Creating Engaging Online Learning Material with the JSAV JavaScript Algorithm Visualization Library

Ville Karavirta and Clifford A. Shaffer

Abstract—Data Structures and Algorithms are a central part of Computer Science. Due to their abstract and dynamic nature, they are a difficult topic to learn for many students. To alleviate these learning difficulties, instructors have turned to algorithm visualizations (AV) and AV systems. Research has shown that especially engaging AVs can have an impact on student learning of DSA topics. Until recently, most AV systems were Java-based systems. But, the popularity of Java has declined and is being supplanted by HTML5 and JavaScript content online. In this paper, we present JSAV: the JavaScript AV development library. JSAV goes beyond traditional AV library support for displaying standard data structures components, to provide functionality to simplify creation of AVs on many engagement levels including interactive exercises. We describe the growing body of content created with JSAV and summarize our three years of experience and research results from using JSAV to build content that supports CS education.

Index Terms—Data structure and algorithm visualizations, algorithm animation, interactive courseware, HTML5, active electronic textbooks, hypertextbook, JSAV

1 Introduction

Computer Science instructors have a long tradition of using Algorithm Visualization (AV) [1], [2], [3] to convey dynamic aspects of algorithmic process. Hundreds of AVs have been created over the years [2]. There is also a history of meaningful interactive exercises for data structures and algorithms (DSA) topics, in the form of algorithm simulation or algorithm proficiency exercises as pioneered by the TRAKLA2 system [4]. But until recently, putting AVs, interactive exercises, and content together on a large scale such as in the form of a complete semester eTextbook has not been feasible due to technological limitations. Prior to the advent of Java in the mid 1990s, AV systems were highly fragile, each able to run on only a limited range of computers. Java permitted an explosion of AVs and AV systems [2]. But Java applets never really meshed well with content in their containing HTML pages, nor have different browsers presented a consistent environment for Java implementations. Java applets are increasingly difficult to present on web pages due to security concerns. As an alternative, Adobe Flash was often used to provide advertisements and streaming media. However, few AVs have been developed in Flash.

Modern web browsers with their relatively consistent support for HTML, many JavaScript APIs, and advanced CSS styling make it easier than ever before to develop interactive content for the web. Thus, HTML5 (as the combination of modern HTML, JavaScript, and CSS is often called) has become an Internet standard. Developers now expect content to run on any user's computer. HTML5 even runs on most tablets and many mobile devices. Modern web technologies include all necessary computational, graphical, and interface support to create sophisticated visualizations. For the first time, there is enough consistency between browsers and a maturity of tools to make it practical to combine AVs, automatically assessed exercises, and tutorial content together at meaningful scale.

In this paper we present JSAV: The JavaScript Algorithm Visualization Library. JSAV is written in JavaScript, and is meant to support development of engaging AVs for online learning material. 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) With the inevitable decline of Java, an HTML5-based solution is needed. There is little existing support for AV development in JavaScript. At this time there are few other major AV development efforts in HTML5/JavaScript, all of which are missing some key features needed by effective AVs.
2) JSAV incorporates 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 well as support for embedding AVs within a tutorial.

JSAV reflects the collective experience of three major AV development groups: Aalto University (the developers of TRAKLA2 [4]), Virginia Tech, and the JHAVÉ community [5]. Each group has written many widely used AVs,

• *V. Karavirta is with the Department of Information Technology, University of Turku, Helsinki, Finland. E-mail: ville@villekaravirta.com.*
• *C.A. Shaffer is with the Department of Computer Science, Virginia Tech, Blacksburg, VA 24061. E-mail: shaffer@vt.edu.*

Manuscript received 10 Mar. 2015; revised 3 Oct. 2015; accepted 7 Oct. 2015. Date of publication 14 Oct. 2015; date of current version 22 June 2016.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TLT.2015.2490673

and the developers' differing perspectives have ensured that JSAV is able to support the needs of a broad community within a development environment (HTML5) that will prove 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 [6], [7], [8], [9], which seeks to provide a complete open-source resource for teaching Data Structures and Algorithms courses.

An earlier version of this paper appeared as [10]. Since that article, JSAV has evolved, and has been used to develop a wide variety of visualizations and interactive exercises. As of this writing, the resulting materials are being used by hundreds of students every semester at about a dozen institutions, with total views of visualizations and interactive exercises numbering in the hundreds of thousands.

The rest of this paper is organized as follows. Section 2 presents some background, introduces requirements for AV systems, and evaluates existing AV systems. Section 3 presents JSAV in detail. Section 4 summarizes three years of experience with using JSAV. Section 5 discusses our contributions and Section 6 presents our conclusions.

2 Related Work

Since there is more previous research on AVs than can fit into the pages of this article, we will focus here on the research themes of recent years, the general requirements for an AV system, and an evaluation of existing systems. Overviews for the history and current state of AV research can be found in [2], [3], [11].

We should note, as does [11], the difference between algorithm visualization and program visualization (PV). Algorithm visualization is the visual presentation for the behavior of an algorithm (perhaps expressed in a particular programming language). As such, it is tutorial in nature. Typically AVs require good quality support for precise control of the presentation, as detailed later in this section. Program visualization in contrast is the automated generation of visual feedback for a program, in particular with minimal intervention by the programmer. Its goal is to provide insight into the behavior of the program, including for such tasks as debugging. By its nature (that is, minimal intervention by the programmer, and thus automatic generation of the visual feedback), detailed control of the visual elements is not a requirement of the tool. Rather, PV tools must focus on aspects such as parsing and interpreting the program code, which is not at all a concern of an AV system. Two popular PV tools used for educational purposes are jGrasp [12] and Python Tutor [13], neither of which support features required by AV systems.

2.1 Algorithm Visualization Research Trends

One 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 [14]. 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 term Visualization-based Computer Science Hypertextbook (*VizCoSH*). Another working group in 2013 used the term *icseBook* to refer to interactive computer science electronic books [16]. Their report includes discussion and guidelines for both pedagogical and technical issues in authoring and using such eBooks.

One attempt to merge visualization with tutorial content integrates the Animal visualization system with hypertext as a Moodle module [17]. 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 [18]. Currently, the largest such project is OpenDSA [6], [7], [8], [9]. OpenDSA uses the term *active eBook* to refer to a merging of content, AVs, and exercises with automated assessment.

Increasing evidence confirms the hypothesis that student engagement is the key to educational effectiveness of AVs [1], [19], [20]. The different types of engagement encountered in AVs have been categorized in the *Engagement Taxonomy* [1]. The taxonomy defines five levels of interaction between a student and an AV:

- *Viewing:* Student passively views an AV, perhaps with the ability to control the animation speed or move step-by-step backward and forward.
- *Responding:* Student responds 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.
- *Changing:* Student changes the AV by, for example, providing input data to the algorithm.
- *Constructing:* Student constructs an AV. A variation on this approach gives a data structure and an algorithm, and expects the student to simulate the algorithm. That is, they need to imitate the steps of an algorithm by manipulating the interface to control the progress of the AV.
- *Presenting:* Student presents an AV 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 requirement of an AV system.

2.2 Requirements for an AV System

There have been attempts to define requirements for an AV system. The majority of this work has been done by Rößling and Naps [21], [22] with further requirements introduced by others [15], [23], [24]. We have summarized these requirements in Table 1. However, since most of these efforts are over a decade old and predate HTML5, we have redefined them to better reflect the current state of AV systems. This updates a similar list introduced by the first author [18]. We can map engagement taxonomy levels to the requirements. Requirements R2, R3, and R5 can increase the control of *viewing*. Level *responding* matches R7, and *changing* matches R9. Level *constructing* is part of requirement R11.

