Experiences on Automatically Assessed Algorithm Simulation Exercises with Different Resubmission Policies

LAURI MALMI, VILLE KARAVIRTA, ARI KORHONEN, and JUSSI NIKANDER
Helsinki University of Technology

In this paper, we present our experiences in using two automatic assessment tools, TRAKLA and TRAKLA2, in a second course of programming. In this course, 500–700 students have been enrolled annually during the period 1993–2004. The tools are specifically designed for assessing algorithm simulation exercises in which students simulate the working of algorithms at a conceptual level. Both of these tools allow students to resubmit their solutions after getting feedback. However, the resubmission policy has changed considerably during the period. Those changes reflect the students performance in the exercises. We conclude that an encouraging grading policy, i.e., the more exercises they solve the better grades they achive, combined with an option to resubmit the solution is a very important factor promoting students' learning. However, in order to prevent aimless trial-and-error problem solving method, the number of resubmissions allowed per assignment should be carefully controlled.

Categories and Subject Descriptors: K.3.1 [**Computers and Education**]: Computer Uses in Education

General Terms: Algorithms, Human Factors

Additional Key Words and Phrases: Automatic assessment, resubmission, algorithm simulation

1. INTRODUCTION

Learning to program is at the heart of learning computer science and a large number of students are enrolling in introductory programming courses annually. A consequence of this is that teachers in many universities and institutes are facing a great number of assignments that require marking and grading.

One way to overcome this workload is to use automatic assessment tools, which are gaining wide acceptance among teachers. In addition to reducing the teacher's work, such tools provide other advantages as well. For instance, they especially promote the student-centered learning as well as the learning

Authors' address: L. Malmi, V. Karavirta, A. Korhonen, and J. Nikander, Helsinki University of Technology, Department of Computer Science and Engineering, PO Box 5400, 02015 HUT, Finland; email: {lma,vkaravir,archie,jtn}@cs.hut.fi

outcomes. This is because of the fact that automatic assessment is a tool to create more exercises for the students and, at the same time, provides them with more feedback, compared with traditional teaching methods [Carter et al. 2003]. This is very beneficial for learning, since programming is a skill that requires solving many practical exercises. Moreover, computerized feedback is very useful in self-studying when there is no instructor assistance.

The mainstream of research on automatic assessment has concentrated on designing and implementing systems for assessing solutions to programming exercises. Examples of such systems include VIOPE [Vihtonen and Ageenko 2002], Ceilidh [Benford et al. 1993], its successor CourseMaster [Higgins et al. 2002], ASSYST [Jackson and Usher 1997], and Scheme-Robo [Saikkonen et al. 2001]. The main purpose of these systems is to check the correctness of the submitted student programs by executing them with predefined input data and analyzing the program output or an internal representation that is produced. Additional features may include checking the program structure, coding style, or program efficiency.

There are, however, other related areas where the idea of automatic assessment can be exploited. These include systems for checking algorithmic exercises (PILOT [Bridgeman et al. 2000], TRAKLA [Hyvönen and Malmi 1993; Korhonen and Malmi 2000] and its successor TRAKLA2 [Korhonen et al. 2003]), as well as for analyzing, for example, object-oriented designs and flowcharts [Higgins et al. 2002].

At the Helsinki University of Technology (HUT), we have used Ceilidh since 1994 and Scheme-Robo since 1999 to check programming exercises. In a typical programming exercise, the assessment is based on testing the submitted program against *specification requirements* instead of *implementation requirements*. Although it is possible to set up exercises where the correctness of the final or intermediate results of a specific algorithm are checked, such checking may well fail to discover incorrect implementations. Moreover, checking most or all intermediate states of the algorithm makes such an approach quite impracticable. Thus, on the Data Structures and Algorithms Course we have had quite a different method for checking exercises.

In this paper, we discuss our experiences in applying automatic assessment in the context of a course on data structures and algorithms. Our aim is that students should first understand the working of an algorithm at a *conceptual level* and, in this way, build a viable mental model [Norman 1983] of the algorithm before they proceed to implementing it. Since 1991, we have used *algorithm simulation exercises* in our course with around 500–700 students a year. In these exercises, students simulate how the given data structures are changed during the execution of the given algorithm. The simulation involves pointing out the changes in the visual representation of the data structure after each step during the execution of the algorithm (a step could be, e.g., inserting a key into an AVL-tree, or coloring a visited node in a graph traversal algorithm). Such a simulation generates a sequence of data structure states that can be compared with a corresponding sequence generated by a real implemented algorithm. We emphasize here that in these assignments our students *are not required to code anything since all operations are carried out at a conceptual*

level. Exercises requiring implementation of data structures and algorithms are purposefully delayed until subsequent courses. This allows us to cover a broad range of sorting, searching, and graph algorithms in our course.

Our first venture into the field of automatic assesment was TRAKLA [Hyvönen and Malmi 1993; Korhonen and Malmi 2000], which was introduced in 1991. Initially, all assignments were sent to the students by email and their submissions were returned by email using a predefined textual format for presenting the data structures. In 1997, a WWW front end was added to allow *graphical construction* of the answers. At the same time, we allowed students to *resubmit* their solutions after they received the email containing the feedback on the solution. Because of the problems with extending the system, a wholly new application framework, called Matrix [Korhonen and Malmi 2002], was built in 2000–2001. This framework supports visual algorithm simulation and animation, and based on Matrix, a new exercise system, TRAKLA2, was built in 2003 [Korhonen et al. 2003]. This system gives a considerably wider selection of operations for the users than TRAKLA. In addition, it gathers much more information about student interaction when they solve the exercises [Silvasti et al. 2004] compared with TRAKLA. Such information helps us to better understand student behavior in their learning process.

Most automatic assessment systems allow students to resubmit solutions to a given exercise. We have adopted this feature into our systems as well, but during the past decade or so we have had many different resubmission policies. Initially, all students had personal randomized data for their exercises in TRAKLA that allowed no resubmissions. All checking was carried out once—after the deadline for submissions had passed. After resubmitting was permitted, we noticed that students really seem to take advantage of it. They improve their performance considerably if the course-grading policy also encourages it. However, because students were allowed to continue solving the same exercise after the feedback, we strictly limited the number of resubmissions. Typically 3–5 submissions were allowed for each exercise. After we changed to TRAKLA2, no such limitation was needed because each time students reinitialize the exercise, they have to restart it using fresh new random data. For example, the set of keys to be inserted into an AVL-tree is different each time. Even though this approach increases students' workload, we observed that the results, i.e., the marks received from the exercises, are now better than in TRAKLA. However, the number of submissions per exercise has also greatly increased.

The reasons behind our decision to support resubmissions are the following. First, resubmission can support the constructivist view of learning: students construct their answers, and analyze the feedback to identify possible errors they have made. In the subsequent submissions, they try to correct the errors, analyze the new feedback, and rethink, if necessary. Thus, the immediate feedback received after each submission works as formative feedback supporting the students' reflection on their learning and, at the same time, it also includes summative evaluation information (received marks or the number of correct steps). In addition, TRAKLA2 provides the model solution for each exercise instance that is formative feedback as well. However, after viewing it, the student cannot submit that solution any longer, and has to start the exercise again

with new input data. Second, we argue that such an iterative learning process produces learning results at least similar to those found among a large group of students participating from closed laboratories, where the tutoring assistant has no opportunity to examine all the solutions submitted by each student, nor give advice if needed. We base this argument on the research we carried out in 2001, where we had two randomly selected groups on the same course, one using TRAKLA and another solving similar exercises in closed labs of 20 students each. We found no statistical difference between the groups in the results of the final examination [Korhonen et al. 2002]. Similar studies with TRAKLA2 confirms the observations.

