Visualization, Assessment and Analytics in Data Structures Learning Modules

Matthew Mcquaigue
The University of North Carolina
Charlotte, North Carolina
mmcquaig@uncc.edu

David Burlinson
The University of North Carolina
Charlotte, North Carolina
dburlins@uncc.edu

Kalpathi Subramanian
The University of North Carolina
Charlotte, North Carolina
krs@uncc.edu

Erik Saule
The University of North Carolina
Charlotte, North Carolina
esaule@uncc.edu

Jamie Payton
Temple University
Philadelphia, Pennsylvania
payton@temple.edu

ABSTRACT

In recent years, interactive textbooks have gained prominence in an effort to overcome student reluctance to routinely read textbooks, complete assigned homeworks, and to better engage students to keep up with lecture content. Interactive textbooks are more structured, contain smaller amounts of textual material, and integrate media and assessment content. While these are an arguable improvement over traditional methods of teaching, issues of academic integrity and engagement remain. In this work we demonstrate preliminary work on building interactive teaching modules for data structures and algorithms courses with the following characteristics, (1) the modules are highly visual and interactive, (2) training and assessment are tightly integrated within the same module, with sufficient variability in the exercises to make it next to impossible to violate academic integrity, (3) a data logging and analytic system that provides instantaneous student feedback and assessment, and (4) an interactive visual analytic system for the instructor to see students' performance at the individual, sub-group or class level, allowing timely intervention and support for selected students. Our modules are designed to work within the infrastructure of the OpenDSA system, which will promote rapid dissemination to an existing user base of CS educators. We demonstrate a prototype system using an example dataset.

KEYWORDS

interactive textbook, student engagement, visualization, learning analytics, data structures, algorithms

ACM Reference Format:
Matthew Mcquaigue, David Burlinson, Kalpathi Subramanian, Erik Saule, and Jamie Payton. 2018. Visualization, Assessment and Analytics in Data Structures Learning Modules. In *Proceedings of The 49th ACM Technical Symposium on Computing Science Education, Baltimore, MD, Feb. 21–24,2018 (SIGCSE '18),* 6 pages. https://doi.org/10.1145/3159450.3159460

SIGCSE '18, Feb. 21–24,2018, Baltimore, MD

ACM ISBN 978-1-4503-5103-4/18/02…$15.00
https://doi.org/10.1145/3159450.3159460

1 INTRODUCTION

Instructors in lecture based courses in introductory computer science have an expectation that the material presented in class is reinforced via textbook reading, lab exercises, assigned homeworks or quizzes. Given the rapid increases in the CS population over the past few years [39], these courses are large, and monitoring the performance of all students and reaching out to at-risk students is a challenge. This has resulted in new ways of teaching such courses, broadly classified as *active learning* techniques, that can include any combination of lab-based instruction, flipped classroom settings, gamification, peer-learning, or use of multimedia content [20, 23, 28, 33], all of which attempt to better engage students.

Our focus in this work is towards building new interactive learning modules and analyzing student performance in data structures and algorithms courses, which have exhibited significant drop rates of 40-60% [3, 39]. These modules can be part of interactive textbooks [35, 38, 40] and help reinforce lecture content outside of class. We also present visual analytic tools that will make it easier for the instructor to monitor and understand student performance during the course, thus allowing timely interventions of students who might be falling behind, or at-risk students. Our learning modules and visualization tools are extensions to OpenDSA [35] and the Canvas Learning Management System (LMS) [12] and have the following characteristics:

Interactive Modules. In contrast to the typical use of quizzes (short answers, multiple choice questions) in interactive textbooks, we propose highly interactive modules, in which students *directly interact with the visually represented data structure or algorithms* to complete the module. Students can practice an exercise any number of times in the learning or training phase (to ensure mastery of the material); this is followed by an uninterrupted assessment phase. Autograding provides immediate feedback on student performance to both the student and the instructor.

Academic Integrity. The constructed modules are built with enough variability so that no two students will see the exact same data structure or algorithm module simultaneously; this reduces academic integrity violations, and eliminates them in controlled situations, such as proctored exams, since neighboring students will receive different but equivalent modules as part of their assessment.

