Algorithm Visualization: The State of the Field

CLIFFORD A. SHAFFER
Virginia Tech
MATTHEW L. COOPER
Microsoft Corp
ALEXANDER JOEL D. ALON, MONIKA AKBAR, MICHAEL STEWART,
SEAN PONCE, and STEPHEN H. EDWARDS
Virginia Tech

We present findings regarding the state of the field of Algorithm Visualization (AV) based on our analysis of a collection of over 500 AVs. We examine how AVs are distributed among topics, who created them and when, their overall quality, and how they are disseminated. There does exist a cadre of good AVs and active developers. Unfortunately, we found that many AVs are of low quality, and coverage is skewed toward a few easier topics. This can make it hard for instructors to locate what they need. There are no effective repositories of AVs currently available, which puts many AVs at risk for being lost to the community over time. Thus, the field appears in need of improvement in disseminating materials, propagating known best practices, and informing developers about topic coverage. These concerns could be mitigated by building community and improving communication among AV users and developers.

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, Measurement, Design

Additional Key Words and Phrases: Algorithm animation, algorithm visualization, AlgoViz Wiki, community, data structure visualization, free and open source software

An early version of this article was published as Shaffer et al. [2007].
This work is supported in part by the National Science Foundation under Grant Nos. DUE-0836940, DUE-0839837, DUE-0937863, and DUE-0946644. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.
Author's address: C. A. Shaffer, Department of Computer Science, Virginia Tech, Blacksburg, VA 24061; email: shaffer@cs.vt.edu.

ACM Reference Format:
Shaffer, C. A., Cooper, M. L., Alon, A. J. D., Akbar, M., Stewart, M., Ponce, S., and Edwards, S. H. 2010. Algorithm visualization: The state of the field. ACM Trans. Comput. Educ. 10, 3, Article 9 (August 2010), 22 pages. DOI = 10.1145/1821996.1821997.
http://doi.acm.org/10.1145/1821996.1821997.

1. INTRODUCTION

Algorithms and data structures form one cornerstone of an undergraduate computer science education. A technique for improving instruction in this critical area is to include algorithm and data structure visualizations and animations (hereafter referred to as "algorithm visualizations" or "AVs") into the curriculum. AVs have a long history in computer science education, dating from the 1981 video "Sorting out Sorting" by Ronald Baeker and the BALSA system [Brown and Sedgewick 1984]. Since then, hundreds of AVs have been implemented and provided free to educators, and scores of papers have been written about them [AlgoViz.org 2010]. Good AVs bring algorithms to life by graphically representing their various states and animating the transitions between those states. They illustrate data structures in natural, abstract ways instead of focusing on memory addresses and function calls.

AVs are naturally attractive to educators, who nearly universally view them positively [Naps et al. 2002]. They are also consistently "liked" by students [Gurka and Citrin 1996; Stasko et al. 2001]. But an important question persists: Are they effective at educating computer science students? There has been some debate in the literature as to whether AVs are effective in practice. Some studies have shown the classic dismissal for many technological interventions in education: "no significant difference" [Gurka and Citrin 1996; Hundhausen and Douglas 2000; Jarc et al. 2000]. Other studies have shown that AVs can indeed improve understanding of the fundamental data structures and algorithms that are part of a traditional computer science curriculum [Lawrence et al. 1994; Byrne et al. 1996; Hansen et al. 2000; Grissom et al. 2003]. An important conclusion from the literature is that to make AVs pedagogically useful, they must support student interaction and active learning [Naps et al. 2002].

Certainly, many AVs exist and are widely (and freely) available via the Internet. Unfortunately, those of high quality can be lost among the many of lower quality.

So we see that (a) while many AVs exist, relatively few are of true value, and (b) some AVs can be demonstrated to have pedagogical value, yet it is also quite possible to use AVs in ways that have no pedagogical effect. These results indicate that creating and deploying effective AVs is difficult. There is a growing body of literature that investigates how to create pedagogically useful AVs [e.g., Hundhausen et al. 2002; Saraiya et al. 2004a; Naps et al. 2002, 2003; Rössling et al. 2006]. Yet, there is still much to be done before we are at the point where good quality AVs on most topics of interest are widely recognized and adopted.

The purpose of this article is to provide a summary of the state of the field in terms of existing AV artifacts. To bound the content area, we focused our

attention on AVs for topics commonly taught in undergraduate courses on data structures and algorithms. We seek an understanding of the overall health of the field as represented by what AVs are currently available, and present a number of open research questions that we and others can work on in the future. Some examples of the questions we seek to address are the following.

—What AVs are available?
—What is their general quality?
—Is there adequate coverage of the major topic areas covered in data structures and algorithms courses?
—How do educators find and use effective AVs?
—Is the field active and improving?
—Is there adequate infrastructure for archiving and disseminating AVs?
—Is there adequate support for a community of AV developers and users?

The remainder of the article is structured as follows. Section 2 presents a brief review of the history of AV development, a survey of some major AV systems, and discussion on the literature of AV pedagogical effectiveness. Section 3 describes the methods that we used to survey the current state of the field, primarily through the data collected and cataloged at the AlgoViz Wiki. Section 4 presents our main findings on the current state of the field of AVs. Section 5 presents our conclusions and future plans.

2. LITERATURE REVIEW

In addition to many hundreds of individual AVs created over the years, there have also been many systems created to support AV development. A significant amount of research has been devoted to understanding the pedagogical effectiveness of AVs. We examine some of the literature in this section.

2.1 Algorithm Visualization Development

A number of AV development systems have become well known within the CS educational community. We can divide their history into two parts: what came before the rise of Sun's Java programming language and widespread uptake of content delivered via the Internet, and what came after. See Saraiya [2002] and Wiggins [1998] for a more complete treatment of older or presently inaccessible systems.

Since the widespread use of Java, development seems to have moved away from AV authoring toolkits, and towards suites of "canned" AVs. Pre-Java systems often came packaged with pre-generated AVs, but allowed educators and even students the freedom to implement other AVs in a scripting language or by annotating a real program. Today's systems are frequently distributed as collections of AVs not tied to any operating system or environment. Some of these collections come as part of AV development systems and some do not.

Brown ALgorithm Simulator and Animator (BALSA) introduced the concept of using program-generated animations to teach fundamental computer science

concepts [Brown and Sedgewick 1984]. Besides undergraduate education, the system was also used for debugging and research in algorithm analysis. Later systems developed by Brown include Zeus [Brown 1992] and JCAT [Brown et al. 1997].

XTANGO [Stasko 1992] is the most recent version of the TANGO visualization system. Developed by John Stasko at Georgia Tech, its goal is to allow students who are not experts in graphical systems to create algorithm animations by implementing the algorithm in C and then adding special calls to the XTANGO graphical library. XTANGO is still maintained and distributed (making it one of the few non-Java AV development systems still available). POLKA (Parallel program-focused Object-Oriented Low Key Animation) [Stasko and Kraemer 1992], another Georgia Tech project, is described as "a general purpose animation system that is particularly well-suited to building algorithm and program animations" [Stasko 2001]. Like XTANGO before it, POLKA allows nonexperts to author AVs without deep knowledge of graphics programming.

