Automatic Tutoring Question Generation During Algorithm Simulation

Ville Karavirta*
Helsinki University of Technology
P.O. Box 5400, 02015 TKK
Finland
vkaravir@cs.hut.fi

Ari Korhonen
Helsinki University of Technology
P.O. Box 5400, 02015 TKK
Finland
archie@cs.hut.fi

ABSTRACT
High user interaction is the key in the educational effectiveness of algorithm visualization (AV). This paper introduces integration of an AV system with an interaction component in order to add support for the responding level of the user engagement taxonomy. We describe the resulting AV system, which is capable of providing algorithm simulation exercises with pop-up questions that guide the student in solving the exercise. This research aims at providing a system usable in teaching as well as in validating the engagement taxonomy.

Keywords
algorithm simulation exercises, automatic assessment, algorithm visualization, engagement taxonomy

1. INTRODUCTION
Algorithm visualization (for the rest of the paper, abbreviated as AV) gives a visual representation of an algorithm on a higher level of abstraction than the actual code. The aim of AV is to help humans to grasp the workings of the algorithm through visualization. Although AV has been used in education for more than two decades with varying results, the current belief is that AV must provide interaction between the animation and the student in order to be educationally effective [2].

Further research on the topic of *AV effectiveness* has come up with a taxonomy to measure the user engagement with the AV (system). This *engagement taxonomy* [13] defines six levels of engagement:

No viewing – no use of AV technology.

Viewing – passive viewing of AV material where the student only controls the visualization's execution, for example, by changing the speed and direction of the animation.

Responding – the student is engaged by asking questions about the AV material. These questions can be, for example, about the efficiency or prediction of the next step of the algorithm.

Changing – requires the student to modify the AV material. This can be done, for example, by changing the input data.

Constructing – the student is required to constructs his/her own AV material related to an algorithm.

Presenting – the student presents a visualization for an audience.

The hypotheses of the taxonomy are that there is no significant difference between levels no viewing and viewing, and that the higher the level of engagement the better the learning outcomes of students.

The above hypotheses have been studied over the years. Before the taxonomy was even defined, Jarc et al. conducted an experiment comparing the levels viewing and responding [4]. The results of the survey suggested, that the students using more interactive material scored better on difficult topics, but poorly in overall. None of the differences were statistically significant, though. Lauer reports on an evaluation comparing three different levels of engagement: viewing, changing, and constructing [8]. Neither this study found statistically significant differences, although the group using changing performed slightly worse on average. Grissom et al. experimented to compare levels no viewing, viewing, and responding [1]. The results show that learning improves as the level of student engagement increases. The difference between no viewing and responding was statistically significant. None of the studies mentioned above have compared, for example, level responding with changing, and constructing.

We have developed a growing set of algorithm simulation exercises that support engagement levels viewing, changing, and constructing. However, as claimed by Rößling and Häussage, the support for the responding level in the exercises in MatrixPro [6] and TRAKLA2 [9] is *"too limited to use for experiments that require this level of engagement"* [15]. Thus, to be able to use TRAKLA2 / MatrixPro in experimental studies requiring engagement in all the levels (except possibly presenting), we wanted to add proper support for responding level as well.

In this paper, we will describe an approach to extend our systems to support the responding level by adopting the AVInteraction software module [15] developed in the Technische Universität Darmstadt, Germany. The same software module is utilized by other AV systems as well. This will allow us to conduct and repeat experiments comparing the different levels of engagement and hopefully validating the hypotheses presented in the engagement taxonomy. We will also point out new research questions we would like to study.

The rest of this paper is organized as follows. Section 2 introduces related work done on this topic. Next, Section 3 describes the concept of algorithm simulation exercises.

*Corresponding author.

Section 4 in turn concentrates on our implementation of responding level interaction in algorithm simulation exercises. Finally, Sections 5 and 6 discuss the usefulness of our approach and concludes this papers main ideas, respectively.