Student Performance: Instructor and Student Views. To analyze all of the data generated by the learning modules and regular assignments, we present visual analyses tools that communicate student performance on a daily or weekly basis to the instructor at multiple levels: individual student, selected subgroups of students, or of the entire class. Instructors can focus on specific groups of students (at-risk students, for instance), and ensure they get the support and attention via teaching assistants, tutors or direct interaction. Analytics is provided to help the instructor make sense of all the grading data, thanks to a biclustering algorithm that identifies highly correlated subsets of the grades. A limited version of these visualizations is also presented to the students.

Contributions. We propose a design that focuses on learning modules of critical material in data structures and algorithms courses, using direct interaction and visualization to promote student learning. The integration of learning and assessment within a visual and interactive environment is novel. The variability in exercises across students completing the same modules preserves academic integrity in most situations, while the autograding promotes scalability to large classes. The use of an open source textbook system (OpenDSA) with an existing base of users and integration with a commonly used Learning Management System (Canvas) will help dissemination and ease of adoption. Visual dashboards of student performance integrated into the LMS will provide instructor new tools for timely intervention and student support. We have built a prototype of the different components of our design[1], that includes five learning modules, and a visual exploratory tool to analyze student performance, powered by clustering algorithms.

2 RELATED WORK

Our emphasis in this work spans the areas of student engagement, interactive exercises, and visualization, targeted at student performance for improved learning outcomes. Interactive textbooks incorporate some of these features in improving the student engagement, by emphasizing a more hands-on approach to learning and assessment. Two such systems that have gained prominence in recent years include zyBooks [17, 37, 40] and OpenDSA [19, 35]. zyBooks is a commercial interactive textbook system that provides interactive tools, animations and responsive questions as part of its textbooks and has a large user base. Edgcomb et al. [17] discuss their use of zyBooks across multiple institutions for reading, studying and homework assignments in introductory programming courses and describe mechanisms to maintain academic integrity. OpenDSA is an open source interactive textbook system that uses the JSAV library [24, 25] for all of its interactions and visualizations. It provides visualizations of all the basic data structures and interaction capabilities. Our work is based on using the OpenDSA infrastructure to augment the interactive and visualization functionality, as well as provide new visual analytic capabilities for both students and instructors.

A large body of work has focused on visualization of algorithms and data structures for improved student engagement [2, 10, 32]. More recently, work by Burlinson et al. [11] combined the use of real-world data and data structure visualizations to improve student engagement. Bart et al. [5–7] have focused on curating a large number of interesting datasets for use in introductory courses in computer science. The use of visual programming (e.g., Scratch and Alice) [16, 34] has shown promise for making the first programming steps easier and more engaging. In addition to providing a graphical interface for piecing together programs; these systems let students build graphically interesting programs and encourage them to explore, experiment, and play. Formal evaluations of Alice [31] have shown increased performance and retention in the programing courses and improved attitudes toward computing, especially for at-risk students.

Researchers have also been looking into adding or exploiting learning analytics capabilities in an effort to improve student outcomes. Carter et al. [13] analyzed the different transition sequences and patterns that students go through when completing programming assignments and related that to student performance. Similarly, researchers have also looked at programming behavior by looking at compilation and hint statistics, in order to understand their impact on student performance in exams [1, 18]. Identifying at-risk students effectively, however, still remains a challenge. Khosravi et al. [26] applied clustering techniques to analyze patterns of summative, formative and behavioral data for a large flipped class. Bringing sophisticated learning analytics to understanding and classifying student groups is a goal of our work.

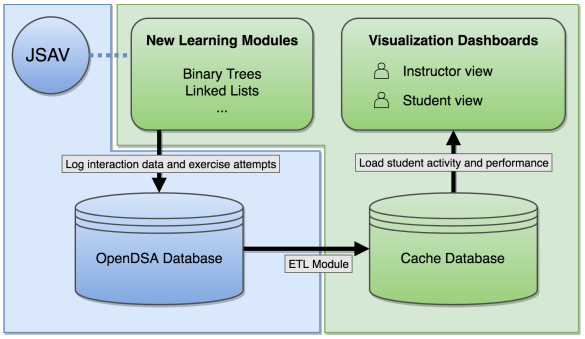

Figure 1: System Design. OpenDSA components are shown in blue and our contributions are in green.

