JHAVÉ: Supporting Algorithm Visualization

Thomas L. Naps
University of Wisconsin, Oshkosh

In his keynote address at the 2005 ACM Special Interest Group on Computer Science Education (SIGCSE) Symposium, Mordechai Ben-Ari lamented:

> From reading press releases, you would think that we experience true revolutions every day. I used to read profiles of startup firms in the newspaper, and invariably, the description of the activities of the startups included the phrase: "X has developed a technology to do Y." As far as I could tell, what they did was to write a program, which is hardly a new technology or a true revolution.[1]

Those of us who use graphics in our teaching of computer science concepts might have indeed fallen prey to this same tendency—that is, overhyping something that is state of the art. In particular, when Marc Brown's dissertation on algorithm visualization (AV) won the 1988 ACM Dissertation of the Year award, it seemed to many computer science educators that the dawn of an algorithms-made-simple era was just around the corner. If educators use computer graphics to depict an algorithm's actions, how could students not come to understand the algorithm more easily and in greater depth? Brown's work was brilliant from a computer graphics perspective. He set the stage for how to display exquisite algorithm visualizations with relative ease.

We appeared to have discovered a new technology (in the Ben-Ari sense) for teaching computer science. Researchers developed a variety of new systems, and Web applets that animated algorithms abounded. Many computer science instructors advised their students who were having trouble understanding algorithm X to go to URL Y and view a great animation that would help them study this difficult material. Before proclaiming how valuable this new technology was, however, we should have read with care the last paragraph in the appendix to Brown's thesis:

> The bottom-line question in the (Brown University) Electronic Classroom is "Do the (graphic) simulations help?" Unfortunately, no controlled experiments could be done to answer this credibly.[2]

Yes, these animations were great if you were a computer science professional, someone who already understood the algorithm's essentials before watching the animation. Such professionals could wax ecstatic about how the stunning graphics captured the essence of the algorithm. However, students attempting to learn the algorithm were often mystified by the high-powered display they saw unfolding in front of them. What had gone wrong? Developers of algorithm visualizations and the systems that facilitate them had too often showcased their work to professionals who already understood the algorithm rather than the intended audience of students trying to learn it. These professionals had the expert knowledge that made them savvy in using such tools to explore the algorithm's subtle nuances. But students lacked this base of experience in using a graphics tool to acquire understanding through exploration. Stern, Søndergaard, and Naish stated that the lack of success of AV in computer science education is because "many of the current algorithm visualizations concentrate on graphics rather than on pedagogy."[3]

JHAVÉ fosters the use of algorithm visualization as an effective pedagogical tool for computer science educators, helping students to better understand algorithms.

Stanford University psychologist Barbara Tversky and her associates noted this inclination to concentrate on computer graphics instead of how to pedagogically employ the graphics: "The advances in the technology of producing attractive graphics often seem to drive and outstrip … the research on their utility."[4]

When does AV work?

As researchers, many of us have been lured into AV by our love of computer graphics, but as educators we now need to admit that AV researcher Chris Hund-

 0272-1716/05/$20.00 © 2005 IEEE

hausen is right when he writes: "The most significant factor (in learning) appears to be not what AV viewers see, but what they do (with the visualizations)."[5]

There is both good and bad news in Hundhausen's conclusion. The bad news is that educationally oriented AV researchers who want to concentrate on nothing but graphics are probably in the wrong field—education takes more than just graphics. The good news is encouraging, however—by guiding students into activities that engage them with the visualization, it appears that we really can improve how well they learn.

This conclusion was substantiated in a comprehensive metastudy conducted by Hundhausen, Douglas, and Stasko.[6] This metastudy examined 21 empirical evaluations of AV effectiveness that had been done by other researchers in the field. The metastudy's top-level conclusion appeared discouraging. Of the 21 experiments, only 13 showed that some aspect of AV significantly impacted learning. However, Hundhausen, Douglas, and Stasko probed a bit deeper and divided these experiments into two groups:

- Those characterized by the high-end nature of the graphical attributes of the visualizations watched by the students.
- Those that involved students more actively.