TABLE 1
Requirements for an AV System, Expanded from [18]

Req	Description of the Requirement
R1	The system's platform should allow the widest possible target audience.
R2	System should support visualization rewinding so that users can return to the place where they lost track of the content.
R3	Learners should be able to adapt the display to their current environment. This includes the choice of display background color to account for lighting situations, transition speed, display magnification.
R4	System should be general-purpose instead of topic-specific, allowing for greater reuse and integration into a given course. General-purpose systems allow a common interface to a large number of animations.
R5	User should be offered the choice between smooth animation transitions and discrete steps.
R6	Visualizations should include accompanying prose. Examples include static material explaining the concept, dynamic explanations aware of the algorithm's state, and pedagogically dynamic explanations that are aware of learner knowledge.
R7	Asking questions about the algorithm's behavior in following states should be supported. Questions should incorporate feedback.
R8	System should support communication with a database for course management facilities. The database can then be used for example to store the points received by answering questions.
R9	System should allow users to provide custom input to the algorithm.
R10	Visualizations should give structural view of an algorithm's main parts, that can be used to jump to associated visualization steps.
R11	System should support visualization construction exercises with automated feedback.
R12	System should include reusable visualization modules, thus aiding in the authoring of future AVs.
R13	It should be possible to easily merge the visualizations with hypertext.
R14	Visualizations should be localizable in two ways: 1) match the written language of the surrounding hypertext and 2) match the programming language of the surrounding hypertext.
R15	System should be able to import/export visualizations in different formats.
R16	Learners should be able to see previous steps of the visualization simultaneously with the current step.

Next we evaluate existing AV systems against these requirements. We focus on HTML5-based systems, as those are in our opinion the technological future of the field. However, we include several of the best-known Java-based AV systems, as these have heavily influenced the design of JSAV.

2.3 Existing Algorithm Visualization Systems

A necessary requirement for any AV system aspiring to wide-spread use is that it should support as wide a target audience as possible. Since the mid-1990s, this has meant implementing the system in Java, which led to Java being the most common implementation technology for AVs until recently [2]. Since the widespread adoption of HTML5 beginning around 2010, combined with recent security concerns and reduced support for Java within browsers, the importance of Java on the web is continually diminishing. In contrast, HTML5 and JavaScript are rapidly gaining in popularity. Thus, in the past few years, HTML5-based AV systems have been developed. Here, we briefly introduce several systems that we are familiar with, and evaluate them based on the requirements presented in Table 1.

ANIMAL [25] provides several different ways to create AVs. It has a graphical editor, a scripting language, and customizable content generators for many algorithms. It supports both graphical primitives and data structures. AVs can include popup-questions about the content, as well as allow custom input from users.

JAWAA [26] is a language for writing animations as well as a system for visualizing those animations. Animations can include graphical primitives, as well as some data structures.

JHAVÉ [5] provides support for multiple visualizers for different animation languages. AVs are mainly built using data structures, and can include popup-questions.

TRAKLA2 [4] is a learning environment that includes algorithm simulation exercises. The exercises require students to construct an AV by simulating the steps for an algorithm. The system supports multiple data structures, has several exercises included, automatically assesses student solutions, and gives visual feedback.

JSXaal [18] was the first JavaScript-based AV system we are aware of. It is a visualizer for AVs written in the XAAL algorithm animation language. While there are several tools available for creating such animations, the viewer has never seen significant use.

Galles has created a comprehensive collection of AVs in JavaScript.[1] It includes AVs where students can provide their own input and see an animation on how the data structure is updated. Some data structures can be manipulated interactively by the user, such as through buttons for operations such as *"Remove Smallest"* for a minimum heap. The sorting AVs allow a student to run the algorithm on a random set of data.

While a system for visualizing algorithms and data structures, Vamonos has features more common in PV systems, such as breakpoints and watch points. The online collection[2] includes sorting and graph algorithms. Currently supported data structures are array and graph.

The Algomation website[3] has an extensive collection of ready-made visualizations on a range of topics. The system includes visualizers for arrays, trees, heaps, and graphs.

VisuAlgo [27] is an online collection of AVs. Users can learn algorithms by seeing their execution (in case of, for example, sorting algorithms) as a slideshow, or by exploring operations on a data structure (for example, a binary heap).

1. http://www.cs.usfca.edu/galles/visualization
2. http://rosulek.github.io/vamonos/demos/index.html
3. http://www.algomation.com/

TABLE 2
Evaluation of Existing AV Systems (x Feature Supported, (x) Partially Supported, and—not Supported)

Req	Animal	JAWAA	Jhavé	TRAKLA2	Vamonos	Galles	Algomation	JSXaal	VisuAlgo	JSAV
R1	(x)	(x)	(x)	(x)	x	x	x	x	x	x
R2	x	(x)	x	x	x	x	x	x	x	x
R3	x	-	(x)	-	-	x	(x)	-	(x)	(x)
R4	x	x	-	-	-	x	x	x	-	x
R5	x	-	-	-	(x)	x	x	x	(x)	x
R6	(x)	-	(x)	(x)	-	(x)	(x)	(x)	(x)	(x)
R7	x	-	x	-	x	-	-	x	(x)	x
R8	x	-	x	x	-	-	-	x	x	x
R9	x	-	x	-	x	x	-	x	x	(x)
R10	x	-	-	-	-	-	-	(x)	-	-
R11	-	-	-	x	-	-	-	-	-	x
R12	x	x	x	x	x	x	x	-	x	x
R13	(x)	(x)	(x)	(x)	x	x	x	x	x	x
R14	(x)	-	-	(x)	-	-	-	(x)	(x)	x
R15	x	(x)	(x)	-	-	-	-	x	-	(x)
R16	-	-	-	-	-	-	-	x	-	-

An interesting feature of the site is the online quiz system, which generates questions about the content for the learner to answer. In many of the questions, the answer is provided by direct interaction with a data structure. The data structures supported include arrays, trees, graphs, and lists.

An evaluation of these systems against the requirements of Table 1 is summarized in Table 2. We have included an evaluation for JSAV. The evaluation has three possible values: supported (marked with x), partially supported (x in parentheses), and not supported (-). "Partially supported" is used for systems that fulfill only some aspects of the requirement. While space restrictions make it impossible to go through the evaluations in detail, we give some examples. Requirement R13 is easy merging with hypertext. Although this is possible with the Java-based systems, the merging is superficial at best compared to the possibilities of JavaScript-based systems. Similarly, while Java used to be the widest possible platform (R1), it is not supported by most mobile devices and no longer comes with all major desktop operating systems. Thus, we evaluate Java-based systems as partially supporting R1. For R6, all systems lack dynamic explanations that adapt to the learner's knowledge level. For R3, systems with (x) in their evaluation are missing some of the display adaptation features listed in the requirement. For R14, most systems lack support for localizing (or changing) the programming language used in the visualization. Finally, for R15, systems with partial support have limited ability to import or export anything beyond their native format.

We do *not* imply that a system fulfilling more requirements is better than one with less requirements fulfilled. Some requirements are clearly more valuable than others, and different systems have different goals and primary use cases. So the table first provides an overview of the features supported by representative systems, from which we might derive further analysis. We note that few systems include support for interactive exercises, even though AV research on engagement implies that this feature is particularly important. On the other hand, few systems support R10 or R16, which probably indicates that most developers do not consider these of importance. We see that learner control (requirements R2 and R5) is typically present in the systems, as is dynamic documentation (R6). Naturally, HTML5-based systems support the widest possible platform (R1) and are easily merged with HTML learning material (R13).

3 JSAV

In this section we introduce the JSAV library. We start by explaining the origins of the project, and then present JSAV's main features.

3.1 JSAV Design Origins

The JSAV design was initially driven by several aspects:

- Desire to support HTML5-based AVs.
- Existing knowledge and experiences of the authors and immediate collaborators from using and developing previous AV systems.
- Careful examination of an extensive catalogue of AV systems and AVs maintained by one of the authors [2], [28].
- Desire to support interactive proficiency exercises such as those in the TRAKLA2 system [4].

From this starting point, the first versions of the APIs were developed during Summer and Fall 2011. The first items created were a Shellsort visualization and a Shellsort proficiency exercise now available in OpenDSA. The first refactoring of material occurred when we realized that the best approach to presenting an initial tutorial on an algorithm (as opposed to a summary AV) involved a static example presented through a series of slideshows interspersed with textual description. This lead to our first full-fledged tutorial module in the OpenDSA system for Shellsort, that included a presentation of the algorithm (including a series of small slideshows), a full AV for the algorithm (allowing either random input or user-specified input to drive the visualization), and a proficiency exercise. This defined a number of major features for the JSAV system, including integration of JavaScript-based visualizations with HTML, and fundamental support for proficiency exercises.