2. RELATED WORK

Our aim is to support the engagement level responding in our existing system. However, as there already are systems supporting this level (see, e.g., Animal [14] or JHAVÉ [12]) we do not want to reinvent the wheel by implementing yet another interaction piece of software, but rather reuse existing work developed by others.

A package intended for the kind of interaction we are looking for is AVInteraction [15]. AVInteraction is a tool-independent interaction component designed to be easily incorporated into Java based AV systems. The component offers a parser and a graphical interface for the interaction events. In addition, it includes an option to evaluate the answered questions. The current version of the component can handle different types of questions:

- *true /false* questions, where the user chooses from two alternatives
- *free-text* questions, where the user can type in free text. The answer is evaluated by string matching that allows several possible correct answers determined in the question specification.
- *multiple choice* questions, where the user selects a subset of the given choices.

The developers of the AVInteraction component have used it in their algorithm visualization system Animal [14]. In Animal, the user is asked questions about the animation while viewing it. Usually, these questions require the student to predict what will happen next in the animation.

The AVInteraction package has been successfully integrated to another system as well. Myller [11] introduced a version of Jeliot 3, a program visualization system for novice programmers [10], that supports automatic generation of prediction questions during program visualization. The approach taken also used the AVInteraction to bring up the questions.

3. TRAKLA2 EXERCISES

TRAKLA2 [9] is an automatic exercise system for learning data structures and algorithms. The system provides an environment to solve *algorithm simulation exercises* that can be automatically graded. In this environment, the students manipulate conceptual views of data structures simulating actions a real algorithm would perform. All manipulations are carried out in terms of graphical user interface operations. The system records the sequence of operations that is submitted to the server. The submitted sequence is compared with a sequence generated by a real implemented algorithm, and feedback is provided for the student. The feedback is based on the number of correct steps in the sequences of operations.

For example, a typical TRAKLA2 exercise is "insertion into a binary heap". In this exercise, the student simulates how a number of keys are inserted into an initially empty binary heap. An array of keys is provided and the task is to move them one-by-one into the tree representation of the heap. In addition, the heap order property must be restored after each step (heapify). The feedback is not provided until all the keys are inserted. Thus, there is a number of ways to solve the exercise incorrectly. This is especially useful in case the user has a misconception of how the algorithm works [16]. For example, in this exercise, the student might have difficulties to understand how recursion works. Thus, he/she might never do the recursive step in the heapify procedure (common error in final examination). This is why the feedback provided should be designed in such a way that it promotes reflective thinking, i.e., the feedback should reveal the misconceptions and change the way of thinking: a process we call learning.

The initial data for each exercise is random. For example, the array of keys in the previous example is randomized for each exercise instance. This allows the student to request grading for the solution an unlimited number of times. However, after each grading action, the student cannot continue solving the exercise with the same data. Instead, the exercise must be re-initialized with new random data.

In addition, the student can request the model solution for the exercise at any time. The solution is presented as an algorithm animation that the student can freely browse backwards and forwards. However, as with grading, the student has to reset the exercise and start with fresh random initial data before he/she can resubmit the solution again. Between two consecutive submissions, the student is encouraged to compare the submission with the model solution and find out the error made (i.e., reflect the possible misconception). Yet, the number of resubmissions can be unlimited.

The user interface of TRAKLA2 is a Java applet that is tailored to each exercise separately. The applet includes visualizations of data structures needed in the exercise, push buttons for requesting *reset*, *submit* and *model solution* for the exercise, as well as buttons for browsing one's own solution backwards and forwards. Simulation is carried out by drag-and-dropping data items (i.e., keys and nodes in data structures) or references (i.e., pointers) from one position to another. Some exercises also include additional push buttons to perform exercise specific operations such as rotations in trees. Thus, more complex exercises can be divided into several smaller tasks. For example, there are separate exercises for single and double rotations in AVL tree (both based on pointer manipulation). The final task, however, is to learn how to insert a number of keys into an AVL tree. This exercise has separate buttons to perform the corresponding rotations (instead of still doing them in terms of pointer manipulation).