In the first case, the visualizations might offer simultaneous multiple views of the algorithm, slick cartoonlike animations instead of mere still pictures, and so forth. The second approach was characterized by students constructing their own input data for the algorithm, answering strategic questions about the algorithm, engaging in interactive predictions regarding future algorithm behavior, and/or programming the algorithm.

They found that 10 out of 12 (83 percent) experiments in the second group produced a significant result, but only 3 out of 9 (33 percent) in the first group could claim any significant effectiveness achieved by AV. Since the publication of this metastudy, I have been involved in two additional studies that would fit into the second group.[7,8] Both of these studies showed significant effectiveness, hence increasing the overall effectiveness rate for this group to 12 out of 14 (86 percent).

The message seems clear—if we guide students into additional activities that are synchronized with the visualizations, then positive results are likely. Because of this, in my own teaching, I gave up using AV displays in lecture settings after I observed students quickly becoming entranced by the movie they were watching as they lounged in those comfortable lecture hall chairs.

However, I observed a payback from the visualizations when students individually came to my office for some help with a particular algorithm. Then I could use my AV system to bring up repeated displays of the algorithm, walk the student through what the display was showing, and explore the algorithm's behavior with the student.

By the example I was setting for my students as I used the system with them, I could begin communicating with them about the significance of the pictures. The AV playing the role of instructional medium allowed me to guide the students toward understanding the algorithm thoroughly. When the students became confused, I could rewind the display to the point where they lost understanding and encourage them to take a longer look at what was about to happen as we started to view again this critical segment of the movie. I could ask students questions about the visualization and play what-if games with them. Through this intense combination of seeing pictures that were beginning to take on meaning and conversing about those pictures, students succeed in making the mental association between the conceptual algorithm and my graphical rendering of it.

Student interaction with AVs

Members of an Innovation and Technology in Computer Science Education (ITiCSE) working group on improving the educational impact of algorithm visualization produced a report codifying—in an engagement taxonomy—modes that students could become active participants in exploring an algorithm with an AV system.[9] The four active categories of engagement defined in that report are

- responding,
- changing,
- constructing, and
- presenting.

In the responding category, educators focus on having students answer questions concerning the visualization that the system presents. These questions can be predictive in nature, asking what the next frame of the animation will look like, or they can be more conceptual, asking students what code segments could have been used to execute the algorithm in the fashion they had just seen, for example. In the responding category, the visualization system becomes a supporting resource for posing questions and helping students answer them—the same role I played in my earlier description of using AV to help an individual student in my office.

In the changing category, students provide input to the underlying algorithm to make the visualization appear a certain way. For example, if students watch a visualization of binary search trees, the educator might direct them to provide a data set that will make the tree take on a certain shape. Or, if the students watch Dijkstra's shortest path algorithm, they might need to provide a data set in which the algorithm explores every vertex in the graph before the shortest path from the start to the goal can be found.

In the changing category of interaction, students use the visualization to hypothesize about the algorithm's behavior in different cases. Students provide the input prior to watching the algorithm and then view it to confirm whether or not the display progresses as they predicted. In effect, the AV system becomes a hypothesis checker for students—the same role I took on in the scenario of using AV to play what-if games with the individual student in my office.

In the constructing category, students construct their own visualizations of the algorithm under consideration. This entails a larger time commitment on the part

of the student. A common technique for constructing a visualization is to first code it and then annotate it with calls to have graphics produced at interesting events during the algorithm's execution. Well-designed class libraries can make this relatively painless for students to accomplish. For example, a data structure object such as a graph might have a display method that the algorithm can call to produce an aesthetically pleasing picture of the object in the AV system without requiring students to do much beyond a method call.[10] However, constructing the visualization does not always entail having the student code the algorithm. Systems such as Rößling's Animal Script[11] and Hundhausen's Alvis[12] provide visualization designers with a collection of modeling tools well suited to algorithm visualization. In effect they allow the user to create a movie about the algorithm working strictly from a conceptual perspective, without ever having to code the algorithm.

