Automatic Feedback and Resubmissions as Learning Aid

Lauri Malmi

Dept. of Computer Science and Engineering
Helsinki University of Technology
P.O. Box 5400, 02015 HUT, Finland
lma@cs.hut.fi

Ari Korhonen

Dept. of Computer Science and Engineering
Helsinki University of Technology
P.O. Box 5400, 02015 HUT, Finland
archie@cs.hut.fi

Abstract

Feedback based on automatic assessment of students' solutions is an important aid for students' learning process in self study and distance learning. Most automatic assessment systems allow students to revise their solutions after getting the feedback and resubmit their work to be able to complete the exercise. In this paper, we analyze the effect of resubmission in detail in the context of automatically assessed algorithm simulation exercises. In the target system TRAKLA2, students can revise their answers as many times as they wish, but each trial requires to restart the exercise with new random data. We present statistical results from a course with 600 students and show that our method that combines resubmissions and exercises with randomized initial data has a positive effect on learning results.

1. Introduction

Learning tools that automatically assess and give feedback on learners' performance provide valuable help for both teachers and learners. From teacher's point of view, the main motivation is to save time and increase the amount of feedback on large courses. For learners, automatic feedback is very useful, because it supports self study and distance learning. It is often much better to get instantly even simple feedback than to get advanced human feedback many days afterwards, or even worse to get no feedback at all.

Many automatic assessment systems have been developed since early 1990's. Some of their main application areas include assessing programming exercises [1, 4, 12], algorithm simulation exercises [2, 5, 7], object-oriented designs and flowcharts [3], or SQL-statements [10]. In such applications the answers can be expressed in a formal way, which makes it feasible to automatically analyse the answers. Automatic assessment of answers composed of free textual sentences, however, is evidently much harder and is not discussed further in this paper.

There are some important features in automatic assessment systems that are relevant to this paper. First, they generally allow learners to *resubmit* their answers after getting feedback from the system. Thus, learners can reconsider their answers to find and understand the errors they have made, and submit a revised version of the answer. The number of allowed submissions is typically a parameter determined by the teacher. Second, some systems utilize *random data* in assignments. For programming assignments, this typically means randomized input instead of fixed input for the programs to be tested. On the other hand, randomization can also mean *personalization* of the assignment. For example, in the TRAKLA system [5] each learner simulates the working of the same algorithm, but each of them has different input data. This feature effectively prevents plagiarism — a problem often encountered when assessing programming exercises.

Although automatic assessment systems have been in production use over 10 years, very little evaluation research has been carried out. Mitrovic [11] analyzed the learning results when SQL tutor was used. He used several different levels of feedback given by the system, such as plain positive/negative evaluation, tips, highlighting one or all errors, and showing a partial or full solution. The main result showed that feedback giving tips or pointing out the errors is the best method supporting learning. Too simple feedback is not useful enough and giving the whole solution does not encourage learners to think enough on their own. Many of them just copy the answer.

Some other results indicate that high or unlimited number of allowed resubmissions discourage active pondering of exercises. Laine [8] has reported experiences with SQL trainer in which each student submitted answers about 5 tries per task. However, a small number of students submitted answers much more often, sometimes without any actual change between two consecutive submissions, if there was no limit for the number of allowed submissions. We have similar experiences with Ceilidh [1] and Scheme-Robo [12], which we use for assessing programming exer-

cises. If the number of submissions is allowed, for example, up to 20 times, many learners do not concentrate on finding the errors in their programs on their own. Instead, they use the automatic assessment system as a kind of debugger: "Try something and look at if it works".

Other evaluation research has been carried out by Korhonen, *et.al.* [6]. In this research, university students were split into randomized groups: one group was solving automatically assessed algorithm simulation exercises in the web, and the other group solved similar exercises in small groups supervised by two tutors. No significant differences were found between the average learning results of the groups. Another research (Malmi *et.al* [9]) concentrated on analyzing statistical results of using the TRAKLA system during a period of ten years. The system allowed the student to re-submit the answers typically 3-5 times and the points from the best submission were recorded. The results indicated that the option for the resubmission does not affect learning very much as such. However, when this option was incorporated with an encouraging grading policy in which students could get better grades by solving more exercises, the learning results turned out to be much better.

