DAVE: A Dynamic Algorithm Visualization Environment for Novice Learners

Euripides Vrachnos and Athanassios Jimoyiannis
Department of Social and Educational Policy, University of Peloponnese
evrachnos@gmail.com, ajimoyia@uop.gr

Abstract

Acquiring knowledge about algorithms and programming skills is a difficult and complex process in particular. Various algorithm visualization systems have been developed, using animation techniques to illustrate the behavior of basic algorithms, to facilitate students’ learning and skills development. This paper presents DAVE, an interactive dynamic algorithm visualization system for the introductory lessons in algorithm design and programming. DAVE allows students’ experimentation not only with sample algorithms, constructed by the designer, but, mainly, with the automatic animation of their own algorithms.

1. Introduction

Acquiring and developing knowledge about algorithms and programming is a highly complex but particularly interesting process, from both the teacher and student perspective. Literature shows that novices in their journey to programming and algorithmic thinking are faced at many difficulties [8, 18, 21]. There are clear differences between novices and experts in programming. The most important one is that expert programmers form abstract representations based upon the purpose of the code whereas novices are limited to surface characteristics of the code and form concrete representations based on how the particular code functions.

An algorithm describes a process which is both abstract and dynamic. Common methods used to teach analysis and design of algorithms seems to be insufficient to help students to construct effective representations and stable mental models about algorithmic concepts and processes. Computer simulations and visualization systems aim at enabling students to grasp the abstract and complex concepts more easily and facilitate conceptual understanding of algorithmic processes.

Over the last years, algorithm visualization has received increasing interest within the education community while numerous visualization systems have been developed [4, 10, 16, 20, 22]. Those systems aim to visually demonstrate the fundamental operations of basic algorithms concentrating more on abstractions of their behavior than on low-level program structures.

Moreover, several experiments have been conducted to investigate the educational value of algorithm visualization systems although [3, 6, 13, 23]. Surprisingly, in many cases, the results showed that algorithm visualization had minimal impact on students’ learning. Hundhausen et al., in their meta-study of 24 evaluation studies on algorithm visualization, concluded that effective algorithm visualization systems should be interactive and promote students’ engagement [11]. Thus, the most significant factor influencing students’ learning appears to be not what algorithm visualizations viewers see but what they do with them.

In this paper we present a fully interactive Dynamic Algorithm Visualization Environment (DAVE) aiming to enhance secondary education students’ learning about algorithms. An overview of the literature outlines the field of algorithm visualization and follows an analytical presentation of the key design features, the architecture and the functionality of the system.

2. Subject Overview

Algorithm animation systems have a long history in education. The first reference to algorithm animation was the famous video ‘Sorting Out Sorting’ which has been presented by R. Baecker on 1981 at the SIGGRAPH Conference [2]. This 30 min video demonstrated the characteristics and operations of nine sorting algorithms, using animation and audio comments. Since then, many researchers have developed various visualization systems to depict the
behavior of certain algorithms and assist students’ deeper understanding of algorithmic and programming concepts [4, 10, 16, 20, 22].

Nowadays, visualization involves more than the visual presentation of information in various media (such as animations, text, static or dynamic pictures, diagrams, etc.). Most algorithm visualizations provide an interactive environment that elicits active student participation using informative animations and multiple representations. Additionally, an effective visualization should not merely attract the visual attention of the learner but should be designed to promote the cognitive attention and engagement of the learner.

In general, there are two main categories of visualization that focus on education: program visualization and algorithm visualization.

2.1 Program Visualization

Program Visualization (PV) systems produce direct representations of programming structures and/or program execution phases (e.g., values of variables, internal program structures, method frames, data structures, objects etc.).

BlueJ [14] and Jeliot [15] are the most popular systems developed for object-oriented programming. In BlueJ, students interact with classes by creating objects and calling their methods through a visual interface. The system is excellent for an introductory course in object-oriented programming with Java. However, it is not an algorithm animation system, and cannot be effectual in visualizing abstract data types and operations on them. Jeliot animates automatically the object-oriented structures of the source code written by the student, but visualizations are restricted to a subset of Java source code.

PlanAni [20] is a program animation system based on the pedagogical approach of teaching the roles of variables in introductory programming. The idea is to utilize a role image for all variables having the same role in the program. Role-based animation of operations provides a meaningful visual representation of role properties and thus provides additional memory elaboration.