In the presenting category, students use a visualization to help explain an algorithm to an audience for feedback and discussion. The presenting student might or might not have created the visualization. According to John Stasko, both the constructing and presenting categories

> force a student to think about the fundamental operations of the algorithm. … The student constructs the algorithm-to-animation mapping and determines what is unique to the algorithm and what deserves to be communicated visually.[13]

JHAVÉ

In the previous sections I have argued that graphics alone are not sufficient to effectively deliver educational applications of algorithm visualization. Unfortunately, the extra tools needed for such effective deployment—namely, interesting hooks to actively engage the student with the visualization—often require more effort to produce than the graphic displays themselves. The Java-Hosted Algorithm Visualization Environment overcomes this impediment.[14] JHAVÉ is not an AV system itself but rather a support environment for a variety of AV systems (called *AV engines* by JHAVÉ). In broad terms, JHAVÉ gives such an engine a drawing context on which it can render its pictures in any way.

In return, JHAVÉ provides the engine with effortless ways to synchronize its graphical displays with

- a standard set of VCR-like controls,
- information and pseudocode windows,
- input generators,
- stop-and-think questions, and
- meaningful content generation tools.

The VCR-like controls let students step through the algorithm's visual display. Hence, the GUI has a standard look and feel that is independent of the AV engine in use.

Information and pseudocode windows are HTML windows where visualization designers can author static or dynamically generated content to help explain the significance of what the student sees in the algorithm's graphical rendering. The distinction between an information window and a pseudocode window is that the former shows higher level conceptual explanations of what is happening, and the latter displays a pseudocode description of the algorithm complete with highlighting of lines that are particularly relevant for the picture currently displayed.

Input generators are objects that gather input data from a student when using a visualization that engages users in the changing category described in the previous section. Students can feed this input data to the visualization, letting them learn whether their data set drives the pictures in the anticipated fashion.

The visualization designer can designate stop-and-think questions in a variety of formats—such as true-false, fill in the blank, multiple choice, and multiple selection (multiple choice with more than one right answer)—to pop up at the algorithm's key stages. The question will typically ask students to predict what they will see next, facilitating the responding category of engagement described in the previous section.

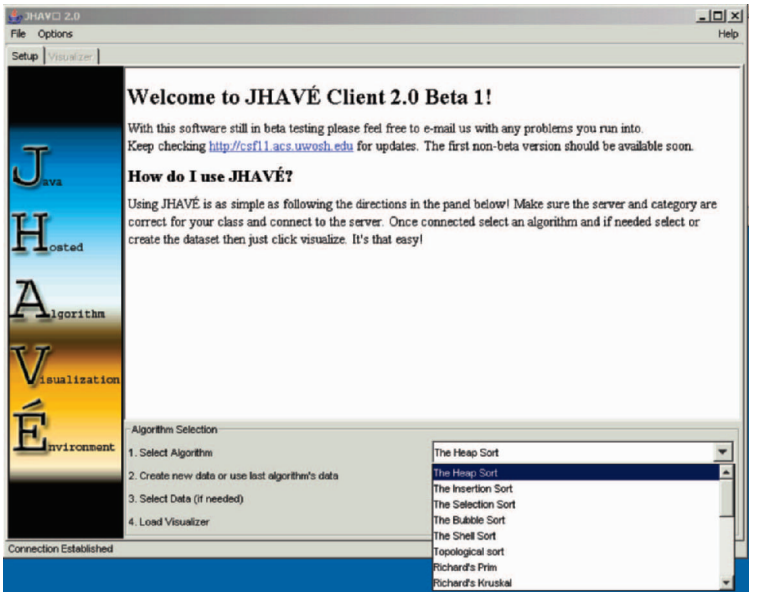
Meaningful content generation tools include a variety of class libraries to help an AV designer (who might be a teacher or a student) create a visualization, including its graphical display and the interaction support tools that JHAVÉ offers.

AV engines and JHAVÉ

What price does an AV engine pay for these benefits? First, it has to be written in Java and derived from an abstract base class called the `Visualizer` class. As a consequence of participating in this object hierarchy, the engine must know how to respond to a standard set of messages sent to it by JHAVÉ. Second, it must produce its visualization of an algorithm by parsing a textual visualization script and then rendering its pictures according to that script's content. Each AV engine is free to define its own scripting language—this is what allows a great deal of variation among the engines. With the script file methodology, a program implementing the algorithm to be visualized executes, and, instead of directly rendering a visualization of the algorithm, it writes visualization commands to the script file. When the algorithm terminates, the AV engine parses and renders this script file. The reasons for this are tied to the JHAVÉ's client–server architecture.