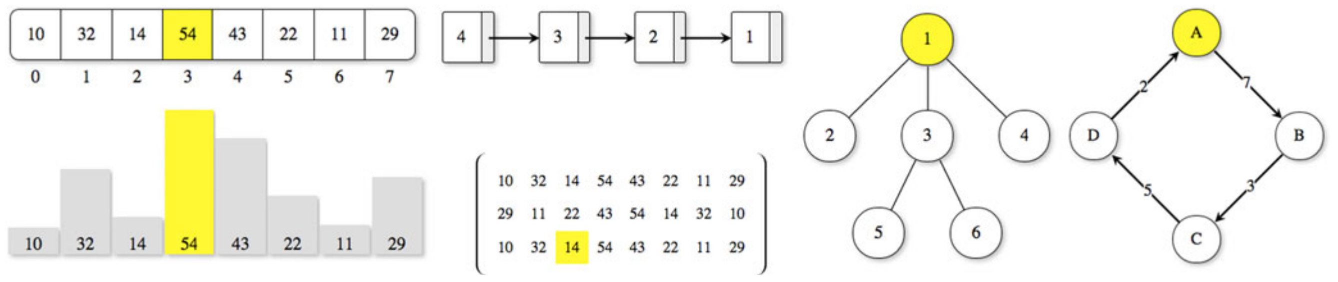


Fig. 1. Default visualizations of JSAV data structures: array and bar layout for an array, 2D matrix, linked list, tree, and graphs (directed and undirected).

The demands of a wide range of new AVs for the OpenDSA project had a major impact on further developments for the JSAV APIs and features. This iterative design and implementation process has ensured that not only does JSAV fulfill the requirements based on literature, it also really works for a wide group of AV developers.

3.2 Main Features of 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 that can report outcomes to an external system such as a learning management system. In the following, we present what we believe to be JSAV's most innovative features.

Levels of engagement. JSAV supports different types of AVs 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.

Slideshows 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 to the end (AV system requirement R2). They can change the speed of transition animations (R3). Setting the speed to the fastest available causes the animation to switch from smooth transitions to discrete steps (R5).

While important as visual aides to accompany tutorial presentations, both static images and slideshows represent only the lowest level (viewing) of the engagement taxonomy. They do not represent the most effective use of AVs. A simple way to increase engagement is to allow students to enter the input data for the algorithm (requirement R9), or to have the example be generated at random (such as random values in an array to be sorted). Both can be created with JSAV.

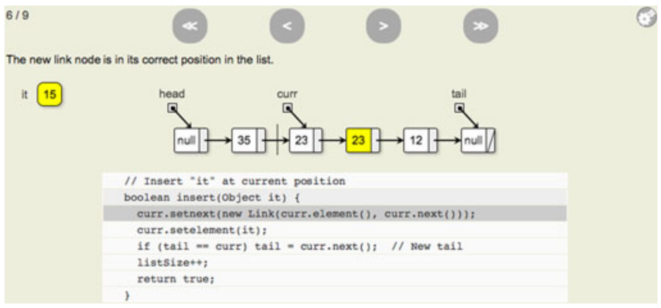
A more significant interaction is to require students to answer questions. JSAV supports pop-up questions in slideshows (requirement R7). These require the student to *respond* during the AV. JSAV supports questions that can be either true/false, multiple-choice, or multiple-select questions. A question is shown as a pop-up 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. Another alternative is to combine JSAV with a library designed to support question types. We have successfully integrated JSAV displays with the Khan Academy (KA) exercise infrastructure,[4] thereby gaining access to the ability to create a wide range of interactive exercises that can allow students to directly manipulate a data structure 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. Simulation of the algorithm is done by clicking directly on the visual representation of a data structure or on user interface buttons. This type of interaction is aided by JSAV's support for binding developer-defined click handlers to JSAV data structures (arrays, trees, etc.). JSAV supports versatile user feedback modes for proficiency exercises, as will be explained later.

Visual components. On all levels of engagement, AVs can be composed from the same selection of objects. The building blocks used in creating JSAV visualizations are similar to those of many existing AV systems. There are essentially three types of objects: *data structures*, *graphical primitives*, and *code constructs*. The graphical primitives make JSAV a general system (requirement R4), whereas the data structures simplify the visual presentation of algorithm-specific content.

The data structures supported are array, (2D) matrix, linked list, (general) tree, binary tree, and graphs. Examples are shown in Fig. 1. "Supported" means that the developer declares an object of the given type, perhaps specifies its placement and some visual characteristics, and JSAV will then handle layout of the object. 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 methods to change the visual appearance for parts of a structure. Calls to operations that change the state or visual appearance of an object are recorded and can be undone and redone by the user in

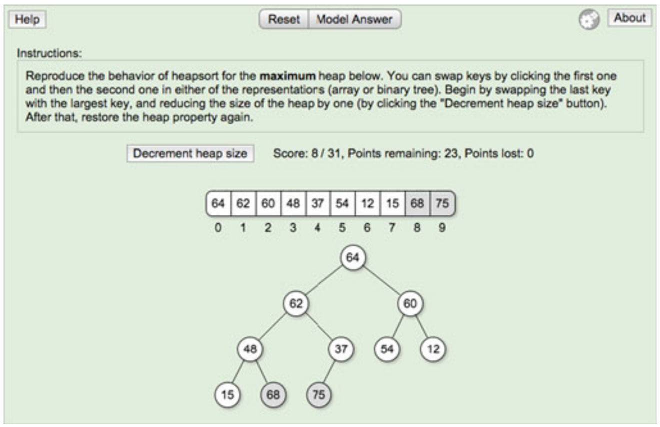
4. https://github.com/Khan/khan-exercises



The new link node is in its correct position in the list.

```
// Insert "it" at current position
boolean insert(Object it) {
  curr.setnext(new Link(curr.element(), curr.next()));
  curr.setelement(it);
  if (tail == curr) tail = curr.next();  // New tail
  listSize++;
  return true;
}
```

Fig. 2. A visualization with a pseudocode component and pointer objects.



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: 8 / 31, Points remaining: 23, Points lost: 0

Fig. 3. A proficiency exercise for Heapsort. Important components include instructions, scoring feedback, and the interactive heap object as user interface.

any slideshow. Default appearance for all visual elements is specified using CSS, and so can be overridden by the developer. We have also built a number of derived data structures from the JSAV basic collection, which help to demonstrate their power. These include the heap (which slaves together an array view and a tree view for the data) and the "array node" tree used to support structures with more complex node types, like the 2-3 tree and the B-tree.

The graphical primitives supported are text, line, circle, ellipse, rectangle, polygon, polyline, and general path. The general path allows drawing arbitrary shapes made of lines and curves.[5] 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. Objects can be scaled, rotated, and moved. Again, all changes are recorded to the animation.

To tie a visualization to code for the corresponding algorithm, JSAV supports visualizing pseudocode, variables, and pointers. The pseudocode element can be given a piece of code as a string, or it can read the code from a file, and display it as shown in Fig. 2. The code object has methods for highlighting specified lines, and can indicate the previously highlighted line when the currently highlighted line is changed. Lines of code can be assigned tags, with highlighting targeted to those tags. This makes it possible to have code in different programming languages and have the same JSAV visualization work for all of them (part of requirement R14). Variable objects can be used to track and visualize variable values in an algorithm. Finally, pointers can be used to represent variables that point to a part of a data structure. The pointer is visualized with the variable name and an arrow pointing to the target structure (see the head, curr, and tail pointers in Fig. 2). For example, a tree traversal algorithm could use a pointer to mark the current node in the algorithm.

For all 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, 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.

Proficiency exercises. One of JSAV's most innovative aspects is its support for proficiency exercises, requiring 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 AV, within constraints that the exercise developer has provided to allow student control. This student AV can be automatically assessed by comparing it to a model solution. The model solution is also available for the student to view as a JSAV slideshow. Fig. 3 shows an example of the Heapsort exercise.