Current selection of exercises deals with binary trees, heaps, sorting algorithms, various dictionaries, hashing methods, and graph algorithms. You can freely access the learning environment on the web, and create a test account to try out the exercises.[1]

3.1 Remark

Implementing new exercises is the hard part of the system.[2] However, it should be noted that any algorithm

[1]http://svg.cs.hut.fi/TRAKLA2/

[2]Currently, designing, implementing, and testing a new exercise takes roughly a week.

simulation exercise can be reused unlimited number of times with unlimited number of students.[3] Thus, after implementing an exercise, the human workload is independent of the number of students and submissions they do. This is due to the fact that the exercise is automatically assessed for each initialization, and each instance of the exercise is different from previous one. Thus, the students cannot copy the answers from each other. Actually, they need to think the solution anew for each submission. In addition, due to the model solution capability, the students can view any number of example solutions before, during, and after solving their own exercise.

4. IMPLEMENTATION

This section describes our implementation of adding the responding level to an existing AV system. Figure 1 shows an example of an exercise with the tutoring questions included. In the exercise, student is expected to color the nodes of a binary search tree to be a valid red black tree. The tutor gives guidance by asking questions whenever any of the red black tree properties are violated:

1. A node is either red or black.
2. The root is black.
3. Both children of a red node are black.
4. All paths from the root to any empty subtree contain the same number of black nodes.

The questions asked from students, however, are not restricted to these (explicit) properties. We can derive a number of (implicit) properties from these that we call rules. In the figure, for example, the question "How many children can a red node have?" is shown. The question was selected due to the fact that one of the nodes in the tree violates the implicit rule that says "a red node cannot have only one child." (but zero or two children). This rule follows from the properties.

PROOF. (Proof by Contradiction.) Assume to the contrary that a red node can have exactly one child. This child must be black by the property 3. Well then, there exist two paths that differ in their black lengths: the one that ends at the other side of the red node, and the longer path that ends after the black child. This violates the property 4, contradicting our assumption that a red node can have only one child. □

We have identified three different forms of questions that can be asked from the student:

tutoring questions – These are questions such as in Figure 1, which are asked while something is wrong within the student's solution. The most simple example of such question is to ask about the correct order of the alphabets if the student has made a mistake with them. The overall aim of this form of questions is to guide the student into the right direction in solving the simulation exercise.

detailed questions – These questions are not directly related to the simulation exercise, but rather ask some extra properties of the data structures manipulated in the exercise. Examples of such questions are (in the binary search tree exercises) questions about the height of the trees, and how many nodes fit in a tree if the height is limited. The aim of this form of questions is to increase the student's knowledge about the topics related to the exercise.

thinking questions – These are kind of mixed tutoring and detailed questions in which the same question might help the student to continue with the simulation, but at the same time, possibly increase the knowledge about the topic. In the red black tree exercise, for example, some students might try to prove the new implicit rule emerging from the question based on the given properties. The proofs are typically simple, as demonstrated above, but difficult for novices. Thinking questions can be promoted by asking them more explicitly, for example, in class room sessions. However, visual algorithm simulation exercises and related tutoring questions can motivate the student to actually think of this proof due to the fact that the "proof can be seen" (in Figure 1, spot the erroneous red node that has only one child, and read the proof again).

There are major differences in the nature of the questions and approaches between our tutor and the other systems using the same AVInteraction component. In ANIMAL the questions are loaded once the animation is loaded and are in a way static questions presented at predefined time. The engagement is mainly a combination of levels viewing and responding. Our approach combines engagement levels responding and changing/constructing[4]. Furthermore, both ANIMAL and Jeliot 3 require the user to respond to the questions[5], where in our implementation, the responding to the questions is voluntary.

Another difference is that ANIMAL and Jeliot 3 are usually used to ask questions about the next step of the animation, so to *predict* the result. Our approach is to use the questions to *guide* or *tutor* a student to construct a correct simulation of the algorithm in question. Actually, the response is not that important at all. That is to say, we know that the students know the correct order of alphabets in the simple tutoring questions described above. Thus, in our approach, the most crucial thing is the thinking process triggered by a question.

