JSAV: The JavaScript Algorithm Visualization Library

Ville Karavirta
Dept. of Computer Science and Engineering
Aalto University
ville.karavirta@aalto.fi

Clifford A. Shaffer
Dept. of Computer Science
Virginia Tech
shaffer@cs.vt.edu

ABSTRACT

Learning abstract concepts in data structures and algorithms (DSA) courses is often difficult for students. To improve understanding of DSA topics, numerous algorithm visualization (AV) systems and stand-alone AVs have been developed, supporting a wide range of algorithms and different engagement strategies. Prior studies show that active engagement of students is necessary to make AVs educationally effective. In this paper, we introduce JSAV, a new JavaScript framework for creating engaging algorithm visualizations with active learning features. JSAV is meant to be used with HTML5-based online learning materials. We describe the special features of JSAV that support active learning, and discuss its potential for use in online education projects.

Categories and Subject Descriptors

E.1 [**Data Structures**]; E.2 [**Data Storage Representations**]; K.3.2 [**Computers and Education**]: Computer and Information Science Education

General Terms

Algorithms, Design

Keywords

Data Structure and Algorithm Visualizations, Algorithm Animation, Interactive Courseware, HTML5, Active Electronic Textbooks, Hypertextbook, JSAV

1. INTRODUCTION

Algorithm Visualization (AV) has a long history of success in the classroom [1, 2, 20]. Nonetheless, many advocates of AVs have been disappointed at how the lack of overall classroom adoption does not match the reported support by students and instructors for AVs on surveys [12, 19]. Based on survey feedback, impediments to adoption include difficulty of finding or using AVs, and the difficulty of "fitting them in" as add-ons to existing lecture material. The second problem can be mitigated by providing complete units of instruction, whether at the single-topic, chapter, or semester level. Multiple initiatives have tackled this by integrating visualizations into learning material [13, 15, 18]. The OpenDSA Project [3, 21] seeks to provide complete instructional materials for a data structures and algorithms course.

Technical difficulties in using AVs stem in part from their historical context [20]. The earliest AV systems were implemented to run first under X Windows and then on proprietary operating systems like Microsoft Windows. Since the late 1990s, the dominant mode of implementation for AVs has been with Java, whether presented as a standalone application, a web applet, or using web-start. In any of these modes, Java has made AVs more portable between operating systems. However, Java is currently losing standing on the Internet, with its support becoming more problematic for instructors.

In contrast, HTML5 (with programming support via JavaScript) has quickly become an Internet standard, and is now supported by all major browsers with relative consistency. This makes it easier than ever before for implementers to create content that they can expect will run on any user's computer with little effort. Indeed, unlike Java, HTML5 even runs on most tablets and many mobile devices. HTML5/JavaScript includes all necessary computational, graphical, and interface support to create the most sophisticated of visualizations.

In this paper we present JSAV: The JavaScript Algorithm Visualization Library. JSAV is written in JavaScript, and is meant to support development of AVs for HTML5. A reader knowledgeable with the history of AVs might question the need for yet another AV support system, given that so many already exist. The two primary motivations for JSAV are:

1. There is little existing support for AV development in JavaScript. To our knowledge, the only other major AV development effort in HTML5/JavaScript is the collection of AVs and associated support environment by Galles[1].
2. JSAV includes a specific set of features designed to support development of AV-based exercises that involve active learning techniques, and visual content that can be easily integrated into online tutorials. In particular, the JSAV API supports special features for creating visual algorithm simulation exercises as first

[1]Data Structure Visualizations Website: `http://www.cs.usfca.edu/~galles/visualization/Algorithms.html`

implemented in the TRAKLA2 system [10], as well as support for creating slideshows for presenting instructional visualizations within a tutorial.

JSAV represents the collective experience of three major AV development groups: Aalto University (the developers of TRAKLA2), Virginia Tech, and the JHAVÉ community. Each group has written many widely used AVs, and the differing perspectives of the developers have ensured that JSAV is able to support the needs of a broad community within a development environment (HTML5) that will prove most significant to the future of online education.

Key features of JSAV include automated layout for a number of traditional data structures, support for presentation slideshows, and support for TRAKLA2-style "proficiency exercises" that require the student to demonstrate proficiency with an algorithm by simulating its key steps. JSAV is the development library for the OpenDSA project, which seeks to provide a complete open-source resource for teaching Data Structures and Algorithms courses.