JSAV proficiency exercises provide three modes for providing feedback to students. 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 an incorrect step. If a step is incorrect (that is, it does not match the corresponding step in the model answer), then it is undone and the student can try that step again.
- *Continuous feedback with incorrect steps fixed:*Again, feedback is given after each operation that 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's implementation to first generate random input data for the exercise and initialize the data structures used. The exercise must also generate the model solution, similarly to how JSAV slideshows are written. The model solution is annotated to mark the steps which are to be graded. Finally, the author needs to add the student interaction interface, that is, attach the needed event handlers to the data structures so that changes are made when the student interacts with them. The 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.

Localization. An AV system should support internationalization (R14). Currently, JSAV's user interface is available in English, Finnish, Swedish, and French. A URL

5. JSAV can draw anything that is possible with the SVG path element (http://www.w3.org/TR/SVG/paths.html#PathData).

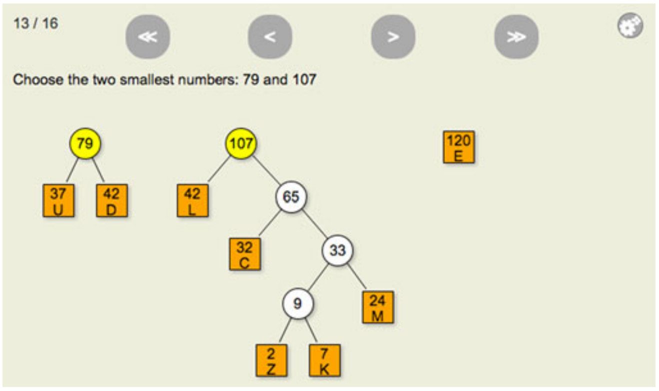


Fig. 4. Example of node appearance customization in the Huffman coding tree slideshow. Leaf nodes have been assigned one style, internal nodes another, and the roots of the first two trees have been temporarily colored to signify that the algorithm is currently acting on them.

parameter to the AV determines the language to be used at run time. To support another language based on a western-style alphabet requires only that the proper set of strings be supplied for the system to integrate into the user interface at run time. JSAV also supports localizing AVs by providing an interpreter function. The interpreter works like localization packages typically do: the content uses keys to access localized strings, which are provided by translation files.

Another form of "localization" allows developers to configure a JSAV AV or proficiency exercise to switch the programming language used at runtime. The code to be displayed is typically read from a file, and specific lines can be highlighted at various points in the visualization. To switch the programming language displayed requires only that the filename from which the code is read be changed, and that logical tags be assigned for the code lines to be highlighted.

In OpenDSA, written language localization is controlled by means of a JSON file that contains the displayed strings to be used for each written language for which a translation of that specific AV is available. Likewise, a section of the JSON file defines for each programming language to be supported the file containing the program display code, and the binding of logical tags to line numbers in the displayed code.

3.3 Technology

JSAV's user interface is HTML with the functionality implemented in JavaScript and the appearance specified with CSS. Thus, the platform is the one that today has the widest target audience (requirement R1). JSAV also takes advantage of some existing, high quality JavaScript libraries. It uses jQuery to help in resolving differences in browsers and working with the DOM. jQueryUI[6] implements animation effects and element positioning. Raphaël[7] eases using SVG to display and animate changes to the graphical primitives. MathJax[8] is used to render mathematics specified in LaTeX format.

Officially, JSAV supports all modern versions of Chrome, Firefox, and Safari browsers. For each of these, JSAV has been extensively tested and used by students. Furthermore, JSAV should work in modern IEs and Opera as well as Mobile Safari on iOS and the Chrome browser on Android 4 and above. JSAV is open source and released under the MIT license. Source code is available from GitHub.[9]

3.4 Integrating with Tutorial/Explanatory Content

Content created with JSAV can be standalone and does not need to be embedded into any other material. This is aided by the fact that each AV can have static learning material as well as explanations on each step in the AV (requirement R6). However, JSAV-based content can be embedded into electronic courseware such as an eTextbook. The underlying web technologies make integrating JSAV AVs within hypertext simple and flexible. This fulfills requirement R13. AVs can be directly included into an HTML page using HTML `div` elements, with specific class names for placement and including JavaScript and CSS files to define the AV. Alternatively, AVs and proficiency exercises can be embedded from standalone HTML pages using the HTML `iframe` element. In this case, the containing page (such as an eTextbook containing a JSAV exercise) can pass parameters to the exercises (through the URL) to control features such as which written language to use, which coding language to use for code snippets, which grading option to use (see Section 3.2), or which version of the exercise to show (such as whether to require a user to make a single or double rotation in an AVL tree rotation exercises).

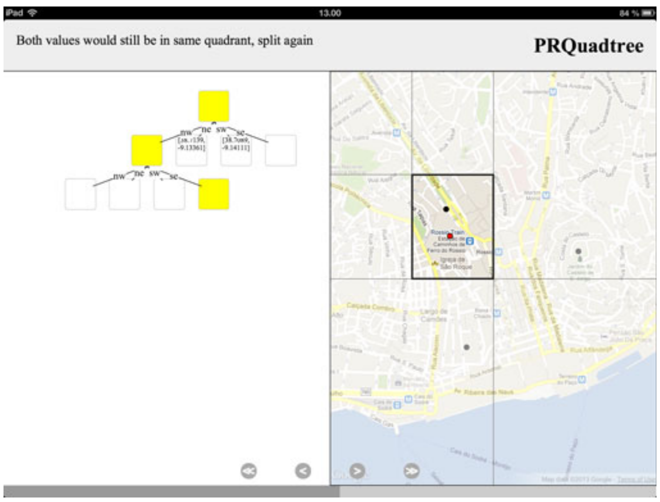
3.5 Extending and Customizing

HTML5 technology makes customizing JSAV easy and flexible. By changing CSS styles, the AV's visual appearance can be changed. JSAV aids CSS styling by using hierarchic HTML `class` attributes to style objects. For example, all nodes in lists, trees, and graphs have CSS class `jsavnode`. They also have more specific classes. Binary tree nodes have additional classes `jsavtreenode` and `jsavbinarynode`. All JSAV visual elements support adding or removing developer-defined CSS classes. This makes it easy to define logical roles and separately define their physical appearance. For example, a developer can assign a given tree node or array cell the "processed" role, and define in CSS the physical representation (e.g., node color, border thickness) for "processed" objects. 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. Fig. 4 shows an example of such customization in the Huffman coding tree slideshow.

Customizing and extending behavior is made easy by the dynamic nature of JavaScript. As JSAV exposes the types used for its visual components, adding or changing functionality is straightforward. For example, a visualization of the BST insert operation could add a function for the BST insert to the core JSAV *binary tree* object definition without changing JSAV's source code. Thus core

6. https://jqueryui.com/
7. http://raphaeljs.com/
8. https://www.mathjax.org/

9. http://github.com/vkaravir/JSAV/



Both values would still be in same quadrant, split again

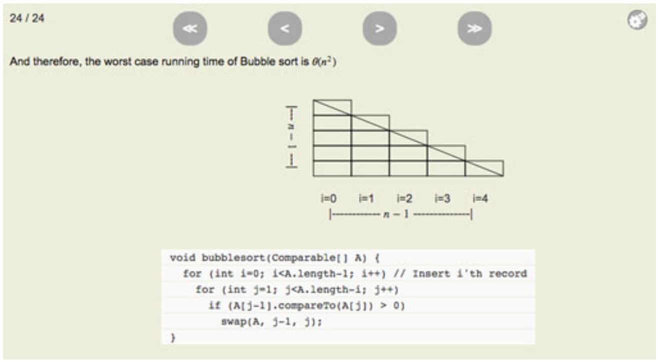
Fig. 5. A JSAV visualization on an iPad with a customized UI using graphical primitives and Google Maps.

functionality can be added outside of JSAV. The heap data structure (see Fig. 3) and the Huffman coding tree (Fig. 4) are examples of new data structures, with new APIs, created in this way. Both of these APIs are built from base JSAV functionality, but separate from the JSAV library itself. To the AV developer, there is no difference between using a “core” JSAV data structure and using an add-on data structure. 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 eTextbook. Thus JSAV fulfills requirement R12: reusable visualization modules.