In this paper, we consider more closely the effect of resubmission on learning. We present the research carried out with the TRAKLA2 system in spring 2003. In this system, there is an important difference compared with the old TRAKLA system. The number of allowed resubmissions is *unlimited*. After each submission, however, the learner can redo the same assignment, but with *fresh random data* instead of the *same* initial data as with the old system. It turned out that the course results were better than with the old system.

We present the functional characteristics of TRAKLA2 in Section 2. Following this, in Section 3, we present the results of the research findings from the Data Structures and Algorithms course. Finally, in Section 4, we conclude the main points of our paper.

2. TRAKLA2

TRAKLA2 is a web-based learning environment dedicated to distribute *visual algorithm simulation exercises* [7] in a Data Structures and Algorithms course. In *algorithm simulation*, the learner manipulates a given data structure using tools provided by the GUI. Typically this means drag & dropping graphical entities such as keys, nodes and references to new positions on the screen to simulate the operations a real algorithm would do. Consequently, the system performs the corresponding changes on the underlying data structures, and redraws the picture.

The system is able to create an applet that portrays individually tailored exercises in which each learner has different initial data structures that he or she manipulates by

means of the available functionality. Actually, an exercise is individually tailored *each time* the exercise is initialized. This feature has several advantages over traditional homework exercises as learners can freely collaborate without the temptation to copy answers from each other. In addition, the system is capable of automatically assessing the exercises by recording the sequence of simulation operations, and comparing it with another sequence generated by the actual implemented algorithm. Based on this, TRAKLA2 can give immediate feedback to the learner. This supports self studying at any place or time.

Figure 1 shows an example exercise in which the learner is to insert (drag and drop) the keys from the given array (initial data) one by one into the binary search tree. After completing the exercise, the learner can ask the system to grade his or her performance. The feedback received contains the number of correct steps out of the maximum number of steps, which is turned to points relative to other exercises. The final points are determined by the best points received among all the submissions for a single exercise. The learner can also view the model solution for the exercise as an algorithm animation sequence. The model solution is shown as a sequence of discrete states of the data structures, which can be browsed backwards and forwards in the same way as learner's own solution.

After viewing the grading results, the learner can either submit the solution to the course database or restart the same exercise with different input data. If the model solution has been requested, grading and submission are disabled until the exercise has been reset again.

The system collects data [13] during the exercise sessions. The idea is to monitor students as a group. The statistical data is gathered to obtain feedback and insight how the students are performing during the exercise sessions, and to improve the system further.

Data is logged each time the user performs one of the following *significant* GUI operations: Exercise initialization (or Reset), Grade, Open/Close model solution, or Submit an exercise. For each record, this data is combined with the following information the learner gives directly or indirectly in the login phase and that is stored into the course database: student ID number, course code (several courses can be active at the same time), and exercise code (selected exercise). Each record also includes a time stamp of the operation. Moreover, to follow the solving process, an additional log entry is inserted each time there is over one minute idle time (the user does nothing) during the exercise session.

Each initialization of an exercise asks the server to generate a new seed that is used to randomize the input data. This seed is also stored in the database in order to be able to reconstruct the exercise afterwards, if necessary. Needless to say, the solution and the points are also stored each time the learner submits his or her answer.

Figure 1. TRAKLA2 applet. The exercise window includes the data structures and push buttons. The model solution window is open in the front.

3. Results

TRAKLA2 was in test use in our Data Structures and Algorithms course in Spring 2003 with over 600 students that were divided in two separate groups. The CS major students ($N=230$) had 8 exercises and the CS minor students ($N=390$) 6 exercises, thus making the overall number of different exercises 14. Some exercises were fairly trivial such as traversing a binary tree in preorder, but others were trickier, for example, inserting items into an AVL tree or a red-black tree. The overall number of exercise initializations (including Resets) during the course was almost 23000.

We analyzed the data gathered during the course to discover what kind of effect *resubmissions* (or the number of grading requests) had on students' overall performance. We also compared the results from the TRAKLA2 to the results from the old TRAKLA system in which the number of resubmissions was limited. We knew from the previous research [6] that there is a significant correlation between the points received from the TRAKLA exercises and the points received from the final examination. Similar correlation was expected with TRAKLA2.

In the following analysis, we do not distinguish the case in which the learner submits the solution to the database

from the case when only grading was requested because the difference is not interesting here. For simplicity, we use the term resubmission consistently for both of these cases.