2.2 Algorithm Visualization

Algorithm visualization (AV) deals with the visual representation of the program at a higher level of abstraction. Usually, the logic behind an algorithm cannot be revealed by just showing how the values of the program variables change. Students need proper graphical representations which fit better to their mental models about the execution of the particular algorithm. AV systems can support this need by visualizing abstract concepts and unfolding the underlying logic of the algorithm under study.

The most popular technique for creating an algorithm animation is by annotating the algorithm code with scripting commands producing the visualization. The first system based on a scripting language has been developed by John Stasko and his colleagues [23] and belongs to a wide family of algorithm visualization systems (Tango, Polka, Samba and JSamba). Animations consist of a file containing graphics instructions which correspond to certain events of the algorithm.

A system similar to JSamba is JAWAA [17] which supports the representation of data structures such as arrays and trees. Its main drawback is that it cannot be easily used for algorithms handling data structures which are not included in the implementation. There are also systems that allow the student to build his own algorithm animations by using a graphical editor, such as MatrixPro [12] and Animal [19]. MatrixPro has no support for source tracing. The Animal editor generates a text script that can be modified by hand, a task that is particularly difficult for the students.

A novel AV system is JHave [16]. JHave cannot be considered as a complete algorithm visualization system but rather as a support environment for displaying a variety of algorithm animations produced with other AV systems (e.g. JSamba, Animal). The special feature of this system is the ‘stop-and-think’ questions and explanations that can appear at any time during the animations, promoting thus students’ active interaction with the visualization of the algorithm.

Alvis Live! is an algorithm visualization system which supports the construction and the interactive presentation of ‘low fidelity’ algorithm visualizations using the SALSA scripting language [10]. In Alvis Live! the students can write their own programs and algorithms using a pseudo-code like scripting language. This language provides only arrays, variables and array indexes as data types. Moreover, this system provides an error checker reporting error messages while the student develops its code.

A different approach for algorithm visualization uses graphics libraries instead of a scripting language. In that case, visual function calls are inserted by hand in the source program to produce the target visualizations. Algorithm Explorer [4] is a system of this type. Students have to use a C++ graphics library to construct their own visualizations of the algorithms under study. This is also a difficult task for novice programmers.
3. Key features of the design framework

Research studies have shown that the learning benefits of an AV system are limited when the student watches passively an animation without having the opportunity to actively interact with it [13, 23]. Although students benefited from their active engagement with the animated algorithm, in most AV systems this is accomplished by a VCR-like control panel which helps their experimentation just by controlling the execution of sample algorithms.

Therefore the critical design principle for an effective algorithm visualization system is the engagement of the student, not only with the execution of sample algorithms, but also with the animation of their own algorithms. However this is extremely difficult for students attending introductory programming courses because they have to program the animation along with the algorithm to be studied. In most cases, they need to learn a scripting language in order to construct an abstract visualization of the source code.

The design philosophy of DAVE is to avoid any kind of annotation of the source code with scripting commands or calls to graphical libraries. An automatic animation of the algorithm, directly generated from the student’s source code, is the best way to promote his cognitive engagement and understanding of the basic characteristics of the algorithm under study. This feature will enhance student’s learning opportunities by adding a new dimension in his interaction with the algorithm. Thus the student watches a graphical representation of the basic features of his own algorithm while, at the same time, he can detect and fix any logical error in the code.

In his classic paper for the design of educational simulations Alessi [1] has proposed a framework consisted of three building blocks. Following a previous adaptation of this framework to the requirements of dynamic visualization systems [5], we have defined three main components in DAVE:

Visualization scenario: Scenario related issues concern the focus of the visualization model and the degrees of model modification and model transparency (for both teachers and students).

Visualization model: The computational model upon which the visualization of the target algorithm is based is related to the appropriate representations. The critical issue is to clarify complex or difficult concepts and to tailor the animation to the students’ prior knowledge and experience.

Instructional overlay: The instructional overlay is made up of those features defining the educational context, the educational approach (constructivist and discovery learning), the tasks and the representational forms used, and, finally, the learner motivation, guidance, assistance and feedback.