JavaScript events can be used to change or control JSAV functionality. JSAV triggers events whenever a user interacts with content, and these events can be reported to the containing environment. For example, both successfully and unsuccessfully completing steps in a proficiency exercise exposes scoring events. OpenDSA uses these events to change displayed student progress in HTML surrounding the AV. Likewise, all user interaction events can be reported to the containing environment, for example to generate detailed user interaction logs. This is also how the learning management system integrations described in Section 4.2 track student progress. JSAV also 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 Fig. 5. Here, the events have been used to add AV controls and a progress bar to the bottom of the iPad view.

3.6 Documentation

JSAV documentation can be found at http://jsav.io/. The documentation presents most of the APIs in detail. Furthermore, it includes many live examples with JavaScript code and CSS styles for the example. This, combined with the tutorials on getting started and creating exercises provides a good starting point for learning JSAV.



And therefore, the worst case running time of Bubble sort is $\theta(n^2)$

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

Fig. 6. A visual explanation for the running time analysis of Bubble sort.

4 JSAV Content and Use

Since 2011, JSAV has been used to create a wide variety of content, integrated into different eLearning environments, and used in many courses and institutions.

4.1 Existing Content

JSAV was the first infrastructure component in 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.

OpenDSA provided motivation to create a significant set of AVs with JSAV. OpenDSA currently has over 200 AVs and over 100 exercises that use JSAV. The topics cover a wide range, with chapters on algorithm analysis, linear structures, binary trees, sorting, and hashing considered to be “complete” and polished. In addition, many more modules have been prototyped and used in at least one course, including basic graph algorithms, memory management, two-three trees, the Union/Find algorithm, tree serialization, buffer pools, string matching algorithms, dynamic programming, NP-Completeness, and a detailed introductory tutorial on recursion. Materials for a programming languages course are in development. The widely used JFLAP project,[10] consisting of a large body of finite state automata simulations, is being re-implemented in JSAV.

While OpenDSA includes most “typical” AV content for arrays, trees, linked lists, and graph algorithms, OpenDSA includes many AVs not commonly available. This include visualizations for analysis of algorithms and NP-completeness. Figs. 6 and 7 show examples of such visualizations.

As explained in Section 3.5, JSAV supports extensions to the data structure implementations provided in the core library. As part of OpenDSA, several such extensions have been developed. These include the AVL tree, red black tree, 2-3 tree, binary heap, and Huffman coding tree. Of particular note is the array tree, where JSAV tree nodes are defined using a JSAV array. The structure is useful for visualizing the 2-3 tree, B tree, and B+ tree. Fig. 8 shows an example illustrating 2-3 Tree insertion.

10. http://www.jflap.org/

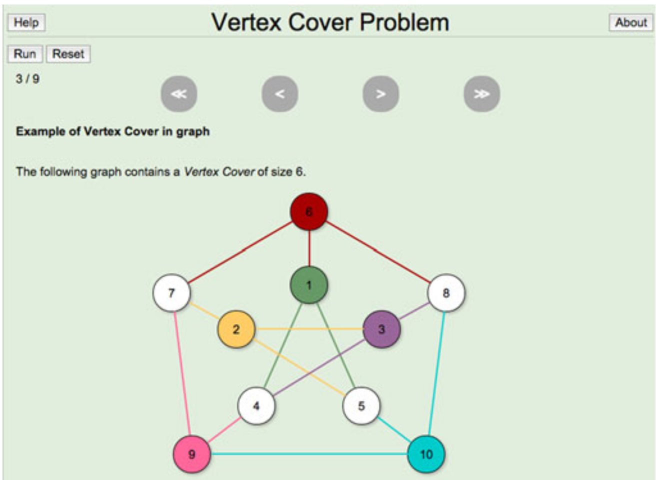

Help
Vertex Cover Problem
About
Run
Reset
3 / 9
Example of Vertex Cover in graph
The following graph contains a Vertex Cover of size 6.

Fig. 7. AV explaining the vertex cover problem.

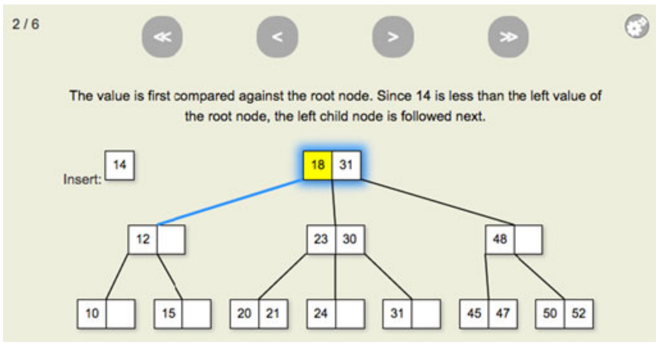
Beyond OpenDSA, materials for introductory CS courses at the University of Turku have been developed with JSAV. Use of JSAV in this and other courses is described in more detail below. The courses at Turku include topics such as the basics of artificial intelligence and the different levels of abstraction from high-level programming language to machine code execution. JSAV-based content has been created for many topics. For example, the course used a JSAV exercise where the student needs to play Tic-Tac-Toe according to the rules for the AI algorithm. So, the student simulates the computer player.

As an another example, Fig. 9 shows a slideshow of executing a program written in a simplified assembly language. Execution of each step is visualized and explained. The same approach was used to create simulation exercises where the student has to click the instructions in assembly code in the order that they would be executed. Other topics with visual content include variable swap exercises, parse tree generation, and more traditional AVs.

4.2 System Integrations

Many instructors use a Learning Management System (LMS) such as Moodle or Sakai to maintain class score information. Such an instructor who also wishes to use JSAV-based visualizations would find it convenient if JSAV exercises could report scores directly to their LMS. Thus far, we have integrated JSAV into the LMSs used at the institutions where JSAV has been used.

The original infrastructure for OpenDSA implemented its own scoring support. However, the OpenDSA project is currently in the advanced stages of replacing its communications layer to use the LTI standard,[11] including support for scoring all JSAV-based exercises, and reporting completion for all JSAV-based visualizations. Once completed, all JSAV proficiency exercises that load the OpenDSA client-side communications library will be able to report scores to a range of LMSs that support the LTI standard, such as Canvas and Moodle. At Aalto University, JSAV was integrated with the TRAKLA2 learning environment [4] in 2012. In 2013, it was integrated into the A+ environment [29] and, in 2014, with the OpenEdX learning management system. At the University of Turku, JSAV was integrated with the ViLLE learning environment [30], [31] in 2014. All these integrations include JSAV-based exercises into the system in a way that makes it seem like native content for the system. In all these cases, student grades from JSAV exercises are submitted and stored within the associated LMS. In some cases (such as the ViLLE integration), a detailed enumeration of the steps used by the student to generate their solution is stored and can be viewed by the course teacher or researchers interested in the data. Technically, all the LMS integrations use JSAV events (explained in Section 3.5) to capture student scores, and then submit them to the LMS in a suitable format.



The value is first compared against the root node. Since 14 is less than the left value of the root node, the left child node is followed next.

Fig. 8. Two-three Tree slideshow illustrating the array tree.

Another type of integration is to use JSAV as a building block in learning tools needing data structures and AVs. An example is the JSAV integration into the Khan Academy exercise framework [8] in order to provide KA-compatible exercises using data structure visualizations and interactions with the visualizations. JSAV has also been used in a program visualization system by Mornar et al. [32]. Their system interprets pseudocode or python programs. It identifies "interesting" events from the execution, and visualizes the data structures manipulated by the visualized program. An iPad application prototype for learning spatial data structures and algorithms uses JSAV to visualize the structures [33]. This app is shown in Fig. 5.

To support reuse of existing algorithm visualizations developed over the years (Requirement R15), there is a proof-of-concept implementation of importing and visualizing AnimalScript [34] animations with JSAV.[12]

4.3 Experiences and Research Results

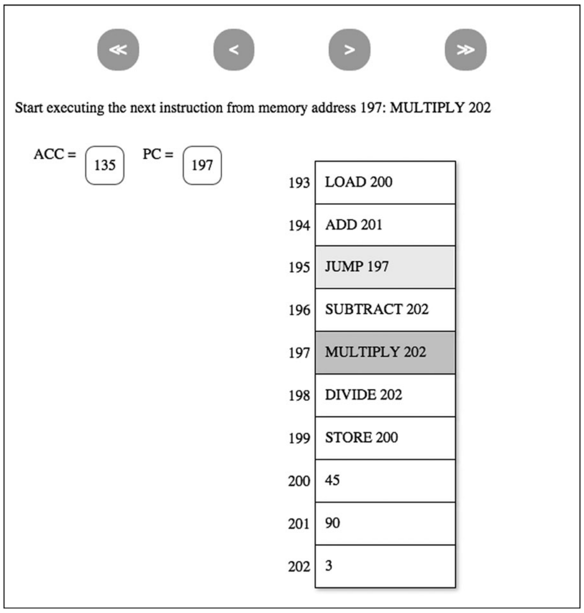
JSAV has been used in multiple institutions in different countries. Here, we briefly introduce the different uses and some of the research findings made thus far.