We are aware that some students have resubmission strategies that are less constructivist. In our programming courses, for example, we have made observations that the assessment tool is used for debugging and not getting feedback. In other words, the students adopt this approach "test the program, make some changes, test it again until you get full points." They do not stop to think what is wrong with their program; instead they work by trial-and-error. This method has also been called *bricolage* [Ben-Ari 2001] and this behavior is related to the number of resubmissions allowed. In algorithm simulation exercises, bricolage is a very inefficient strategy since the solution space is typically very large. Thus, by limiting the number of allowed resubmissions or by changing the data for each submission, as happens with TRAKLA2, we hope to eliminate this strategy.

Our belief is that students' behavior is influenced by the quality of feedback. If students do not see that they are making any progress in solving the exercise, some of them will resort to bricolage. On the other hand, too detailed feedback is not effective either. Mitrovic [Mitrovic et al. 2000] observed that if partial or full-model solutions were provided for the students, they performed more poorly than those students receiving only hints for finding a solution. Thus, the feedback should be *constructive*, giving hints and supporting the learning process. In the initial TRAKLA version, the feedback was not constructive, because it was given only once and not directly connected to the learning *process* at all. In the latter versions of the system, and especially in TRAKLA2, we have started to pay attention to this issue.

The structure of this paper is as follows. In the next section, we present the principles of the TRAKLA and TRAKLA2 systems. In Section 3, we present and analyze several different sets of statistics gathered during the years we have applied the two systems. Section 4 concludes the paper by summarizing our main observations.

2. TRAKLA AND TRAKLA2

In this section, we introduce the two concepts *algorithm simulation* and *algorithm simulation exercises*. In addition, we present the main features (an overview) of the TRAKLA and TRAKLA2 systems. We look at the systems mainly from the learners' point of view, because we are interested in the resubmission policies of these systems and, particularly, how the changes in these policies may or may not affect the learners' behavior and learning outcomes.

As a consequence, the teacher's point of view is mostly omitted. However, the TRAKLA2 system is described more thoroughly in the paper by Malmi et al. [2004] and readers who would like to know more about designing and implementing new exercises are encouraged to read this material.

2.1 Algorithm Animation and Simulation

Algorithm animation is a process where an algorithm manipulates data structures and the resulting changes in the structures are visualized for the user as conceptual diagrams, typically including nodes, references, and/or arrays. The algorithm code could also be animated to emphasize the correspondence among execution of the code and changes in the structures. A great number of systems with many different technical solutions have been implemented to create algorithm animations (e.g. [Baker et al. 1999; Brown 1988; Haajanen et al. 1997; Mukherjea and Stasko 1993; Naps et al. 2000; Pierson and Rodger 1998; Rößling and Freisleben 2002; Stasko 1997]). In most of the systems, the animation is intended to be observed by the user in order to assist in the understanding of the working of the actual algorithm code that is implemented.

Algorithm simulation is a process where the *user* directly manipulates data structures by using tools available through a user interface. Typically, this means dragging and dropping graphical entities, such as keys, nodes, and references, into new positions on the screen in order to simulate the operations of a real algorithm. Consequently, the system performs the corresponding changes on the underlying data structures, and finally visualizes the outcome. The sequence of operations performed makes up a stepwise animation sequence, which can be browsed forward and backward. No user written code is needed.

Algorithm simulation exercises are assignments where the student is requested to simulate a given algorithm with given input data structures. For a solution, the student should give a conceptual diagram that depicts the states of the data structures during and after the execution. Examples of typical exercises could be: *"Insert the keys A L G O R I T H M one by one into an initially empty AVL tree and show the state of the tree after each insertion,"* or *"Sort the following keys IMPLEMENTATION into ascending order using Quicksort when the pivot is taken from the right end of the sorted area."* However, many other types of assignments are possible, as pointed out by Korhonen and Malmi [2004]. The student could solve the exercise by drawing the requested diagrams on a piece of paper,[1] or he/she could use a computer to construct the answer. In this context, we see *visual algorithm simulation* to be a process where the system actually understands the underlying data structures. The assignment allows the student to use either primitive operations, such as changing keys and references, or to invoke abstract data type (ADT) operations, such as insertions into binary search tree, rotations etc. In the latter case, the exercise could be, for example, to build a worst case red–black tree by using a given set of keys.

[1]Such a process could be called *manual algorithm simulation*.

2.2 TRAKLA

TRAKLA [Korhonen and Malmi 2000] is a framework for algorithm animation and simulation, created at HUT. The system is used for automatic assessment of algorithm simulation exercises and has been employed on our data structures and algorithms courses from 1991 to 2003. Currently, the course enrollment is about 500 students a year, comprising 150 CS majors and 350 students from other engineering curricula. The primary purpose of the TRAKLA system is to automatically assess and give feedback on algorithm simulation exercises. Initially, the system was an email-based system [Hyvönen and Malmi 1993] whereby the assignments were sent to students by email and they returned their solutions to the TRAKLA server by email. For each data structure, a conceptual textual layout was defined. For example, trees and graphs were presented as adjacency lists. In 1997, we added a graphical Java front end that allowed learners to construct the solutions by using graphical user interface (GUI) operations. The front end also allowed the user to browse the solution backward and forward, and, finally, to convert it into a form acceptable to the server. The submitted solutions were then compared against model solutions generated by actually implemented algorithms using several different heuristics and the feedback was sent to the student by email.

A highly important feature in TRAKLA was that each student had personalized random data for each exercise. This allowed the exchange of ideas, but prevented the straightforward copying of answers from one student to another. What we wanted to do was to encourage fruitful cooperation through discussion of the solutions.

Up until 1996, the whole checking process was carried out as a batch process after the deadline for submitting the exercises had passed. Thus, giving feedback could take several days. In 1997, we built in a mechanism to provide feedback after a period of only a few minutes; it also allowed students to resubmit answers to the exercises after getting that feedback. Typically, three to four resubmissions were allowed per exercise.

2.3 TRAKLA2

TRAKLA2 is a web-based learning environment dedicated to *visual algorithm simulation exercises* [Korhonen et al. 2003] used in a data structures and algorithms course. The system was introduced in 2003 and totally replaced the TRAKLA system in 2004. In 2004, TRAKLA2 was also adopted by the University of Turku. In 2005, Tampere University of Technology as well as Helsinki Polytechnic Stadia joined us. Student response to the new system has been very positive [Laakso et al. 2004].

The system is able to create individually tailored exercises in which each student has different initial data that he or she manipulates in terms of visual algorithm simulation. An exercise can be individually tailored *each time* the exercise is initialized. This is different from its predecessor, TRAKLA, where the initial data had to remain the same for each resubmission. Naturally, the new system is also capable of preserving the initial values between submissions,

if required. In addition, the system is capable of assessing the student's performance when solving the exercises by recording the sequence of simulation operations and comparing that sequence with the sequence generated by an actual implemented algorithm. Based on this, TRAKLA2 can give immediate summative feedback to the student. By contrast, in TRAKLA, the evaluation was based on comparing the final states of the structures in the sequences generated by the user with those produced by the implemented algorithm. In some assignments, one intermediate state was also used for comparison. However, in TRAKLA2, all states in the simulation sequence are considered in the comparison. This comparison can be made both on the client side, where the TRAKLA2 Java applet portrays all the data structures needed to solve the exercise, as well as on the server side, where the points received from the exercises are stored in the database.