In this architecture, the server application manages the available algorithms and generates the visualization scripts that the client can display. In a standard session, a student first launches an instance of the client application, which displays a listing of available algorithms—tailored to the particular subject that the viewer is studying (see Figure 1, next page).

When the user selects an algorithm from that list, the client sends a request to the server. The server then runs a program that generates the script for that algorithm and sends a URL back to the client from which the user can read the script. The client then parses and renders the script with the appropriate engine. If the algorithm requires input from the user, the server sends an input-generator object to the client. This is just a frame with appropriate input areas for the user. Once the user fills out these areas, the client returns the user's input to the server as a data set to use when running the algorithm



1 Client initially connects to server. Menu of algorithms displayed on lower right corner.

2 Input generator gets data from user to direct the visualization.

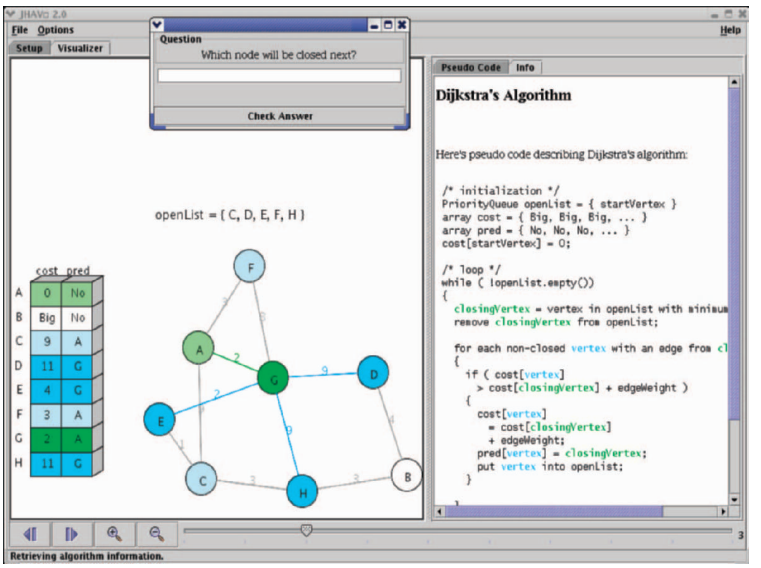
(see Figure 2). After the server generates the script, the client downloads, parses, and renders it. The rendering is staged using JHAVÉ's VCR-like viewing controls, stop-and-think questions, and information/pseudocode windows (see Figure 3).

Transforming renderers to JHAVÉ plug-ins

AV engines that presently plug into JHAVÉ as clients in the fashion described previously are Rößling's AnimalScript,[11] Stasko's JSamba (http://www.cc.gatech.edu/gvu/softviz/algoanim/jsamba),[15] and my own Generalized Algorithm Illustration through Graphical Software (GAIGS).[16] Each of these exists also as an AV system in itself, but in that existence outside of JHAVÉ they are not able to annotate their scripts with the engagement tools that JHAVÉ offers. In effect, they become mere renderers, and the students using them become passive viewers.

To become a JHAVÉ AV engine, a renderer must declare that it is a subclass of JHAVÉ's abstract `Visualizer` class. As a consequence of this, it then must inform JHAVÉ of its capabilities by calling the `setCapabilities` method of the `Visualizer` class. JHAVÉ allows a `Visualizer` to support any or all of the following capabilities:

- Stepping forward to the next picture in the visualization.
- Stepping backward to a prior picture in the animation.



3 AV engine renders picture in the left pane, with support from the JHAVÉ environment. The controls shown are for the Generalized Algorithm Illustration through Graphical Software (GAIGS) visualizer, allowing stepping forward and backward, direct access to a particular frame, and zooming.

- Going directly to a particular frame in the animation, skipping the rendering of all frames in between that frame and the current frame.
- Zooming in or out on the picture in the current visualization frame.
- Delivering a visualization as an animated motion picture rather than a slide show. Slide show mode is the default unless a `Visualizer` declares otherwise. JHAVÉ uses this information about the visualizer's capabilities to display the appropriate subset of VCR controls in its GUI for that visualizer (see Figure 3).

