, M.: Investigating the presentation and format of instructional prompts in an electrical circuit analysis computer-based learning environment, IEEE transactions on education. 48 (2005) 531-539
4. Roth, C. H.: Fundamentals of logic design. 5th edn. Thomson Brooks/Cole (2003)
5. Glass, A. L., Holyoak, K. J.: Cognition. 2nd edn. Random House, New York (1986)