Figure 1 shows an example exercise. The student has 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 student can ask the system to grade the solution. The feedback received shows the number of correct steps out of the maximum number of steps. In addition, the student can view the model solution for the exercise as an algorithm animation sequence, i.e., as a sequence of discrete states of the data structures that can be browsed backward and forward in the same way as the student's own solution.

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

Finally, we note that because the system is used for summative assessment that affects the course grade, each student is identified by a unique student number that is entered when he or she logs into the system. Moreover, the final marks are determined by the best points received among all the submissions for a single exercise. In order to use the system for formative assessment only,[2] one can simply omit the submission phase. Table AI (see Appendix A) lists the current set of exercises in TRAKLA2, since the system replaced TRAKLA in spring 2003.

For a more detailed description and technical details of the TRAKLA2 system see Malmi et al. [2004]. In that paper, we conclude that the system works very well. Although creating new exercises is time consuming (approximately 2–3 workdays/exercise) and requires Java programming skills, we argue that the system saves time in the long run, because, once an exercise has been implemented, it can be utilized over and over again by any number of students and with almost no costs at all. This contrasts with traditional exercises, where the writing of an assignment is straightforward. It is the assessment, which depends on the number of students, that is time consuming. However, this and all the other benefits from the teacher's point of view that the TRAKLA2 system brings along are beyond the scope of this paper.

[2]This kind of exercises are freely available at http://www.cs.hut.fi/Research/TRAKLA2/

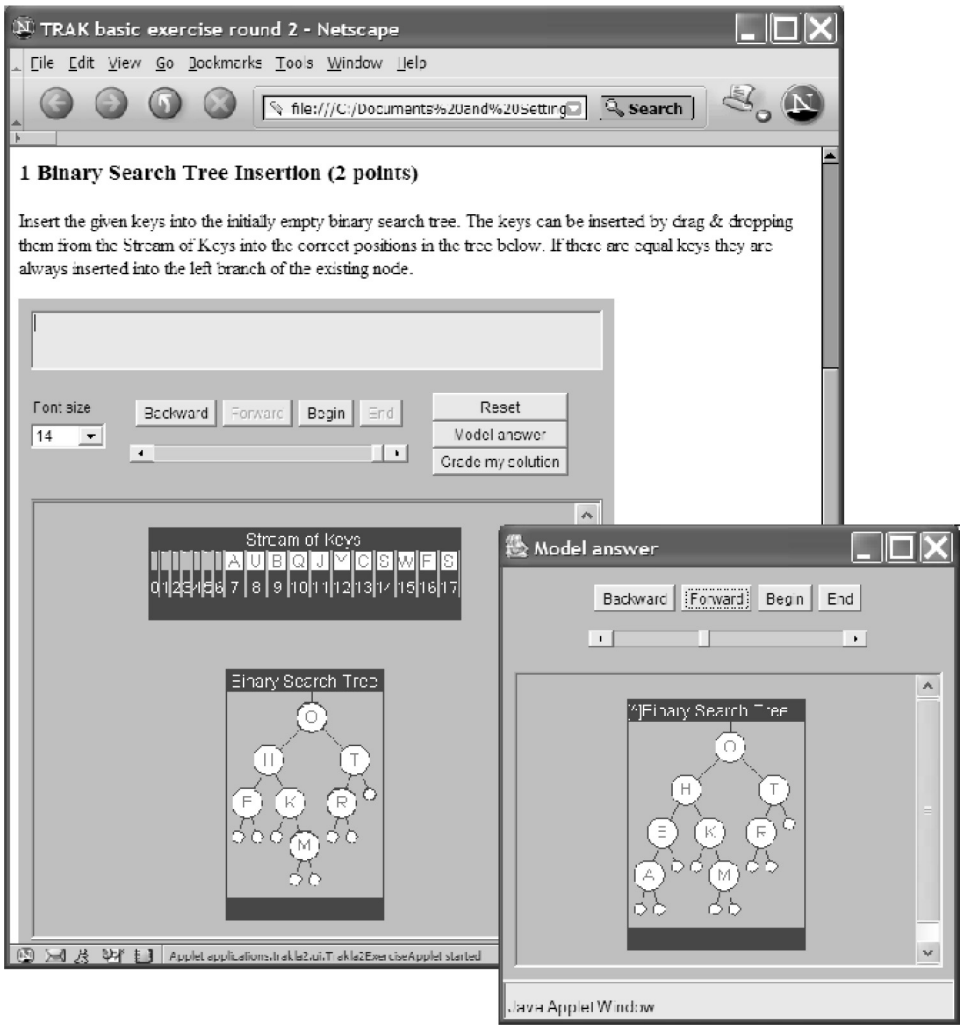

Fig. 1. TRAKLA2 applet. The exercise window comprises the data structures and push buttons. The model solution window is open in the front.

2.4 Gathering Statistical Data

TRAKLA recorded only the submitted solutions, including a time stamp when the solution was received. TRAKLA2, on the other hand, logs much more data on user operations during the exercise sessions [Silvasti et al. 2004]. The idea is to monitor the students as a group, and not as individuals. The statistical data is collected to obtain valuable feedback and insight into how the students are performing during the exercise sessions and, thereby, to improve the system further.

Data is sent each time the user performs one of the following *logged* GUI operations: Initialize (or reset) an exercise, Grade a solution, Open model solution, Close model solution, or Submit an exercise. Each log record includes

the following information given directly or indirectly by the student during the login phase: id (the student identification number), course code (several courses can be active at the same time), exercise round (exercises can be arranged into groups, each having typically five to eight exercises), and the exercise number. In addition, each record has a time stamp. Moreover, an additional log entry is inserted whenever there is idle time (i.e., when the user does not interact with the applet) exceeding 1 minute during that exercise session.

Each initialization of an exercise asks the server to generate a new seed for randomizing the input data. This seed is also stored in the database so that we are able to reconstruct the exercise afterward, if required. If the learner views the model solution for the exercise, the number of visited states in the algorithm animation sequence is stored, as well. In addition, the mark for summative assessment is stored each time the learner submits his or her answer.

Finally, the entire simulation process performed by the user is stored in each submission, and this serialized Java object can be used to discuss possible grading problems with the student. In addition, storing such data now will, in the future, enable us to analyze more deeply in which steps of the simulation sequences most errors occur.

3. RESULTS

In this section, we will first introduce the target course and how the course policy has changed during the longitudinal study. After that, we present the main observations and data gathered during the research.

3.1 Target Course

Our target course is the Data Structures and Algorithms Course at HUT. This is the second course of programming for both CS majors and minors. The course gives a broad overview of data structures and algorithms: basic data structures, sorting algorithms, priority queues, searching, hashing, and graph algorithms. There are 40 hours of lectures, a compulsory final examination at the end of the course, and the compulsory algorithm simulations exercises that we discuss in this paper. In addition, some other forms of exercises, such as analytical exercises in a classroom or design exercises solved in groups, have been used in some years.

The course does not include programming assignments. Instead, the subjects are taught at a conceptual level. The lecturer and teacher-in-charge have changed several times during the research period 1993–2004: the first time was in autumn 1996, the second time was in 2002, and, after that, more frequent staff changes have occurred. However, the lecturing style and subjects covered in the lectures have changed only slightly during the period. The course enrollment has varied between 350 and 700 students.