3 METHODS

3.1 System Design

The OpenDSA infrastructure [19, 35] supports robust data collection from a variety of learning artefacts, including visualizations, interactive exercises, code authoring, and traditional multiple choice and response strategies, all of which accompany brief textual descriptions and instructions in a book instance on a Learning Management System (currently Canvas LMS is supported). Data related to exercise attempts, scores, and proficiencies for student work are stored as granular interaction logs as students progress through the visual and interactive components of a course.

Fig. 1 shows a system design overview with existent OpenDSA components on the left (in blue) and our contributions on the right (in green). Interactive exercises are created using the JavaScript

[1] A limited version of the learning modules and visualizations can be found at http://sigcse-87-2018.herokuapp.com/

Algorithm Visual Library (JSAV) [24, 25], which provides building blocks for web-based data structure and algorithm visualizations and an API for triggering built-in and custom logging events. Once a textbook and its exercises have been compiled and the corresponding course is generated on an LMS using the OpenDSA framework, these events can be logged to the OpenDSA MySQL database.

In addition to the new learning modules, we are in the process of adding three components to the OpenDSA framework to facilitate flexible visualization of the collected interaction and exercise attempt data (See Fig. 1). (1) An ETL (Extract, Transform, Load) module will read recent student and course data from the OpenDSA database, compute statistical metrics and aggregations to reshape the data, and send the resultant data to our secondary data store. (2) The secondary database will cache the output of the ETL module. This includes data related to student exercise progress, topic proficiency, and overviews of the progress of a class within recent chapters and modules. The database will be implemented with MySQL. (3) A visualization module will let students examine their progress and proficiency and allow instructors to view relevant information about the performance of the students in their course. The visualization module will be implemented alongside other OpenDSA components, using Ruby on the server side and web technologies (HTML, CSS, JavaScript) on the client side. The charts in the respective visualization dashboards will be built using a mix of Highcharts [22], a JavaScript visualization library, and D3 [15], a JavaScript library for manipulating and visualizing data. Each chart supports operations for reordering, filtering, and exploring the student data to provide timely insight into performance, and will judiciously utilize color schemes (from ColorBrewer [14]) and visual metaphors wherever possible.

3.2 Interactive Module Creation/Assessment

Our approach to building learning modules as part of an interactive textbook has the following design goals:

- Make the modules highly interactive and engaging, by using visual representations of the data structures that students *directly interact with* to master the underlying algorithms. In addition, modules should make students reflect on a given algorithm, as to what it does or what it might construct or produce as a result.
- Integrate assessment as part of the interactive learning modules, as opposed to having traditional quiz or short answer style questions, after the exercise.
- Maintain academic integrity by using a combination of automatic generation of examples from each module instance (each user sees a different but equivalent example) and autograding of that instance (OpenDSA provides a large amount of flexibility for building data structures and algorithms that can be customized to the goals of the learning module).
- Support for assessment at finer scales by assessing multiple instances of an exercise that a student works through, to obtain a better estimate of the student's mastery of the algorithm or concept.

Figs. 2 and 3 illustrate two example modules that are part of our system. The first module illustrates an exercise of inserting integer keys into a binary search tree. The student is presented with a

