Abstract

Algorithm visualization is a means of simplifying the understanding and analysis of computer programs and the data structures they utilize. Much research has been focused on algorithm visualization with the intent to develop tools that can be used in an academic environment. While it has been found that algorithm visualizations can be effective, their adoption within curricula has not been forthcoming. JSAVE is an algorithm visualization tool for the Java Collection Framework that addresses both the requirements for effective visualization tools and the deficiencies that have prevented their widespread use. It provides a simple and effective means for using algorithm visualization both in an academic setting and for personal programming inspection.

1. Introduction

Algorithm visualization is an attempt at presenting the behaviors involved with an algorithm through a visual medium. It provides a means by which the teaching of algorithms can be simplified and the behaviors of algorithms can be better understood by classes or individuals. Research in the field of algorithm visualization in the past has led to criteria for features that ought to be present in algorithm visualization tools. [3] [8] Tools already exist that support much of these older criteria, but newer research has focused on the efficacy and acceptance of algorithm visualization tools. [13] [14] JSAVE is an algorithm visualization tool based off of a previous prototype [7] and designed to address the functionality of older research and the design objectives of newer research while providing increased capabilities in the context of the Java Collection Framework.

JSAVE, or the Java Simple Automated Visualization Environment, is a set of classes that provide complete visualization functionality for Collection classes. The core classes of JSAVE parallel the Collection classes of Java, providing complete compatibility with the standard classes while also providing automatic facilities for algorithm visualization. Only the List interface is currently supported, but this is easily extendible to the Java Set interface as well. Two design goals of JSAVE were to develop an effective visualization tool that: was intuitive to use while creating or viewing visualizations; and provided insight into the different implementations of the Java Collection interface.

2. Fulfill design objectives of older research

More traditional research into the field of algorithm visualization has produced a wealth of design characteristics, objectives, and features that are beneficial in the development of visualization tools.

2.1 Model-View-Controller design

JSAVE is based upon the Model-View-Controller design for visualization. [2] This paradigm divides a program into three parts:

- **Model.** The model contains the behaviors that are being analyzed. In JSAVE, the model consists of JSAVE data structures, which communicate with the view and controller components.
- **View.** The view represents the behaviors to the active user. In JSAVE, a special Visualizer is able to hold many different views, or representations, at once.
- **Controller.** The controller is what dictates the behavior of the model. For JSAVE, the controller can be any program that uses Java Collection classes to be able to use JSAVE for visualization. Visualizations can be created within a program by loading a saved visualization script.

2.2 Event driven vs. state driven design

Most algorithm visualization systems can be categorized as either event driven or state driven. In the event driven, or imperative, approach, the visualization is determined by the important or interesting events that occur during an algorithm. [8] The state driven approach creates a visualization automatically from the status of the data involved with an algorithm. There are several lists of
the advantages and disadvantages of both systems. [3] [5] [4] While event driven approaches are intuitive and promote abstract representations, they require a great deal of modification to an algorithm's source code and have difficulty representing invariance. State driven designs can create visualizations without much code modification, but they cannot be easily customized, abstractions are more difficult to represent, and state driven visualizations lack smooth transitions.

JSAVE uses the recommended approach of a hybrid of these two systems that is effectively state and event driven. [5] This increases the control over a visualization while decreasing the invasiveness of the visualization developing process. This is achieved by monitoring the changes in state for a JSAVE Collection class and creating the interesting events automatically. In addition, optional JSAVE directives can be entered into an algorithm, allowing an increasing level of control with an increasing level of invasiveness. The JSAVE Visualizer also provides a means of refining previously made visualizations without adding invasive directives to an algorithm's source code.

2.3 User control

Allowing stronger user control and customization of an algorithm animation gives the user, or active viewer, of the visualization the ability to make it best serve his or her needs. Previous research has suggested that giving active viewers control of an animation aids in the efficacy of the visualization. [9] [6] [3] [4] [15] JSAVE provides the ability for user control of color, navigation, and representation of algorithm visualizations.

2.3.1 Color

Color is one of the more aesthetic features of an algorithm visualization, but it also has potential as an aid in visualization clarity and understanding. All components of an algorithm visualization within JSAVE have colors that are customizable and can be dynamically changed during the execution of an algorithm. This allows active viewers to alter the colors of visualization components, highlighting component types and giving a better understanding of the different parts involved with an algorithm or data structure. Such a feature could benefit users who may not have previous experience with a certain data representation and do not understand what different visual components are. Further, color customization allows people who may not be able to distinguish between the default colors in a visualization to make the colors more suited to their unique needs.

2.3.2 Navigation

A person may not understand an algorithm animation the first time it is presented or if it is presented too quickly. JSAVE allows full control of the navigation through a visualization, controlling speed, direction, and the animation style.