Swan [Shaffer et al. 1996], developed at Virginia Tech, provides visualizations of C and C++ programs annotated with calls to SAIL, the Swan Annotation Interface Library. Unlike most program annotation systems, SWAN supports the ability to provide significant user control over the behavior of the annotated program.

ANIMAL (A New Interactive Modeler for Animations in Lectures) [Rössling et al. 2000] incorporates lessons learned from pedagogical research (see Section 2.2). Developed at the University of Siegen in Germany, ANIMAL's developers believe that visualization systems should supplement (instead of supplant) traditional approaches to CS education. Animal AVs tend to have limited user interaction, which is not crucial when used as a lecture aid.

JAWAA (Java And Web-based Algorithm Animation) [Pierson and Rodger 1998] was developed at Duke University by Willard Pierson and Susan Rodger. It is a Java applet-based system that runs on script files generated by any programming language. It is similar to Animal in that the AVs it generates provide little user interaction. JAWAA assists the animation developer by providing primitives in a range of granularities, from basic graphics (circles, squares) up to data structures (arrays, trees). Generally only a few lines of JAWAA script are required to animate an algorithm: one to create and display the structure, the others to perform operations on it.

JHAVÉ (Java-Hosted Algorithm Visualization Environment) [Naps et al. 2000] is a client-server AV system written in Java, and most typically run as a Web-start application from the browser. JHAVÉ does not itself provide an animation system, but rather a network-based framework into which animation systems can be (and have been) embedded. This network model provides delivery of the AV as well as a question-and-answer platform to improve the pedagogical effectiveness of the animation plugin [Naps et al. 2000]. JHAVÉ represents some key initial steps in attempting to support interoperability with other AV development systems, such as Animal.

AlViE 3.0 [Crescenzi 2010] is maintained by Pilu Crescenzi at University of Florence. AlViE is a "post-mortem" algorithm visualization environment,

presenting a series of "interesting events." The AVs are tied to a widely-used Italian data structures textbook [Crescenzi et al. 2006], and so AlViE has many users.

Animal, JAWAA, JHAVÉ, and AlViE are all similar in that they provide frameworks for playing AV "scripts," where these scripts can be generated in various ways independent of the system (such as by instrumenting output from actual programs). By their nature of running scripts, they tend to be limited in their ability to support sophisticated user interaction such as steering the algorithm or running it on user-supplied data. In contrast, a number of other systems take the approach of providing a canned suite of AVs. This limits the ability of those outside the project (such as instructors wishing to use the system) to create their own AVs, but it often provides greater design flexibility to "expert" developers.

Developed in Germany at the University of Marburg, the Data Structure Navigator (DSN) is a project intended to "facilitate the understanding of all kinds of data structures" [Dittrich et al. 2000]. Instructors might find it to be a good starting point for a suite of AVs to be used during an entire data structures course. A major drawback of DSN is that it cannot load and save data structures to disk, and development appears to have ceased.

Data Structure Visualization (DSV) [Galles 2006] was created at the University of San Francisco by David Galles. DSV aspires to be a comprehensive suite of AVs. DSV provides animations for a wide variety of algorithms and data structures, including several sorts, multiple varieties of trees, and a number of graph algorithms.

Interactive Data Structure Visualizations (IDSV) [Jarc et al. 2000] was created by Duane Jarc while at George Washington University. It is a collection of Java applets covering a range of topics related to search trees, graph algorithms, and sorting algorithms. IDSV is Similar to JHAVÉ in that it periodically gives the user questions to answer while viewing the animation.

A large collection of AVs for spatial data structures is available from the University of Maryland [Brabec and Samet 2003].

The Algorithms in Action project [Stern 2001] is directed by Linda Stern of University of Melbourne. The original version was made available in 2000, and was quite sophisticated for its time in its use of multiple windows in a Java applet to track pseudocode, the visualization, and supporting explanatory text. An extensive collection of additional applets was released in early 2010 after classroom testing.

The TRAKLA2 project [Korhonen et al. 2003] is directed by Ari Korhonen and Lauri Malmi of the Software Visualization Group at the Helsinki University of Technology. TRAKLA2 is particularly innovative in its extensive use of interactive exercises where students demonstrate their proficiency with an algorithm by providing information about the various key events that happen during execution. This is done through a variety of graphical interface actions such as dragging values to various nodes or array positions, selecting from a list of choices, etc. While the current collection of exercises tends to work better for simpler algorithms and data structures, this approach points the way forward towards deeper student interaction with the AV. TRAKLA2 is widely used

throughout CS Departments in Finland, and as such has perhaps the highest number of (student) users of any AV system.

2.2 Pedagogical Effectiveness of Algorithm Visualizations

AVs can provide a compelling alternative to other types of instruction, particularly written presentations such as pseudocode and real code. In lecture situations, students routinely report liking AVs [Gurka and Citrin 1996; Stasko et al. 2001], and faculty appear to strongly support them in principle [Naps et al. 2002]. The literature, however, has not demonstrated that AVs are always effective in practice. Results form a continuum from "no significant difference" [Gurka and Citrin 1996; Hundhausen and Douglas 2000; Jarc et al. 2000] to demonstrations that AVs can indeed improve understanding of data structures and algorithms [Shaffer et al. 2007; Lawrence et al. 1994; Hansen et al. 2000; Hundhausen et al. 2002]. The existing body of research helps to illuminate some of the features of AVs that make them effective.

Gurka and Citrin [1996] found little to no evidence that visualizations are effective teaching tools in a survey of previous experiments. They point out that while many fields are concerned with false positives in evaluation, the visualization community may need to concern itself with false negatives. They suggest repeating some "failed" experiments from previous projects with tighter controls on issues such as usability, algorithm difficulty, animation quality, and experimental design. They caution, however, that the community must be prepared to accept an ultimate finding of "no significant difference."

Hundhausen and Douglas [2000] found that students who construct their own visualizations might be distracted by the creation process. Their experiment was based on the finding that learner involvement increases effectiveness, but technology does not necessarily improve the result. They found that student-created low-fidelity visualizations made of art supplies had no less impact on educational outcomes than high-quality computer-mediated AVs also created by the students.

Jarc et al. [2000] found no difference in educational outcomes for students who used an interactive AV system compared to their peers who did not. While AV users performed better on some questions, they performed worse overall, despite spending significantly more time with the educational materials than the nonusers.

In contrast to these studies, a growing body of evidence indicates that certain uses of AVs do have a measurable impact on student learning. The most important factor appears to be *engagement* of the students' attention [Naps et al. 2002; Hundhausen et al. 2002]. Work in this area continues to home in on exactly what features make for effective AVs.