3.1. Student performance

The overall performance in the following six TRAKLA2 exercises was studied: insertion to binary search tree (BST-insert), deletion from binary search tree (BST-delete), insertion to AVL tree (AVL-insert), red-black tree (RBT-insert), digital search tree (DST-insert) and radix search trie (RST-insert). Typically, in each exercise, students had to insert 10-15 keys into the dictionary, and thus they had to understand the working of the algorithm properly to be able to complete the exercise successfully. When so many keys were to be inserted, a trial-and-error method was not very useful.

In Figure 2 we see the correlation between the number of resubmissions and the results of the red-black tree exercise. A similar exercise was available for CS majors using TRAKLA and CS minors using TRAKLA2. However, in TRAKLA the number of resubmissions was limited, and the initial data remained the same. This example provides evidence that resubmissions promote learning: students who submitted their solution only once got on average only half of the maximum points, and with the number of resubmissions the results steadily improved. However, it is interesting that in TRAKLA the results of the first submission were much better than with TRAKLA2.

Figure 2. The number of resubmissions vs. the average percentage of points in the red-black tree exercise.

A similar schema was regularly observed with other exercises, as well. The main difference between the exercises was the average number of resubmissions performed. With

easier exercises there were considerably lower number of resubmissions than with red-black trees.

3.2. Time on task

Next we compared the time spent on exercises with the results. This study confirmed the fairly obvious assumption that the time used for the exercise and the iterations pay off. Figure 3 shows the cumulative time spent with the red-black tree exercise before the last grade and the corresponding points received. Each dot corresponds to a single student. The maximum points available for this exercise was four. Similar charts were received with other assignments.

Figure 3. The relation between the time used and the points received in the red-black tree exercise.

Based on the time stamps, it was possible to calculate the total time the students spent for each different exercise (excluding the idle time). The net total times for the exercises are shown in Figure 4. We see that obviously the most difficult exercises were insertion to AVL tree and red-black tree, and the easier ones were insertion to binary search tree, digital search tree and radix search trie.

3.3. Examination results

Figure 5 depicts the correlation between the final examination points after the course and the percentage of the exercise points received during the course. There is a threshold of 50% to take the examination because that was the minimum points required to pass the exercises before the examination. The examination took place immediately after the course and had several different types of assignments, which tested knowledge on definitions, applying algorithms to solve problems, and comparing algorithms. Only one assignment out of four was based on algorithm simulation. The chart includes the sum of both courses (minor/major), even though they had slightly different assignments.

Figure 4. The average time spent on solving the exercises.

During the course, students solved 6-8 TRAKLA2 exercises but 23-24 TRAKLA exercises. We therefore provide here the comparison for the TRAKLA results only, which indicate a clear correlation between the exercise and examination points. The corresponding chart for the TRAKLA2 exercises was not so obvious, even though those who failed the examination had received considerably less exercises points than those who passed it. One reason for the minor correlation obviously is there were fewer exercises, and those who passed the exam had received almost full points of the exercises.

Figure 5. The examination points vs. the average percentage of TRAKLA points.

4. Conclusion

In this paper, we have analyzed our experiences on the TRAKLA2 learning environment with the algorithm sim-

ulation exercises in which resubmission has been combined with randomization of initial data in a novel way. The students can submit and revise their solution to an exercise arbitrary number of times, but each time they have to restart the assignment with fresh random initial data. This obviously means more challenge and work for the students, compared with the previous resubmission policy in TRAKLA, where the number of submissions was tightly limited but the students were allowed to continue solving the same assignment after getting the feedback.

Due to such a change in the resubmission policy we assumed that people would give up more easily because the exercise must be restarted with new input data. On the contrary, the results indicated that this was not the case. Students got clearly better results, actually better than we assumed. The average results were well over 90% of maximum value with such non-trivial assignments as inserting many keys to different types of dictionaries. However, we also observed that even though using a trial-and-error method in solving the exercises is not very useful, the results of the first submissions were considerably lower than in the older system. Obviously many students prepared their first solution more carelessly in TRAKLA2 than in TRAKLA.

A closer analysis revealed that students took the resubmission option seriously to solve the assignments correctly. Since the system allows no guessing how to proceed in solving the assignment, students really had to think what they were doing, and learned better. Solving the same exercise several times with different initial data aids building a viable mental model of the topic. One has to understand the algorithmic principles involved in order to be able to pass such an exercise.