4.3.1 *JSAV and OpenDSA*

As previously described, OpenDSA relies on JSAV as the foundation for all AVs and interactive exercises. Since Fall 2013, OpenDSA has already been used in over a dozen institutions around the world outside of its home collaborators at Virginia Tech, Aalto University, and University of Wisconsin-Oshkosh. New institutions adopt OpenDSA materials each semester. Primary use so far has been for the DSA

11. http://www.imsglobal.org/toolsinteroperability2.cfm

12. https://github.com/vkaravir/jsav-asu-import



Start executing the next instruction from memory address 197: MULTIPLY 202

ACC = 135 PC = 197

193	LOAD 200
194	ADD 201
195	JUMP 197
196	SUBTRACT 202
197	MULTIPLY 202
198	DIVIDE 202
199	STORE 200
200	45
201	90
202	3

Fig. 9. Example of a step-by-step visualization of execution of a simplified assembly language.

content for CS2 courses, and as the semester "textbook" for DSA courses following CS2 (we will refer to these as "CS3"). We have collected extensive survey information from students regarding OpenDSA use, along with a small number of student interviews, feedback from instructors, and interaction log data from student use. Details of resulting analysis of these data sources can be found in [6], [9], [35]. We present here a brief overview of these findings.

Since Fall 2012, we have collected literally millions of individual user interactions with the OpenDSA content. We have done a number of analyses that reveal information about student behavior with eTextbook material. Knowing about these behaviors are important, as they can help us to prioritize a redesign of the relevant system components and user interactions. Briefly, we have determined the following (see [9], [35] for details). First, we have learned that many students engage in a number of behaviors that we term "credit seeking" in contrast to more positive behavior that we term "learning behavior". This includes:

- Students tend to skip tutorial content to move directly to exercises that are required for a homework grade. They only look at the tutorial content as necessary to solve the exercises.
- When slideshows are given a small amount of credit, students will either click through them quickly or skip to the end, whichever will give them participation credit.
- Students will often skip tutorial content related to the analysis of an algorithm entirely. This content historically has not included a visual component.

We have also made other observations based on log data and feedback from students that can be viewed more positively.

- Students appreciate and take advantage of "gamification" aspects, such as giving checkmarks for completing slideshows, and "module complete" messages for doing all activities.
- Students will react to a gradesheet or table of contents that highlights in green "completed" exercises or modules.
- Students do use OpenDSA exercises as study aides.

We have learned [9] both from interaction log data and from surveys that students show little interest in using smartphones or tablets to view OpenDSA materials, far preferring to use a laptop or desktop computer.

From Fall 2012 through Fall 2014, approximately 200 students have been surveyed regarding their experience with OpenDSA. Both in the surveys and in numerous unsolicited comments, students have voiced overwhelming support and appreciation for OpenDSA as a learning tool. Students credit it with improving their understanding of the algorithms covered in courses.

We note that while subjective student reaction to OpenDSA is strongly positive, there is only weak evidence as of yet that OpenDSA positively affects learning outcomes. In Fall 2012 [6] a study was done comparing student performance on a midterm between students using OpenDSA materials for the topics of Sorting and Hashing, versus a control group in another section of the course that did not use OpenDSA. While students in the OpenDSA section had a mean score about one half of a standard deviation higher than those in the control section, the results were just short of being statistically significant. Other evidence for positive learning outcome comes in a correlation between level of use by students using OpenDSA as a study aid (specifically, doing additional exercise instance correctly) and doing better on the associated exams. However, we do not know at the present time if there is a causal relationship.

OpenDSA at present is far stronger on its presentation of descriptive material (the AVs themselves) than it is on either exercises or presentation of analytical material. While there is no doubt that the students believe AVs to be improving their understanding, we hypothesize that the true improvements to learning come from working exercises, and studying material that they otherwise historically ignore. As both of those aspects improve, we hope to be able to detect stronger evidence of learning gain. This is in line with the literature on "No Significant Difference" [36]. The AVs themselves are not providing a new learning experience (just possibly one whose content is easier to grasp). In contrast, bulk interactive practice exercises are something previously not practical. Likewise, while more visual presentation of analytical material might itself not be significantly better than textual presentation, we have evidence that the use of slideshows and careful engineering of the circumstances of presentation will require the students to engage the analytical material [9]. We know from our interaction log data that many students completely ignore the analytical content, so a design that requires students to study the content should improve learning outcomes as compared to when they do not study that content at all.

4.3.2 JSAV Use at Aalto University

Some of the first proficiency exercises created with JSAV were on the topic of binary heaps (see Fig. 3), and were used at Aalto University in the Spring of 2012. The exercises replaced one round of TRAKLA2 [4] exercises in a data structures and algorithms course. Student interaction with

TABLE 3
Percentages of Student Interaction with Tree and Array Visualizations in Binary Heap Exercises

Exercise	Tree Clicks	Array Clicks
Heapsort	94.8%	5.2%
Heap Insert	68.9%	31.1%
Buildheap	87.7%	12.3%
Heap Delete	92.3%	7.7%

the exercises was collected and analyzed [37]. From the collected log data, the misconceptions that students had could also be analyzed. The analysis showed that students had the same misconceptions of the algorithms in JSAV exercises as they had with the original TRAKLA2 system. Thus, JSAV exercises were at least no worse than the corresponding TRAKLA2 exercises in terms of performance and misconceptions. More importantly, the study included analysis to identify the steps in the algorithm that were difficult for students. This shows the power of data logging in JSAV, which provides new information that can lead to improvements in the future that otherwise might not be possible. An example of such an improvement already implemented is enhanced heap exercises used in Spring 2013 to ensure that the input for the algorithm is such that a student getting full points on the exercise does not have any of the identified misconceptions.

The heap exercises also collected data on whether students interacted with the tree or the array in the heap exercises. The distribution of interaction (mouse click) with tree and array are shown in Table 3. As can be seen, a majority of the interaction is done on the tree visualization. This is natural, since the tree better shows the relationship of the data items in the heap. However, it is interesting how much the distributions differ between exercises. This is another example of the kind of insight that we can get from detailed log data.

In Fall 2014, the old TRAKLA2 exercises were replaced with JSAV counterparts, many developed at Aalto University. Some students solved the exercises through the OpenEdX environment, while others had the choice of solving them embedded in OpenDSA material or through the A+ environment. The score achieved by the student was stored in A+ whichever way the student accessed them.

4.3.3 JSAV Use at University of Turku

At the University of Turku, JSAV-based content has been used since Fall 2014 (see [31] for details). It was first used in an introductory CS course for data structures content. Proficiency exercises on swapping the values of variables and indexes on arrays and matrices were included. Results of a student questionnaire on JSAV exercises are shown in Fig. 10. The claims were as follows: 1) The exercises helped my learning, 2) The exercises were easy to use, 3) The feedback of the exercise helped my learning, and 4) The exercises should be used in the future. Students found the JSAV exercises both easy to use and helpful for their learning.

JSAV exercises were also used in an introductory CS course that covered a wide range of topics such as logical gates, machine code, assembly code, code parsing and compiling, and artificial intelligence. JSAV was used to visualize machine and assembly code, parser tree generation, and AI

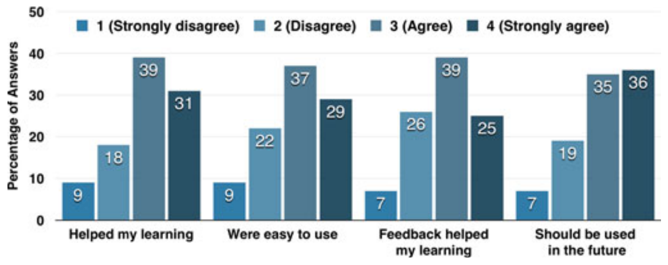


Fig. 10. JSAV student feedback at university of turku.

algorithms. The JSAV content provided an important part of the course, and could not have been implemented with other tools used for the course.

