Design and architecture of an interactive eTextbook – The OpenDSA system

Eric Fouha, Ville Karavirta, Daniel A. Breakiron, Sally Hamouda, Simin Hall, Thomas L. Naps, Clifford A. Shaffer

Abstract

The OpenDSA Project seeks to provide complete instructional materials for data structures and algorithms (DSA) courses. Our vision for a highly interactive eTextbook involves the use of many algorithm visualizations (AVs) and a wide range of interactive exercises with automated assessment. To realize this vision we require a mix of third-party and custom software components that make up a client/server-based web application. The massive amount content development required compels us to adopt an appropriate mix of open-source practices that will encourage broad contribution to the project. In this paper we describe the OpenDSA system architecture and the design goals that led to the present version of the system.

Keywords: eLearning eTextbook Automated assessment Algorithm visualization Data structures and algorithms

1. Introduction

Data Structures and Algorithms (DSA) topics play a central role in every Computer Science curricula [1], defining the transition from learning programming to learning computer science. However, students often find this material difficult to comprehend because so much of the content is about the dynamic process of algorithms, their effects over time on data structures, and their analysis (determining their growth rates). Dynamic process is extremely difficult to convey using static presentation media such as text and images in a textbook. During lecture, instructors typically draw on the board, trying to illustrate dynamic processes through words and constant changes to the diagrams. Many students have a hard
time understanding these explanations at a detailed level. Another difficulty is the lack of practice with problems and exercises that is typical with DSA courses. Since the best types of problems for such courses are traditionally hard to grade, students normally experience only a small number of homework and test problems, whose results come only long after the student gives an answer. The dearth of feedback to students regarding whether they understand the material compounds the difficulty of teaching and learning DSA.

There is a long tradition among CS instructors of using computer software in the form of Algorithm Visualization (AV) [2–4] to convey dynamic concepts. AV has been used as a means both to deliver the necessary dynamic exposition, and to increase student interaction with the material [5]. However, despite the fact that surveys show that instructors are positive about using AVs in theory, and students are overwhelmingly in favor of using AVs when given the opportunity, many AV advocates have been disappointed at the relatively low level of sustained adoption for AV in actual classes. Surveys of instructors [2,6] show many impediments to AV adoption, including finding and using good materials, and “fitting them” into existing classes.

The OpenDSA project [7–9] seeks to remedy these impediments by providing complete units of instruction on all traditional DSA topics. OpenDSA’s goal is to build a complete open-source, online eTextbook for DSA courses that integrates textbook quality text with algorithm visualizations and a rich collection of interactive exercises available as a series of tutorials implemented using HTML5 technology. In this way, the modules are available for use on any modern browser with no additional plugins or software needed, and can even run on tablets and many mobile devices. OpenDSA modules combine content materials in the form of text, slideshow, simulation, and various types of assessment questions. All exercises are assessed automatically with immediate feedback to the student on whether the exercise was answered correctly. Thus, we posit they offer solutions to all of the problems mentioned above. They provide easy access, they illustrate dynamic process through dynamic visualization, and they allow for far more practice with the material than has traditionally been possible. As a result, students will gain far more practice by working on OpenDSA exercises than is possible with normal paper textbooks.

OpenDSA is designed with the following educational goals in mind.

1. **Leverage technology to present the material in a way that makes learning easier.** Surveys have measured the positive attitude towards AVs by educators and learners [2,6,10]. Despite the fact that measuring pedagogical effectiveness of AVs is not an easy task, several AV systems have led to the improvement of learner’s performance [11,12]. Thus, a key goal is to make it easier to deliver AVs to a wider audience.

2. **Include activities to engage and motivate students.** AVs can engage students at different levels. The taxonomy introduced by Naps et al. [2] defines a hierarchy for learner engagement. Higher engagement levels tend to provide a richer way for learners to interact (changing input data, responding to questions, etc.) with the technology. We seek to make it as easy as possible to deliver materials that provide meaningful engagement.

3. **Include assessments to evaluate how well students learn.** Each module should include mechanisms for students to self-gauge how well they have understood the concepts presented. Self-assessment can increase learner’s motivation, promote students’ ability to guide their own learning and help them internalize factors used when judging performance [13,14].

Each module should then integrate:

- Text and images.
- Dynamic presentation (visualizations and interactive exercises).

The following issues arise as a consequence of our vision. These drive the architecture and implementation of OpenDSA.

- The instructor for a given course should have control over the content, either to easily make use of existing materials, pick-and-choose from among existing materials, or to alter or add to existing materials. This leads to various requirements related to our licensing structure (an open source project), and our authoring system (easy to use, easy to modify content, easy to include or not include various dynamic components such as visualizations and exercises).

- Support for algorithm visualization is a core requirement. Thus, our system must be able to support presentation of dynamic content (in the sense of visualizations and interactive activities). While ease of authoring for dynamic content is desirable, we do not consider this to be a core requirement. In contrast, ease of reuse of AV artifacts within various combinations of the content is a core requirement.

- Support for a wide and flexible array of interactive exercises with automated assessment is another core requirement. We want to support everything from static multiple choice questions to dynamically-generated random problem instances for interactive exercises with non-trivial user interfaces. Again, we are more concerned that exercises once created can be easily reused than we are with making it easy to create the exercises.

- Support for tracking student progress (which exercises were attempted and/or completed) is a core requirement. Both students and their instructors must be able to see how a given student is progressing through the content. At the
same time, we would like as much ability as possible for students to work exercises when the data collection server is unavailable. Thus, we require some sort of client/server approach where students view exercises, and their results are reported to the server.

- Support for tracking student interactions with the content is highly desirable for many reasons, including support for studying the pedagogical effectiveness of various approaches and support for gathering data about usability of system components for future improvement.
- We would like the system to be easy to access, and portable across a wide range of computing platforms. This requirement places many constraints on the content deployment technology.