Saraiya et al. investigated a number of features that might or might not make AVs effective. They found that the ability to control the pace of the visualization, versus watching an animation, had the single greatest impact on AV effectiveness of the factors studied [Saraiya et al. 2004b; Saraiya 2002]. Other features such as the presence of a good data set and logical break-down of steps also showed promise as indicators of effectiveness. However, providing

pseudocode with an AV did not appear to improve understanding despite the fact that it made the students spend more time with the content.

Lawrence et al. [1994] found that allowing students to create their own visualizations led to better test results compared with students who merely viewed AVs or did not access any visualizations. Students who viewed but did not interact with the visualization actually did slightly worse than students who only listened to the lecture. Again, this points to engagement as a differentiating factor.

Hansen et al. [2000] found that although traditional approaches to AV use had at best mixed results, a particular approach could lead to significant positive impacts on student learning. Specifically, Hansen's group leaned heavily on pedagogical theories such as learning objectives, scaffolding, and chunking. They point out the use of analogies for priming students' understanding of the underlying algorithm.

In 2002, Hundhausen et al. performed a metastudy of 24 studies of AV use. By breaking the studies into analytical groups based on educational theory espoused, Hundhausen found that how students use an AV has more effect on learning outcomes than what they see. They concluded that AVs are an effective tool when used to actively engage students' attention.

3. DATA COLLECTION AND THE ALGOVIZ WIKI

The major source of data for our investigation into the state of the field of algorithm visualization is the AlgoViz Wiki [2010]. This growing collection of curated resources includes the largest AV link collection ever created. This section outlines the data collections available, and the process used to collect the data. Section 4 presents analysis of the dataset.

3.1 Data Contents: The Collections

A wiki is a user-editable Web site, usually supporting a simplified markup syntax and directed towards a particular topic. The AlgoViz Wiki is built on the MoinMoin Wiki Engine [Waldmann and Hermann 2010]. The wiki concept is preferable to a standard HTTP server for the goals of the original AlgoViz site because it allows for the community access and editing that we desire.

The primary content of the AlgoViz Wiki is its catalog of links to AVs. Section 4 gives information on topic coverage and other summary statistics about the catalog. We now give a brief description of the other resources collected in the wiki.

The wiki contains a list of related software including major AV projects, toolkits for authoring AVs, and visual debuggers. Many of the toolkits include collections of AVs, so they are represented multiple times: once for each topic they cover and once on the Toolkits page. Visual debuggers superficially resemble data structure visualizations in that they present information about data paths and memory contents in a step-by-step, visual manner. However, visual debuggers are distinct from AVs because they show physical, machine-specific implementation details about code execution for the purpose of creating and

debugging software, rather than conceptual views of data structures for the purpose of education [Stasko et al. 1998; Diehl 2007].

The other major component of the wiki is an annotated bibliography of AV-related research [AlgoViz.org 2010]. In creating this bibliography, we seek to provide a starting point for newcomers to the field as well as a repository of important work to which practitioners can refer. The annotated bibliography now contains more than 500 papers with BibTeX citations for all and brief descriptions and reviews for about 100 of the papers.

3.2 AV Catalog Information

Our catalog of AV links is a valuable resource in its own right. However, we have taken the idea of a collection a step further by summarizing, describing the benefits, and evaluating a given AV. While not all entries currently possess descriptions and evaluations, the ultimate goal is to create a curated catalog. This section will describe in further detail some of the characteristics that we collect about each AV.

Key information about AVs includes topic, author data (names and institutions), delivery mechanism (e.g., Java applet), when created, what project (if any) the AV is part of, and language of presentation (e.g., English, German). Fields are also available to provide screenshots of the AV and references to publications about the AV.

After an AV has been evaluated, it is assigned an editors' recommendation rating of "Recommended," "Has Potential," or "Not Recommended." These ratings are intended to help instructors rapidly find a suitable AV for teaching while also providing concise feedback to creators of AVs about where to allocate resources for improvement. The small set of categories reflects the fact that it is difficult to place a visualization into a finer-grained categorization (such as a 1–10 rating or A–F grade scheme). Having only three buckets (besides "Unrated") allows us to reduce the amount of subjectivity.

Subtly different than a recommendation is the idea that different AVs can be "Good For" different purposes. The catalog currently identifies several (not mutually exclusive) "Good For" labels.

—Teaching the Concept. An instructor could use this visualization as part of a lecture or give instructions in a homework/lab assignment for students to follow. It must include a good default dataset that shows some interesting (or at least typical) behavior of the algorithm. These AVs tend to be conceptual rather than focused on implementation.

—Exploring the Concept. As students gain knowledge about the concept, this visualization can be used to experiment on their own without explicit direction. Often, it allows users to enter their own data. It might be conceptual or focused on implementation.

—Debugging. The AV serves to illustrate the exact steps that a student's code should follow or outcome that it should produce. The student can then debug her output by comparing it to the visualization. These AVs tend to focus on implementation, but their output may be conceptual or machine oriented.

On rare occasion, an AV might support an API or mechanism for plugging into student code, so students can see visually what their code is doing in the machine.

—Comparison. Students or instructors could use the AV to illustrate the differences and similarities between competing data structures or algorithms that operate in the same domain.

—Lecture Aid. The AV is best used during lecture by an instructor who is using it to illustrate an algorithm that he or she is simultaneously explaining to the class. Without this accompanying explanation by an instructor, the AV itself does not sufficiently explain or make clear the algorithm that it portrays.

—Lab Exercise. The AV offers ways of interactively testing the learner's understanding of the algorithm. Examples would be pop-up questions that ask the viewer to predict what will happen next in the algorithm's execution or the opportunity to create input data sets that exercise the algorithm in some prescribed fashion.

—Self Study. The AV is provided in the context of supplementary text that describes the algorithm in a fashion that could replace a textbook explanation of the algorithm.

The "ActivityLevel" field indicates whether the AV is purely an animation or whether the user has control over the pacing; whether the data sets are predetermined, random, or input by the user; and whether the user answers questions or otherwise provides interactive feedback.

Originally, the catalog had only a single "Description" field where subjective and objective information intermingled. As we added more links to the collection, it became apparent that these two were quite distinct and should be recorded as such. "Description" has been reduced in scope to only objective, descriptive text about an AV. "Evaluation" was added to capture more subjective statements about AVs. Even if a user of the catalog disagrees with the subjective evaluation of the AV, she can be sure that she is also receiving an objective description and make up her own mind.

3.3 Populating the Catalog

There are two distinct ways to go about finding an AV on a given topic. One is to "Google for it", that is, use a favorite Internet search engine to search on keywords that will hopefully find what you need. The other is to look in an AV collection, a courseware repository, or a curated link collection.