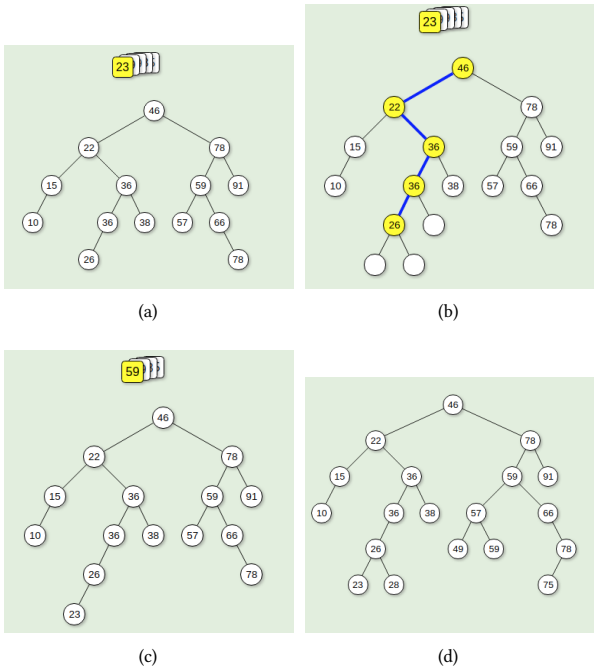

Figure 2: Interactive exercise: Inserting elements into a binary search tree, (a) Initial view showing a stack of values (at the top) to be inserted into the tree, (b) Inserting 23, after interactive identification of the path followed by the insertion algorithm, (c) Insertion of 23 into its correct location, (d) Final tree after insertion of all values. The user will use mouse clicks to identify the path taken by the insertion algorithm and will repeat this exercise for each of the values in the stack, followed by pressing the grade button for immediate assessment.

brief description of the algorithm and the required interaction; in this instance, the user clicks on the involved elements (Fig. 2(b)), until the final insertion position is found (Fig. 2(c)). Fig. 2(d) shows the final tree after all elements have been inserted into the tree. Three key features of the module are (1) the user can practice this exercise until he/she is comfortable, (2) each instantiation of the exercise creates a new tree with a different set of keys, and (3) the user can view a model solution of the module being completed step-by-step to understand how the module works. After viewing the module solution, the user must reset the module to be given a different set of keys. Once the user has mastered the exercise, and completed their final iteration of practice, the 'grade' button can be pressed to get the final grade. Note that the since each tree and set of keys are generated randomly, no two students will get the exact same problem. This makes collusion between students more time consuming since the solution of one instance will not be the solution of an other one. We paid attention to generating similar instances for all student (i.e., trees with good balance) to keep the difficulty of the different instances of the problem comparable.

Fig. 3 illustrates a second example. We are interested in training students to study an algorithm and illustrate its functionality or

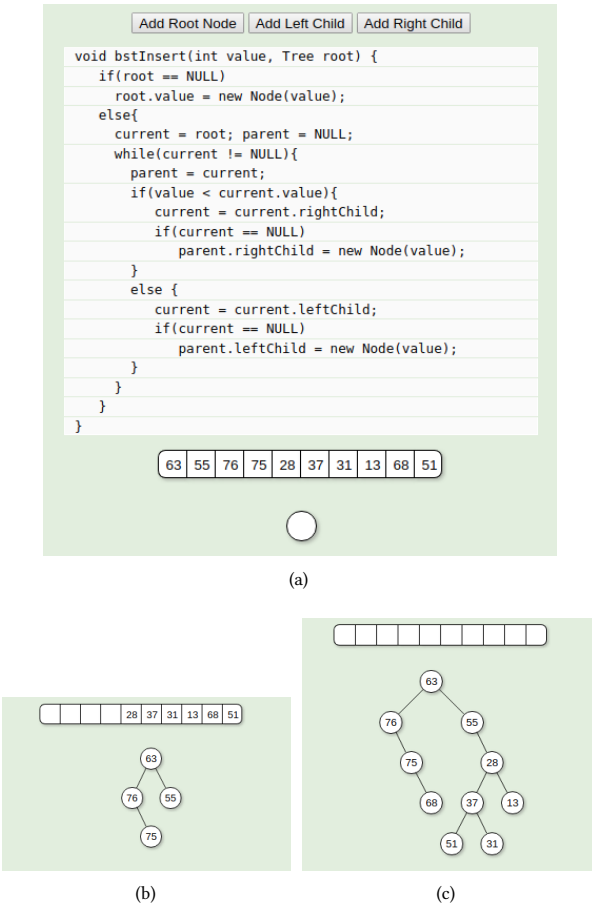

Figure 3: From an Algorithm to a Data Structure. In this exercise, the user interactively creates the data structure constructed by the given algorithm. User can create nodes and place them in left or right subtree of a node. In this example a set of values are to be inserted into an initially empty binary search tree. Intentionally, this algorithm will place smaller key values on the right subtree and larger values on the left subtree, (a) algorithm and values to build the tree, (b) after insertion of 4 values, (c) final tree.