Next, the renderer must implement a constructor that receives an input stream as its only parameter. When JHAVÉ instantiates a particular visualizer object, it calls this constructor, passing in the animation script, which exists as a URL on the server, for the input stream parameter.

The renderer then implements three additional methods that are pure abstract methods in the base `Visualizer` class: `getCurrentFrame` returns the index number of the current frame in the visualization, `getFrameCount` returns the total number of frames in the visualization, and `getRenderPane` returns the Java component in which the engine renders the visualization's graphic artifacts.

For each capability that it declares itself capable of supporting, the renderer must override a method in the base `Visualizer` class to ensure the appropriate action is taken when the viewer uses the JHAVÉ GUI to trigger the event associated with that capability.

During the course of parsing a visualization script, if an AV engine encounters a tag for a question or a pseudocode and documentation window, it calls a `fireQuestionEvent` or `fireDocumentationEvent` method in the `Visualizer` base class. These methods handle everything connected with processing that event and hence free the particular AV engine from concerning itself about the details of parsing and displaying the information associated with them. Instead, the visualizer base class handles these tasks. The AV engine itself must worry about nothing but displaying graphics in its rendering pane.

Academic use

JHAVÉ has been used as the underlying AV vehicle in two published empirical studies conducted at three universities.[7,8] Its client–server architecture has proved remarkably resilient.

Because it is written in Java, the server application itself can be relocated from one operating system to another without making any changes. Programs that implement algorithms producing visualization scripts, however, can be written in any language. They are merely executed at the direction of the Java server application. This means that, under the JHAVÉ environment, instructors can provide their own visualizations for students without having to write them in Java. Hence there is a great deal of variation in the ways you can produce visualizations that ultimately are viewed in JHAVÉ. For example, prior to the advent of JHAVÉ, computer science instructors had written many programs in other languages that produced GAIGS, AnimalScript, and Samba visualization scripts. These instructors now must merely modify their old programs so that, inside the scripts they produced, annotations are inserted as to when information and pseudocode windows and stop-and-think questions should pop up during the course of the visualization.

This methodology also allows students in a course to write Web-viewable visualizations even though they are not programming in Java. The instructor can load the students' script-producing programs on the departmental Web server, and JHAVÉ can make the visualizations produced by these programs available for everyone. Moreover, having control of the script-producing programs on a central server makes it easier for a course instructor to keep materials current. All the students need to participate in a JHAVÉ-staged visualization is the relatively lightweight client that is easily deployed using Java's Web Start methodology (http://java.sun.com/products/javawebstart). Instructors can make last-minute additions and changes to particular visualizations on the server and be confident that no students in the course will have out-of-date material.

Conclusion and future directions

Other systems that directly support AV with hooks that force students into more active engagement with visualizations include Jarc's IDSV,[17] Hundhausen's Alvis,[12] and Laakso et al.'s Trakla2/Matrix.[18] Neither JHAVÉ nor any of these other systems that directly support AV with engagement hooks are in widespread use yet.

However, these systems bode well for the future because they all reflect an awareness of the maturation process that education-oriented AV has gone through in the 15 years since Marc Brown's dissertation. As often happens with eye-catching new technologies, we have gone through what in retrospect were predictable phases: An initial, almost giddy, awe at the remarkable graphic effects, followed by disenchantment that the visualizations did not achieve their desired purpose in educational applications. Then we went through a period of empirical evaluation in which we began to sort out what worked and what did not.

We are now ready to leverage that knowledge. Such leveraging will certainly involve some graphics development, but we must remember that graphics display techniques are not the missing piece in this puzzle. Rather, we suffer from a lack of those materials that would have students engage with visualizations in the context of the taxonomy described previously. AV researchers—particularly those oriented toward educational applications—must begin to examine many questions.

How can we enhance the type of responding from students when they interact with a visualization? The questions that students are asked when engaging in this category of the taxonomy are ultimately produced by software that supports the AV system. How can we make such machine-produced questions more meaningful in their content? Can we make the system sense a student's mastery of certain points and hence only generate questions directed toward areas where the student displays a weakness?