Assume that there exists on the Internet a suitable AV on a specific topic. In that case, one hopes that standard Internet search technology will allow educators to find it. If so, this might alleviate the need to create and maintain specialized repositories or link collections for courseware in general, and AVs in particular. Unfortunately, whether any given instructor will be able to find an existing artifact depends a lot on the instructor's ability to supply the right keywords to yield successful results.

Internet search keywords need both to capture the desired artifacts (known as *recall*) and avoid generating overwhelming numbers of false-positive

responses (known as *precision*). Therefore, keywords must identify both the type of material desired (AVs as opposed to syllabi, static tutorials, or project descriptions), and the topic or content area desired. Some keywords, while technically specific, might lead to a wealth of non-related information. For example, looking for "Huffman Trees" is likely to give results related to the data structure, while looking for terms such as "lists," "queues," "sequences," or "trees" is likely to return information unrelated to computer science. Unfortunately, some data structures have common, everyday names.

AVs constitute only a small part of the courseware materials available on the Internet. Far more artifacts can be described as content presentations (lecture materials or tutorials), projects or exercises (assignments), or course management materials such as syllabi. Thus, to find a given AV, restrictive keywords are necessary for searches on most topics. Unfortunately, the providers of "visualizations" often do not label them as such, nor is there any standard alternative synonym that captures the typical visualization, although "animation" sometimes works (especially for older systems). Since the vast majority of AVs written since the mid-1990s were written in Java, and many of these are delivered as applets, "applet" is often a successful choice of keyword. Unfortunately, this will tend to leave out those AVs that exist within a visualization system, since they are not typically presented to the world labeled as "applets."

In one experiment, we counted the number of sorting AVs that appear in the first 30 responses from Google searches on various keywords. "Sorting visualization" yielded only seven AVs, while "sorting applet" yielded 20, and "sorting animation" yielded 21.

Unfortunately, using "applet" as a keyword results in generating a self-fulfilling prophecy in that if you search for applets, then the only AV presentation mechanism you will find will be applets. Initially, this skewed the balance of the materials found in our catalog away from projects with integrated applications, since non-applets were intrinsically harder to find. Since then, a conscious effort has been made to catalog non-applets, and these now make up a significant fraction of the current total in our catalog.

The main alternative to keyword-based Internet search is to look in courseware repositories or link collections. Unfortunately, there is currently little in the way of good repositories for AVs. Repositories were not a significant factor in developing the AlgoViz catalog, although we believe their potential importance could be great.

We used a number of techniques to locate AVs for the catalog. We began with a list of all AV systems that we were aware of from our general knowledge of the field. We developed a topic list based on our experiences teaching relevant courses. We considered what search terms would be most productive for locating AVs via Internet searches. Based on our topic list, we then performed searches using Google to find whatever we could. Once we had generated a base of AV links, we then examined these pages to try to locate other AVs, since developers of a given AV often have others available. Sometimes we could find these other AVs from direct references on the pages we already had, and other times we could deconstruct the URLs to find more AVs. Whenever we came

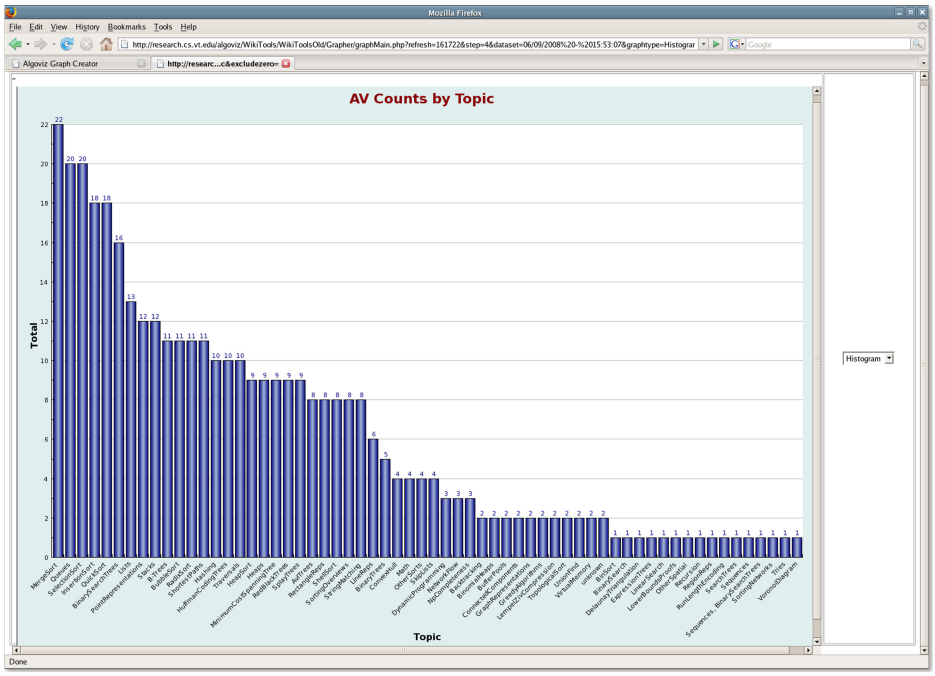

Fig. 1. Sample graph output from the query tool showing AV counts by topic.

across a page that had links to collections of AVs, we would follow those links to capture any new ones not yet in our collection.

3.4 Analysis Tools

We created a number of tools to automatically extract data from the AlgoViz catalog, including a catalog query tool and a link checker. The query tool loads each AV entry in our catalog and reformats its data as a row in a comma-separated values file. The user can make queries against this file that cause the tool to return a histogram of the AV entries that match the query. Figure 1 shows a sample graph generated by the tool.

Our link checker tests each link in the catalog and records its HTTP status code. This allows us to track AV disappearance over time. While there are a variety of possible HTTP codes, we only need to separate Web server responses into three buckets. “200 OK” means the page was found normally; “301 Moved Permanently” and “302 Found” mean that the page has been moved (with redirection to the new location). Any other code (in particular those with return values of 0 or 404) indicates that access to that AV is likely lost. The link checker allows us to track data about the rate at which AVs are lost, as reported in Section 4. We run our link checker once per week and archive the results.

4. THE STATE OF THE PRACTICE

This section reports our findings regarding the state of the practice in AV development, derived from analyzing the data contained in the AV catalog. From data generated using the catalog, we seek an understanding of the overall health of the field.

4.1 How Many Are Available?

As of March, 2010, the collection contains links to more than 500 AVs. Many of these are individual applets or programs, but a significant fraction appear as parts of integrated AV collections (see "Who Makes Them?" below).

We found considerable difficulty in deciding what to count as separate AVs for the purpose of defining catalog entries. Some collections come packaged as single applets or applications containing a number of different visualizations (perhaps a collection of sorting algorithm visualizations packaged along with search tree data structure visualizations). If a given applet or program contains multiple AVs (for example, a single Java applet that embodies separate AVs for various sorting algorithms), it is counted multiple times—once for each distinct AV.