The rest of this paper is organized as follows. Section 2 presents some background and related research. Section 3 presents JSAV in detail. Section 4 presents a discussion of lessons learned so far. Finally, Section 5 presents our conclusions.

2. BACKGROUND

Since there is more previous research on algorithm visualization than can fit into the pages of this article, we will focus here on the research themes of recent years and the general requirements for an AV system. Overviews for the history of AVs can be found in [1, 2, 20].

One recent trend in AV research has been integrating interactive visualization components with hypertext tutorial content. Early work on this topic was done by Ross and Grinder [13]. They defined the term *hypertextbook* to mean more than hyperlinked documents. They saw that it should include visualizations and active learning objects. An ITiCSE working group in 2006 provided guidelines on how to integrate visualizations into hypertext and course management systems [15], coining for such systems the acronym *VizCoSH* for Visualization-based Computer Science Hypertextbook. Since then, there have been other attempts to merge visualization with tutorial content. The Animal visualization system has been integrated into hypertext as a Moodle module [18]. In that project, visualizations were launched from the hypertext using Java Web Start. Another proposed solution is JSXaal, a visualization system implemented in HTML5 and JavaScript [5]. Finally, there is the already mentioned OpenDSA project [3, 21]. OpenDSA uses the term *active eBook* to refer to a merging of content, AV, and exercises with automated assessment. The visualizations in OpenDSA are written using JSAV.

Increasing evidence confirms the hypothesis that student engagement is the key to educational effectiveness of AVs [4, 12]. The different types of engagement encountered in AVs have been categorized in the *Engagement Taxonomy* [12]. The taxonomy defines five levels of engagement between a student and an AV:

- *Viewing* - The student passively views a visualization, perhaps with the ability to control the animation speed or move step-by-step backward and forward. Most AV systems support this minimal level of engagement.
- *Responding* - The student must respond to questions about the content while viewing an AV. These are most often pop-up questions where the student is required to select or type the correct answer. This type of engagement is used, for example, in Animal [14] and JHAVÉ [11].
- *Changing* - The student must change the visualization by, for example, providing input data to the algorithm. For example, the JHAVÉ system supports this in some visualizations.
- *Constructing* - The student must construct a visualization. A variation on this approach is taken in MA&DA [9] and TRAKLA2 [10], where the student is given a data structure and an algorithm and is expected to simulate the algorithm. That is, they need to imitate the steps of an algorithm by controlling the visualization. This approach is also called *visual algorithm simulation* [8].
- *Presenting* - The student is presenting a visualization to others.

The hypothesis of the taxonomy is that the higher the level of engagement, the more educationally effective the AV is. Therefore, the possibility for creating engaging material can be considered the most important feature of an AV system.

Over the years, more proposals on the required features of an AV system have been introduced [5]. Most of the requirements have been introduced by Rößling and Naps [16, 17]. Their first requirement is that the AV system should support as wide a target audience as possible. Since the mid-1990s, this has meant implementing the systems in Java and has led to Java being the most common implementation technology [20]. However, since the widespread adoption of HTML5 the importance of Java on the web is continually diminishing, and browser support is becoming more difficult for instructors to maintain. In contrast, HTML5 and JavaScript are rapidly gaining in popularity.

This shift in technology has not yet been widely reflected in the technologies used in AV systems. Work by Galles already mentioned is probably the first large-scale JavaScript-based algorithm visualization collection. However, those visualizations are geared towards viewing the algorithms and offer no engaging content. JSXaal is a viewer for visualizations in the Xaal algorithm animation language [5]. In terms of engagement, it supports pop-up questions, changing input data, and student annotations. JSXaal does not, however, support algorithm simulation exercises. Furthermore, it is not intended for creating visualizations, but merely viewing visualizations created in existing systems.

To summarize, there is a clear need for a system that seamlessly integrates AVs with hypertext learning material, supports multiple levels of engagement through the use of exercises and simulations, and is implemented with HTML5 and JavaScript.

3. JSAV

JSAV has adopted what we consider to be the best features from the large number of existing AV systems that we are familiar with. What makes it different from other libraries for building AVs is the ease with which it can be integrated into hypertext and its support for creating engaging exercises. In the following, we will present JSAV's most innovative features.

Levels of Engagement.