4.1 Technical Aspects

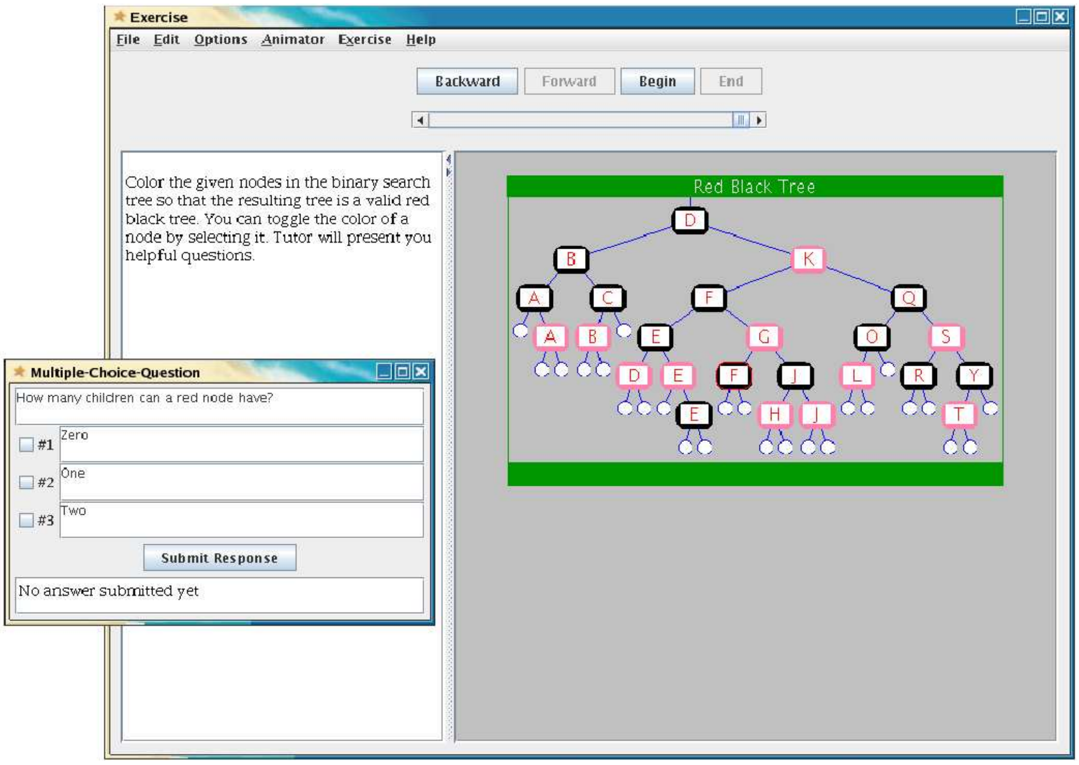
We used the AVInteraction component to add support for the responding level of the engagement taxonomy to our algorithm simulation exercises. As the exercises currently assume that the student always has the control, we included the questions in a way where the student is not forced to answer them. Due to this, we do not use the responses for grading purposes, but merely to give feedback and guide the process of solving an exercise.

The integration of the AVInteraction component to the exercises was straightforward and simple. Our approach,

[3]There are some 40 exercises implemented that already cover most common data structures and algorithms.

[4]The level of engagement depends on the nature of the exercise. For a more detailed discussion, see [7].

[5]In Jeliot 3, the user can turn on/off the questions but cannot leave a shown question unanswered.



Color the given nodes in the binary search tree so that the resulting tree is a valid red black tree. You can toggle the color of a node by selecting it. Tutor will present you helpful questions.

How many children can a red node have?

#1 Zero

#2 One

#3 Two

No answer submitted yet

Figure 1: An example algorithm simulation exercise with tutoring questions. The question "How many children can a red node have?" is asked because the red node E has only one child node, which is against the red black tree properties.

however, required dynamic generation of questions (the questions depend on the exercise instance that is randomized). Thus, some problems were caused by the design of the component in which all the questions were loaded upon initialization.