What teacher has not been asked, Will I need to know this for the test? The obvious implication on the part of the student is, If I'm not being graded on it, I do not have time to learn it. What does this scenario mean with respect to AV? In particular, we realize that many students are more intense in their level of engagement when they know that the responses are being evaluated. It would then seem to follow that some monitoring by the system of how students respond to questions when they work with an AV system would heighten their level of engagement—particularly if this monitoring played a small role in their grade. Yet the only empirical study done in this regard produced mixed results.[17] More work needs to be done here, but that will entail a lot of server-side database development to record how students are interacting with the system.

How can we make it easier for students to provide meaningful input to an algorithm that they will watch (using an AV system) in the changing mode of engagement? The input generators that exist in most AV systems (including JHAVÉ) are relatively unsophisticated and do not provide convenient ways for the student to specify data for complicated problems such as graph algorithms.

How can we make it easier for students to engage in the constructing and presenting modes of the taxonomy? Class libraries and other forms of content-generation materials are needed here.

As we make progress in developing tools to enhance these modes of engagement, can we do so in a way that maximizes their portability? Computer science educators will want a choice of AV systems, not just a single option. But the empirical results we have cited clearly demonstrate that any pedagogically effective AV system should support these various engagement modes in some fashion. Does that mean that every AV system developer will now need to reinvent the tools needed for these engagement modes? JHAVÉ has illustrated one way in which different AV systems could share various forms of interaction support. But are there ways to extend tools like interactive question generators and input generators across an even broader range of AV systems?

Toward that end, researchers have already begun exploring the packaging of such AV support tools in XML format.[19] Hopefully this exploration will soon begin to achieve results, allowing vastly different AV systems to easily exchange supporting content, with each system processing that content in its own way. If this work succeeds, it will consequently be much easier to remain with the AV system that best suits your teaching style and still incorporate into it the engagement hooks that we now realize are necessary.

Beyond everything else involved in making AV a good pedagogical tool, we must also realize that the major worry of teachers who might consider using AV is the perceived large amount of time it will take to learn and use. This was borne out in survey results that appeared in Naps et al.[9] In broad terms, these results indicated that nearly all educators intuitively felt that AV should help students. The survey then asked if educators feel so strongly that AV can help, why don't they use it more extensively? The option that two-thirds of respondents cited as a major impediment was the time it takes to develop visualizations. Four other options were listed as major impediments by 45 to 48 percent of the respondents. These were the time required to search for good examples, the time it takes to learn the new tools, the time it takes to adapt visualizations to an individual's teaching approach and/or course content, and the lack of effective development tools.

Clearly materials must be developed that address these concerns about time. A repository where harried educators could go to find not just graphics but complete, peer-reviewed AV resource collections would enable more widespread use of AV. Development of such resources will certainly require a lot of work on the part of AV researchers, and much of that work will only have tangential connections to actually doing graphics. Instead, it will emphasize materials that effectively mesh with the graphic end of AV systems. However, all the empirical evidence cited previously indicates that such work will yield positive results in the learning our students achieve.

The JHAVÉ client is available at http://www.jhave.org. All the source code for both the client and the server is also available through a concurrent versions system server on the same machine. JHAVÉ is an open source project, and interested contributors should contact the author by email. ■

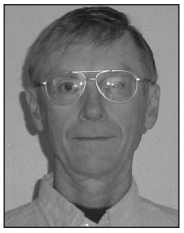
Acknowledgment

The National Science Foundation's Course, Curriculum, and Laboratory Improvement program (DUE awards 0126494 and 0341148) has supported JHAVÉ.

