Algorithm Simulation with Automatic Assessment

Ari Korhonen and Lauri Malmi
Department of Computer Science and Engineering
Helsinki University of Technology
Finland
{archie,lma}@cs.hut.fi

Abstract

Visualization is a useful aid for understanding the working of algorithms. Therefore many interactive algorithm animation tools have been developed. However, students may misinterpret the visualization and therefore the correctness of their interpretation should be confirmed by tests supplemented with feedback.

In this paper, a learning environment for data structures and algorithms is presented. The combination of algorithm animation and simulation with automatic assessment provides a way to give meaningful feedback to the students. Our experience shows that this combination is of great value for the students studying algorithms.

1 Introduction

Data structures and algorithms belong to the core issues in computer science education. Students should learn how various data structures are constructed and how different algorithms manipulate them. In addition, the students should gain a proper understanding of the analytical properties of algorithms and their applicability to different practical cases.

Since data structures and algorithms are often complex topics, different approaches have been attempted to present their dynamic nature in realistic and easily understandable forms. *Algorithm animation* is one of such approaches. Many tools for creating algorithm animations have been developed during the past decade, and new useful features have been introduced in recent years. For example, the visual debugging system presented in [4] included a possibility to view the execution of the program source code during the animation. We call this feature *program animation*. In Jeliot [1] this idea was enhanced by allowing the user to specify the code to be animated and visualized. The "Algorithms in Action" tool presented in [5] has a novel approach where students have the option of stepwise refinement of algorithm code and the corresponding animation.

There are, however, still challenges to be met from the teacher's point of view. It is quite possible that some students consider algorithm and program animation as entertainment without seriously trying to learn from them. Even if this is not true, the students may believe that they have understood how the algorithm works, when in practice they have misinterpreted something. We should give them an opportunity to test their understanding.

An obvious method for such testing is *manual algorithm simulation*. With this we mean simple exercises where students have to demonstrate how a given algorithm manipulates a given data structure step by step. When appropriate feedback is given, such exercises apparently help the students to capture the logic of exercised algorithms. The problem, however, is that if the exercises are solved on paper and checked manually, it is hard to provide immediate feedback.

At Helsinki University of Technology, we have extensively used manual algorithm simulation to support learning. In 1991, we implemented the TRAKLA system [2] which automatically generates algorithm simulation exercises and *automatically assesses* students' answers, returned by email. For the whole of the 1990's the system has been successfully used in a course with a yearly enrolment of 500 students. Many new features have been added. Currently the students solve the problems using a graphical editor, TRED, on a web page [3]. These tools interestingly combine manual algorithm simulation and algorithm animation with automatic assessment, since the students can view their solution by stepping the states of the data structure

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
ITiCSE 2000 7/00 Helsinki, Finland
© 2000 ACM 1-58113-207-7/00/0007...$5.00

forward or backward before they submit the solution for evaluation. We have monitored the learning results and we have observed that they are very good.

We present the key ideas of TRAKLA and TRED in Section 2. In the next section we discuss the principles of implementation and automatic assessment. In section 4 we present our experiences when the system has been used for a mass course, and the final section includes the conclusion.

2 Overview of the TRAKLA system

TRAKLA is a system for teaching basic computer science data structures and algorithms including sorting, searching and graph algorithms. The heart of the system is the TRAKLA server which provides the following facilities:

1. distribution of individually tailored algorithm simulation exercises (by email or by Java applet),
2. accurate and automatic assessment of the exercises,
3. meaningful and immediate feedback about the answers to improve learning,
4. distribution of model solutions for the exercises, and
5. automatic administration of the course.

The other part of the system is TRED which provides a graphical interface for manipulating a selection of data structures. TRED combines the idea of algorithm animation and simulation, i.e., it provides tools for manipulating visual data structures step by step and for browsing the constructed sequence of steps forward and backward.

An older, email-based interface for using TRAKLA is also supported, since some students prefer using it when lacking graphical browser access to the Internet. They do manual simulation by paper and pencil and submit the solution by email. Then, additional knowledge of the the submit format of the answer is required, and there is neither animation nor continuous interaction between the user and the system.

In both ways the user can submit the answer and get almost immediate feedback generated by the automatic assessment tool. Based on this feedback, the students can revise the answers a few times, typically 3-5 times per exercise, if necessary[1].

In Figure 1 the logic of the system is presented on a general level. Registration is needed in order to interact with the system. For all messages sent to the system, a receipt is sent back to the user as a notification that the message was received.

A typical exercise gives an initial data structure and/or an input data set. The student has to apply the given algorithm using this data and present the resulting data structures(s). An example: "Insert the keys X H R U F G I J C M A K in this order into an initially empty AVL tree. Give the tree both after 6 insertions and after all keys have been inserted." The initial data is individually tailored for each student, i.e., all students have the same assignment with different input parameters.