We have used the TRAKLA system since 1991 and most TRAKLA assignments have remained unchanged since 1993. In 1997, we introduced some more challenging exercises, such as insertions into AVL trees and B trees, as well as Dijkstra's algorithm. A year later, some simple assignments (e.g., manipulating stacks and queues) were left out, because the visual interface turned them too

simple compared with the older email-based system. In 2003, we introduced TRAKLA2. Initially, it was used in parallel with TRAKLA: in some exercises CS majors used TRAKLA and minors used TRAKLA2—and vice versa, as well. However, for majority of the exercises, only TRAKLA was used, since, at that time, TRAKLA2 did not contain enough exercises to cover the entire course.

By the following year, all students were using TRAKLA2, because we had implemented a sufficient number of new exercises to be able to totally omit TRAKLA exercises. Most TRAKLA2 exercises deal with the same topics in a similar way to the corresponding TRAKLA exercises. However, there are some totally new exercises, as well.

3.2 Resubmission Policy and Student Behavior

The resubmission policy changed considerably during the observation period. In years 1993–1996, no feedback was given on the submissions before the deadline. Students could submit their solution anew, but then only the last submission was used for the evaluation. Effectively, this meant that they had one submission. Then, in 1997, we allowed to resubmit solutions several times before the deadline, and provided the feedback on the exercises almost immediately.[3] For more difficult exercises, a greater number of resubmissions was allowed than for the easier ones. Because the number of exercises has varied during the years, the average number of maximum allowed resubmissions per exercise has also varied each year.

In TRAKLA2, we have not limited the number of resubmissions, because the assignments have new random data in each reset. Earlier, it was self-evident that if the data remained the same each time a student did an exercise, the opportunity to resubmit an unlimited number of solutions to an exercise would be unfruitful. The students could just use the feedback to see their mistakes, and, in turn, continuously amend the answer using aimless trial-and-error problem solving method. In TRAKLA2, however, each time a student resets an exercise, new input data is generated, rendering such a bricolage style useless. As a result, the student must reflect on the feedback, and find out not only why a mistake was made but also why the answer was wrong, and then apply this new insight to the new set of data.

Table I summarizes the number of exercises in years 1997–2004. The number is broken down into categories that show the number of allowed resubmissions per exercise. The number of exercises that fall into each category is shown in the columns. Note that in 1997, there are three exercises that have only one resubmission. These exercises were graded manually. At the other extreme, the new TRAKLA2 exercises have an unlimited number of allowed resubmissions (shown in the row as ∞). For years 1999 and 2003, there are two rows, because in these years the CS major (e.g., 2003T) and CS minor (2003Y) course versions had each a different number of exercises or allowed resubmissions. In all the other years, the two groups are treated as a single course, because they included exactly the same exercises and each exercise had the same number of allowed resubmissions.

[3]Typical delay for the feedback was a few minutes.

Table I. Total Number of Exercises[a]

Year	N	1	2	3	4	5	6	7	8	∞	Ave
1997	34	3	1	24	5	1					3.1
1998	30			21	8	1					3.3
1999T	28			16	5	1	3		3		4.1
1999Y	28			8	4	1	11		4		5.1
2000	28			19	8	1					3.4
2001	28			19	8	1					3.4
2002	28			19	8	1					3.4
2003T	32			16	6	2				8	N/A
2003Y	29			16	5	2				6	N/A
2004	26									26	N/A

[a] *N* is the total number of exercises in each year. Values 1–8 and ∞ (unlimited) indicate the number of resubmissions allowed, and the values in corresponding rows the number of exercises in each year. *Ave* is the average number of allowed resubmissions per exercise.

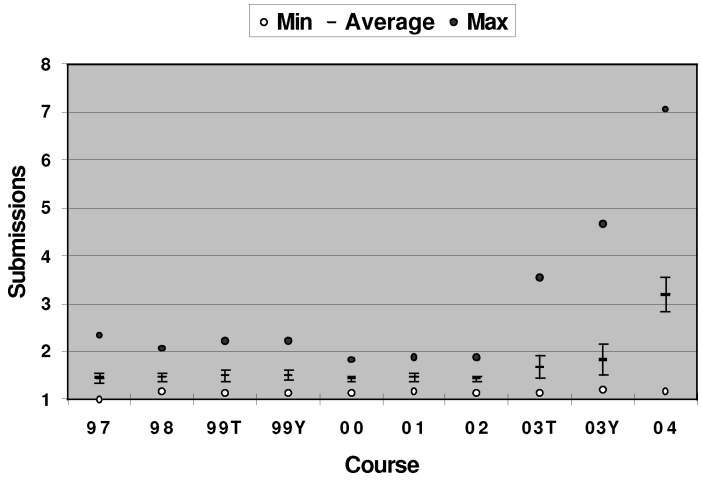

Fig. 2. The minimum, maximum, and average numbers of resubmissions as well as the confidence intervals for the average from 1997 to 2004.

We have examined the data by comparing the different populations (CS majors and CS minors) to each other. Generally, there were no significant differences, and we, therefore, do not consider them separately in the following discussion.

Figure 2 shows the effect on student behavior of the change in allowed resubmissions during the period 1997 to 2004. The figures were calculated as follows: for each separate assignment, the average number of resubmissions per student was calculated. From these average values, the minimum, maximum, and average were calculated and plotted on the diagram. As was anticipated, when the TRAKLA2 assignments were introduced with an unlimited number of resubmissions allowed, the maximum and average numbers rose considerably. In 2003, one-fourth of the exercises were performed using TRAKLA2. However, the average number of resubmissions approximately doubled. The same

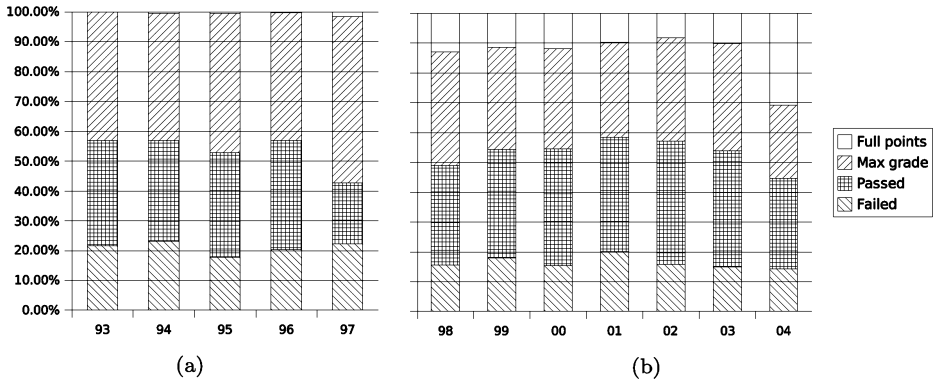

Fig. 3. Distribution of exercise grades. (a) Years 1993–1997; (b) years 1998–2004.

trend continued strongly when only TRAKLA2 was used in the next year. The maximum average value in 2004 is almost twice as high as in 2003. However, both maximum values originate from the red-black tree exercise in TRAKLA2, which was exactly the same in both years. We do not have a sound explanation for this phenomenon. Our assumption is that in 2003, when most assignments were performed with TRAKLA, and with only a small number of allowed resubmissions, students considered all exercises more carefully before submitting their solutions. This behavior carried over to TRAKLA2 exercises, as well. In 2004, on the other hand, the number of resubmissions was unlimited in *all* exercises and thus it was "easier" to submit the solutions more carelessly than in the previous year.

3.3 Results of the Assignments

The students should receive a certain minimum number of points for each of the five to six exercise rounds. Each round typically includes four to eight exercises related to a specific course topic. If a student fails to get the required minimum points from one or more rounds, there is an option to do them by completing extra assignments. The students can also raise their grades by solving these additional exercises. Usually, 50–100 students use this option. However, the procedure for doing the extra assignments has changed quite considerably over the years. Thus, we present the results without these additional points.