The AVInteraction package did not fill all the needs we had once starting the implementation. There is room for improvement in the use of visualizations with the questions. For example, it would be good if one could add visualizations as part of the questions. Furthermore, a feature that allows students to answer the questions by manipulating the visualizations in the exercise instead of through the AVInteraction GUI would be useful. In a way, this has already been implemented in our version. For example, in the red black tree coloring exercise, the question "What color is the root node of the red black tree" can be "answered" by coloring the root node black, which causes the question to be changed.

One important aspect of introducing such tutor to more exercises is the amount of work required to add the responding functionality. The introduction of the questions does require some effort in addition to the implementation of a new exercise. However, this effort is mostly needed to design a sensible set of questions and the logic used to determine when and why to ask them. Adding the questions to the exercises only requires implementation of a Java interface with one method.

5. DISCUSSION

The aim of this project was to extend the TRAKLA2 system to support more of the levels of engagement taxonomy. We first review the levels and give examples of the use cases possible with this extended version. It should be noted that these are merely examples, and it is possible to design simulation exercises for our system where some certain levels are supported. For example, an exercise with support only for responding level.

Viewing is allowed in several phases during the simulation process. First, the concept of visual algorithm simulation is based on the visualizations of data structures that the user is supposed not only to view, but also to modify during the process. In addition, in the algorithm simulation exercises, TRAKLA2 provides model solutions in terms of algorithm animations that also correspond to the viewing level.

Responding is fully supported after integrating AVInteraction module to be accompanied with the exercises. The major differences between our approach and some previous experiments are the following:

1. Our system allows *dynamic questions* to be asked that are related to the current state of the simulation exercise. This is different, for example, compared with ANIMAL in which the questions are static and loaded together with the animation. Thus, in ANIMAL, the questions are predefined and always the same from one run to another. Our approach, however, allows that the question is parametrized and depends on the current run.

2. Our system has *non-blocking questions* during the animation / simulation. For example, in Jeliot 3, it is possible to ask dynamic questions, but the student cannot continue the first task (animation) until the question is answered. Our approach would allow both, blocking and non-blocking questions, but

we have decided to use non-blocking questions since the focus is on the simulation that is graded. Thus, the animation / simulation can be continued, and the question might even be answered by doing the correct step in the simulation. This is an especially useful feature if the question is intended to guide the student more than address some detail related to the topic.

3. Algorithm simulation exercises intrinsically have the property that the student is required "to predict" the next step in the simulation process. In ANIMAL and Jeliot 3, this is something that is promoted through responding: the student is asked an explicit question "predict what happens next". Thus, in our case, we consider the "responding level" to be something that is already incorporated in the "changing" and "constructing". Of course, this new implementation makes responding and this kind of predicting more explicit.

Changing entails modifying the visualization. In general, visual algorithm simulation supports this level very well. For example, in MatrixPro, the user can freely choose any input for the algorithm under study in order to explore the algorithm's behavior in different cases. Such free experimentations could be connected to some open ended questions, but the TRAKLA2 exercises are more closed in this respect. Thus, in the current set of algorithm simulation exercises, this level is not promoted very well. However, the form of engagement is more on the next level.

Constructing requires the students to construct their own visualizations of the algorithms under study. In terms of visual algorithm simulation, this means that students simulate a given algorithm step-by-step by manipulating visual representations of data structures. In TRAKLA2, this process is supported by a Java applet that automatically updates the representations according to the modifications. Thus, the student can concentrate on the logic and behavior of the algorithm, i.e., content, and ignore the hard part of drawing complex illustrations. The students construct visualizations, but do not draw them, which means that the focus is on the process, not on a single outcome.