In this research, TRAKLA2 assignments dealt only with various types of dictionaries, traversing binary trees and binary heap operations. We have implemented many new exercises on hashing methods, sorting algorithms and graph algorithms, which are used on the course in spring 2004. Based on this larger data we can analyze students' learning behavior even better. An interesting challenge is to analyze the submitted simulation sequences in more detail for better understand what kind of errors there occur most. This information will then be used to tune the course material.

Acknowledgments Many results in this paper are based Petteri Torvinen's work in his Master's thesis. We also wish to thank the members of the Matrix group who have aided us in developing TRAKLA2: Ville Karavirta, Jussi Nikander, Harri Salonen, Panu Silvasti, and Petri Tenhunen.

References

- [1] S. Benford, E. Burke, E. Foxley, N. Gutteridge, and A. M. Zin. Ceilidh: A course administration and marking system.

- In *Proceedings of the 1st International Conference of Computer Based Learning*, Vienna, Austria, 1993.
- [2] S. Bridgeman, M. T. Goodrich, S. G. Kobourov, and R. Tamassia. PILOT: An interactive tool for learning and grading. In *The proceedings of the 31st SIGCSE Technical Symposium on Computer Science Education*, pages 139–143. ACM, 2000.
- [3] C. Higgins, P. Symeonidis, and A. Tsintsifas. The marking system for CourseMaster. In *Proceedings of the 7th annual conference on Innovation and technology in computer science education*, pages 46–50. ACM Press, 2002.
- [4] D. Jackson and M. Usher. Grading student programs using ASSYST. In *Proceedings of 28th ACM SIGCSE Tech. Symposium on Computer Science Education*, pages 335–339, San Jose, California, USA, 1997. ACM.
- [5] A. Korhonen and L. Malmi. Algorithm simulation with automatic assessment. In *Proceedings of The 5th Annual SIGCSE/SIGCUE Conference on Innovation and Technology in Computer Science Education*, pages 160–163, Helsinki, Finland, 2000. ACM.
- [6] A. Korhonen, L. Malmi, P. Myllyselkä, and P. Scheinin. Does it make a difference if students exercise on the web or in the classroom? In *Proceedings of The 7th Annual SIGCSE/SIGCUE Conference on Innovation and Technology in Computer Science Education, ITiCSE'02*, pages 121–124, Aarhus, Denmark, 2002. ACM.
- [7] A. Korhonen, L. Malmi, and P. Silvasti. TRAKLA2: a framework for automatically assessed visual algorithm simulation exercises. In *Proceedings of Kolin Kolistelut / Koli Calling – Third Annual Baltic Conference on Computer Science Education*, pages 48–56, Joensuu, Finland, 2003.
- [8] H. Laine. SQL-trainer. In *Proceedings of Kolin Kolistelut / Koli Calling – First Annual Baltic Conference on Computer Science Education, Report A-2002-1*, pages 13–17, Joensuu, Finland, 2002. University of Joensuu.
- [9] L. Malmi, A. Korhonen, and R. Saikkonen. Experiences in automatic assessment on mass courses and issues for designing virtual courses. In *Proceedings of The 7th Annual SIGCSE/SIGCUE Conference on Innovation and Technology in Computer Science Education, ITiCSE'02*, pages 55–59, Aarhus, Denmark, 2002. ACM.
- [10] A. Mitrovic. Learning SQL with a computerized tutor. In *Proceedings of the 29th SIGCSE Technical Symposium of Computer Science Education*, pages 307–311. ACM, 1998.
- [11] A. Mitrovic, B. Martin, and M. Mayo. Using evaluation to shape its design: Results and experiences with SQL-tutor. *User Modeling and User-Adapted Interaction*, 12:243–279, 2000.
- [12] R. Saikkonen, L. Malmi, and A. Korhonen. Fully automatic assessment of programming exercises. In *Proceedings of The 6th Annual SIGCSE/SIGCUE Conference on Innovation and Technology in Computer Science Education, ITiCSE'01*, pages 133–136, Canterbury, United Kingdom, 2001. ACM.
- [13] P. Silvasti, L. Malmi, and P. Torvinen. Collecting statistical data of the usage of a web-based educational software. In *Proceedings of the IASTED International Conference on WEB-BASED EDUCATION*, pages 107–110, Innsbruck, Austria, 2004. IASTED.