3 Manual simulation and automatic assessment

In order to be able to automatically assess the simulated behaviour of a data structure, a well defined simulation model is needed.

The simulation model consists of an exercise algorithm (EA), input and output data structures needed by the algorithm and an execution environment which produces the initial simulation case, i.e., it initializes the input data structures with individually tailored data.

3.1 Exercise algorithms and data structures

An exercise algorithm is the logic and topic for an exercise. It describes the actions that should take place in order to complete the exercise. If the focus of an exercise is not on the algorithm but on a data structure, we can see EA as an operation of an abstract data type (ADT) and the data structure as an implementation of this ADT. This is because the nature of the simulation is not static (like a data structure) but dynamic (an algorithm). In the AVL tree exercise described above, the ADT is the balanced search tree and EA is the insert operation of the ADT. Of course, there exists also an actual implementation for EA in order to complete the automatic assessment.

3.2 Setting up an exercise

The execution environment includes a random input generator and an input data validator. The student identification and other parameters are used as a seed for the random generator which creates individually tailored input data structures for a specific EA.

The input data is validated by checking that it satisfies some exercise dependent constraints. In the AVL tree example, these constraints could be, "the initial structure contains at most 12 items to be inserted and during the insertions at least two single rotations and at least one double rotation must be performed". If the initialization fails under these constraints, a new attempt is made with a slightly modified seed. The initialization procedure is, however, a deterministic algorithm. Thus, for every attempt to produce an individually tailored

[1] The average count of submissions among more than 1000 students has been 1.4.

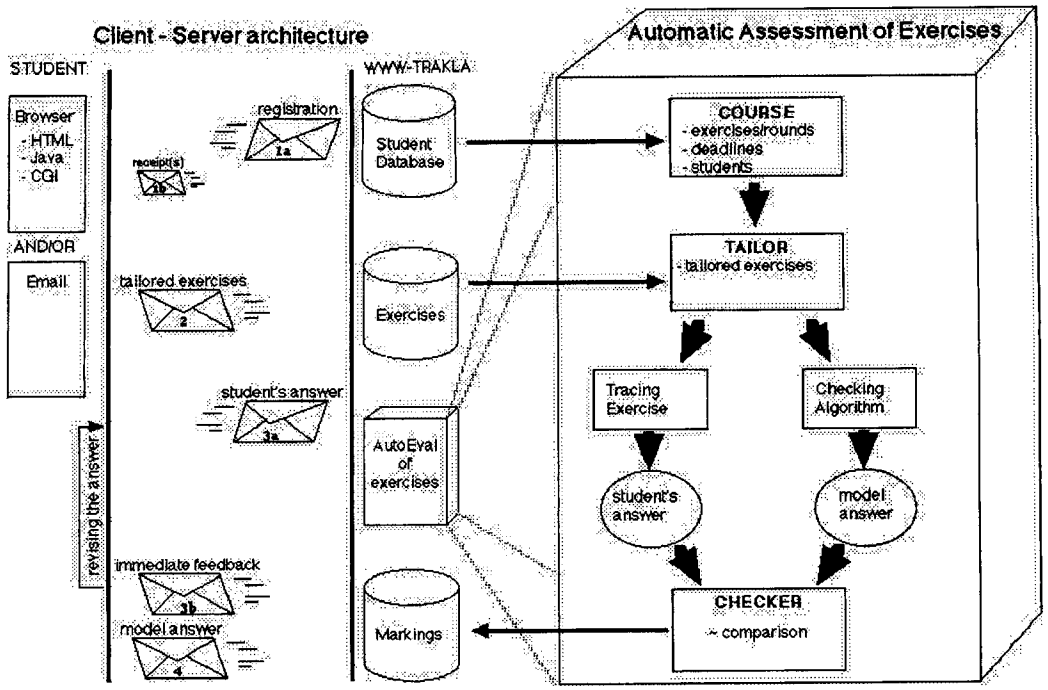

Figure 1: Flow of submissions and the overview of the automatic assessment procedure.

input data for a given exercise, the initialization terminates with the very same output.

3.3 Students' view of the exercise

From the students' point of view, the simulation model is a combination of a predefined assignment (textual description) and the input data. The assignment gives guidelines what to do, how input data should be understood, and in which format to submit the answer (data structure). As the exercise, the students manually simulate the given exercise algorithm and produce an output structure or a set of output structures depending on the assignment.

The input data has to be represented in some predefined format. In our example, the set of input items could be represented simply as a string or as an array of alphabets. When appropriate, more complicated forms like adjacency lists are used. Output structure formats are usually very similar to initial input structure formats. However, when TRED is used, the user does not need to consider formats.

3.4 Automatic assessment

After the student has submitted the answer to the system, TRAKLA initializes the exercise again precisely as in the creation phase. Now, in addition to the initial input structures the resulting output structure(s), which are required to follow the execution of EA, are created. Depending on the assignment, the output structure is one data structure or a sequence of states of the data structure. This state or these states serve as the model solution of the exercise.