4. DAVE

A common technique to produce algorithm animations is by annotating the algorithmic code with calls to visualization routines. Firstly, the actions which play an important role in relation to the visualization process have to be identified. These actions are critical for the inherent logic of the algorithm and are referred hereafter as interesting events. Every interesting event in the code is associated to a suitable animated action. This association is critical from both the pedagogical and the technological perspective of the animation design. In the case of array algorithms (searching, sorting, etc.), for example, interesting events are the comparison between two array elements or the swapping of two array elements.

Moreover, the animation uses graphical effects to denote specific algorithmic operations depending on the category of the algorithm. For example, in the sorting algorithm the animation highlights key-operations, such as comparing or swapping of array elements. The aim is to help the student to associate every command in his code with a graphical representation that is closer to his mental model, to focus on the important properties of the algorithm, and, finally, to understand its underlying logic.

The main goal of DAVE is not only the active engagement and experimentation of students with sample algorithms, but more importantly, the automatic visualization of the algorithm that was designed by the student himself. The system associates each algorithmic command with the appropriate graphical representation of data in order to highlight the fundamental features of the algorithm.

4.1. Interface and Functionality

Figure 1 shows a screenshot of DAVE with the animation of the bubble sort algorithm. The user interface of DAVE consists of a top-level application window with the following components:

- The source code editor, where the student can write its algorithm using pseudo-code or a programming language (left side)
- The animation area which displays the animation, e.g. the instant changes of the graphical
representations associated with the operation of the current algorithm (right side)

- The animation control panel with buttons which control the progression and the speed of the animation (down left). The main buttons support various operations (start, pause, restart, forward-backward running).
- The main control panel containing the menu commands and buttons dealing with operations as open, save, compile and run a program file.
- The output area where the system displays the program output or messages for the students (down right).

![Screenshot of DAVE’s user interface](image)

**Figure 1. Screenshot of DAVE’s user interface**

During its evolvement animation highlights the source code command which corresponds to the current event in algorithm visualization. This helps the student to map a programming command to the corresponding graphical action in the animation panel. We have adopted a popular and effective representation for the array elements (bars with different height) used also by JHave [16] and Halvis [7] systems in sort algorithms.

DAVE is written in Java, therefore it runs practically on any operating system and in the Web as an applet as well. The prototype uses a two dimensional visualization but the system can be extended to 3D visualizations using Java OpenGL libraries (JOGL, Java3D).

Currently the prototype supports the pseudo-code programming language, named GLOSSA, which is used in Greek upper secondary education schools in an introductory course about algorithms and programming. DAVE can easily be extended by adding new languages, such as Pascal C, C++ or Java, in order to support courses in programming at the university level. The system can be extended to support any procedural programming language and basic algorithmic structures, such as sequence, selection, loops, iteration structures etc.

4.2. The Architecture

The DAVE system consists of three building components, as shown in Figure 2: the user interface, the compiler and the animation engine.

![Basic Architecture of the system](image)

**Figure 2. Basic Architecture of the system**

We have used the ANTLR [9] tool for the design and the development of the compiler in Java. The compiler maps the data objects of an algorithm to their visual counterparts in the animation. This is done by translating the variables in the source code to special data types that are provided with special animation functions for the algorithm to be animated. Every algorithm corresponds to a special data type, since the interesting events that need to be highlighted are different in general. The generated animation code is executed by the Java interpreter.

The prototype supports 2D animations on array algorithms (searching and sorting). We can add a new category of algorithms just by adding a new visual data type into the compiler, in the form of a Java class, using methods that identify the interesting events in student’s code. Our future plans are directed to add visualization of graph algorithms.

5. Summary and Future Work

This paper presented DAVE, which is a user friendly, dynamic algorithm visualization environment
Aiming to promote students’ active engagement with visualizations of algorithms. The main advantage of DAVE is that allows the engagement of the student, not only with the visualization of sample algorithms, but also with the visualization of their own algorithms. The system does not demand from the students to learn any scripting language or to annotate the source code with graphics function calls.

From the educational perspective, DAVE may offer to introductory courses about algorithms and programming not only by facilitating constructivist and discovery learning activities but also by supporting different types of learners. To our point of view, the educational potential of the dynamic algorithm visualization educational environments is that they provide opportunities for active learning, enable students to perform at higher cognitive levels, promote conceptual change, and help students to build abstract representations that reflect the inherent logical structures of algorithms. The following step of our research is the empirical evaluation of the system in real classroom settings by developing and applying proper learning scenarios.

References