In Figure 3, we can see the distribution of points (excluding the extra exercises) awarded for the algorithm simulation exercises during the periods (a) 1993–1997 and (b) 1998–2004. The number of students on the course increased from 334 in 1993 to 710 in 2000, but has diminished to about 500 in 2004. There are four categories of students. First, those who achieved less than 50% of the maximum points possible and who consequently failed the exercise part. Second, the students who received at least 50%, but less than (a) 80% or (b) 90% of the maximum points and passed the exercises. Third, those who received at least (a) 80% or (b) 90% of the maximum points and received the maximum grade for the exercises, and, finally, those students who solved all the exercises correctly. The figure is split in two parts because in (a) 1993–1997, the grading

policy was different from that used in 1998–2004 (b). Thus, the categories are slightly changed as well. Until 1997, the third and fourth category received +1 in their final grade (that was otherwise solely based on the grade from the final examination), except in those cases where the student failed (grade 0) or got the best grade 5. After 1998, the grading policy was changed: 50–60% of the exercise points corresponded to grade 1, 60–70% to grade 2, etc. Thus, at least 90% of the maximum points were required for the best grade, i.e., 5. The simulation exercises accounted for 30%, the analytical exercises 30%, and the examination 40% of the final course grade.

There are significant changes in the distribution. Up to 1996, the overall distribution remained roughly constant, but, thereafter, the students achieved far higher grades, particularly in 1997 and 1998. Another big change appears in 2004, when about 30% of students achieved full points from the exercises.

By applying the F test, we compared the grade distributions of years 1997 and 1998 to see whether they were homogeneous. The result of this test as well as the χ^2 test for years 1993–1997, show that the differences in the distributions are statistically very significant ($p < 0.001$). On the other hand, the same test showed that the variation in grade distributions in years 1993–1996 is not significantly different ($p = 0.67$).

By applying χ^2 test, we compared the grade distributions of years 2003 and 2004. The results show that the difference in distributions is statistically very significant ($p < 0.001$). Likewise in years 1998–2003, the variation in the results is not significantly different ($p = 0.11$).

3.3.1 *The Role of New Tools and Features.* In the following, we compare these results with changes that have taken place in the course. Between 1993 and 1996, both the TRAKLA system and the course requirements remained unchanged. In 1997, we introduced two major enhancements to the system. First, we allowed resubmissions. Second, we introduced the graphical web-based front end, called WWW-TRAKLA, for solving the assignments. Previously, the only submission method was to send an email message in a predefined format to the TRAKLA server. All data structures had to be presented in ASCII text, which was somewhat clumsy, for example, in the case of trees. WWW-TRAKLA allowed students to construct the algorithm simulation sequence using simple drag-and-drop operations. They could, in addition, review their solution sequence by stepping it forward and back before submitting it. Finally, the front end took care of transforming the required states into the format acceptable to the TRAKLA server. In this way, simple formatting errors could be avoided in the solutions.

As might be expected, the results in 1997 were somewhat better than in 1996. The most conspicous detail is the definite appearance of the fourth category in which all the students obtained maximum points, although this has no direct influence on their grade. This group has existed in the results ever since, but the phenomena is still somewhat inexplicable.

The phenomena was even greater in 1998. There is, however, a number of reasons for that. Resubmissions clearly have a role to play, but, in addition to this, for example, the proportion of students who used WWW-TRAKLA

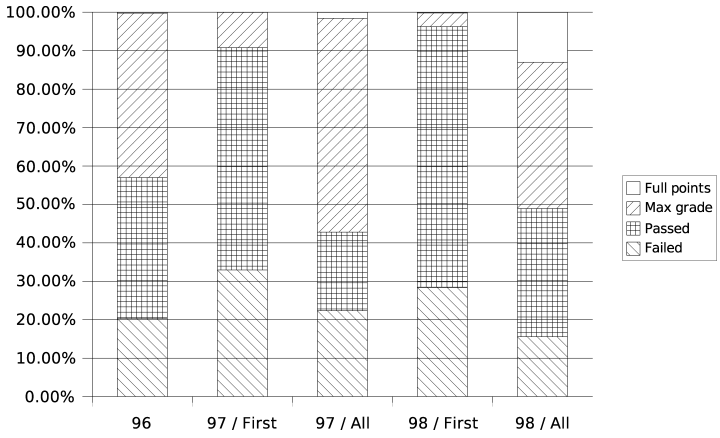

Fig. 4. Comparison of the results of the first submission and all resubmissions in 1997 and 1998. In 1996, only one submission was allowed.

increased from 23.7 (1997) to 42.6% (1998). Thus, it is not clear what kind of impact the GUI has to the overall phenomenon. Moreover, the change in the grading policy makes the comparison between 1997 and 1998 difficult. Thus, all we can state, is that the combination of resubmission, GUI, and encouraging grading policy have a great impact on student performance.

There are, however, some negative effects as well. On average, students submitted an answer to an assignment 1.45 times in 1997 and 1.47 times in 1998. These are relatively low numbers (even compared with 1996 when each student typically submitted the answer only once, as there was no feedback between submissions). However, a closer examination revealed that the first submissions in 1997 and 1998 were worse than those in 1996. Thus, it seems that after resubmission was introduced, students submitted their first answers more carelessly than before. However, the overall performance was still slightly better in 1997 and 1998 (see Figure 4). The other issue is that an increase in resubmissions raise the possibility that aimless trial-and-error methods were used to solve the exercises. We revisit these issues later in this paper.

3.3.2 *The Role of Course Contents and Requirements.* In 1998, several changes in the course syllabus took place. To begin with, we eliminated simple exercises such as manipulating stacks and queues, because they were too trivial in a graphical form. In their place, we brought in a number of more challenging exercises such as inserting items into an AVL tree, red-black tree, and B^+ tree as well as applying extendible hashing, and generating minimum spanning and shortest-path trees.

Another change was to introduce a different type of exercises to be done in closed labs, submitted on paper, and checked manually. These were considerably more challenging than the web-based exercises. They were analytical and constructive exercises and included algorithm analysis and small algorithm design problems. Finally, the grading policy of the simulation exercises was changed, as mentioned above.

The consequence of these changes was that the course requirements and the course workload were increased quite considerably in 1998. The students' results, however, were far better under these new circumstances. We assume that two factors played a major role. First, the change in the grading policy, especially combined with the option to use a GUI and resubmit the exercises, motivated the students to strive for better grades, because the exercises had a more fine-grained impact on the final grade. This has mostly influenced the average and weak students in 1998; thus, we see a smaller number of failed students. Second, the more advanced exercises raised overall motivation and increased the learning curve, as a whole. Previously, the course had been easier to pass than we thought and thus did not motivate the students to participate and learn through these activities. The more demanding exercises seem to make the students more competitive; thus, we see a greater number of students achieving the maximum points. However, it is uncertain how each of these changes interacted with the overall performance.

Moreover, because of the increase in the workload for both the teachers and the students in years 1999–2001, we replaced the analytical exercises with a more general half-semester project assignment to be solved in small groups. During this time, the course results seemed to slightly deteriorate. Thus, it appears that the difficulty of the exercises motivates the students to perform better. We observed the same effect in the research carried out in spring 2001 [Korhonen et al. 2002]. A group of students were randomly chosen to do exercises that were more challenging than the algorithm simulation exercises. Their results in the examination were significantly better than those who did only the simulation exercises. On the other hand, their drop-out was high. In 1998, such drop-out did not occur. We suspect that this was because the students had no comparison group that simultaneously carried out "easier tasks." Thus, their motivation remained the same throughout the course.