JSAV supports different types of visualizations on multiple engagement levels. The simplest are *static images* of data structures. JSAV allows easy generation of figures under programmatic control for use in illustrating learning material. The main advantage of such images compared to image formats like PNG is the ease in changing visual appearance and the data presented. Since the images are generated programmatically (often representing state of an algorithm run to a particular point), it is easy to adjust the algorithm or associated input to generate a new image at another specified point in the algorithmic process.

The second type of visualization we call *slideshows*, which show a series of steps to animate the behavior of an algorithm. The student is merely *viewing* the AV. Students can control the slideshow by moving a step backward or forward, to the beginning or the end. They can change the speed of transition animations. Implementing a slideshow using JSAV feels much like writing a presentation using a script-based slideshow package such as Beamer. Again, programmatic control makes for powerful integration of the visualization with the algorithm being visualized.

While important as visual aides to accompany tutorial presentations, both static images and slideshows represent only the viewing level of the engagement taxonomy. Viewing is the lowest level of engagement and therefore does not represent the most effective use of AVs. A simple way to increase engagement is through pop-up questions that can be used in slideshows. These require the student to *respond* during the AV. JSAV supports questions that can be either true/false, multiple-choice, or multiple-select questions. A question is shown as a popup to the student. When a question is answered, the correctness of the answer is given. If the answer was incorrect, the student is allowed to try again. While not strictly a part of JSAV, we have also successfully integrated JSAV displays with the Khan Academy exercise infrastructure[2], thereby gaining access to the ability to create a wide range of interactive exercises that can allow students to directly manipulate the data structures as part of specifying the answer.

Finally, JSAV supports algorithm simulation exercises on the engagement level known as *constructing*. We refer to these as *proficiency exercises*, since the student has to show his/her proficiency with the algorithm by simulating the steps taken by the actual algorithm. In JSAV proficiency exercises, simulation of the algorithm is mainly done by clicking the data structure visualizations or buttons. JSAV supports versatile feedback modes for proficiency exercises, as will be explained later.

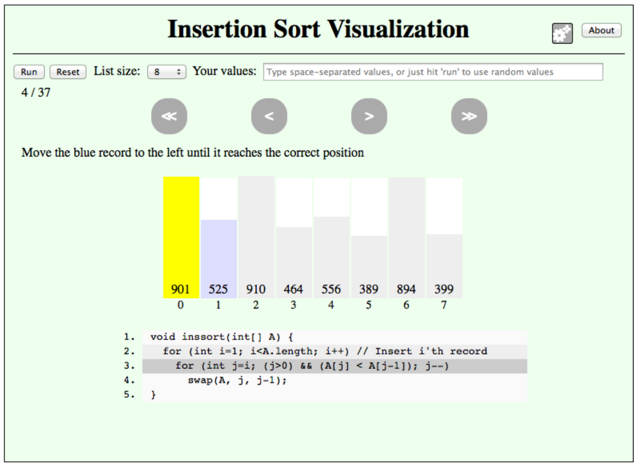
Visual Components.

On all levels of engagement, visualizations can be composed from the same selection of objects. The building blocks used in creating JSAV visualizations are similar to many existing AV systems. There are essentially three types of objects: *data structures*, *graphical primitives*, and *code*.

The data structures supported are array, linked list, tree, binary trees, and graphs.[3]. Examples for all these structures are shown in Figure 1. The structures support operations that one would expect, like set/get values of array elements, add/remove/get children of a node in a tree, set/get the next node in a linked list, or add/remove nodes and edges in a graph. There are also methods to change the visual appearance for parts of a structure. Calls to any operations that change the state or visual appearance of the objects are recorded and can be undone and redone by the student when in slideshow mode. Default visual appearance for all visual elements can also be specified using CSS. Finally, all data structures support automatic layout of their elements as well as allow manual positioning.

The graphical primitives supported are text, line, circle, ellipse, rectangle, polygon, polyline, and a general path. The general path allows drawing arbitrary shapes made of lines and curves[4]. In addition, multiple graphical primitives can be combined into a *set*. The visual appearance can be changed through method calls on the objects, or their default appearance can be specified using CSS. Furthermore, the objects can be scaled, rotated, and moved. Again, all changes are recorded to the animation.

To tie the visualization with code for the algorithm, JSAV supports visualizing pseudocode and variables. The pseudocode element can read a piece of code from a file and display it as shown in Figure 2. The code object has methods for highlighting specified lines or the current line. It can also automatically indicate the previous line when the currently highlighted line is changed. Variables can be used to track and visualize variable values in an algorithm.