At the other extreme, separate URLs might be provided for what is otherwise the same visualization program presented in different languages, or for examples of the same visualization run on different input data. We classify these under a single entry. Some "gray area" cases are difficult to assign. For example, we decided to provide a single entry under "quadratic sort" for systems that provide insertion, selection, and bubble sorts, since many systems do provide all three.

Although it is difficult to accurately define (much less determine) how many distinct programs or pieces of software are represented, one way of determining this information is to count the number of unique links in the collection. By this metric, there are more than 450 distinct programs.

These totals do not include many older systems like XTANGO and Balsa. Due to difficulties in getting these older systems to run in modern computers, we have not yet integrated their AVs into the main part of the catalog, although the wiki does contain information about many of these systems. Cataloging the AVs embodied in older systems would add over one hundred more entries.

Given the level of effort that we have devoted to populating the catalog (see Section 3.1), the amount of publicity that the catalog has received, and the number of AV developers and users that we have interacted with over the past four years, we believe that we have so far captured the bulk of what is publicly available, including the vast majority of easily found, better-known AVs. The search procedure models the process an educator would employ for locating an AV. If an AV cannot be found by our methods, it seems unlikely that an instructor would easily locate it via standard Internet search engines either. We continue to actively collect new links.

4.2 How Are They Implemented?

Virtually all AVs and AV toolkits that we located with creation dates since the mid-1990's were implemented in Java. Just over two thirds of available AVs can be viewed directly in a Web browser. Most often they are delivered as applets or initiated using Java Webstart, but a few are presented using Flash, Javascript, or other mechanism through a Web page. Of the other third, nearly all are Java applications that must be downloaded and opened locally, while only about three percent of AVs developed since the mid-1990s are implemented any other way.

We believe that AVs directly available on Web pages will get more attention from potential users, since they need not go through the additional step of downloading and unpacking an AV or AV system. Thus, these are more likely to be linked by others, resulting in a higher profile among search engines. On the other hand, a significant number of links in the collection were located via AV link collections or AV systems provided as downloads.

4.3 How Are They Disseminated?

Almost none of the AVs in the catalog are available from large, organized repositories. Many AVs are cataloged by link sites, such as the AlgoViz catalog, that attempt to link to AVs that the site's managers consider "worthy." However, most of these link sites are small in scale, linking to perhaps 10–20 favorite AVs for some course or textbook. A small number of efforts exist to produce comprehensive catalogs of AVs. These include the Hope College collection [Hope College 2001] (last edited in 2001) with approximately 100 entries, and Guido Rössling's collection [Rössling 2006] (last edited in May 2006), containing 233 entries. The AlgoViz catalog now incorporates nearly all entries from Rössling's collection that are still available on the Internet.

4.4 What Are They About?

The catalog's topic list was originally seeded with subjects typically covered in freshman and sophomore courses on data structures and algorithms. Over time, we have widened the scope of topics to accept everything that might be considered computer science. However, we have found that few AVs have been created on topics other than traditional data structures and algorithms. While we have found AVs covering areas such as networks, operating systems, programming languages, and numerical algorithms, they are rare. Our top-level categories for grouping AVs, along with their current population counts, are given in Tables I and II. Each major category shows the total number of corresponding AVs in the catalog, along with a breakdown into major subcategories.

As Figure 1 and Tables I and II show, there is wide variation in level of coverage. About 9% of all AVs demonstrate linear structures such as stacks and queues, even though these probably present less difficulty to students than many other topics. Over a quarter of all collected AVs demonstrate sorting

Table I. Counts by Major Category (with Significant Subcategories) for Visualizations of Data Structures in the AlgoViz Catalog

Linear Structures	46	
Lists		13
Stacks & Queues		32
Search Structures	76	
Binary Search Trees		16
AVL Trees		8
Splay Trees		9
Red-Black Trees		13
B-Trees and variants		17
Skiplist		6
Spatial Search Structures	31	
Point representations		13
Rectangle representations		10
Other Data Structures	25	
Heap/Priority Queue		11

Table II. Counts by Major Category (with Significant Subcategories) for Visualizations of Algorithms in the AlgoViz Catalog

Search Algorithms	18	
Linear/Binary Search		5
Hashing		11
Sort Algorithms	130	
Sorting Overviews		8
Quadratic Sorts		25
Shell Sort		13
Quicksort		24
Mergesort		25
Heapsort		16
Radix and Bin Sort		14
Graph Algorithms	68	
Traversals and Searches		15
Shortest Paths		20
Spanning Trees		17
Network Flow		4
Compression Algorithms	18	
Huffman Coding		13
Networking & OS	10	
Dynamic Programming	9	
Computational Geometry	20	
String Matching	6	
$\mathcal{NP}$-complete Problems		9
Other Algorithms	47	
Recursion & Backtracking		10
Mathematical Algorithms		8

algorithms. While sorting is a key topic for undergraduate data structures and algorithms courses, this disproportionate representation overstates its importance. Further, many of these sorting AVs are variations on the classic "Sorting Out Sorting" video [Baecker and Sherman 1981] and merely show bars being swapped. In contrast, most specialized and advanced topics are poorly covered.

Table III. Counts for Projects or Institutions Producing Significant Numbers of AVs

"One-offs" (1–5 visualizations)	161		31%
Small shops (6–10 visualizations)	52		10%
Algorithms In Action Project (Melbourne)		6	
Borowski's Sorting Demos		6	
University of Pittsburgh		6	
JAVENGA		6	
Michael Goodrich's Collection		7	
University of Patras		7	
Kovac's Tree Project		7	
JCAT		7	
Prolific teams (10+ visualizations)	127		25%
Jacobs' Animated Lectures		12	
University of Oldenburg/OLLI		13	
University of Auckland		15	
IIT Kanpur		18	
Virginia Tech		19	
University of Canterbury		22	
University of Maryland Spatial Data Structures		28	
Major Visualization Project	182		35%
JAWAA and JFLAP (Duke University)		11	
IDSV (University of Maryland, University College)		12	
Data Structures Navigator (Phillips-University, Marburg)		15	
JHAVÉ (University of Wisconsin, Oshkosh)		22	
TRAKLA2 (Helsinki University of Technology)		28	
ALVIE (University of Florence)		30	
Animal (TU Darmstadt)		31	
Data Structures Visualization (University of San Francisco)		33	

For instance, only 18 AVs (3.5%) cover compression algorithms and 13 of those are on Huffman coding. Even worse, a fundamental topic for upper-division courses like $\mathcal{NP}$-completeness gets attention in only 9 AVs ($< 2\%$). This imbalance shows a need for new AVs that address underrepresented areas.