3.3.3 *Introducing TRAKLA2.* One interesting feature in the results since 1998 is that approximately 10% of the students achieved full points in the exercises, although they knew that this did not bring them any extra benefit. They were apparently motivated by the assignments. In addition, the learning environment provided tools for enhancing their performance as they could resubmit exercises until they were satisfied. Such behavior became much more evident after TRAKLA2 had become fully operational in 2004. Of students 30% achieved maximum points from the exercises, but there was no difference in the grading policy. Moreover, the assignments in the TRAKLA2 system corresponded very closely to the TRAKLA assignments. Some new assignments were introduced when TRAKLA2 became fully operational and a few TRAKLA assignments had no counterpart in the new system. However, most TRAKLA2 assignments were just new versions of the old exercises.

As we noted previously, the statistical difference between grade distributions in years 2003 and 2004 is very significant. There are several explanations for this. First, a new system has always some kind of novelty value that encourages students to use it more than they would do if the system were familiar to them. Second, the number of resubmissions increased dramatically compared to year

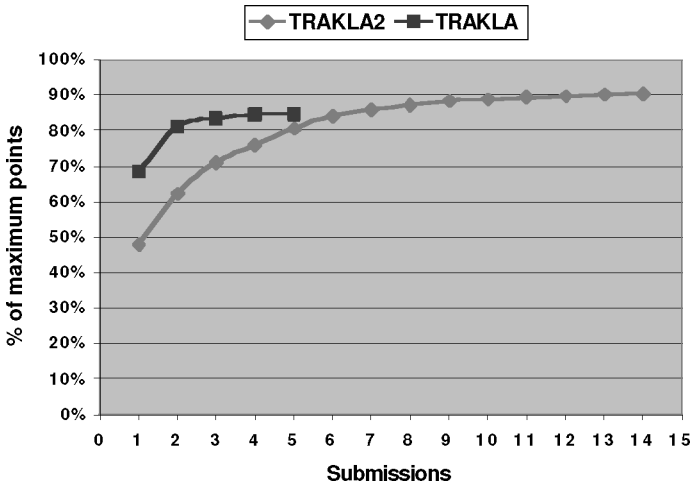

Fig. 5. Number of resubmissions versus points achieved in red-black tree assignment.

2003. We discuss this issue more in the next section. Third, there were fewer exercises in 2004 than in the previous year. However, the difference is not large (26 exercises in 2004 versus 29–32 exercises in 2003—in both years, the exercises were split into several rounds).

3.4 Resubmission Versus Results

Here we present some data and analysis of how the exercise results compare to the number of resubmissions used. In Figure 5, we can see how the results improved when compared with the number of resubmissions students had used in the red–black tree assignment. The results are from the CS minor courses in 2002 and 2003. In 2002, only TRAKLA was used, and, at most, five resubmissions were allowed. In 2003, the assignment was solved in the TRAKLA2 system. A few students used more than 15 resubmissions, but we left them out of the figure, since they did not affect the overall performance to any significant extent.

Each point on the graph corresponds to the average results of those students who had submitted the solution at most N times. We have left out the number of students per point, but, of course, the number decreases as we proceed to the right in the diagram.

When comparing the results of the first submissions, we observe that the TRAKLA results are better than TRAKLA2 results. The TRAKLA results initially increase rapidly with the number of submissions, but, after the third attempt, the improvement in the results is very small. In TRAKLA2, the number of resubmissions is far greater and the performance improves steadily until about the 10th resubmission. Thereafter, the improvement is very small. In addition, in TRAKLA, the average value rises only to about 84% of the maximum points, whereas in TRAKLA2 the average results exceed 90%.

We examined the data even further (not shown in the figure) and it seems that the average number of points a student receives after his/her last resubmission

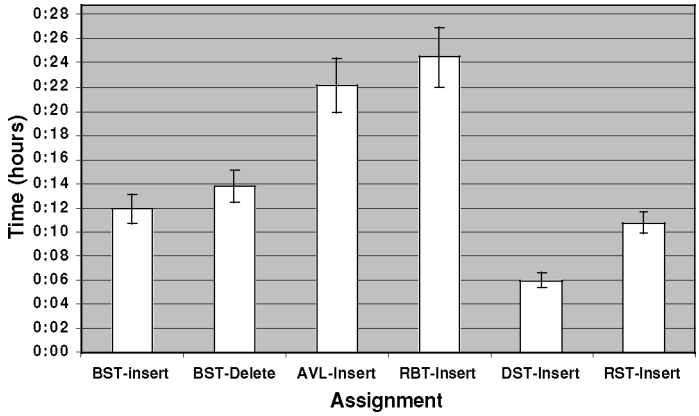

Fig. 6. The average time spent on solving the exercises: Insertion into/deletion from binary search tree, insertion into AVL, red-black, digital search trees, and radix search tries.

is almost constantly 90%. This, of course, implies that after achieving the grade that they consider acceptable, students stop trying. However, such a phase in their work does not seem to correlate with the number of submissions they use.

The results are very similar with those of other assignments. Students really take advantage of the option to resubmit many times and this resulted in very good average points. There is, however, an important aspect that we must remember. In TRAKLA, the students continue to use the same initial data structures when resubmitting their work, whereas in TRAKLA2 each resubmission means working with new data. Thus, we assume that they finally learned better with TRAKLA2, because they had to solve many assignments with different data. This is also one reason for the higher number of resubmissions.

The improved performance is a two-way process, i.e., it is related to the time used and *vice versa*. TRAKLA2 records the time stamp for each significant user interface operation and, based on this, it is possible to calculate the total time a student spends on each exercise (excluding the idle time). The net total times for some exercises are shown in Figure 6. As we anticipated, the most difficult exercises were those requiring insertion into AVL tree and red-black tree; the easier ones were insertions into binary search tree, digital search tree, and radix search trie. Finally, when we counted the number of idle time stamps per exercise, the same difficult exercises had most idle time stamps, as well.

3.5 Examination Results

Figure 7 depicts the correlation between the points achieved in the final examinations, which follow the course, and the percentage of the points received for exercises during the course. The graph is for the year 2004. Each bar indicates the average exercise points of students who received the corresponding examination points below the bar. The chart includes the sum of both courses (CS major/minor), although they had slightly different assignments. The examination took place immediately after the course and there is a threshold of 50% before the students can take the examination. This is because 50% is the

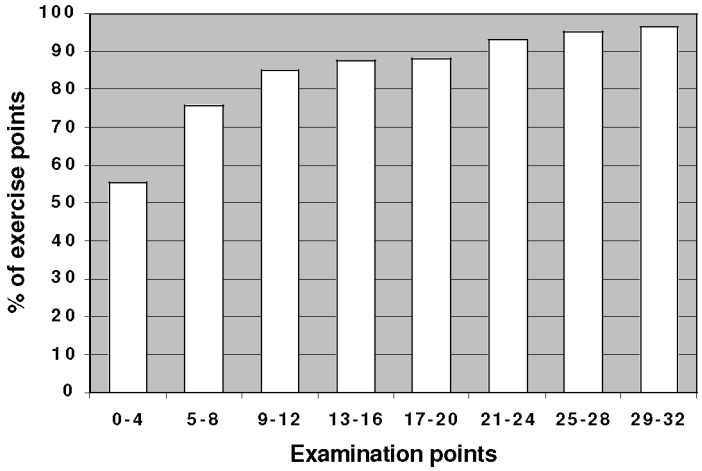

Fig. 7. The examination points versus the average percentage of TRAKLA2 points in 2004.

minimum number of points required to pass the exercises before taking the examination.