Figure 4: An example data set of student performance in a course. Top panel illustrates the average student performance (blue curve) across a set of assignments, quizzes and exams throughout the semester, with the darker regions representing inter-quartile (25-75%) performance. Bottom panel shows a matrix plot of the same data, sorted by their overall average grade: students on the Y axis and assignments on the X axis. The data has been anonymized and actual grades randomized to only show the overall distribution.

what outputs it might generate. In this instance, the algorithm constructs a binary search tree by inserting a set of elements, starting from an empty tree. However, in this instance (Fig. 3(a)), we have altered the algorithm so that larger values will be inserted on the left subtree and smaller values into the right subtree (this is still a binary search tree, albeit an unconventional one). Fig. 3(a) shows the algorithm and Fig. 3(b), shows the partially constructed tree as more values get inserted into the tree, and Fig. 3(c) shows the final tree. For this exercise, the user is provided with interactive capabilities to create a node, assign a node to its left or right child, all done with mouse interaction. Once the exercise is completed (all the nodes in the array (Fig. 3(a)) are processed), it is automatically graded. Similar to the earlier example, we can vary the given algorithm and create a number of variations on the original insertion algorithm (for instance, insert elements only on the left or the right side of the tree). The ability to have such variability has 2 advantages, (1) preserves academic integrity (as students will see different algorithms when working on the same module), and (2) forces students to understand and reflect on the given algorithm.

In the current implementation, we have created modules for the basic operations on binary search trees (insert, delete, find) as well as multiple algorithms for constructing and operating on binary search trees. We have also begun implementing algorithms relating to linked lists (insertion of elements).

3.3 Visual Analytics

Introductory courses in computer science are generally large, and typically divided into a number of lab sections. Monitoring student performance is a challenge, and identifying at-risk students is an even bigger challenge. Introducing the assessed modules provide additional information to understand student behavior and performance in the class. But at the finer grain, each exercise or

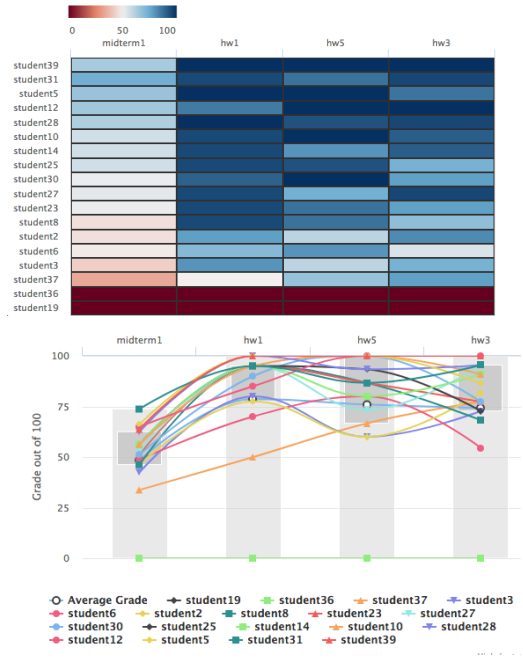

Figure 5: The flood of data contained in a grade book can be made sense of by using analytics such biclustering algorithms. Here one bicluster extracted by the CPB algorithm is shown spanning 18 students and 4 grades. This bicluster highlights that the grades on three homeworks and one midterm are correlated.

activity produces a grade which can create multiple grades per week. Because the instructor of the class will be flooded with low level information, we provide visualization capabilities to look at student performance at multiple scales, from individual student, to subgroups of students, all the way to the entire class. Data visualizations have been used by many researchers to look at post-mortem data, ranging from simple bar charts [29] to more sophisticated charts like parallel coordinates [18], and cluster visualizations [26].