Previous publications have addressed other aspects of the OpenDSA system. In particular, [8,9] address the vision for the OpenDSA system, [7] discusses initial pedagogical evaluation efforts, and [15] presented a description of the JavaScript Algorithm Visualization library (JSAV) used to support development of the AVs and some exercises used by OpenDSA. Details on the OpenDSA event logging system and the types of evaluations that it makes possible are discussed in Broader discussions regarding the previous state of use of AVs in education and their effectiveness are presented in [3,4], and we only briefly touch on those subject in this paper as necessary.

Our focus in this paper is on the OpenDSA system architecture and the design goals that lead to the present version of the system. Section 2 describes prior work that has affected the design of OpenDSA. Section 3 describes the architecture of OpenDSA, while Section 4 focusses on the various interactive components and assessment activities as seen by users of an OpenDSA eTextbook. Section 5 discusses some non-technical considerations that influenced our design choices, especially as they relate to OpenDSA’s status as an open-source project. Section 6 reviews evaluation studies that have been done or are planned for OpenDSA. Finally, Section 7 presents our future plans.

2. Related work

ETextbooks today are quite common, in the sense of static documents like the ePub format [16], eBooks from sites like Amazon, and other PDF-like static presentations. Despite recent prototypes of interactive eBooks [17], these formats generally are not suitable for dynamic content like algorithm visualization and non-trivial exercise interfaces. A number of papers have surveyed various aspects of interactive eTextbooks, including the role of algorithm visualization [8,18,19] so we will not discuss that further here. Instead we focus on prior and ongoing work that has directly influenced the OpenDSA project. We recognize that there are also many recent efforts in online, automatically assessed exercises. Many are commercial, and they are so new that their impact has yet to be felt (such as Zyante and recent efforts by Intel and Microsoft). But the most important for our purpose is the non-commercial exercise infrastructure from Khan Academy [20], which we discuss in further detail below since we make use of it in OpenDSA.

Miller and Ranum’s interactive eTextbook for Python programming [21], and CS Circles [22] by Pritchard and Vasiga are Python courses for novice programmers. To the best of our knowledge, these are the closest projects to our idea of an interactive eBook. Miller and Ranum produced a complete book that includes embedded video clips, active code blocks that can be edited within the book’s browser page by the learner, and a code visualizer that allows a student to step forward and backward through example code while observing the state of program variables. Like OpenDSA, they use reStructuredText (ReST) [23] and Sphinx (http://sphinx.pocoo.org) as their authoring system. Sphinx is a document processing tool that makes it easy to output documentation to HTML, LaTeX, and other formats. Miller and Ranum’s book runs on the Google App Engine and includes assessment activities requiring students to write a small function to perform a single action. Their grading system is rudimentary and only provides students with simple pass/fail feedback upon completing an exercise. In our opinion, the most important thing missing from this effort is a wider variety of automated assessment with immediate feedback.


Both Miller and Ranum’s book and CS Circles use Guo’s online Python tutor [25], an embeddable web program visualization for Python. The online Python tutor takes a python source code as input and outputs an execution trace of the program. The trace is an ordered list containing the state of the program at each line of code. Each execution point contains the line number of the code that is about to be executed, a map of global variables to their current values, an ordered list of stack frames with each frame containing a map of local variables to their current values, the state of the heap, and the program output up to the execution point. The trace is encoded in JSON format and sent to the user’s browser for visualization via HTTP GET request. The backend can work on any webserver with CGI support or on the Google App Engine.

TRAKLA2 [5,26] comprises a large collection of Algorithm Visualization simulation exercises with automatic feedback. TRAKLA2 exercises are called “visual algorithm simulations” because they ask students to determine a series of operations that will change the state of the given data structure to achieve some outcome. For example, students might build a tree data structure by repeatedly dragging new values to the correct locations in the tree. Alternatively, the student can gain understanding by examining a step-by-step execution of the algorithm (called the model solution). Many TRAKLA2 exercises include some tutorial text along with pseudocode to explain the algorithm, but their main purpose is to provide an interactive proficiency exercise. A database stores user information, including submissions and grades as well as information about
courses and exercises such as deadlines and maximum points. OpenDSA makes heavy use of the TRAKLA2 visual algorithm simulation concept for its “proficiency exercises”, and the TRAKLA2 developers are part of the OpenDSA development team.

3. Architecture

To the reader, the fundamental unit of OpenDSA is the module, which is a single browser page containing text, graphics, visualizations, code snippets, and exercises of various types. A module corresponds to a section in a traditional textbook or a topic that might be covered in a single lecture or part of a lecture. So they are generally equivalent to around 15–30 minutes of instruction, though some modules are much shorter. Modules can be grouped to form “chapters” as shown in Fig. 1. Modules are authored using reStructuredText.

OpenDSA includes a configuration mechanism that allows instructors to select from among existing modules to compile into an “eTextbook” instance that contains only those modules that they wish to use for their course. Based on the configuration file, the module source and the indicated collection of independent visualizations and exercises are compiled together to produce a set of HTML pages (one per module) that define an instance of an OpenDSA eTextbook. Navigation through a given eTextbook is currently done through an index of modules that looks much like a traditional table of contents, as shown in Fig. 2. In the future, alternative navigation systems could be devised that operate within the existing framework, such as navigation via concept map. The authoring and configuration phases of the process are described in more detail in Sections 3.1 and 3.2.

The resulting eTextbook operates by means of a client-server implementation. An overview of the OpenDSA architecture is shown in Fig. 3. The content server delivers the HTML documents along with embedded visualizations and exercises that make up the eTextbook. The data collection server is a web application that we built using the Django web framework with a MySQL database for data persistence. The functionality provided by the content server and data collection server are described in Sections 3.3 and 3.4.