We already know that there is a correlation between the exercise points and examination points in TRAKLA [Korhonen et al. 2002]. In the 2004 TRAKLA2 results, the linear correlation is very significant ($\rho^2 = 0.14$, $p < 0.001$). The small correlation follows, because the average TRAKLA2 exercise points are so high.

In spite of the correlation, we can see that high points achieved for exercises do not imply equally high examination points, because of the nature of the examination: it comprised several different types of assignments, which tested knowledge of definitions, applying algorithms to solve new problems, and comparing algorithms. Upon examination, only one assignment out of five was based on algorithm simulation. Obviously, knowledge of how algorithms work does not directly imply the ability to write clear definitions or to argue why different algorithms can be applied to different problems, or imply the ability to write new algorithms. We need different exercises to equip learners with such skills.

4. CONCLUSIONS

We have reported a number of statistical results derived from data gathered from studying the behavior of students who used automatically assessed algorithm simulation exercises in our Data Structures and Algorithms Course in years 1993–2004.

During this period, we applied several different resubmission policies, i.e., how many times students are allowed to resubmit their solution to a single assignment and receive feedback. A central objective in this research has been to determine how the resubmission policy correlates both to the number of resubmissions actually used and to the points achieved.

Students achieve better points when they are allowed to resubmit their work. However, our observations demonstrate that improved points alone do not

intrinsically motivate all students. Indeed, most students are not motivated to do extra work if they do not get credit in terms of better marks for the course. However, there seems to be a fairly large group of students who are ready to do more work in order to achieve the maximum points for the exercises, although no extra credit is given for this achievement. This phenomenon is connected to the number of allowed resubmissions.

We assumed that people would give up more easily if an exercise had to be restarted with new input data (TRAKLA2) rather than simply being allowed to continue to solve the same exercise between attempts (TRAKLA). However, this proved not to be the case. Students achieved clearly better results, in fact better than we assumed. The average results were well over 90% of maximum value with such nontrivial assignments as inserting many keys into different types of balanced search trees.

A closer analysis revealed that students took the resubmission option seriously to help solve the assignments correctly. Since the system allows no guessing how to proceed in solving the assignment, we argue that students really had to think about what they were doing. Thus, solving the same exercise several times with different initial data aids building a viable mental model of the subject under study. One has to understand the algorithmic principles involved in order to be able to work through such an exercise successfully.

Another clear observation is that if there is no limit to the number of allowed resubmissions, some students try to solve the exercises fairly carelessly without deep consideration. Although the exercises are very hard to solve using the trial-and-error method (sometimes called bricolage [Ben-Ari 2001]) because of the vast size of the solution space, there are students who are still tempted to work in this manner.

Our initial assumption with TRAKLA2 was that such behavior becomes pointless when the student faces new initial data each time he/she continues to attempt to solve the same exercise. However, the results suggest that this assumption may be wrong. Under the TRAKLA2 system, the average number of resubmissions is very much higher compared with the TRAKLA system, which indicates that there are learners that might use the trial-and-error method. Apparently, student behavior has changed. This became obvious when we compared student behavior in the two different courses in 2003 and, at that time, TRAKLA2 was only partially being utilized in these courses. The average number of resubmissions in the investigated exercises was much higher in the same exercise in 2004, when only TRAKLA2 was used. Our assumption is that in 2003, TRAKLA, and its small limited number of allowed resubmissions, pushed students to rethink the exercises more seriously and this carried over into TRAKLA2 exercises. In 2004, the students had the freedom to submit as many times as they liked and they used this feature much more. Thus, TRAKLA2 possibly tempted them to use considerably more time on the exercises than previously. A closer analysis, not included in this paper, revealed that for about 10% of the students, the time spent on the exercises was, perhaps, unreasonable when measured against their success in the examination. Thus, we must consider seriously limiting the number of resubmissions, although the students get new initial data for each new submission.

Finally, when we compared the points awarded for exercises and the final examination scores, we found that there is a rather small, but very significant, correlation between them. However, in the examination, only one assignment out of five was an algorithm simulation exercise. Therefore, this correlation probably relates more to students' overall effort in learning the subject than the actual learning outcomes of the TRAKLA2 assignments. However, we recall here the research we carried out in 2001, when we had randomly chosen student groups who used either TRAKLA in self-study or solved similar problems in closed labs with the assistance of tutors. The results of the survey showed no significant difference between the groups in the results of the final examination [Korhonen et al. 2002]. Currently, the TRAKLA2 system provides better interaction tools and better feedback on the solutions, both of which support the learning process. We expect that this actually leads to better learning than occurs with the TRAKLA system. We are going to repeat the 2001 research with randomized groups to confirm this.

ACKNOWLEDGMENTS

We would like to acknowledge the work of Petteri Torvinen, who, in his Master's thesis, analyzed the data collected; a number of the results in this paper were based on his detailed work. [Torvinen 2003].

We also wish to thank the members of the Software Visualization Group at HUT who have been involved in developing TRAKLA2: Panu Silvasti, who implemented the TRAKLA2 server and data logging facilities; and Harri Salonen, Otto Seppälä, and Petri Tenhunen, who have implemented many of the exercises in the system.

REFERENCES

BAKER, R. S., BOILEN, M., GOODRICH, M. T., TAMASSIA, R., AND STIBEL, B. A. 1999. Testers and visualizers for teaching data structures. In *Proceedings of the 30th SIGCSE Technical Symposium on Computer Science Education*. New Orleans, LA, ACM Press, New York, 261–265.

BEN-ARI, M. 2001. Constructivism in computer science education. *Journal of Computers in Mathematics and Science Teaching 20*, 1, 45–73.

BENFORD, S., BURKE, E., FOXLEY, E., GUTTERIDGE, N., AND ZIN, A. M. 1993. Ceilidh: A course administration and marking system. In *Proceedings of the 1st International Conference of Computer Based Learning*. Vienna, Austria.

BRIDGEMAN, S., GOODRICH, M. T., KOBOUROV, S. G., AND TAMASSIA, R. 2000. PILOT: An interactive tool for learning and grading. In *Proceedings of the 31st SIGCSE Technical Symposium on Computer Science Education*. ACM Press, New York. 139–143.

BROWN, M. H. 1988. *Algorithm Animation*. MIT Press, Cambridge, MA.

CARTER, J., ENGLISH, J., ALA-MUTKA, K., DICK, M., FONE, W., FULLER, U., AND SHEARD, J. 2003. ITICSE working group report: How shall we assess this? *SIGCSE Bulletin 35*, 4, 107–123.

HAAJANEN, J., PESONIUS, M., SUTINEN, E., TARHIO, J., TERÄSVIRTA, T., AND VANNINEN, P. 1997. Animation of user algorithms on the Web. In *Proceedings of Symposium on Visual Languages*. Isle of Capri., Italy, IEEE, 360–367.

HIGGINS, C., SYMEONIDIS, P., AND TSINTSIFAS, A. 2002. The marking system for CourseMaster. In *Proceedings of the 7th Annual Conference on Innovation and Technology in Computer Science Education*. ACM Press, New York, 46–50.

HYVÖNEN, J. AND MALMI, L. 1993. TRAKLA—a system for teaching algorithms using email and a graphical editor. In *Proceedings of HYPERMEDIA in Vaasa*. 141–147.

JACKSON, D. AND USHER, M. 1997. Grading student programs using ASSYST. In *Proceedings of 28th ACM SIGCSE Symposium on Computer Science Education*. 335–339.