Fig. 4 illustrates two visualizations of student performance in a course across a combination of assignments, projects and exams. The top plot illustrates average student performance with quartile performance. The lower plot shows the data of each student in a matrix form: students on the Y axis and the assignments on the X axis. Our system supports sorting this visualization based on average student grades, reordering the columns to reflect comparison across categories (homeworks, projects, exams, etc).

This flood of data can be hard to make sense of. We believe that automated analyses are necessary to help extend the visualization in order to reveal patterns that are difficult to see otherwise. As such, we propose to make use of biclustering techniques, which became very popular recently in the field of bio-informatics to analyze the expression level of thousands of genes across hundreds of patients [27]. In our problem a bicluster is a sub-matrix of the matrix grade, composed of a subset of the students and a subset of the grades. Algorithms for biclustering can search for different type of clusters, for instance OPSM [8] tries to identify high-value, low-value which could make sense when analyzing student grades.

We selected in this work the CPB algorithm [9] which looks for biclusters where rows are highly correlated and columns are highly correlated, according to Pearson Correlation Coefficient (PCC). The reasoning is that identifying a group of students who performed similarly across a group of activities can help understand why the students performed the way they did. For instance, if a biclusters links some students performance between an early recall-how-to-program assignment and project with programming components, one could hypothesize that the students performance in the projects are linked to their early programming skills. Note that a student who improved in programming during the semester would likely not be part of such a cluster. If any part of the class was designed to improve programming skills, one could hypothesize that the effort was not successful for the students in that bicluster.

Our visualization supports looking at particular biclusters returned by the algorithm. We computed biclusters using CPB on our test dataset looking for clusters of PCC greater than 0.95. Fig. 5 shows both the matrix and the grade plots for a particular bicluster. It shows that for a group of 18 students in that class, their score in 3 of the homeworks and the first midterm was correlated. Once again, it does not mean that these 18 students all performed well, or all performed poorly. It indicates there is a link between their performance in these 4 grades. In this particular case, the relation between homework 1 and 3 and the midterm is easily understood as the midterm covered the topics of homework 1 and 3, all using mathematical proofs and expressions. The biclusters do not indicate causality, just correlation. But that is enough in many cases for the instructor who knows the structure of the class to gain a better understanding of grades of the class.

Looking at a single bicluster can be helpful. But looking at multiple biclusters could be insightful as well. Visualizations can be done by reordering the grade matrix to improve the locality of the biclusters (which is done by BicAT [4] or BiCluster viewer [21]) or by representing the biclusters as a graph (which is done by ClusterMaker [30] or Furby [36]). One of the advantages of being able to see multiple biclusters is that one can try to understand a student's grade by looking at the biclusters that this student participate in, or does not participate in.

Hypothetically, one could see that the student's good midterm grade is explained by his or her good homework grade which highlights that the student understands the theoretical part of the class well. Also, one could see that the student's bad grade on the first project is correlated with bad grades in the early programming labs, showing that the student's programming skills were poor. But a lack of correlation with later projects could hint that the student's programming skills improved during the semester, maybe after the instructor suggested the student meets with the TA to strengthen his or her programming skills.

We believe that combining analytics and visualization can provide instructors tools to better understand the student population of a class and their performance. And while we showed how biclustering could help, we believe more complex analysis and visualization could shed a light on effects that are hard to understand from raw grades.