3.1. The OpenDSA authoring system

Since OpenDSA aims to integrate text, dynamic content, and automated assessment, we needed an authoring tool that allowed us to produce authored content in a web-accessible form while also enabling us to glue the dynamic elements together (that is, the AVs and exercises that presumably would not be created using the content authoring system). We wanted a text-based format for documents (in the sense that HTML and LaTeX are text-based as they use ASCII-text source from which the final product is created), since most binary document formats (like Microsoft Word or Apple’s iBooks Author) are proprietary and invariably have cross-software and device compatibility issues. Previous experiences with hypertextbook authoring made us prefer that the authoring environment should be a standalone system not attached to a particular web development framework or environment [27]. We chose the reStructuredText (ReST) markup language and its compiler Sphinx (http://sphinx.pocoo.org) as our authoring tool for content.

ReST is a so-called “lightweight” markup language, originally designed to produce Python program documentation. Sphinx is the engine that converts reST files to HTML, LaTeX, or PDF documents according to specific “directives”. A directive in Sphinx is equivalent to a macro in Microsoft Office documents or a markup command in LaTeX. A directive is
implemented by a small Python program that controls how Sphinx should process the text included in the directive. Sphinx has many features of traditional document processing software such as cross referencing, tables of contents, image support, etc. One important feature of Sphinx for our purpose is that authors can include "raw HTML" code in the ReST document and have it pass through to the output (HTML) file. This feature allows us to embed AVs and exercises from external sources (specifically, as iframes in the HTML file). We wanted our eTextbook to be platform independent, so we opted for HTML5 output files and wrote several custom Sphinx directives to support our needs for dynamic content on webpages. For example, while ReST supports the ability to pass through raw HTML to the output HTML pages, we do not actually want module authors to do so directly. So directives are used to control this process. Many of our custom directives allow
convenient embedding of dynamic (JavaScript-based) content. We wrote several directives that augment support for creating book-length documents that Sphinx currently lacks.

- **Embedded dynamic content**: The “avembed” directive embeds into a ReST document an HTML iframe containing an AV and/or interactive exercises.
- **Code snippet support**: We designed OpenDSA to be programming language independent (in the sense of what language the code examples on a page will be in). We provide module authors the ability to embed code located in separate files.
- **Numbered cross references**: We display module numbers with the ability to divide into author-defined “chapters”. Citing book elements will produce the appropriate numbering as is done in other authoring systems such as LaTeX.
- **“ToDo” text boxes**: During the development phase of the project we find it helpful to be able to specify items for module developers “to do” later. This directive provides necessary markup and generates a page listing all TODO elements.
- **Prerequisites**: Provides support whereby authors can define topics that are covered by modules, and topical requirements that must be covered by other modules. In this way, when a book instance is compiled the author can know whether important topics are missing from the book.

To support our augmented cross reference numbering scheme and other configuration support in ways that we could not do with Sphinx directives, we created a document pre-processor script that embeds the numbering as needed for hyperlinking text.

3.2. The OpenDSA book configuration system

The starting point when building an OpenDSA book instance is to define all book details within a configuration file. OpenDSA configuration files are in JSON format, with an example shown in Fig. 4. The configuration file lists the modules to be included in the book, along with the AVs and exercises referenced within each module. That is, an AV or exercise can only appear in a module if it was included in the ReST file by the module author, but the configuration file provides additional control. A given AV or exercise can be left out, or it can be included in the module but “hidden” behind an HTML button if defined to do so by configuration options. The configuration file controls the number of points (if any) to be awarded for completing a given AV or exercise. The configuration file can be seen as a serialization of a book instance. It collects all settings and preferences required to configure an instance of an OpenDSA book in a single, portable file that can be easily shared among instructors. The advantages of a central configuration point include:

- **Ease of use.** While in its current version the configuration system is difficult to edit as a JSON text file, it was designed with the intention of being edited by instructors through a custom GUI that is under development.
- **Ease of personalization.** A configuration file contains all environment-dependent settings such as paths and target URLs. For example, if developers want to point their content server code at different instances of the data collection system (that is, the gradebook and tracking infrastructure), they simply make the change in their own configuration file. In this way, each institution can collect data for its own students. Instructors can include their own resources like AVs or exercises in the book. Instructors can share all OpenDSA content, but when they build the book, it will be built using their personalized settings.
- **Ease of replication.** Once a configuration has been created, instructors can make identical copies without going through the configuration process. This is helpful for separating students within different sections of a course for grading purposes.
- **Fine-grained control.** Existing configuration files provide sensible defaults, but allow instructors to control aspects such as how many points a specific exercise is worth or whether it is required for module credit.
The configuration file is parsed by a configuration script written in Python. The configuration script uses the settings in the configuration to build an OpenDSA textbook using Sphinx. The script performs the following tasks.

- Sets sensible defaults for optional configuration settings.
- Appends necessary external JavaScript and CSS links, as well as configuration information, to every module.
- Appends configuration information to exercises or removes them from modules based on configuration settings.
- Generates the table of contents page.
- Provides necessary support for checking that declared module topic prerequisites are honored.

3.3. Content technologies

HTML5 is the latest revision of the HTML standard. It introduces elements and attributes that allow developers to create more dynamic and functional web pages. HTML5 is supported by all major browsers as well as most tablets and many smartphones, and thus content that conforms to HTML5 standards will generally run on any of these.

OpenDSA makes heavy use of HTML5's localStorage feature. This is a set of functions and procedures that allow web developers to store data in a web browser. We use localStorage to (temporarily) store all user interactions with the eTextbook and all data received from the content and data collection servers. This allows OpenDSA's client-side processing to keep all transactions in a coherent state, and allows interactions with the server to be cached and transferred in batches to reduce network traffic. Fig. 5 shows the role of localStorage within OpenDSA.

We use JavaScript as our scripting language and Asynchronous JavaScript and XML (AJAX) for communication with OpenDSA's server-side processing. JavaScript is natively supported by all modern browsers and when coupled with AJAX, allows us to send (asynchronous) event-based requests to the OpenDSA data collection server through HTTP/S methods (such as GET and POST). We encode all data exchanged between the client and the data collection in JSON. Since JSON is a subset of JavaScript, this approach simplifies parsing and handling of objects within the browser. It is also supported by most modern web browsers.
OpenDSA’s client-side JavaScript libraries support the client as follows.