In these courses, the JSAV exercises were embedded within learning material (locally called tutorials) similar to the OpenDSA approach to presenting material. In addition, some stand-alone homework assignments used JSAV-based exercises. Finally, a collection of JSAV exercises were used in a Data Structures and Algorithms course as homework assignments. These were mainly exercises created by the OpenDSA project, which were adapted to work in the ViLLE environment.

5 Discussion

JSAV has evolved concurrently with it being used to create material on an ever widening variety of topics. At this point, the library has been used by dozens of developers to create hundreds of visualizations and exercises on diverse topics ranging from visualizations on algorithms that manipulate arrays, trees, and graphs, to proofs of NP-completeness, displays of Finite State Automata, basic programming techniques, and parsing. Key to this is the proven ability of developers to extend base JSAV capabilities beyond the scope originally envisioned, without being forced to make changes to the library itself.

There is a learning curve that must be climbed to develop content with JSAV. It requires knowledge of JavaScript and HTML, with CSS and web developer tool skills also being important. While not all student and faculty members are skilled in these areas, it is getting more common with the growing role of web technologies. Developing JSAV-based content has served as a good motivation and introduction to web technologies for a number of students. Over 30 undergraduate students and a dozen graduate students at Virginia Tech, University of Wisconsin-Oshkosh, Duke University, Linköping University, and Aalto University have already participated in developing AVs and exercises. Their motivations include getting course credit to learn a useful technical skill (JavaScript/HTML/CSS web development), but also a sense of "giving back" to the OpenDSA project that they credit with helping them learn DSA content.

The technical learning curve is not a major hurdle, with students able to begin developing useful content once they have mastered basic JavaScript programming. It helps to speed the learning that there now exists a large body of content to be used as examples, as well as complete API documentation for the library. A typical starting place is to create a static image, which can be done in a few minutes when working from an example. Implementing a simple tutorial slideshow using JSAV feels much like writing a presentation using a script-based slideshow package such as Beamer.[13] The next stage in a developer's progress is to move from a procedural style of "put this element here" to instrumenting a working algorithm. Programmatic control of the visualization (through library calls to JSAV) makes for powerful integration of visualization with the algorithm being visualized. The hardest thing to learn is not how to use the library to do what you want. The hardest skill is developing good judgment for what level of detail should be presented about the working of an algorithm.

A library like JSAV can never be complete. As more and more material is being created with it, new requirements and old bugs emerge. It is, however, complete enough for us to call the current version 1.0. When looking back at the requirements for an AV system introduced in Section 2, there are still some features missing. It is not possible for the user to change the background color (part of requirement R3). That said, it could be added by the AV developer to any given AV through the flexible "settings" mechanism provided by JSAV. A standard feature of most of the AVs is a user-configurable settings panel, which a developer could easily extend to include simple color changes to the UI and similar user controls. Furthermore, like the existing HTML5-based AV systems, there is no support for explanations which adapt to the knowledge of learner as well as the state of the algorithm (part of R6). The feedback in pop-up question support is limited to correct/incorrect (R7). However, the primary reason why there is only limited support for questions within JSAV itself is the fact that we have found it relatively easy to integrate JSAV with other systems (such as the Khan Academy Exercise Framework) that directly support a rich set of question types. In the long run, it is probably a better design approach to allow integration with other tools, than to create a monolithic system that supports all potential features. Features lacking completely are a table-of-contents-like structural view of the algorithm (R10) and viewing of the previous step in the algorithm simultaneously to the current step (R16). Our experiences suggest that the need for a structural overview depends on the way the AV content is designed. The first AV developed with JSAV was a Shellsort slideshow going through the complete algorithm. However, we quickly realized that this was not the best approach to explain the algorithm, and ended up breaking the slideshow into a series of smaller slideshows, each focusing on the different phases of the algorithm. These slideshows were then embeded within the tutorial material. That said, for standalone (comprehensive) AVs, a structural view would be beneficial. Overall, while not all requirements are met or are only partially fulfilled, JSAV does support more requirements than the any of the existing HTML5-based systems, or even any of the previous Java-based systems. This, however, does not mean that JSAV is better than the other systems for every use case.

JSAV has been heavily inspired by previous work, and it includes many of the features found necessary in prior systems. Thus, most features have been previously introduced in some system. However, when compared to the existing AV systems evaluated in Section 2.3, JSAV is the only system supporting viewing, responding, and constructing levels of engagement. This makes it suited for more use cases than any single existing system. Furthermore, it has more advanced support for proficiency exercises. JSAV supports multiple feedback options, and is the only system that supports fix mode to put students back on track after a mistake during an exercise. JSAV's model answer feedback can contain any explanations and visualizations, including smooth animation. This more versatile feedback is what sets JSAV apart from TRAKLA2, which has had support for proficiency exercises for years. JSAV's use of HTML5 and JavaScript make it more customizable and it can reach a broader audience than Java-based systems. JSAV is the only AV system that we know of that supports 'localization' for both natural language and programming language. Finally, the ability to integrate JSAV with LMS backends enables rich logging and analysis of many aspects of the use of the content.

There is yet a long list of issues in JSAV's GitHub repository,[14] and there are bigger future goals for the project as well. One such goal is to add built-in support for identifying misconceptions as well as generation of data to expose student misconceptions. Previous efforts in misconception-aware input generation was implemented by the first author, and was only specific to the binary heap algorithms. Another bigger issue to tackle is the security and trustworthiness of the grading. Currently, the grading is done on the client side and is thus vulnerable to cheating by anyone skilled enough with JavaScript and browser web developer tools. Setting up a way to do server-side regrading of the submissions would provide another level of trustworthiness to the results. A solution to this has been prototyped at Aalto University, but has not seen wider use. Finally, since the pop-up question support is somewhat limited in its features, we will explore options to integrate some more comprehensive question library instead of evolving our own implementation.

6 Conclusions

We have introduced the JavaScript Algorithm Visualization library. JSAV is the first HTML5-based AV library supporting multiple levels of engagement, including algorithm simulation exercises. There now exists a vast body of content created with JSAV, by a considerable number of developers, and spanning a wide range of topics and purposes. JSAV has been integrated into multiple learning environments and educational systems. We discussed our experiences and research done on JSAV use. Student response to the resulting visualizations has been strongly positive.

Acknowledgments

This work was supported in part by the US National Science Foundation (NSF) grants DUE-1139861, IIS-1258571, and DUE-1432008.

References

[1] T. Naps, G. Rössling, and nine more authors, "Exploring the role of visualization and engagement in computer science education," in *Proc. Working Group Rep. ITiCSE Innovation Technol. Comput. Sci. Edu.*, 2002, pp. 131–152.