4.5 Who Makes Them?

Just under one third of all AVs in our catalog are essentially single efforts by their respective authors. Another 10% are provided by "small shops" that have created 5–10 AVs, mostly by hand, often as individual Java applets. These might have each been created by the same individual over some number of years (typically a faculty member who is teaching relevant courses), or they might have been developed by a small number of students working for a faculty member. About 60% of all AVs in our catalog are created by groups who have developed over 10 AVs. In the majority of these cases, they are part of an AV development system, but a significant minority are parts of collections we cannot characterize as a system. Table III lists the major developers of AVs since the mid-1990s.

In addition to categorization problems for how to define distinct entries in the catalog as described above, there is also the question of when to draw the

Table IV. Year of Last Change by Project

Year	96	97	98	99	00	01	02	03	04	05	06	07	08
Project	1	1	1	3	4	2	3	2	1	1	3	1	8
Smaller	16	16	13	14	5	6	7	2	6	10	6	1	3

line on what is an AV and what is static course material. We are still struggling with how to decide this. We also note that because our search procedure consisted of a great deal of manual link following and URL manipulation, once a single AV from a collection or small shop was added to the catalog, the entirety of that collection was soon added. This ensures that we probably have cataloged AVs from the groups and projects with at least some visibility. The AVs we missed are presumably heavily skewed towards one-offs and smaller efforts.

4.6 When Were They Made?

Some well-known systems were developed in the early 1990s for creating AVs [Brown and Sedgewick 1984; Stasko 1992; Stasko and Kraemer 1992; Hansen et al. 2000]. However, most of these are now no longer available or so difficult to access due to changes in computer operating systems that they are not currently a factor in education. Considering primarily the development of AVs since the mid-1990s (i.e., Java), AV development as indicated by last-changed dates appears to be continuing relatively evenly over the years. Table IV shows a table of the last-change dates for those AVs in the catalog for which the information is available. These counts are "by project" rather than by individual AV, in that if a given AV or AV system provided visualizations for multiple algorithms, then it only counts once in the table. Note that AV developers rarely do a good job of documenting their development histories, and so there is a lot of inference and some guessing on these dates. While these data appear to show a decrease after 1999, it is important to look at the effects of large projects versus smaller "one-off" AV production. The large numbers in the late 1990s mostly come from individual AVs, rather than from major projects or active groups. The number of smaller efforts has dropped off in later years. In contrast, the last-changed dates for larger efforts is roughly steady, with a number of large projects still active. The big increase recorded for 2008 merely indicates that a number of projects have not stopped maintaining their software. Unfortunately, ongoing projects do not appear to be attuned to addressing gaps in topical coverage.

We speculate that students were readily available to create AVs during the Java boom of the mid to late 1990s, when Java and applets were new and the "in thing" for students to learn. But now there are other "in things" competing for students' attention. The nationwide drop in computer science enrollment might also mean that there are fewer students available to do such projects. This is not necessarily a bad outcome, as such student-driven one-off implementations tend to be of low pedagogical value. Larger projects appear to be ongoing at rates similar to 10–15 years ago.

4.7 Will We Find Them Again?

Like everything on the Internet maintained by individuals, AVs have a turnover rate in their accessibility. To measure this, we track the status of the links on our catalog. Each week, a snapshot of each link's accessibility is recorded. These weekly checks have been performed since May 2007. At various points in time, we have analyzed the link data and attempted to recover AVs at broken links. For instance, as of August 2009, there were 382 unique links in our catalog (multiple entries in the catalog can share the same link). Of these, 60 had become lost at some point (15.7%). Fortunately, many of those links were recoverable with some searching—they had been moved but their old links were not redirected. Through manual efforts to locate and restore lost links (including queries to the authors in some cases), we were able to recover 49 of the 60. Thus, of the 382 unique links in the catalog in August, 2009, 11 (2.9%) were lost permanently (so far as we know) and 49 (12.8%) moved without redirection but were found again through manual intervention. As of March 2010 we had 455 unique links in the catalog. Of these, 20 (4.4%) were to AVs that we could no longer locate, representing an additional loss of nine AVs in six months.

4.8 How Are They Licensed?

All AVs in the catalog are freely available for educators to use. Although this might be an artifact of the search process, we have yet to find any non-gratis AV or system. 57% of all AVs provide access to the source code in some way. Of 447 AVs in our collection for which we have identified the status of their source code availability, 193 (43%) do not make source code available; 68 (15%) make the source code available with no defined license beyond possibly a copyright notice in the code itself; 140 (31%) attempt to place some sort of recognizable license restriction on the code (but only 80 of these do it in any formal way, such as citing the GNU General Public License (GPL)), 44 (10%) say sourcecode is available by special request, and one AV explicitly puts its source code in the public domain.

Many of the source-available AVs seem to lack a license not because the developers wish to prevent redistribution, but due to a lack of understanding of the process. It is reasonable to expect that most AV developers who provide access to their source code would willingly make it officially open source if there were greater awareness on their part of the process and advantages for doing so. And it is possible that many who do not make source code available would be willing to do so if they knew a simple mechanism for putting it under open source license.

5. CONCLUSIONS

While many good AVs are available, the need for more and higher quality AVs continues. There are many topics for which no satisfactory AVs are available. This is especially true for more difficult material such as B-trees or $\mathcal{NP}$-completeness. As yet there seems to be no organized movement to satisfy this

need. On the other hand, the theoretical foundations for creating effective AVs appear to be steadily improving. More articles are appearing regarding effective use of AVs in courses, and a body of work has begun to form regarding how to develop effective AVs in the first place. While more fundamental research on how to develop and use AVs is necessary, we also simply need more implementers to produce more quality AVs. Once these AVs are created, educators must be able to find them.

Currently, much AV development seems to occur in a social vacuum. While the larger developers are aware of each other's efforts (as evidenced by participation in activities such as ITiCSE working groups [Naps et al. 2002, 2003; Rössling et al. 2006]), most smaller developers of AVs do not seem aware of lessons learned from effectiveness studies. Instead, they seem to be repeating the same mistakes as their predecessors. Additionally, it is not easy for users to find guidance on issues such as how to find the best AVs to use, and what are pedagogically effective ways to use them. Our proposed remedy for this situation is to bring various AV stakeholders—developers, educators, researchers, and end users—together into a community. We believe that enhanced communication will lead to dissemination of knowledge and good ideas, higher quality AVs, and better educational outcomes from the use of AVs.

What will it take to create a community of AV stakeholders? We believe that three things are necessary (or at least desirable): opportunities for interaction, a good courseware repository, and up-to-date statistics.

While digital communities probably will never replace the traditional birds-of-a-feather sessions and conference workshops at conferences such as SIGCSE and ITiCSE, having an online place for informal discussion and knowledge dissemination should help improve the field [Harrison and Dourish 1996]. Online communities typically facilitate asynchronous communication using discussion forums and/or e-mail listservs.