- Send the information necessary to store a new book, module, or exercise in the database.
  One advantage to having the module and exercise configuration information available on the client is that it allows
  new modules and exercises to be dynamically loaded into the database stored on the data collection server. If a user
  is logged in when a module page is loaded from the content server, then an AJAX request containing the configuration
  information is sent to the OpenDSA data collection server. If the module or any of its exercises do not appear in the
  database, the data collection server creates new objects using the information provided in the request. The response
  from the data collection server contains information about the user’s proficiency with the module and each exercise in
  the module. This technique reduces the cost of configuring the database and allows the entire system to automatically
  adapt to new content.
- Manage users’ score and proficiency data locally.
  When an exercise is completed, our AV support library (JSAV) generates an event which is captured by a listener. When a
  JSAV event is captured, a series of actions is triggered. First the module page caches the user’s proficiency status locally,
  displays the appropriate proficiency indicator to the user, and sends the score to the data collection server. When the
  server responds, the proficiency indicator and local proficiency cache are updated based on the server’s response and a
  check is performed to determine whether the user is now proficient with the module. If so, a visual sign is provided to
  the user (such as a red button turning green for an exercise, or a green checkmark by an AV).
- Keep multiple OpenDSA pages in sync.
  We make sure that actions such as logging in, logging out, or gaining proficiency are reflected across all OpenDSA pages
  open within the browser.
- Collect and transmit user-interface interaction data to the data collection server.

3.4. Data collection server technologies

We built OpenDSA’s data collection server using the Django web framework. Django is an open-source Python Web
development framework that implements the model/view/controller architectural pattern.

We wanted our data collection infrastructure to be easily integrated into course management systems (CMS). We have
integrated OpenDSA data collection server into A+, a CMS created and used by our collaborators in Finland [28]. We use A+
  to manage the relationship between courses, eTextbooks, instructors and students. A book is attached to a course instance
  that is taught by an instructor. When using a particular book instance, students are automatically enrolled in the associated
  course. There are also OpenDSA eTextbook instances that are created for testing purposes.

OpenDSA uses the **REpresentational State Transfer** (REST) design model for web applications [29]. REST’s goal is to
  simplify use, development, and deployment of web applications. REST requires applications to comply with the following
  constraints.

- **Use of Client-Server architecture**: OpenDSA is composed of a content server (set of HTML documents created using
  Sphinx) that delivers HTML pages that are viewed on the user’s browser. The HTML pages are clearly distinct from the
  actions of the data collection server (Django application). Neither the user’s browser nor the content server are involved
  in any data persistence operations required by the data collection server. This constraint allows the components to work
  independently.
- **Stateless**: Each message between the user’s browser and the data collection server contains all information necessary
  (in the form of JSON objects) for the service to perform its function, with no client context being stored on the server
  between requests.
- **Cacheable**: OpenDSA’s webpages uses the localStorage object to cache information from the server’s responses.
Layered system: The data collection server and user’s browser communicate via an API. The API, based on the Model/View/Controller architecture pattern implemented by Django, makes the system scalable.

Uniform Interface: Within OpenDSA, all communication between the browser and the data collection server is mediated through HTTP/S GET and POST methods.

We used django-tastypie (http://tastypieapi.org/) to build OpenDSA’s data collection API. An overview of the API is shown in Fig. 6. Cross-site HTTP requests are authorized only to sites explicitly listed in the data collection server configuration settings. This allows us to control which sites can serve OpenDSA content “textbooks” to a given instance of the data collection system. Likewise, it also allows third parties to independently set up just an OpenDSA content server (and find a willing partner with a data collection server that will accept their input) or both a content server and data collection server. The API exposes the following services.

1. **Authentication** allows users to login and/or logout. OpenDSA uses an API Key authentication mechanism. In that model, a new key is generated every time the user connects to the server. After the initial login only the key is used to authenticate the user. This mechanism allows us to forbid connection from a different host at the same time by the same user.

2. **Book components management**: In OpenDSA, book components such as module information (name, author, topic, etc.), and exercise information are stored in the database through the API. The client sends all the information via a HTTP POST request to the API, and the book components management service is responsible for storing the data.

3. The **exercise attempt manager** is responsible of storing all user-exercise interaction data such as time of the attempt, exercise name, attempt correctness, etc.

4. The **content events manager** allows us to record every action that occurred on OpenDSA content, like button clicks made by the students, windows and HTML iframe openings and closings, etc.

5. The **student progress manager** provides the client with progress data on a specific exercise, such as how close the student is to reaching the proficiency criteria for that exercise. This allows us to provide appropriate feedback to the user.

6. The **Student proficiency manager** returns the student’s proficiency status on an exercise or a module.

7. The **Student/Class manager** provides instructors with views to see their students’ progress and use of the book.

4. **Interactive content and exercises**

   Engaging, interactive content and automated feedback are the key ingredients that give eTextbooks the potential to be a significantly better learning experience than traditional paper textbooks. In this section we briefly describe the many different types of interactive content and exercises used in OpenDSA. Their delivery, their assessment, and tracking of the
results presents the greatest technological challenges to the project, and drives much of the architecture that was described in Section 3.

Automated assessment of exercises is a major feature of OpenDSA. We consider it to be the most important aspect of the project, that students are given the opportunity to do substantial amounts of practice on all aspects of the content that they are studying in a DSA course. Lack of practice is a significant problem in a typical DSA course. Since the best types of problems for such courses are traditionally hard to grade, students normally experience only a small number of homework and test problems (in the tens of exercises), and feedback in the form of grades that result from human grading comes only long after the student gives an answer. The dearth of feedback to students regarding whether they understand the material compounds the difficulty of teaching and learning DSA. In contrast, OpenDSA seeks to give immediate feedback on the state of a student’s understanding, and over the course of a semester to allow a student to work hundreds of problems.