References

1. M. Ben-Ari, "The Concorde Doesn't Fly Anymore," *Proc. 36th SIGCSE Technical Symp. Computer Science Education*, ACM Press, 2005, p. 196; http://portal.acm.org/citation.cfm?doid=1047354.
2. M.H. Brown, *Algorithm Animation*, MIT Press, 1988.
3. L. Stern, H. Søndergaard, and L. Naish, "A Strategy for Managing Content Complexity in Algorithm Animation," *Proc. 4th Ann. ACM SIGCSE/SIGCUE Conf. Innovation and Technology in Computer Science Education* (ITiCSE), ACM Press, 1999, pp. 127-130.
4. J.B. Morrison, B. Tversky, and M. Betrancourt, "Animation: Can It Facilitate?" *Int'l J. Human Computer Studies*, vol. 57, 2003, pp. 247-262.
5. C.D. Hundhausen, "Toward Effective Algorithm Visualization Artifacts: Designing for Participation and Negotiation in an Undergraduate Algorithms Course," *Proc. Computer–Human Interaction Conf.*, ACM Press, 1998, pp. 54-55.
6. C.D. Hundhausen, S.A. Douglas, and J.T. Stasko, "A Meta-Study of Algorithm Visualization Effectiveness," *J. Visual Languages and Computing*, vol. 13, no. 3, 2002, pp. 259-290.
7. S. Grissom, M. McNally, and T. Naps, "Visualization in CS Education: Comparing Levels of Student Engagement," *Proc. ACM Symp. Software Visualization*, ACM Press, 2003, pp. 87-94.

8. S. Grissom and T. Naps, "The Effective Use of Quicksort Visualizations in the Classroom," *Proc. 9th Ann. Consortium for Computing Sciences in Colleges (CCSC) Midwest Conf.*, Consortium for Computing Sciences in Colleges, 2002, pp. 88-96.
9. T.L. Naps et al., "Exploring the Role of Visualization and Engagement in Computer Science Education," *ACM SIGCSE Bull.*, vol. 35, no. 2, 2003, pp. 131-152.
10. J. Lucas, T.L. Naps, and G. Rößling, "VisualGraph: A Graph Class Designed for Both Undergraduate Students and Educators," *Proc. 34th SIGCSE Technical Symp. Computer Science Education*, ACM Press, 2003, pp. 167-171.
11. G. Rößling, M. Schüler, and B. Freisleben, "The Animal Algorithm Animation Tool," *Proc. 5th Ann. ACM SIGCSE/SIGCUE Conf. Innovation and Technology in Computer Science Education* (ITiCSE), ACM Press, 2000, pp. 37-40.
12. C.D. Hundhausen and S. Douglas, "SALSA and ALVIS: A Language and System for Constructing and Presenting Low Fidelity Algorithm Visualizations," *Proc. IEEE Symp. Visual Languages*, IEEE Press, 2000, pp. 67-68.
13. J. Stasko, "Using Student-Built Algorithm Animations as Learning Aids," *Proc. 28th ACM SIGCSE Technical Symp. Computer Science Education*, ACM Press, 1997, pp. 25-29.
14. T. Naps, J. Eagan, and L. Norton, "JHAVÉ: An Environment to Actively Engage Students in Web-Based Algorithm Visualizations," *Proc. 31st ACM SIGCSE Technical Symp. Computer Science Education*, ACM Press, 2000, pp. 109-113.
15. J. Stasko, "TANGO: A Framework and System for Algorithm Animation," *Computer*, vol. 23, no. 9, 1990, pp. 27-39.
16. T.L. Naps and E. Bressler, "A Multi-Windowed Environment for Simultaneous Visualization of Related Algorithms on the World Wide Web," *Proc. 29th ACM SIGCSE Technical Symp. Computer Science Education*, ACM Press, 1998, pp. 277-281.
17. D.J. Jarc, M.B. Feldman, and R.S. Heller, "Assessing the Benefits of Interactive Prediction Using Web-Based Algorithm Animation Courseware," *Proc. 31st SIGCSE Technical Symp. Computer Science Education*, ACM Press, 2000, pp. 377-381.
18. M.-J. Laakso et al., "Multi-Perspective Study of Novice Learners Adopting the Visual Algorithm Simulation Exercise System TRAKLA2," *Informatics in Education*, vol. 4, no. 1, 2005, pp. 49-68.
19. G. Rößling and T. Naps, *Development of XML-Based Tools to Support User Interaction with Algorithm Visualizations*; http://www.ahrgr.de/AVPortal/WG2005.



Thomas L. Naps *is a professor of computer science at the University of Wisconsin, Oshkosh. His research interests include algorithm visualization and its application in computer science education. Naps has a PhD in mathematical logic from the University of Notre Dame. Contact him at naps@uwosh.edu.*