The ability to allow user controlled speed of a visualization provides a means for active viewers unfamiliar with an algorithm to have the visualization presented at a slower speed, allowing them more time to understand what is being presented. Also, users more familiar with an algorithm could view portions that they understand at a higher speed, thus allowing them to focus on parts of an algorithm that they do not understand as well. The utility of speed control is well noted in previous research. [15] [10]

Allowing the direction of play to be controlled aids in understanding a visualization. If an active viewer does not understand how a certain point in an algorithm was reached, reversibility makes it possible to back up to a point that is familiar and proceed at a slower pace. Reversibility also allows an active viewer to trace the steps involved in an algorithm backwards through their execution. Through this, the causal relationships between the steps of an algorithm can be better observed and understood. Directional control is well accepted as a necessary part of algorithm visualization. [15] [4] [3]

Directional control in JSAVE is provided within a navigation control graphical interface similar to a CD player. This navigational control also allows either stepping or playing through the interesting events that are involved in an algorithm, controlling the style of the animation. If a user is knowledgeable about an algorithm, he or she can simply play through the events involved in rapid succession. Otherwise, the user can step through the individual events involved in an algorithm, making sure that each step is understood before moving on to the next. Step by step analysis allows a very fine grained perspective of an algorithm, enabling an active viewer to ensure that a visualization is presented at the optimal pace for understanding. [8]

2.3.3 Representation

The representation of an algorithm visualization is the mapping from data structures and actions of an algorithm to the visual components and their behaviors in the visualization environment. It has been found that having multiple representations is beneficial. [3] Because the representation is a key part of a user's perception of an algorithm, it is important that the proper representational model is used for the context of a visualization. [10] However, the best representation of an algorithm in one context may not be the best representation in another.
Having multiple representations in a visualization allows different models to be used as they are appropriate.

It has also been found that using only one representation can lead to inaccuracies in understanding if any particular part of that model was unclear or conceptually incomplete. [9] Multiple representations allow additional perspectives, resolving differences between the user's internal understanding of an algorithm and the various representational models. [9] This provides a refined understanding of the actual algorithmic concepts themselves.

JSAVE supports multiple representations in a visualization and works to make this feature as simple and automated as possible. The set of possible representations to of an algorithm is automatically determined by JSAVE. Thus, if a structure holds numeric data, a numeric representation is added to the set. If a structure holds strings, a representation that emphasizes relative comparisons is added. There are currently five different representations in JSAVE, based on data structure and contents, and more representations can easily be added in the future. The user is also free to choose which of the appropriate representations is active at any given time.

JSAVE also supports multiple implementation representations for a single interface. When an implementation of a Java List interface is visualized in JSAVE, the representations of all other implementations are added to the set of possible representations. Thus, an active viewer can dynamically switch between ArrayList and LinkedList representations during the visualization, no matter which implementation of a List is actually being used in the code that utilizes a JSAVE data structure. This allows the different data structures and their behaviors to be actively compared within the JSAVE system.

2.4 Postmortem Visualization

Allowing visualizations to be recorded and separated from the execution of an algorithm is beneficial. [8] [9] Recording a visualization greatly simplifies preparing presentations in advance, and ensures proper and desired functionality when the visualization is played back. This is useful in a classroom setting, where live demonstrations can have problems and waste class time. It is also possible that executing a program may be costly, time consuming, or have other drawbacks. In these cases, recording a visualization allows it to be observed later without the necessity of executing the program again.

JSAVE fully supports viewing visualizations independent of program execution. All visualizations within JSAVE can be saved as XML scripts. These scripts can then be loaded within the JSAVE Visualizer, allowing them to be viewed as visualizations no differently than if they were being generated by a live program.

2.5 Visualization Delivery

The use of a scripting system for postmortem visualization also allows scripts to be delivered or shared between people. This enables professors to create visualizations as learning aids that can be accessed at the most convenient time for students, better facilitating learning. [9] Issues that must be handled for effective delivery have been addressed by previous research:

- **Separating roles.** Designers, developers, visualizers, and end users may be different people. [6] JSAVE fully supports role separation, partially through the Model-View-Controller design, and partially through the robustness of the JSAVE Visualizer.
- **Language independence.** [6] The scripts that JSAVE uses to load previously saved visualizations are in an XML format that is itself language independent. If programs in other languages output these XML scripts, JSAVE will visualize them properly.
- **Multiplatform.** [9] [14] JSAVE is itself written in Java, allowing it to function on platforms supporting the Java SDK 1.4.1 and above.
- **Code Protection.** It may be desirable to prevent executable code from an algorithm from being distributed. [6] This is especially a problem with Java code that can be readily decompiled. Saved JSAVE visualizations do not require any code from the algorithms they are derived from. JSAVE uses a system of metadata, so that saved visualizations are completely separated from the classes used in an algorithm itself.

Actual delivery of visualizations through JSAVE scripts is most easily performed by using an applet. The JSAVE distribution JAR file automatically acts as an applet and needs only the location of a visualization script in order to present a visualization to a user over the Internet. Script files can also be shared manually and loaded into a Visualizer to generate a visualization.