4.1. Slideshows

"Slideshows" show a series of steps to animate the behavior of an algorithm. Figs. 7 and 8 show two types of slideshow. Fig. 7 shows a simple slideshow embedded within the content of a module. These typically present a static example to illustrate a particular point or part of an algorithm. Fig. 8 shows a more complete AV for an algorithm that allows the student to provide their own input or specify that a random input should be generated. In both cases, the student is merely viewing the AV, but this is appropriate for expositional portions of a tutorial. Slideshows (as well as many exercises) are built with JSAV: the JavaScript Algorithm Visualization Library [15]. JSAV has adopted what we consider to be the best features from the large number of existing AV systems that we are familiar with. What makes it different from other libraries for building AVs is the ease with which it can be integrated into HTML pages and its explicit support for creating engaging exercises.

JSAV includes a specific set of features designed to support development of AV-based slideshows and exercises that involve active learning techniques, and visual content that can be easily integrated into online tutorials. JSAV supports the visual building blocks needed to create AV content for the topics covered by OpenDSA. The most important of these building blocks are the common data structures, including arrays, linked lists, trees, and graphs. For each, JSAV provides both automatic and customizable layouts. Furthermore, JSAV supports code display, allowing for step-through highlighting of code lines in sync with the associated data structures. Finally, there is support for a number of graphical primitives, largely through pass-through from the JSAV API to the Raphaël graphics library (raphaeljs.com).
4.2. Simple questions

By “simple” questions, we mean a range of standard question types such as true/false, multiple choice, fill-in-the-blank, etc. Individually, they generally do not take a long time to complete. While we consider this type of exercise to be less desirable than some of the more sophisticated (and more interactive) question types discussed below, they do play an important role in OpenDSA. Well crafted multiple choice and other simple question types can be quite useful pedagogically. So a rich infrastructure to support such question types is important.

We use the Khan Academy Exercise Infrastructure [30] to store and present “simple” exercises. We will refer to it as KAEI. KAEI is a flexible exercise delivery framework written in JavaScript, and compatible with our practice of delivering dynamic content embedded in module pages as iframes, while being HTML5 compatible. Another important feature of KAEI is that it supports a flexible mechanism for defining “parameterized” questions. For example, a simple question can be generated that involves random variables that are bound as problem instances to be presented to the student. Part of defining the correct answer for evaluation purposes is computing the corresponding function for generating the answer to the particular problem instance. This means that question developers must do more work (designing a good question often means being able to do some JavaScript programming to define the evaluation function), but it provides for a truly rich exercise development mechanism. Simpler questions with no such randomized component (such as text-only questions for static true/false or multiple choice answers) are generally easy to define.

We have successfully integrated JSAV with KAEI to develop a number of more sophisticated exercises. In some cases a student views a data structure with randomized values that define a specific problem instance. Depending on the exercise, the student might specify an answer by selecting from a multiple choice list, or filling in a number. In other cases, the student might directly manipulate the data structure, such as putting values in order the way it would be done by a specific sorting algorithm. Such exercises are simpler forms of our “proficiency exercises” described below.

4.3. Proficiency exercises

A key exercise type for OpenDSA is the visual algorithm simulation or proficiency exercise. These exercises were pioneered at Aalto University (an OpenDSA collaborator) through their TRAKLA2 system [5]. Proficiency exercises aim to verify that students understand how a given algorithm works by requiring them to simulate its behavior. A student’s proficiency is measured by one or a set of exercises using visualization that will ask the learner to:

- Predict the next step of the algorithm;
- Provide input data that will make the algorithm behave in a specific way; and
- Simulate the execution of the algorithm or the behavior of a data structure.
Fig. 9. Example of a JSAV proficiency exercise for Heapsort.

JSAV provides a framework for developing proficiency exercises. Fig. 9 shows an example of the Heapsort exercise. In particular, at each step the student will interact with the presented data structure, and the relevant aspects of the interaction are stored. For example, a student might have to click on some elements in an array to put them in order. Depending on the demands of the exercise, the exact sequence of steps might be captured if that is what is being tested. In other exercises, it might be required only that the student recognize which items in the array need to be put in order, and so the final order of those elements is all that is captured for that step. In either case, JSAV will then compare the results of that step as captured by the exercise against the equivalent behavior in the model answer that is generated by the evaluation function. If the captured result from the student’s actions match the equivalent actions in the model answer, then the step is deemed to be correct. If they do not match, then various outcomes are supported.

- **No immediate feedback:** Students go through all of the steps of the exercise, and are told at the end how many steps were done correctly.
- **Step reset:** A problem with “no immediate feedback” is that once a student makes a mistake, it might not be possible to get back on track. When detecting an error, “Step reset” will reset the exercise to the beginning of the step, and require the student to try again.
- **Step feedback and correction:** A problem with “step reset” is that it rewards guessing, which is always a concern with automated assessment [31]. When detecting an error, “Step feedback and correction” will alert the student that the step was done incorrectly, then set the exercise to where it would be if the step had been done correctly. The student does not get credit for that step, but is in position to learn from the mistake and successfully continue the exercise. Typically the scoring for the exercises requires that the student reach some threshold such as 90% of the steps done correctly to get credit for completing the exercise.

4.4. Programming exercises

We have developed a few small-scale programming exercises with automated assessment. Our goal is to allow students to do a range of exercises where the answer is typically a single function requiring a few lines of code, and the definition for a “correct” answer is fairly clearcut. We recognize two distinct approaches to automatically assessing programming exercises:

- **Output-based assessment** where the output or result of the student’s code is compared against the output or result of a model code on a specific suite of test cases.
- **Trace-based assessment** where a trace of state changes to key variables is captured in both the student answer and the model answer, and the traces compared.
In both cases, the interface to the student is the same. Typically, the function signature is given to the student and an editor is shown to him to be able to write or paste his code. The student inputs his code then clicks on the Check Answer button. The student code is then sent to an evaluation process run on server side, and then feedback is returned to the student. The feedback has three cases:

- Correct: when the answer is perfectly correct in that it either matches the output by the model answer on the test cases or when the state trace matches the model state trace.
- Try again: when there are no syntax errors but the answer is not correct. In that case the message appears to the student shows the expected answer and the student’s answer.
- Syntax error: when there are syntax errors. A full listing of the errors generated by the compiler is shown to the student.