Insertion Sort Visualization
About
Run
Reset
List size:
8
Your values:
Type space-separated values, or just hit 'run' to use random values
4 / 37
≪
<
>
≫
901
525
910
464
556
389
894
399
0
1
2
3
4
5
6
7

Move the blue record to the left until it reaches the correct position

```
void inssort(int[] A) {
  for (int i=1; i<A.length; i++) // Insert i'th record
    for (int j=i; (j>0) && (A[j] < A[j-1]); j--)
      swap(A, j, j-1);
}
```

Figure 2: Example of a JSAV visualization that includes the pseudocode component.

For all the visual components, positioning can be defined in three ways. First, elements can be added to the HTML document tree before or after certain elements. This leaves the exact positioning to the browser's layout engine. Second, the objects can be positioned using absolute pixels relative to the left, right, top, and bottom of the AV canvas. Finally, components can be positioned relative to other components.

[2]`https://github.com/Khan/khan-exercises`

[3]The graph support is new, and so AVs built with graphs have not yet been used in a course.

[4]JSAV can draw anything that is possible with the SVG path element `http://www.w3.org/TR/SVG/paths.html#PathData`.

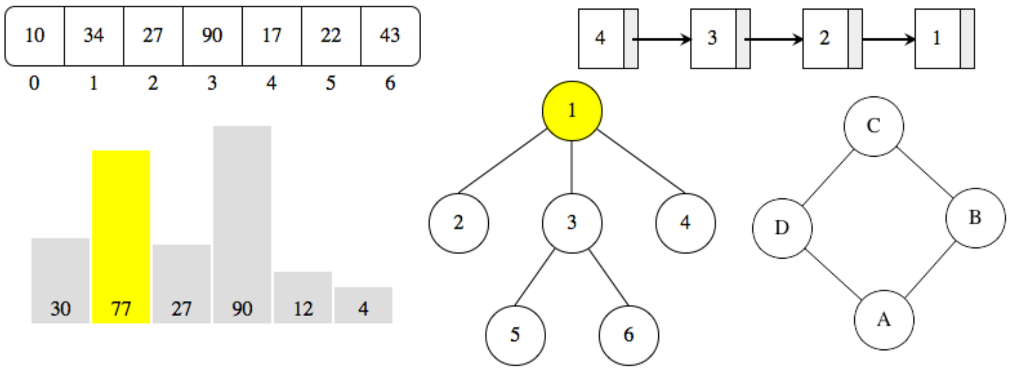
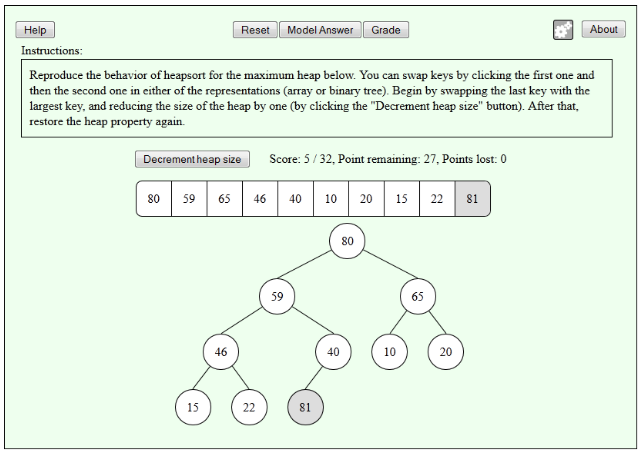


Figure 1: Examples of JSAV data structures array, bar layout for array, linked list, tree, and graph.

Proficiency Exercises.

One of JSAV's most innovative aspects is its support for algorithm simulation, which we call *proficiency exercises.* Proficiency exercises require the student to simulate the workings of an algorithm by, for example, clicking on array indices or tree nodes to swap the items. The student is essentially constructing their own algorithm visualization, within the constraints that the exercise developer has provided to allow student control. This student visualization can be automatically assessed by comparing it to a model solution. The model solution is also available for student view as a JSAV slideshow. Figure 3 shows an example of the Heapsort exercise.



Instructions:

Reproduce the behavior of heapsort for the maximum heap below. You can swap keys by clicking the first one and then the second one in either of the representations (array or binary tree). Begin by swapping the last key with the largest key, and reducing the size of the heap by one (by clicking the "Decrement heap size" button). After that, restore the heap property again.