The student's answer and the model solution are presented as strings. The automatic assessment is performed by searching exercise dependent regular expressions from the student's answer string. This allows some mismatches like typos to be ignored. Some exercises generate model solutions for a few different versions of the algorithm, e.g., the pivot item of quicksort can be taken from the left or the right end of the array. The percentage of correct matches in the string defines the grade of the exercise.

After the marking has been completed, the student is informed of the results. The level of feedback depends on the exercise. Assessment log information, including the generated model solution and the grading trace is stored each time so that the student can discuss the evaluation with an assistant, if necessary.

4 Pedagogical issues

An important feature of the system is that we can generate a unique initial input structure for each student. Thus, although the predefined assignment is the same for all students, each one has to solve a personally tailored exercise. This has obvious advantages for learning. They cannot copy their answer from anybody. Moreover, we can now encourage natural co-operation be-

tween the students. They can discuss the problem freely as long as they do not solve each other's exercise totally.

The other important feature is the immediate feedback of exercises. This allows the students to learn from their mistakes. We stress here that they have to think about their solution anew for each new submission, since the solution space of the exercises is simply too large for using any mechanical trial-and-error method.

As mentioned before, the system includes the actual exercise algorithms for initializing the input structures and for solving the exercises. These very same exercise algorithms are also used for producing model answers to the exercises. The model answers can be delivered to the students after some deadline. There is also an option to create example exercises which can be examined and solved by the students before solving the actual exercises.

All these properties are features we possibly can't achieve without a computer aided learning environment. For example, there is a huge difference between feedback received within a few minutes instead of within a few days or weeks, which is often the case in mass courses.

During the last two years, when TRED has been extensively used by our students, we have had excellent results. On a course of over 500 students, each student had to do about 25 simulation exercises covering most basic algorithms and data structures in sorting, searching and graph applications. Over half of the students got at least 90 percent of the maximum points and almost 2/3 of them got at least 80 percent. This guaranteed that the students know the basic topics well, and that they were well prepared for the more advanced design exercises. The feedback from the students has been very positive, too. An important reason for this is that TRED can be used on any WWW browser supporting Java applets, enabling the students to solve their exercises wherever and whenever they wish.

5 Conclusion

The TRAKLA system has been used since 1991. During these years many new ideas have been encountered for developing the system further, and a great number of them have been incorporated in the system, e.g., the graphical user interface TRED. However, many new ideas remain and a number of problems are still to be solved.

Development of new exercises is currently rather a slow process. We would like to get rid of the fact that developing a new exercise relies on the technical skills of a content expert. The goal is to develop a system which requires only programming skills to produce an actual working exercise algorithm. All the other functionality should be derived from this. This means that the assessment process and the feedback generation should be separated from the actual algorithm producing the model solution. Then, we could set up programming exercises for students, such that introduce new exercises on algorithms and data structures to the system.

The generalization of automatic assessment is currently being implemented. It enables us to observe, assess and comment on every step of the algorithm during the simulation. For the teacher this provides a tool for giving detailed automatic comments on the solutions submitted by the students. The same tool can be used for making on-line examples where the students can browse the states of a data structure during the execution of an algorithm stepwise forwards and backwards. This is a useful aid in understanding complicated structural changes in a data structure, e.g., AVL tree rotations. An even better solution would be to include program animation with each step of the algorithm.

Currently, the client-server architecture requires active on-line access to the server to get any feedback. Our aim is to implement a system in which feedback could be provided without communication with the server. Then, the system could also be used as an *electronic exercise book*.

As a whole, our work has shown that the development of an automatic assessment system is feasible and useful tools can be provided. Such a system can be of great help to anyone who has to assess algorithmical exercises.

References

[1] Haajanen J. et al.: *Animation of user algorithms on the Web*, Proceedings of the IEEE Symposium on Visual Languages, VL'97, pp 360–367, 1997.

[2] Hyvönen J., Malmi L.: *TRAKLA - A System for Teaching Algorithms Using Email and a Graphical Editor.* Proceedings of HYPERMEDIA in Vaasa'93, pp. 141–147, 1993.

[3] Korhonen A.: *World Wide Web in Computer-Aided Learning of Algorithms and Data Structures.* MSc Thesis, Department of Computer Science, Helsinki University of Technology, Finland (in finnish), 1997.

[4] Mukherjea, S. and Stasko, J.T.: *Toward Visual Debugging: Integrating Algorithm Animation Capabilities within a Source Level Debugger*, ACM Transactions on Computer-Human Interaction, Vol. 1, No. 3, pp 215–244, 1994.

[5] Stern, L. et al.: *A Strategy for Managing Content Complexity in Algorithm Animation*, Proceedings of The 4th Annual SIGCSE/SIGCUE Conference on Innovation and Technology in Computer Science Education, ITiCSE'99 pp 127-130, 1999.