Educators also need a dependable collection of AVs to draw on, either directly in service of teaching or as a base for creating new AVs. Small-scale developers of AVs would benefit from access to existing AV implementations for ideas and perhaps for code to quickly get their own projects prototyped. The computer science education community has created some courseware repositories, such as the collection of materials submitted to JERIC [JERIC 2008] (now renamed *The ACM Transactions on Computing Education*) and the CITIDEL repository [CITIDEL 2007]. In particular, since the advent of the ACM Digital Library [Association for Computing Machinery 2010], CITIDEL (now being replaced by the Ensemble project) has repositioned itself from a collection of mainly publications (i.e., journal articles and conference papers) to a repository for educational resources. *The ACM Transactions on Computing Education* and its contributed courseware are indexed as part of the ACM Digital Library, hypothetically allowing for easy searching. While the ACM Digital Library is a huge collection, it does not appear to contain significant amounts of courseware in general or AVs in particular. Equally important, it does not provide adequate search tools for courseware or AVs. The bulk of ACM's materials are papers, typically organized by publication venue. There is no support for browsing of courseware separate from the (overwhelming) body of noncourseware

content. SIGCSE provides a small but growing collection of links to CS-related educational software [SIGCSE 2010]. Broader courseware repositories include SMETE [SMETE 2010], and Connexions [Connexions Scholarly Content Repository 2010]. While these are promising beginnings, current incarnations of courseware repositories are small in scope and in visibility. None of these repositories have large collections of AVs. Further, in all the hours we have spent conducting Google searches for various AVs, not a single AV within any of these repositories was discovered. Thus, these repositories appear to provide no search engine visibility for the few AVs they might contain.

Section 4 gave a detailed overview of the current state of practice within the field of algorithm visualization. However, much of the information was (at least partly) manually derived, a labor-intensive process. Worse, much of that information will soon be out of date. The community needs dynamically updated statistics similar to those contained in Section 4. We have begun to develop software for extracting such data in the form of graphing tools for exploring the our AV catalog, but there continues to be room for more and deeper automated analysis.

By providing dynamic statistics about the collection in the catalog, we make it possible for developers to choose topics and features based on evidence instead of speculation. In addition to numbers, developers might find commentary from forums/listservs to be valuable as well. Similarly, educators can get reports from fellow educators about which AVs work and how to successfully integrate them into a curriculum. Any opinions, experiences, and lessons learned from others can save time and allow educators to make better choices when picking AVs to incorporate. The difficulty comes with establishing a community resource where users and developers can gather and share their experiences.

A potentially powerful way to provide information to educators and to influence developers is to provide a ratings system for existing AVs. Ratings can come from two fundamental sources. The first is from a source of "expertise" such as catalog editors or curators of a site such as ours. Our catalog provides a coarse editorial rating structure of "Recommended," "Has Potential," and "Not Recommended." Over three quarters of the AV entries have been rated as of this writing. Of these, 18% are ranked as "Recommended," 42% are ranked as "Has Potential," and 30% are ranked as "Not Recommended." We plan as future work to do a formal study regarding the sensitivity of our current editorial rating process to subjective bias. That is, we will examine whether most knowledgeable people would agree with a given set of ratings. This should lead to improvements in our rating process.

A second source of rating information should be the broad community of instructors, developers, students, and other users of AVs. By allowing the community to rate AVs and provide comments and feedback, we would hope that developers will respond to requests for improvement, or at least observe what has proved successful in the past before they develop their next AV.

The AlgoViz Wiki was created with the goal of supporting the needs of an Algorithm Visualization community. The Wiki has been under continuous development since 2006. It was advertised at SIGCSE 2009 and to the SIGCSE Listserv multiple times during 2009. A major change began in Summer 2009

when we developed the initial version of the AlgoViz Portal.[1] The AlgoViz Portal is implemented in the Drupal content management system, and is expected to completely replace the MoinMoin implementation during 2010. While not technically a wiki, the AlgoViz Portal implementation provides many of the same features (such as user-editable pages) and more (user forums and a searchable bibliography). The AlgoViz Portal seeks to provide many opportunities for community involvement in the site.

ACKNOWLEDGMENT

We thank an anonymous associate editor for suggestions on how to create a more rigorous framework for evaluating our ratings process.

REFERENCES

ALGOVIZ WIKI. 2010. Data structures and algorithm visualization wiki. http://web-cat.cs.vt.edu/AlgovizWiki.

ALGOVIZ.ORG. 2010. Annotated bibliography of the AV research literature. http://algoviz.org/biblio.

ASSOCIATION FOR COMPUTING MACHINERY. 2010. The ACM digital library. http://portal.acm.org.

BAECKER, R. AND SHERMAN, D. 1981. Sorting out sorting. Video.

BRABEC, F. AND SAMET, H. 2003. Maryland spatial index demos. http://donar.umiacs.umd.edu/quadtree/.

BROWN, M. 1992. An introduction to Zeus. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI'92)*. 663–664.

BROWN, M., NAJORK, M., AND RAISAMO, R. 1997. A java-based implementation of collaborative active textbooks. In *Proceedings of the IEEE Symposium on Visual Languages (VL'97)*. 372–379.

BROWN, M. H. AND SEDGEWICK, R. 1984. A system for algorithm animation. In *Proceedings of the 11th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH'84)*. 177–186.

BYRNE, M. D., CATRAMBONE, R., AND STASKO, J. T. 1996. Do algorithm animations aid learning? Tech. rep. GIT-GVU-96-18, Georgia Institute of Technology.

CITIDEL 2007. Computing and information technology interactive digital educational library. http://www.citdel.org.

CONNEXIONS SCHOLARLY CONTENT REPOSITORY. 2010. http://cnx.org.

CRESCENZI, P. 2010. Alvie 3.0. http://alvie.algoritmica.org/.

CRESCENZI, P., GAMBOSI, G., AND GROSSI, R. 2006. *Strutture di Dati e Algoritmi*. Pearson Education Addison-Wesley.

DIEHL, S. 2007. *Software Visualization: Visualizing the Structure, Behavior, and Evolution of Software*. Springer.

DITTRICH, J.-P., VAN DEN BERCKEN, J., SCHÄFER, T., AND KLEIN, M. 2000. Data structure navigator. http://dbs.mathematik.uni-marburg.de/research/projects/dsn/.

GALLES, D. 2006. Data structure visualization. http://www.cs.usfca.edu/galles/visualization/.

GRISSOM, S., MCNALLY, M., AND NAPS, T. 2003. Algorithm visualization in CS education: Comparing levels of student engagement. In *Proceedings of the ACM Symposium on Software Visualization (SoftVis'03)*. 87–94.

GURKA, J. AND CITRIN, W. 1996. Testing effectiveness of algorithm animation. In *Proceedings of the IEEE Symposium on Visual Languages (VL'96)*. 182–189.