Score: 5 / 32, Point remaining: 27, Points lost: 0

Figure 3: Example of a JSAV proficiency exercise for Heapsort.

JSAV proficiency exercises provide three different pedagogical modes. They are:

- **Limited feedback:** In this mode, the student is given only the number of steps correct so far in the student solution, and only when the student requests it.
- **Continuous feedback with incorrect steps undone:** The student gets feedback after (almost) every operation he/she does. If the latest step the student took is incorrect (that is, it does not match the corresponding step in the model answer), then it is undone to the last correct step and the student can try that step again.
- **Continuous feedback with incorrect steps fixed:** Like in previous mode, feedback is given after each operation the student does. If the step is incorrect, however, the state of the solution is automatically corrected to the one in the model solution. The student does not get a chance to redo that step, and does not get a point for that step.

Developing proficiency exercises requires the AV author to first write JavaScript to generate random input data for the exercise and initialize the data structures used. The second required task is to write the model solution similarly to how JSAV slideshows are written. The model solution needs to be annotated to mark the steps which are to be graded. Finally, the author needs to add the student interaction, that is, attach the needed event handlers to the data structures so that changes are made when students interact with them. The continuous feedback with step fixing mode requires the author to supply a function that will take a state of the model solution and fix the student solution to match that state.

Technology.

The user interface of JSAV is all HTML with the functionality implemented in JavaScript and the appearance specified with CSS. This makes integrating JSAV visualizations within hypertext simple and flexible. The visualizations can be written/loaded directly into the hypertext material, or be implemented within their own HTML documents and embedded with `iframe` HTML elements.

JSAV takes advantage of some existing, high quality JavaScript libraries. It uses jQuery to help in solving differences in browsers and working with the DOM. jQueryUI implements animation effects and element positioning. Finally, Raphaël eases using SVG to display and animate changes to the graphical primitives.

Officially, JSAV supports all modern versions of Chrome, Firefox, and Safari browsers. For each of these, the library has been tested and used by students. Furthermore, the library should work in IE and Opera as well as Mobile Safari on iOS and the default browser on Android 4. JSAV is open source and released under the MIT license. Source code is available from a public GitHub repository[5].

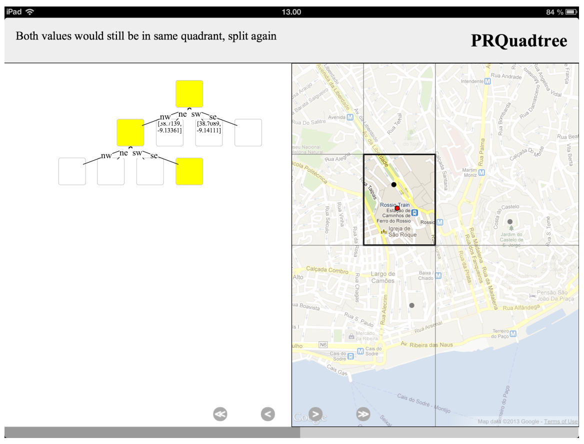
[5]`http://github.com/vkaravir/JSAV/`

Extending and Customizing.

The technologies used makes customizing JSAV easy and flexible. By changing CSS styles, the visual appearance of the AV can be changed. JSAV aids CSS styling by using hierarchic HTML `class` attributes when styling objects of different types. For example, all nodes in list, trees, and graph have CSS class `jsavnode`. They also have more specific classes with binary tree nodes having additional classes `jsavtreenode` and `jsavbinarynode`. When many JSAV visualizations are used together in a document or project, they should all use a common stylesheet to allow changing the appearance of all the AVs simultaneously.

Customizing and extending the behavior is made easy with the dynamic nature of JavaScript. As JSAV exposes all of the types used for its visual components, adding or changing functionality is straightforward. For example, a visualization of the *binary search tree* insert operation could add a function for the BST insert to the *binary tree* component without changing the JSAV source code. New core functionality can also be added outside of JSAV. In fact, most of the functionality in JSAV itself is implemented using the extension mechanism to add new functionality in independent modules. This makes it possible to build smaller versions of JSAV with only the features needed by a specific AV or hypertextbook.