13. https://www.ctan.org/pkg/beamer?lang=en

14. https://github.com/vkaravir/JSAV/issues

[2] C. Shaffer, M. Cooper, A. Alon, M. Akbar, M. Stewart, S. Ponce, and S. Edwards, "Algorithm visualization: The state of the field," *ACM Trans. Comput. Edu.*, vol. 10, pp. 1–22, Aug. 2010.
[3] E. Fouh, M. Akbar, and C. Shaffer, "The role of visualization in computer science education," *Comput. Schools*, vol. 29, pp. 95–117, 2012.
[4] L. Malmi, V. Karavirta, A. Korhonen, J. Nikander, O. Seppälä, and P. Silvasti, "Visual algorithm simulation exercise system with automatic assessment: TRAKLA2," *Informat. Edu.*, vol. 3, no. 2, pp. 267–288, Sep. 2004.
[5] T. Naps, "Jhavé: Supporting algorithm visualization," *IEEE Comput. Graph. Appl.*, vol. 25, no. 5, pp. 49–55, Sep./Oct. 2005.
[6] S. Hall, E. Fouh, D. Breakiron, M. Elshehaly, and C. Shaffer, "Evaluating online tutorials for data structures and algorithms courses," presented at the ASEE Annu. Conf., Atlanta, GA, USA, Jun. 2013, Paper #5951.
[7] C. Shaffer, V. Karavirta, A. Korhonen, and T. Naps, "OpenDSA: Beginning a community Active-eBook project," in *Proc. 11th Koli Calling Int. Conf. Comput. Edu. Res.*, Koli National Park, Finland, Nov. 2011, pp. 112–117.
[8] E. Fouh, V. Karavirta, D. Breakiron, S. Hamouda, S. Hall, T. Naps, and C. Shaffer, "Design and architecture of an interactive Etextbook – the OpenDSA system," *Sci. Comput. Program.*, vol. 88, no. 1, pp. 22–40, Aug. 2014.
[9] E. Fouh, D. Breakiron, S. Hamouda, M. Farghally, and C. Shaffer, "Exploring students learning behavior with an interactive etextbook in computer science courses," *Comput. Human Behavior*, pp. 478–485, Dec. 2014.
[10] V. Karavirta and C. Shaffer, "JSAV: The javascript algorithm visualization library," in *Proc. 18th Annu. Conf. Innovation Technol. Comput. Sci. Edu.*, Canterbury, U.K., Jul. 2013, pp. 159–164.
[11] S. Diehl, *Software Visualization*, no. 2269. New York, NY, USA: Springer, 2002.
[12] T. Hendrix, J. Cross, and L. Barowski, "An extensible framework for providing dynamic data structure visualizations in a lightweight IDE," in *Proc. 35th Techn. Symp. Comput. Sci. Edu.*, 2004, pp. 387–391.
[13] P. J. Guo, "Online python tutor: Embeddable Web-based program visualization for CS education," in *Proc. 44th SIGCSE Techn. Symp. Comput. Sci. Edu.*, 2013, pp. 579–584.
[14] R. Ross and M. Grinder, "Hypertextbooks: Animated, active learning, comprehensive teaching and learning resources for the web," in *Proc. Int. Seminar Dagstuhl Castle Software Visualization*, 2002, no. 2269, pp. 269–284.
[15] G. Rößling, T. Naps, M. Hall, V. Karavirta, A. Kerren, C. Leska, A. Moreno, R. Oechsle, S. Rodger, J. Urquiza-Fuentes, and J. Velázquez-Iturbide, "Merging interactive visualizations with hypertextbooks and course management," in *Proc. ITiCSE-WGR Working Group Rep. ITiCSE Innovation Technol. Comput. Sci. Edu.*, 2006, pp. 166–181.
[16] A. Korhonen, T. Naps, C. Boisvert, P. Crescenzi, V. Karavirta, L. Mannila, B. Miller, B. Morrison, S. H. Rodger, R. Ross, and C. A. Shaffer, "Requirements and design strategies for open source interactive computer science eBooks," in *Proc. ITiCSE Working Group Rep. Conf. Innovation Technol. Comput. Sci. Edu.-Working Group Rep.*, 2013, pp. 53–72.
[17] G. Rößling and T. Vellaramkalayil, "First steps towards a Visualization-based computer science hypertextbook as a moodle module," in *Proc. 5th Program Vis. Workshop*, 2009, pp. 47–56.
[18] V. Karavirta, "Seamless merging of hypertext and algorithm animation," *ACM Trans. Comput. Edu.*, vol. 9, no. 2, pp. 1–18, 2009.
[19] C. Hundhausen, S. Douglas, and J. Stasko, "A meta-study of algorithm visualization effectiveness," *J. Vis. Languages Comput.*, vol. 13, pp. 259–290, Jun. 2002.
[20] J. Urquiza-Fuentes and J. A. Velázquez-Iturbide, "A survey of successful evaluations of program visualization and algorithm animation systems," *Trans. Comput. Edu.*, vol. 9, pp. 9:1–9:21, Jun. 2009.
[21] G. Rößling and T. Naps, "A testbed for pedagogical requirements in algorithm visualizations," in *Proc. 7th Annu. Conf. Innovation Technol. Comput. Sci. Edu.*, 2002, pp. 96–100.
[22] G. Rößling and T. Naps, "Towards intelligent tutoring in algorithm visualization," in *Proc. 2nd Int. Program Vis. Workshop.* 2002, pp. 125–130.
[23] S. Diehl, *Software Visualization: Visualizing the Structure, Behaviour, and Evolution of Software*. New York, NY, USA: Springer, 2007.
[24] J. Morris, "Algorithm animation: Using algorithm code to drive an animation," in *Proc. 7th Australasian Conf. Comput. Edu.*, 2005, pp. 15–20.
[25] G. Rößling and B. Freisleben, "ANIMAL: A system for supporting multiple roles in algorithm animation," *J. Vis. Languages Comput.*, vol. 13, no. 3, pp. 341–354, 2002.
[26] A. Akingbade, T. Finley, D. Jackson, P. Patel, and S. H. Rodger, "JAWAA: Easy web-based animation from CS 0 to advanced CS courses," in *Proc. 34th SIGCSE Techn. Symp. Comput. Sci. Edu.*, 2003, pp. 162–166.
[27] S. Halim, Z. C. Koh, V. B. H. Loh, and F. Halim, "Learning algorithms with unified and interactive web-based visualization," *Olympiads Informat.*, vol. 6, pp. 53–68, 2012.
[28] AlgoViz.org. (2011). The AlgoViz portal [Online]. Available: http://algoviz.org
[29] V. Karavirta, P. Ihantola, and T. Koskinen, "Service-oriented approach to improve interoperability of E-learning systems," in *Proc. 13th IEEE Int. Conf. Adv. Learn. Technol.*, 2013, pp. 341–345.
[30] T. Rajala, M.-J. Laakso, E. Kaila, and T. Salakoski, "Effectiveness of program visualization: A case study with the ViLLE tool," *J. Inf. Technol. Edu.: Innovations Practice*, vol. 7, pp. 15–32, 2008.
[31] V. Karavirta, R. Haavisto, E. Kaila, M.-J. Laakso, T. Rajala, and T. Salakoski, "Interactive learning content for introductory computer science course using the ville learning environment," in *Proc. Learn. Teaching Comput. Eng.*, Apr. 2015, pp. 9–16.
[32] J. Mornar, A. Granić, and S. Mladenović, "System for automatic generation of algorithm visualizations based on pseudocode interpretation," in *Proc. Conf. Innovation Technol. Comput. Sci. Edu.*, 2014, pp. 27–32.
[33] V. Karavirta, "Location-aware mobile learning of spatial algorithms," in *Proc. IADIS Int. Conf. Mobile Learn.*, Lisbon, Portugal, Mar. 2013, pp. 158–162.
[34] G. Rößling and B. Freisleben, "AnimalScript: An extensible scripting language for algorithm animation," in *Proc. 33nd SIGCSE Techn. Symp. Comput. Sci. Edu.*, 2001, pp. 70–74.
[35] D. A. Breakiron, "Evaluating the integration of online, interactive tutorials into a data structures and algorithms course," Master's thesis, Virginia Tech, Blacksburg, VA, USA, 2013.
[36] T. Russell, *The No Significant Difference Phenomenon: A Comparative Research Annotated Bibliography on Technology for Distance Education*, 5th ed. Chicago, IL, USA: IDECC, 2001.
[37] V. Karavirta, A. Korhonen, and O. Seppälä, "Misconceptions in visual algorithm simulation revisited: On UI's effect on student performance, attitudes, and misconceptions," in *Proc. Learn. Teaching Comput. Engineering*, Macau, China, 2013, pp. 62–69.

Ville Karavirta received the PhD degree from Helsinki University of Technology. He is a senior research fellow in the Department of Information Technology, University of Turku. His current research interests include mobile learning, eTextbooks, program and algorithm visualization, and automated assessment of programming assignments. His web address is http://www.villekaravirta.com/.

Clifford A. Shaffer received the PhD degree from the University of Maryland. He is a professor in the Department of Computer Science, Virginia Tech. His current research interests include digital education, eTextbooks, bioinformatics, visualization, algorithm design and analysis, and data structures. His web address is http://www.cs.vt.edu/shaffer.