An example programming exercise is shown in Fig. 10.

In the case of output-based assessment, a test suite is used to test the student’s code. The student code is embedded into a test harness (since it is only a function and must be part of a complete program to execute). Test cases are designed to call the student code and get back the output of the code, and then the evaluator checks if the output is correct or not.

The trace-based assessment approach, while harder to implement, has advantages in flexibility and feedback. An example is the JhavéPOP system [32]. The current form of JhavéPOP runs the student code through an interpreter, and returns an XML-based description of the program trace for visualization. JhavéPOP supports a small subset of Java, and is intended for small exercises in pointer manipulation, such as for linked lists. We are working with the original JhavéPOP developer to produce a new version that integrates with OpenDSA to crate JS AV-based visual traces of the student’s program behavior along with trace-based automated assessment of their answer. Fortunately, trace-based assessment is quite similar to the way that we process proficiency exercises. That is, a trace for the key steps of a model answer is generated. A trace for the equivalent key steps of the student answer is also generated. The traces are then compared for equivalence.

4.5. Other activities

OpenDSA already includes a small variety of other activities that cannot be characterized as simple questions, proficiency exercises, or programming exercises. These include a number of “calculators” that demonstrate an algorithmic process or equation. For example, calculators are in place for calculating somewhat complex hash functions, such as string hashes. An example of an equation calculator presents the result of calculating the “birthday” equation. The question for which this is named is “How many people must be in a room before the odds are at least 50–50 that two people share a birthday?” This is directly related to the probability of collision resolution in a hash table, and the actual calculator will calculate the odds of collision for any size table and number of records. We anticipate that every major equation in OpenDSA will be made “live” through the use of calculators and associated visualizations as appropriate.

Another type of activity is a performance simulator. One OpenDSA activity has the student provide a proposed increment series for Shellsort (see Fig. 11). The increment series is a key parameter for the behavior (and therefore the efficiency) of Shellsort. By providing an increment series and discovering through simulation the resulting cost, students gain hands-on experience with this parameter, rather than just reading a statement about what constitutes a good series. We anticipate that this technique can be used to provide active experimentation with many performance issues.
5. Non-technical design considerations

A number of external considerations affected our design decisions, aside from the core functional requirements of support for developing dynamic content (the AVs) and a rich collection of interactive exercises with self assessment. These include a desire to operate as an open-source project in order to attract the participation to develop a large body of content, a desire for maximum accessibility to the content by users (both in terms of platform independence and in terms of openness of the content), and a desire for maximum flexibility in terms of instructor-driven development of customized eTextbooks for individual classes.

An important feature of OpenDSA is that it is an open-source project. All materials are provided under the MIT open-source license. This means that they can be reused freely, even by commercial entities (the only restriction is that the copyright/license notice on the materials must be preserved). This has a number of implications that we consider fundamental not only to our core values, but also to practical aspects of operating the project.

Most obvious, being freely reusable and freely modifiable will hopefully encourage instructors to use OpenDSA materials in their courses. The content is deliberately “unpacked” as much as possible, allowing instructors to use just a single visualization or exercise on its own, perhaps with or without data logging (and grading) support from the data collection server. Alternatively, modules can be used singly or in combination as a specially constructed eTextbook for a course. Flexibility and reuse considerations drove various design decisions, such as the use of iframes and oembed for the exercises and AVs. Use of the Khan Academy exercise infrastructure (with its design of exercises as separate HTML pages) also supports our reuse goals.

A design consideration for the JSAV library is its relative independence from the rest of the OpenDSA infrastructure, allowing developers to use JSAV independent of OpenDSA. JSAV can also be used in combination with other AV development toolsets, which can in turn feed back into the OpenDSA project. For example, our redesign of the JhavéPOP project described above begins with simply converting the existing JhavéPOP exercises to generate JSAV visualizations. This allows an initial version of these exercises (without automated assessment) to be immediately incorporated into OpenDSA. We are also in the process of developing a translator from AnimalScript [33] to JSAV. This will both allow access through HTML5 to the full range of Animal animations, as well as make it possible to include Animal AVs in OpenDSA modules.

Our design of the student grade tracking system includes as a core feature the ability for users to work exercises – and get feedback on their answers – without requiring them to actually log in to the system or otherwise rely on the availability of the data collection server. Users only need to log in if they want their performance to be tracked (which will be desirable to many users since it does provide the useful capabilities of retaining their progress and providing their results to their instructor for a score). It fits with our core philosophy of openness that our exercises are not hidden behind a login wall (and certainly not behind a pay wall). This allows prospective instructors or student users to “try things out”, a feature that is rarely available except for a small number of demonstration exercises in most existing online exercise systems. This open access feature (with value added by login) is also a hallmark of the Khan Academy exercise infrastructure, and so was easy for us to adopt for those exercises. But we made the conscious decision to ensure that our entire exercise framework preserved this capability.

Another important consideration when adopting an open-source approach is to encourage other AV developers to join or at least contribute to OpenDSA. Developing a complete collection of DSA materials with associated AVs and exercises is a massive undertaking, and gaining additional developers is important to our eventual success. Aside from doing conversions
of existing AV materials as just described (and so quickly generating an initial pool of AVs), the OpenDSA infrastructure will hopefully prove attractive to both old and new AV developers who want to adopt the HTML5 standard. There is a longstanding tradition of many student developers creating AVs as independent study projects, or instructors developing a small number of AVs for their own instructional purposes [3]. OpenDSA can provide a vehicle to allow them to proceed with their development much faster and much further than they could have hoped to do on their own. At the present time, few AVs outside of OpenDSA are HTML5 compatible, and the large back catalog of AVs is almost entirely written in Java. As Java becomes increasingly problematic for use on the Internet (especially with mobile and tablet devices), JavaScript/HTML5 will become increasingly attractive to AV developers. Since it will be easy to both contribute AVs or other material to OpenDSA, and to leverage OpenDSA materials (such as text) to add value to the developer’s own efforts, we hope to gradually attract developers and content by being open source and supporting their efforts.