Finally, browser events can be used to change or control JSAV functionality. JSAV triggers many events whenever a user interacts with it. For example, both successfully and unsuccessfully completing steps in a proficiency exercise exposes scoring events. Such events can be used to, for example, change displayed student progress in HTML surrounding the visualization (as is done in OpenDSA). Furthermore, JSAV listens to some events on its container. These events can be used, for example, to move backward or forward in the animation, enabling building of different kinds of student controls in addition to the default ones provided by JSAV. An example of customizing the UI is shown in Figure 4. Here, the events have been used to add AV controls and a progress bar to the bottom of the iPad view.



Both values would still be in same quadrant, split again

Figure 4: Example of a JSAV visualization on iPad which uses graphical primitives and Google Maps.

4. DISCUSSION

The development of JSAV began at the same time as the OpenDSA project. OpenDSA's goal is to build a complete open-source, online eTextbook for DSA courses, that integrates textbook quality text with algorithm visualizations and a rich collection of interactive exercises. All exercises are assessed automatically. As a result, students gain far more practice by working on OpenDSA exercises than is possible with normal paper textbooks.

Since JSAV is the framework for building visualizations within OpenDSA, there already exists a significant set of visualizations built with JSAV. OpenDSA already has complete chapters on sorting and hashing[6]. These chapters include over thirty AVs ranging from static images to proficiency exercises. By Fall 2013, a full semester DSA course will have been developed for OpenDSA using JSAV.

Many JSAV visualizations have already been used by students in large courses. In Spring 2012 around 80 computer science majors used four binary heap proficiency exercises[7] in a DSA course at Aalto University. Results of the students were in line with results from previous years using TRAKLA2. Furthermore, the student opinions were highly positive with negative comments on some technical bugs that have since then been fixed. More details can be found in Karavirta *et al.* [7].

OpenDSA was used to replace three weeks worth of standard lecture materials on sorting and hashing during October 2012 by around 60 students in a DSA course at Virginia Tech. Students in the treatment section using OpenDSA scored slightly higher on the resulting exam than the control group, but not significantly so. Student evaluations were highly enthusiastic, with mean scores on preference for interactive online tutorials over standard lecture going up after actual use of the materials. More details on this evaluation for the OpenDSA project can be found in Hall *et al.* [3].

Most existing visualizations in OpenDSA are on sorting and hashing, which can be considered simple topics to visualize. The majority of the JSAV AVs created thus far use only the array data structure. The Quicksort AV shows the hierarchical decomposition of the array partitionings, and linked lists are used in both the Radix Sort visualizations and the Open Hashing visualization. A few AVs on more complex topics have been implemented, such as Huffman coding and dynamic programming. Furthermore, there are AVs on spatial algorithms (namely, the PR QuadTree and the kd-Tree) which visualize both a tree data structure and an area using a map [6]. These AVs use the location of the student to initialize the exercise with local data point and they use Google Maps to show a map (see Figure 4 for an example). Thus, they give a good indication on the flexibility of JSAV as well as the ease of which it can be integrated with other web technologies and libraries. Given this body of existing content, we are certain that JSAV is suitable for visualizing even more complex algorithms.

The main obstacle in creating more AVs with JSAV is the learning curve for the library. With the existing Java AV systems, most potential AV developers were already familiar with the Java language and needed only to learn an API for the specific AV system. With JSAV being HTML5/JavaScript based, a potential AV developer will need to first learn JavaScript, HTML5, and CSS to be able to take full advantage of the library. This can currently put off some devel-

[6]This material is available from `http://algoviz.org/OpenDSA/`.

[7]The exercises were heap insert, heap delete, build-heap algorithm, and heapsort.

opers, but we argue that as web technologies further gain in popularity, more and more educators will know or be motivated to learn the technologies needed. Roughly a dozen graduate and undergraduate students have successfully implemented JSAV-based AVs or exercises, mostly as part of independent study courses or as volunteers.

5. CONCLUSIONS

The major contribution of this paper is the introduction of a new educationally oriented algorithm visualization framework called JSAV. JSAV has been built based on years of research on developing both AVs and AV systems, and combines features of several well-known AV systems. We have explained the main features of JSAV, and discussed existing materials and early experiences with using the materials in teaching. JSAV is the first AV development system implemented in HTML5/JavaScript with support for both animated slideshows and engaging automatically assessed exercises. The collection of visualizations and exercises is growing continuously. We hope that more teachers will use the material and join the community effort to continue development of the framework and the materials.

6. REFERENCES

[1] S. Diehl. *Software Visualization*. Number 2269 in Lecture Notes in Computer Science. Springer, 2002.