3. Difficulties with visualization acceptance

While research in algorithm visualization has progressed a great deal, its acceptance within the academic community has been lacking. More recent research has developed reasons for this hesitance and ways to overcome it. [14] [13] The meta-study of Hundhausen et al. [13] observed that professors did not use algorithm visualization because:

- It took too much time to learn the processes involved.
- Visualizations used up too much class time.
- Visualization tools required too much effort to use.
- Doubts existed as to the efficacy of visualization tools.

Other polls have found similar results emphasizing that visualization tools are perceived as too complex and time intensive, and that they might not be effective. [14] JSAVE
addresses the issues of time and complexity as well as the issue of visualization efficacy.

3.1 Addressing complexity and time requirements

One of the core design goals of JSAVE was simplicity. Thus, to create a visualization in the JSAVE system, it is only necessary to use a JSAVE data structure. These data structures implement all of the methods and interfaces of the standard Java Collection classes, so they are fully compatible with any existing programs that use the Java Collection classes. JSAVE structures can be essentially “dropped” into an existing program by replacing only the line on which a Collection is instantiated. This simplicity avoids excessive time requirements for learning how to use the system and develop visualizations, and because JSAVE data structures can be “dropped” into a previously written program, any existing source code examples can be transformed automatically into visualizations.

The JSAVE Visualizer also uses a simple and informative user interface. The components of the interface are given standard names to take advantage of an active viewer's prior computer experience. Controls for navigation are organized in a well understood format similar to the controls of a CD player. Unfamiliar elements are presented in forms analogous to familiar forms. For instance, an operation history, or list of the events that have occurred during an algorithm, is presented in the same fashion as a history of previously visited web pages in a common web browser. This simple interface design makes presenting visualizations effortless for teachers and students.

3.2 Addressing effectiveness

Current research has confirmed that algorithm visualization can be an effective tool when it emphasizes interactivity and active participation. [13] [11] JSAVE is designed to promote intuitive interactivity. The guidelines for interaction put forth by Hundhausen et al. [13] are addressed below.

Part of the interaction capability of JSAVE is inherent to the Model-View-Controller design. The controller module, or client, can be literally any program that uses a Collection implementation. Because of this, interactivity through user input and interactivity through learner designed visualizations can be easily achieved by using a client that provides the appropriate features. Several clients that allow for user controlled basic operations on List implementations are included with JSAVE, providing extremely basic examples of how input may be taken from a user as part of generating a dynamic and interactive visualization.

Strong annotation or textual message capabilities in JSAVE allow for communication between active viewers and visualization designers. Through this facility, contextual questions or instructions for prediction can be embedded directly into an animation. This form of responsive interaction has been found to be effective by multiple sources. [13] [11] The annotation features in JSAVE also provide a means through which learners and educators can clearly communicate with each other about an algorithm. This makes JSAVE beneficial as a discussion aid for algorithms. [9] Annotations in JSAVE can be associated with any event that occurs as part of an algorithm, or with any key graphical component of a representation, such as an iterator or a node in a linked list. This flexibility allows for greater clarity by distinctly associating a message with the part of a visualization to which it applies.

4. Conclusion

While several visualization tools currently exist, there are few that have tried to adapt to the criteria and requirements that new research have put forth for gaining the acceptance of algorithm visualization in the classroom. JSAVE has approached these issues by ensuring that it is:

- **Simple and automated** for speedy and intuitive use.
- **Interactive** to provide an active learning environment.
- **Informative** to help user's better use and analyze an algorithm's visualization.
- **Flexible** so that it can be expanded itself or incorporated into another body of work.

4.1 Future Work

There are several possibilities for improving the breadth and strength of JSAVE:

- **Extend to Set interface.** JSAVE currently only supports visualization of List interface implementations. This could easily be extended into supporting Sets or even other nonstandard data structures.

- **Address large datasets.** The current visualization system has difficulty maintaining clarity when datasets are large. Allowing visualizations to automatically cope with large datasets would improve the visualizations of real world scenarios. [4]

- **Event clustering.** More visualization control could be gained from a refined ability to cluster the events occurring during the execution of an algorithm. This would enable the user to step through different operations of the algorithm more easily and at different levels of detail, providing a better understanding of the concepts. [12]

- **Predictive views.** Representations may be able to observe patterns or foresee events that occur on data structures, allowing them to more smoothly transition from the visualization of one event into another. [8]
• **Expand annotation facilities.** The existing annotation system could be enhanced to facilitate more interaction with the user or more convenient communication. [1]

• **Measure efficacy.** Numerous and varied forms of evaluation can help to assess the tool’s effectiveness and provide even more possibilities for improvement. [14] [13]

**5. Acknowledgments**

This work was funded in part by the National Science Foundation Research Experiences for Undergraduates program through grant CDA-0097464. The authors wish to acknowledge the contributions of their faculty mentor, Herbert L. Dershem.

**6. References**