Presenting entails showing an algorithm visualization to the audience for feedback and discussion. In general, visual algorithm simulation is an effortless way to produce such visualizations [3], but our experiences are limited to lecture situations where it is the lecturer that gives the presentation. However, MatrixPro could be used in this engagement level, for instance, in a setup that requires the student to prepare the visualization on-the-fly. For example, the students could be told which algorithms and data structures to get familiar with, but the actual visualization must be constructed during the presentation (they can learn to simulate the algorithm with any input they wish, but do not know which input they must use in the final presentation).

Some tools are difficult to place in the engagement taxonomy. This might be due to the fact that the levels are not orthogonal (the semantic distance among the levels could be higher if we choose some other levels instead of these four; omitting "no viewing", and recalling that the 1st level, viewing, was considered to be included in all the other four in the first place). We back up our claim by arguing that 1) it is not always clear whether some activity belongs to a certain engagement level or perhaps on many levels at the same time; and 2) it is not always clear how wide is the scope of a certain level. First, in algorithm simulation exercises, the student is constantly supposed to "predict the next step" something typically considered to be on the **responding level**. Still, while doing these exercises the students are clearly **changing** the data structures involved as well as **constructing** the animation sequence (i.e., algorithm visualization) submitted as an answer. Second, if we ask questions about the AV material, it is not clear whether this is included in the changing and constructing levels. For example, our approach to integrate AVInteraction is such that the student can "answer" the question (e.g., what's the color of the root node in a red black tree?) explicitly by choosing the correct alternative, or by **changing** the visualization while **constructing** the next step (e.g., toggling the color of the root node). If the student picks up this latter alternative, is this action considered to belong in the responding, changing, or constructing level?

Another interesting measure to consider is whether or not the inclusion of the responding level in the algorithm simulation exercises lowers the number of submissions students use. In a previous study, we have found out that there exists a group of students who use resubmissions carelessly [5]. Thus, our hypothesis is that including the responding level in terms of tutoring questions could improve the performance of these students.

6. CONCLUSIONS

In this paper, we have described work in progress on the next generation of the visual algorithm simulation (exercises) that support also interactive questions, i.e., responding level in the engagement taxonomy by Naps et al. [13]. While the prototype implementation has a number of enhancements, we focus on discussing the meaning and importance of such new functionality in our context that has certain differences compared with other AV systems. This has also an impact on how to interpret the levels of the engagement taxonomy, especially while planning research setups to test the hypotheses set by the authors of the engagement taxonomy (e.g., whether an effective instructional AV must allow the visualization designer to ask questions of the student).

The aim was to support the responding level also in the context of algorithm simulation exercises. The purpose of such questions typically include the idea of predicting the next step in an algorithm animation. However, as the algorithm simulation exercises intrinsically include the idea of "predicting the next step", we wanted to allow more options to ask questions. First, the questions do not need to block the simulation process, but the student can answer the question either explicitly (by providing the correct answer) or implicitly by continuing the simulation until the question is fulfilled by other means. This requires that the questions are dynamic in such a way that they are dependent on the current state of the exercise (i.e., not only the input of the algorithm has an effect to the question and its answer(s), but also the state of the simulation). This is different compared with algorithm animations that have only one possible execution path after the input is set. In algorithm simulation, there are many execution paths (even though only one of them is considered correct) as the exercises should also allow the students to make errors. Second, the scope of the questions do not need to be limited to predicting what happens next, but they can also guide the students to broaden their knowledge about the topic. Such questions can include tutoring questions (to guide how to proceed with the simulation) and detailed questions (to focus on certain properties of the data structures and algorithms) as well as some mixture of these two. Finally, the responding level questions can easily be combined with other engagement levels. Our hypothesis is, however, that it might be difficult to compare certain levels of engagement directly (i.e., responding with constructing). Yet, this work makes it explicit that the responding level exists if required. This should be taken into account while planning new research setups. For example, if the idea is to compare responding with changing, it might turn out that it is intrinsically impossible to have a setup in which changing does not include any kind of responding (unless changing is forced to be implemented in an unnatural way). This point of view may also give a valuable insight into previous studies that have come up with mixed results.

Acknowledgments

We would like to thank Tomi Lehto for making the implementation work for this research.