[2] E. Fouh, M. Akbar, and C.A. Shaffer. The role of visualization in computer science education. *Computers in the Schools*, 29:95–117, 2012.

[3] S. Hall, E. Fouh, D. Breakiron, M. Elshehaly, and C.A. Shaffer. Education innovation for data structures and algorithms courses. In *Proceedings of ASEE Annual Conference*, Atlanta GA, June 2013.

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

[5] V. Karavirta. Seamless merging of hypertext and algorithm animation. *ACM Transactions on Computing Education*, 9(2):1–18, 2009.

[6] V. Karavirta. Location-aware mobile learning of spatial algorithms. In *Proceedings of the IADIS International Conference on Mobile Learning 2013*, pages 158–162, Lisbon, Portugal, March 2013.

[7] V. Karavirta, A. Korhonen, and O. Seppälä. Misconceptions in visual algorithm simulation revisited: On UI's effect on student performance, attitudes, and misconceptions. In *Proceedings of Learning and Teaching in Computing and Engineering*, Macau, 2013.

[8] A. Korhonen. *Visual Algorithm Simulation*. Doctoral dissertation (tech rep. no. tko-a40/03), Helsinki University of Technology, 2003.

[9] M. Krebs, T. Lauer, T. Ottmann, and S. Trahasch. Student-built algorithm visualizations for assessment: flexible generation, feedback and grading. In *Proceedings of the 10th annual SIGCSE conference on Innovation and technology in computer science education*, pages 281–285, New York, NY, USA, 2005.

[10] 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, September 2004.

[11] T.L. Naps. Jhavé: Supporting algorithm visualization. *IEEE Computer Graphics and Applications*, 25:49–55, September 2005.

[12] T.L. Naps, G. Rössling, V. Almstrum, W. Dann, R. Fleischer, C. Hundhausen, A. Korhonen, L. Malmi, M. McNally, S. Rodger, and J.Á. Velázquez-Iturbide. Exploring the role of visualization and engagement in computer science education. In *Working Group Reports from ITiCSE on Innovation and Technology in Computer Science Education*, pages 131–152, 2002.

[13] R.J. Ross and M.T. Grinder. Hypertextbooks: Animated, active learning, comprehensive teaching and learning resources for the web. In S. Diehl, editor, *Software Visualization*, number 2269 in Lecture Notes in Computer Science, pages 269–284. Springer, 2002.

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

[15] G. Rößling, T. Naps, M.S. Hall, V. Karavirta, A. Kerren, C. Leska, A. Moreno, R. Oechsle, S.H. Rodger, J. Urquiza-Fuentes, and J.Á. Velázquez-Iturbide. Merging interactive visualizations with hypertextbooks and course management. In *Working Group Reports from ITiCSE on Innovation and Technology in Computer Science Education*, pages 166–181, 2006.

[16] G. Rößling and T.L. Naps. A testbed for pedagogical requirements in algorithm visualizations. In *Proceedings of the 7th annual conference on Innovation and technology in computer science education*, pages 96–100, New York, NY, USA, 2002.

[17] G. Rößling and T.L. Naps. Towards intelligent tutoring in algorithm visualization. In *Proceedings of the 2nd International Program Visualization Workshop*, pages 125–130, Aarhus, Denmark, 2002.

[18] G. Rößling and T. Vellaramkalayil. First steps towards a visualization-based computer science hypertextbook as a Moodle module. In *Proceedings of the 5th Program Visualization Workshop*, volume 224 of *Electronic Notes in Theoretical Computer Science*, pages 47–56, 2009.

[19] C.A. Shaffer, M. Akbar, A.J.D. Alon, M. Stewart, and S.H. Edwards. Getting algorithm visualizations into the classroom. In *Proceedings of the 42nd ACM Technical Symposium on Computer Science Education (SIGCSE'11)*, pages 129–134, 2011.

[20] C.A. Shaffer, M.L. Cooper, A.J.D. Alon, M. Akbar, M. Stewart, S. Ponce, and S.H. Edwards. Algorithm visualization: The state of the field. *ACM Transactions on Computing Education*, 10:1–22, August 2010.

[21] C.A. Shaffer, V. Karavirta, A. Korhonen, and T.L. Naps. OpenDSA: Beginning a community hypertextbook project. In *Proceedings of the 11th Koli Calling International Conference on Computing Education Research*, pages 112–117, Koli National Park, Finland, November 2011.