6. Evaluation

We have conducted a number of initial evaluations of OpenDSA, both for pedagogical effectiveness and for user satisfaction by students and instructors. This section reports on these efforts.

6.1. Class evaluations

As of this writing, OpenDSA and associated visualizations have been used by hundreds of students in three countries.

Four JSAV binary heap proficiency exercises have been used in a DSA course at Aalto University for two years. In Spring 2012 around 80 computer science majors and in Spring 2013 around 400 CS majors and minors at Aalto used the exercises. The course has a long tradition of using similar TRAKL2 algorithm simulation exercises. Results of the JSAV exercises were in line with results from previous years using TRAKL2. Furthermore, the student opinions were highly positive both years. More details can be found in Karavirta et al. [34].

OpenDSA was used to replace three weeks worth of standard lecture materials on sorting and hashing during October 2012 by around 60 students in a DSA course at Virginia Tech. Students in the treatment section (that used OpenDSA) scored slightly higher on the resulting exam than the control group (that did not use OpenDSA), but not significantly so. Student evaluations on OpenDSA were highly enthusiastic, with mean scores on preference for interactive online tutorials as compared to standard lecture going up after actual use of the materials. The instructor of the treatment section reported being able to spend a greater fraction of lecture time on content related to the more abstract and difficult topics, and less on the mechanics of the sorting and hashing algorithms. More details on this evaluation can be found in Hall et al. [7].

During Spring 2013, OpenDSA was used to teach sorting and hashing in three separate sections of data structures courses, two at Virginia Tech and one at Alexandria University in Egypt. All three instructors are not directly involved in OpenDSA development. Roughly 120 students in all used OpenDSA during this time. During Fall 2013, two sections of a data structures class at Virginia Tech, a total of about 150 students, are using OpenDSA as their textbook for the entire semester.

As OpenDSA reaches the point where there is sufficient stable material to teach a full semester class, we hope to perform controlled experiments to compare learning gains between regular textbook-and-lecture and OpenDSA in various settings. In the meantime, our initial testing clearly shows that students using OpenDSA do at least as well as students using regular textbook and paper homeworks, while at the same time reducing the grading burden to instructors through automated assessment of exercises.

6.2. Student satisfaction

After using OpenDSA during Fall 2012 and Spring 2013, students were surveyed to describe their overall experience with OpenDSA and to report any bugs or usability issues that they encountered. Student reactions were highly positive. Details of the 2012 results were presented in [7]. As another interpretation of the results, we have computed word clouds images from the student survey results, shown in Fig. 12.

The word cloud shows that students used positive terms to describe their experience with OpenDSA. “Good” and “helpful” were heavily used by students in all three sections. It is interesting to note that the section in which OpenDSA was used in class during lecture period have their top words as “exercises” and “good”. Logs analysis showed that students used the book to review for exams, thus confirming the usefulness of OpenDSA as a study aid. Figs. 13 and 14 present timelines showing exercise completions. Clusters of interactions appear not only at homework due dates as expected, but also immediately prior to the midterm.

When analyzing log data for how students used OpenDSA, we found that a majority of students do the exercises first and then read the text if they get an incorrect answer, rather than reading sequentially [35]. This suggests that summary exercises at the end of chapter helped the student decide what part of the lesson to focus on, and that great care needs to be made in developing the exercises to encourage reading of the complete materials.

6.3. Instructor experience

There are a number of ways that an interactive eTextbook can be used, either as lecture aides and homework exercises in traditional lecture-based courses, as support for “flipped” classes where students read the OpenDSA-based material
before engaging in classroom discussion, or as support for in-class labs. We have conducted several instructor interviews. Although we require more instructor use to make meaningful conclusions, the previous user of OpenDSA provided us with encouraging results. A summary of instructors answers to some questions is shown in Table 1.

Two of the three instructors used OpenDSA explicitly as an in-class teaching aid. One instructor used Powerpoint slides, and switched to OpenDSA to show the visualizations to the students. The instructor who used OpenDSA directly to cover material expressed the desire to be able to create course note using OpenDSA (HTML slides + AVs). We are working now on implementing that support based on ReStructuredText and Sphinx in a way that is fully integrated with the rest of OpenDSA. All instructors used OpenDSA assessment activities as homework assignments worth a fraction of total course grade, and they all indicated that they would like to use OpenDSA in-class as lecture aides in the future. We can hypothesize from the above findings that instructors will be inclined to use OpenDSA in ways that go beyond the use of traditional textbooks (readings assignments and homeworks).

Some of the net benefits of using OpenDSA mentioned by instructors included **Time** and **more practice** opportunities for students. One instructor noted that using OpenDSA allowed him to spend less time creating assignments, and that it is particularly helpful to new instructors, since they do not already have a bank of previous year assignments.
Table 1  
Instructor responses.

<table>
<thead>
<tr>
<th>Inst.</th>
<th>Useful?</th>
<th>Student feedback</th>
<th>Features requested</th>
<th>Future use</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Yes</td>
<td>Yes my students enjoyed using OpenDSA. Now they want all their courses to use OpenDSA model of eTextbook</td>
<td>Support for the generation of course notes (PowerPoint style slides) based on OpenDSA content</td>
<td>Yes</td>
</tr>
<tr>
<td>B</td>
<td>Yes</td>
<td>The students asked for more lectures with interactive examples, and more interactive exercises</td>
<td>More content, more coding exercises, and a bigger bank of questions</td>
<td>Yes</td>
</tr>
<tr>
<td>C</td>
<td>Yes</td>
<td></td>
<td>More content/topics</td>
<td>Yes</td>
</tr>
</tbody>
</table>

6.4. *Uniqueness of the OpenDSA technology*

OpenDSA implements several features that make it stand out among similar efforts.