This work was partially supported by the Academy of Finland under grant number 210947.

7. REFERENCES

[1] S. Grissom, M. F. McNally, and T. Naps. Algorithm visualization in CS education: comparing levels of student engagement. In *Proceedings of the 2003 ACM symposium on Software visualization*, pages 87–94, New York, NY, USA, 2003. ACM Press.

[2] C. D. Hundhausen, S. A. Douglas, and J. T. Stasko. A meta-study of algorithm visualization effectiveness. *Journal of Visual Languages and Computing*, 13(3):259–290, June 2002.

[3] P. Ihantola, V. Karavirta, A. Korhonen, and J. Nikander. Taxonomy of effortless creation of algorithm visualizations. In *Proceedings of the 2005 international workshop on Computing education research*, pages 123–133, New York, NY, USA, 2005. ACM Press.

[4] D. J. Jarc, M. B. Feldman, and R. S. Heller. Assessing the benefits of interactive prediction using web-based algorithm animation courseware. In *The proceedings of the thirty-first SIGCSE technical symposium on Computer science education*, pages 377–381, Austin, Texas, 2000. ACM Press, New York.

[5] V. Karavirta, A. Korhonen, and L. Malmi. On the use of resubmissions in automatic assessment systems. *Computer Science Education*, 16(3):229 – 240, September 2006.

[6] V. Karavirta, A. Korhonen, L. Malmi, and K. Stålnacke. MatrixPro - A tool for on-the-fly demonstration of data structures and algorithms. In *Proceedings of the Third Program Visualization Workshop*, pages 26–33, The University of Warwick, UK, July 2004.

[7] A. Korhonen and L. Malmi. Taxonomy of visual algorithm simulation exercises. In *Proceedings of the Third Program Visualization Workshop*, pages 118–125, The University of Warwick, UK, July 2004.

[8] T. Lauer. Learner interaction with algorithm visualizations: viewing vs. changing vs. constructing. In *Proceedings of the 11th annual SIGCSE conference on Innovation and technology in computer science education*, pages 202–206, New York, NY, USA, 2006. ACM Press.

[9] L. Malmi, V. Karavirta, A. Korhonen, J. Nikander, O. Seppälä, and P. Silvasti. Visual algorithm simulation exercise system with automatic assessment: TRAKLA2. *Informatics in Education*, 3(2):267 – 288, 2004.

[10] A. Moreno, N. Myller, E. Sutinen, and M. Ben-Ari. Visualizing programs with Jeliot 3. In *Proceedings of the International Working Conference on Advanced Visual Interfaces*, pages 373 – 376, Gallipoli (Lecce), Italy, May 2004.

[11] N. Myller. Automatic prediction question generation during program visualization. In *Proceedings of the Fourth Program Visualization Workshop*, 2006. To appear.

[12] T. L. Naps, J. R. Eagan, and L. L. Norton. JHAVÉ: An environment to actively engage students in web-based algorithm visualizations. In *Proceedings of the SIGCSE Session*, pages 109–113, Austin, Texas, Mar. 2000. ACM Press, New York.

[13] T. L. Naps, G. Rößling, V. Almstrum, W. Dann, R. Fleischer, C. Hundhausen, A. Korhonen, L. Malmi, M. McNally, S. Rodgers, and J. Ángel Velázquez-Iturbide. Exploring the role of visualization and engagement in computer science education. *SIGCSE Bulletin*, 35(2):131–152, June 2003.

[14] G. Rößling and B. Freisleben. ANIMAL: A system for supporting multiple roles in algorithm animation. *Journal of Visual Languages and Computing*, 13(3):341–354, 2002.

[15] G. Rößling and G. Häussage. Towards tool-independent interaction support. In *Proceedings of the Third Program Visualization Workshop*, pages 110–117, The University of Warwick, UK, July 2004.

[16] O. Seppälä, L. Malmi, and A. Korhonen. Observations on student misconceptions – a case study of the build-heap algorithm. *Computer Science Education*, 16(3):241–255, September 2006.