KORHONEN, A. AND MALMI, L. 2000. Algorithm simulation with automatic assessment. In *Proceedings of The 5th Annual SIGCSE/SIGCUE Conference on Innovation and Technology in Computer Science Education, ITiCSE'00*. Helsinki, Finland, ACM Press, New York. 160–163.

KORHONEN, A. AND MALMI, L. 2002. Matrix—Concept animation and algorithm simulation system. In *Proceedings of the Working Conference on Advanced Visual Interfaces*, Trento, Italy, ACM Press, New York. 109–114.

KORHONEN, A. AND MALMI, L. 2004. Taxonomy of visual algorithm simulation exercises. In *Proceedings of the Third Program Visualization Workshop*, A. Korhonen, ed. Warwick, UK, 118–125.

KORHONEN, A., MALMI, L., MYLLYSELKÄ, P., AND SCHEININ, P. 2002. 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*. Aarhus, Denmark, ACM Press, New York. 121–124.

KORHONEN, A., MALMI, L., AND SILVASTI, P. 2003. 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*. Joensuu, Finland. 48–56.

LAAKSO, M.-J., SALAKOSKI, T., KORHONEN, A., AND MALMI, L. 2004. Automatic assessment of exercises for algorithms and data structures—a case study with TRAKLA2. In *Proceedings of Kolin Kolistelut/Koli Calling—Fourth Finnish/Baltic Sea Conference on Computer Science Education*. Helsinki University of Technology, 28–36.

MALMI, L., KARAVIRTA, V., KORHONEN, A., NIKANDER, J., SEPPÄLÄ, O., AND SILVASTI, P. 2004. Visual algorithm simulation exercise system with automatic assessment: TRAKLA2. *Informatics in Education 3*, 2, 267–288.

MITROVIC, A., MARTIN, B., AND MAYO, M. 2000. Using evaluation to shape its design: Results and experiences with SQL-tutor. *User Modeling and User-Adapted Interaction 12*, 243–279.

MUKHERJEA, S. AND STASKO, J. T. 1993. Applying algorithm animation techniques for program tracing, debugging, and understanding. In *Proceedings of the 15th international conference on Software Engineering*. IEEE Computer Society Press. 456–465.

NAPS, T. L., EAGAN, J. R., AND NORTON, L. L. 2000. JHAVÉ: An environment to actively engage students in web-based algorithm visualizations. In *Proceedings of the SIGCSE Session*. Austin, Texas. ACM Press, New York. 109–113.

NORMAN, D. A. 1983. Some observations on mental models. In *Mental Models*, Gentner D. and A. Stevens, eds. Lawrence Erlbaum Associates, Mahwah, NJ, 7–14.

PIERSON, W. AND RODGER, S. 1998. Web-based animation of data structures using JAWAA. In *Proceedings of the 29th SIGCSE Technical Symposium on Computer Science Education*. Atlanta, GA, ACM Press, New York. 267–271.

RÖSSLING, G. AND FREISLEBEN, B. 2002. ANIMAL: A system for supporting multiple roles in algorithm animation. *Journal of Visual Languages and Computing 13*, 3, 341–354.

SAIKKONEN, R., MALMI, L., AND KORHONEN, A. 2001. 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*. Canterbury, UK, ACM Press, New York. 133–136.

SILVASTI, P., MALMI, L., AND TORVINEN, P. 2004. Collecting statistical data of the usage of a web-based educational software. In *Proceedings of the IASTED International Conference on Web-Based Education*. IASTED, Innsbruck, Austria, 107–110.

STASKO, J. T. 1997. Using student-built algorithm animations as learning aids. In *The Proceedings of the 28th SIGCSE Technical Symposium on Computer Science Education*. San Jose, CA, ACM Press, New York. 25–29.

TORVINEN, P. 2003. Tilastollinen analyysi algoritmisten harjoitustehtäväsovelmien käytöstä (statistical analysis of usage of algorithmic exercise applets). Master of Science Thesis, Helsinki University of Technology. (in finnish)

VIHTONEN, E. AND AGEENKO, E. 2002. Viope-computer supported environment for learning programming languages. In *The Proceedings of Int. Symposium on Technologies of Information and Communication in Education for Engineering and Industry (TICE2002)*. Lyon, France. 371–372.

Received March 2004; revised January 2005; accepted March 2005

APPENDIX A

Table AI. The Visual Algorithm Simulation Exercises in the TRAKLA2 System[a]

Name	Description
Tree traversal algorithms: (i) preorder, (ii) inorder, (iii) postorder, and (iv) level order	The learner has to show the correct visiting order.
Preorder tree traversal with stack	The learner has to simulate how the stack grows and shrinks during the execution of the preorder algorithm on a given binary tree.
Sorting using (i) quicksort, and (ii) radix-exchange sort	The learner sorts an array of keys by partitioning the data and simulating the recursive function calls.
BuildHeap algorithm	The learner has to simulate the linear time buildheap algorithm on 15 random keys.
Binary heap insertion and delete min	The learner has to insert 15 random keys into an initially empty binary heap and perform three deleteMin operations while preserving the heap order property.
Sequential search: (i) binary search, and (ii) interpolation search	The task is to show which indexes the algorithm visits in the given array of 30 keys by indicating the corresponding keys.
Binary search tree (BST) (i) search (ii) insert, and (iii) delete	The learner has to (i) perform search for two given keys (successful/unsuccessful search) by selecting the correct subtree in each level, (ii) insert random keys into an initially empty tree by dragging and dropping them into the correct positions, and (iii) delete three keys (leaf, one subtree, two nonempty subtrees) from a given BST by performing the corresponding pointer manipulations.
Faulty binary search tree	The learner has to show how to bring the following binary search tree in an inconsistent state: duplicates are allowed and inserted into the left branch of an equal key, but the deletion of a node having two nonempty subtrees relabels the node as its successor.
Insertion into (i) digital search tree, and (ii) radix search trie	The learner has to insert random keys into an initially empty search structure by dragging and dropping them into the correct positions.
AVL tree (i) insertion, (ii) single rotation, and (iii) double rotation	The learner has to (i) insert 13 random keys into an initially empty AVL tree. The tree (i–iii) has to be balanced by rotations. The rotation exercises (ii–iii) require pointer manipulation, while the insertion exercise (i) provides push buttons to perform the proper rotation in a selected node.
Red-black tree insertion	The learner has to insert 10 random keys into an initially empty red-black tree. The color of the nodes must be updated and the tree must be balanced by rotations.
Red-black tree coloring	The learner has to color a given binary search tree (that is sufficiently balanced) to satisfy the red-black tree properties.

(*continued*)

Table AI. The Visual Algorithm Simulation Exercises in the TRAKLA2 System[a] (Continued)

Name	Description
B-tree insertion	The learner has to insert random keys into an initially empty 2-3-4 tree. The model answer reveals whether to use top-down or bottom-up balancing, as well as where to put duplicates. Full nodes can be split by selecting a node and pressing a specific button.
Open addressing methods for hash tables: (i) linear probing, (ii) quadratic probing, and (iii) double hashing	The learner has to hash a set of keys (10–17) into the hash table of size 19.
Graph algorithms: (i) depth-first search (DFS), and (ii) breath-first search (BFS)	The learner has to visit the nodes of a given graph in DFS, and BFS order, respectively.
Graph algorithms: (i) Prim's Algorithm, and (ii) Dijkstra's Algorithm	The learner has to add edges to a spanning tree by following the given algorithm.

[a]The column *name* describes the subject and the *description* specifies the exercise. The roman numerals (i–iv) indicate the separate exercises.