- **OpenDSA infrastructure supports the creation and the use of multiple books.**

  As an open source project, it is of course possible for others to modify the content. OpenDSA provides explicit support whereby instructors can modify existing content in convenient ways to create a subset that is appropriate for their class. Configuration options include picking the modules and exercises to be included in the book, selecting exercises to be completed by the students in each module, and setting the points awarded to students upon completion of an exercise. Currently this configuration process is done by editing an extensive configuration file. We are working on a graphical user interface to make it easier for instructors to do configuration.

- **OpenDSA tutorial content supports multiple programming languages.**

  Instructors can create a version of a “eTextbook” to teach DSA using JAVA, or Python, or JavaScript as code examples within the visualizations, examples, and exercises. Ideally, versions of the example source code will be provided in each of these languages, and a given instructor merely needs to select a configuration option to get the desired language for the code examples.

- **OpenDSA eTextbooks can be used locally as a standalone system without the data collection/tracking infrastructure.**

  OpenDSA supports client-side grading and storage of data (using the HTML5 localStorage capability) for student self-monitoring. Unlike most courseware that operates behind a login wall, OpenDSA eTextbooks, AVs, and exercises can be used by anyone at anytime to get automatic feedback. The purpose of logging in is to allow the system to preserve and relate students’ progress, and report that to the instructor. An individual who does not need those features is free to use the system without logging in.

- **Many AVs and interactive exercises can be embedded in third party educational sites.**

  Most of the AVs and exercises are implemented in separate HTML pages with the intention of being embedded into the module pages. But they can be accessed directly, as well. When used this way, an instructor can decide if they want to use the OpenDSA data collection server to track their students’ progress.

  To further support independent use of our AVs and exercises, we also implemented an oEmbed (http://oembed.com/) web service. oEmbed is an open format which allows embedding content of a website on third party sites. The oEmbed endpoint allows instructors to seamlessly embed our AVs and exercises into their own online course materials.

- **OpenDSA can be integrated with other systems.**

  OpenDSA’s data collection server exposes an API that allows other systems to enter scoring data into OpenDSA’s database. Existing assessment systems can connect to our API and use HTTP requests to send and retrieve their data. Security is maintained by granting explicit permission to specific content server/sites through HTTP access control. However, third parties can also install a copy of our server-side system in order to support their own assessment process.

- **OpenDSA can be used as a testbed for detailed pedagogical experiments.**

  The ability to modify content and configure specific versions of eTextbooks make it easy to develop alternative treatments to test a number of pedagogical questions. Examples of experiments that we have already conducted or are planning in the near future include: (i) comparing the effects of giving credit for viewing slideshows versus not giving credit for viewing slideshows; (ii) comparing the effects of the various proficiency exercise feedback methods described in Section 4.3, and (iii) comparing text versus audio narration in algorithm presentations, as discussed in Section 6.1.

7. Conclusions and future work

As of this writing, OpenDSA content has been completed for lists and other linear structures; binary trees including BSTs and heaps; algorithm analysis; and hashing. A chapter on graphs is nearing completion, and there is initial text on a number of other topics. Material on topics such as file processing and buffer pools, dynamic programming, and string algorithms is also nearing completion. By the end of Fall 2013, sufficient material for a complete semester course will be available.

As it develops more complete content, OpenDSA will provide a rich collection of resources for instructors of DSA courses at all levels of the CS curriculum. As such, OpenDSA does not dictate how a given instructor will use those resources in the context of their classes. One simple option is for instructors to simply use OpenDSA in place of their existing textbook.
and homework exercises, or perhaps to use OpenDSA AVs to support their lectures. A slightly greater change would be for instructors to modify the emphasis for what gets covered in their lectures. Currently, OpenDSA does best at getting across the workings of algorithms, what might be considered as the more mechanical or procedural aspects of the topic. OpenDSA is not so good (or at least, not much better than a paper textbook) at presenting more abstract material such as algorithm analysis or issues related to which data structures to use in a given circumstance. Where traditionally instructors had to spend time presenting algorithm mechanics (and often not doing a good job due to the limitations of static media), now they can rely on OpenDSA to handle that part while they focus on the more difficult conceptual topics.

OpenDSA also provides support for radically different modes of teaching. OpenDSA modules could be done in a “closed lab” setting during class time. OpenDSA modules could play a key role in a “flipped classroom” organization. In this model, students are responsible for “doing the lecture” in advance, on their own. During class, the instructor has now freed up lecture time that can be used on activities such as working exercises, answering questions, leading discussions, or focusing on programming project design instead of traditional DSA lecture content. An important advantage of OpenDSA over standard textbooks within the flipped classroom model is that now, when the instructor says “do the reading before class”, OpenDSA lets the instructor know which students have completed their assignment.

OpenDSA could play an important role in distance learning. Massive Online Open Courses (MOOCs) have now entered the pedagogy lexicon and many are struggling to make them effective. OpenDSA supplies an important piece that is missing or less effective in many distance learning systems, and especially with many MOOC implementations. Since OpenDSA can go far beyond simple multiple choice questions, it allows automated assessment at scale for a rich set of exercises and assignments.

There remains much work to be done, with many opportunities for others to become involved as contributors within our open source model. In this paper we discussed a number of ongoing efforts to expand the existing OpenDSA core capabilities: translators from existing AV systems into JSAV; automated assessment of small coding exercises such as those provided by JhavéPOP; and pedagogical studies for a number of deployment approaches such as text versus audio. So far our major concern has been to develop the fundamental infrastructure and an initial body of content. As we move past that initial stage, we are rapidly moving to expand the number of early-adopter deployments in classes. So far we have not done proper learning gains studies since we had insufficient content necessary to have a major impact on a course such that we can see a meaningful effect size. Demonstrating pedagogical effectiveness will become our next major focus. Our other major goal will be expanding the types of interactive exercises that OpenDSA can support, especially small programming exercises.

References