HANSEN, S., NARAYANAN, N., AND SCHRIMPSHER, D. 2000. Helping learners visualize and comprehend algorithms. *Interact. Multimedia Electron. J. Comput.-Enhanc. Learn. 13*, 3, 291–317.

[1]http://algoviz.org

HARRISON, S. AND DOURISH, P. 1996. Replacing space: The roles of place and space in collaborative systems. In *Proceedings of the ACM Conference on Computer Supported Cooperative Work (CSCW'96)*. 67–76.

HOPE COLLEGE. 2001. Complete collection of algorithm visualizations. http://www.cs.hope.edu/~dershem/ccaa/ccaa.

HUNDHAUSEN, C. AND DOUGLAS, S. 2000. Using visualizations to learn algorithms: Should students construct their own, or view an expert's? In *Proceedings of the IEEE Symposium on Visual Languages (VL'00)*. 21–28.

HUNDHAUSEN, C. D., DOUGLAS, S. A., AND STASKO, J. T. 2002. A meta-study of algorithm visualization effectiveness. *J. Vis. Lang. Comput. 13*, 3, 259–290.

JARC, D. J., FELDMAN, M. B., AND HELLER, R. S. 2000. Assessing the benefits of interactive prediction using Web-based algorithm animation courseware. In *Proceedings of the 31st SIGCSE Technical Symposium on Computer Science Education (SIGCSE'00)*. 377–381.

JERIC 2008. *J. Educ. Resour. Comput.* http://www.acm.org/pubs/jeric.

KORHONEN, A., MALMI, L., AND SILVASTI, P. 2003. Trakla2: A framework for automatically assessed visual algorithm simulation exercises. In *Proceedings of the Koli Calling 3rd Annual Baltic Conference on Computer Science Education (KOLI-CALLING'03)*.

LAWRENCE, A. W., STASKO, J., AND BADRE, A. 1994. Empirically evaluating the use of animations to teach algorithms. In *Proceedings of the IEEE Symposium on Visual Languages (VL'94)*. 48–54.

NAPS, T., COOPER, S., KOLDEHOFE, B., LESKA, C., RÖSSLING, G., DANN, W., KORHONEN, A., MALMI, L., RANTAKOKKO, J., ROSS, R., ANDERSON, J., FLEISCHER, R., KUITTINEN, M., AND MCNALLY, M. 2003. Evaluating the educational impact of visualization. *SIGCSE Bull. 35*, 4, 124–136.

NAPS, T., EAGAN, J., AND NORTON, L. 2000. JHAVÉ—an environment to actively engage students in Web-based algorithm visualizations. In *Proceedings of the 31st Technical Symposium on Computer Science Education (SIGCSE'00)*. 109–113.

NAPS, T., RÖSSLING, G., ALMSTRUM, V., DANN, W., FLEISCHER, R., HUNDHAUSEN, C., KORHONEN, A., MALMI, L., MCNALLY, M., RODGER, S., AND ÁNGEL VELÁZQUEZ-ITURBIDE, J. 2002. Exploring the role of visualization and engagement in computer science education. In *Proceedings of the Working Group Reports from ITiCSE on Innovation and Technology in Computer Science Education (ITiCSE-WGR'02)*. 131–152.

PIERSON, W. AND RODGER, S. 1998. Web-based animation of data structures using jawaa. In *Proceedings of the 29th SIGCSE Technical Symposium on Computer Science Education (SIGCSE'98)*. 267–271.

RÖSSLING, G. 2006. Animation repository. http://www.animal.ahrgr.de/animations.php.

RÖSSLING, G., NAPS, T., HALL, M., KARAVIRTA, V., KERREN, A., LESKA, C., MORENO, A., OECHSLE, R., RODGER, S., URQUIZA-FUENTES, J., AND ÁNGEL VELÁZQUEZ-ITURBIDE, J. 2006. Merging interactive visualizations with hypertextbooks and course management. *SIGCSE Bull. 38*, 4, 166–181.

RÖSSLING, G., SCHÜER, M., AND FREISLEBEN, B. 2000. The animal algorithm animation tool. In *Proceedings of the 5th Annual SIGCSE/SIGCUE ITiCSE Conference on Innovation and Technology in Computer Science Education (ITiCSE'00)*. 37–40.

SARAIYA, P. 2002. Effective features of algorithm visualization. M.S. thesis. Virginia Polytechnic Institute and State University.

SARAIYA, P., SHAFFER, C., MCCRICKARD, D., AND NORTH, C. 2004a. Effective features of algorithm visualizations. In *Proceedings of the 35th SIGCSE Technical Symposium on Computer Science Education (SIGCSE'04)*. 382–386.

SARAIYA, P., SHAFFER, C., MCCRICKARD, D., AND NORTH, C. 2004b. Effective features of algorithm visualizations. In *Proceedings of the 35th SIGCSE Technical Symposium on Computer Science Education (SIGCSE'04)*. 382–386.

SHAFFER, C. A., COOPER, M. L., AND EDWARDS, S. H. 2007. Algorithm visualization: A report on the state of the field. In *Proceedings of the 35th SIGCSE Technical Symposium on Computer Science Education (SIGCSE'07)*. 150–154.

SHAFFER, C. A., HEATH, L., AND YANG, J. 1996. Using the Swan data structure visualization system for computer science education. In *Proceedings of the 35th SIGCSE Technical Symposium on Computer Science Education (SIGCSE'96)*. 140–144.

SIGCSE 2010. External links. http://sigcse.org/resources/external-links.

SMETE 2010. Digital library. http://www.smete.org.

STASKO, J. 1992. Animating algorithms with XTANGO. *SIGACT News 23*, 2, 67–71.

STASKO, J. 2001. POLKA animation system. http://www.cc.gatech.edu/gvu/softviz/parviz/polka.html.

STASKO, J., DOMINGUE, J., BROWN, M. H., AND PRICE, B. A. 1998. *Software Visualization: Programming as a Multimedia Experience*. MIT Press, Cambridge, MA.

STASKO, J., KEHOE, C., AND TAYLOR, A. 2001. Rethinking the evaluation of algorithm animations as learning aids: An observational study. *Int. J. Hum.-Comput. Stud. 54*, 2, 265–284.

STASKO, J. AND KRAEMER, E. 1992. A methodology for building application-specific visualization of parallel programs. Tech. rep. GIT-GVU-92-10. Georgia Institute of Technology.

STERN, L. 2001. Algorithms in action. http://www.cs.mu.oz.au/aia/.

WALDMANN, T. AND HERMANN, J. 2010. MoinMoin Wiki Engine. http://moinmoin.wikiwikiweb.de/.

WIGGINS, M. 1998. An overview of program visualization tools and systems. In *Proceedings of the 36th Annual Southeast Regional Conference (ACM-SE'98)*. 194–200.

Received June 2009; revised August 2009, March 2010, April 2010; accepted April 2010