4 CONCLUSIONS

In this work, we have presented new learning modules that can be used as part of an interactive textbook in data structures and algorithms courses. Our modules are built to work with the OpenDSA system, using JSAV library for generating the module visualizations that have been customized to provide a highly interactive, visual and engaging experience for students. Learning and assessment are part of the same interaction, rather than a post-test, which is current practice in interactive textbooks. We have used the JSAV library to generate the modules with sufficient variability in the module instances, to ensure academic integrity. We also illustrate examples that demonstrate the ability to incorporate basic data structure algorithms into these modules and the means to grade automatically, thus incorporating complex and foundational concepts in an interactive and more engaging environment. We also illustrated visual analyses of student performance data using biclustering algorithms, so as to make sense of the grades generated by the learning modules

The current implementation demonstrates a prototype of the parts of the system described in Fig. 1. Much work remains to be done in connecting the various components and integrating the data into the visualization modules, prior to routine use as part of the LMS. This will be followed by formal user studies in data structures and algorithms courses. Finally, more work is needed in developing the visual analytics system to better understand the needs of the instructors as well as identification of at-risk students for timely intervention and support.

5 ACKNOWLEDGEMENTS

This work was supported by a grant from the National Science Foundation (DUE-1245841, DUE-1726809), and a summer research grant from UNC Charlotte.

REFERENCES

[1] Amjad Altadmri and Neil CC Brown. 2015. 37 million compilations: Investigating novice programming mistakes in large-scale student data. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*. ACM, 522–527.
[2] Ronald Baecker. 1998. Sorting out sorting: A case study of software visualization for teaching computer science. *Software visualization: Programming as a multimedia experience* 1 (1998), 369–381.
[3] L. Barker and T. Camp. 2015. Booming enrollments - What is the impact? *Computing Research News* 27, 5 (may 2015). Computing Research Association.
[4] S. Barkow, S. Bleuler, A. Prelic, P. Zimmermann, and E. Zitzler. 2006. BicAT: a biclustering analysis toolbox. *Bioinformatics (Oxford England)* 22, 10 (2006).
[5] A.C. Bart, E. Tilevich, S. Hall, T. Allevato, and C.A. Shaffer. 2014. Transforming Introductory Computer Science Projects via Real-time Web Data. In *Proc. of the 45th ACM Technical Symposium on Computer Science Education*. 289–294.
[6] Austin Cory Bart. 2016. CORGIS Datasets PRoject: The Collection of Really Great, Interesting, Situated Datasets. (2016). https://think.cs.vt.edu/corgis/.
[7] Austin Cory Bart, Ryan Whitcomb, Dennis Kafura, Clifford A. Shaffer, and Eli Tilevich. 2017. Computing with CORGIS: Diverse, Real-world Datasets for Introductory Computing. *ACM Inroads* 8, 2 (March 2017), 66–72.
[8] A. Ben-Dor, B. Chor, R. Karp, and Z. Yakhini. 2002. Discovering local structure in gene expression data: The order-preserving submatrix problem. In *International Conference on Computational Biology*. 49–57.
[9] D. Bozdağ, J. Parvin, and Ü Çatalyürek. 2009. A Biclustering Method to Discover Co-regulated Genes Using Diverse Gene Expression Datasets. In *Proc. of 1st Int'l. Conf. on Bioinformatics and Computational Biology (Lecture Notes in Computer Science)*, Springer (Ed.), Vol. 5462. 151–163.
[10] Marc H Brown and Robert Sedgewick. 1984. *A system for algorithm animation*. Vol. 18. ACM.
[11] David Burlinson, Mihai Mehedint, Chris Grafer, Kalpathi Subramanian, Jamie Payton, Paula Goolkasian, Michael Youngblood, and Robert Kosara. 2016. BRIDGES: A System to Enable Creation of Engaging Data Structures Assignments with Real-World Data and Visualizations. In *Proceedings of ACM SIGCSE 2016*. ACM, New York, NY, USA, 18–23.
[12] Canvas. Last accessed, Aug. 2017. (Last accessed, Aug. 2017). https://www.canvaslms.com/
[13] Adam Scott Carter and Christopher David Hundhausen. 2017. Using Programming Process Data to Detect Differences in Students' Patterns of Programming. In *Proceedings of the ACM SIGCSE 2017*. New York, NY, USA, 105–110.
[14] Colorbrewer. [n. d.]. ([n. d.]). http://colorbrewer2.org Last Access: July 2017.
[15] D3. 2017. (2017). https://d3js.org/
[16] Wanda P. Dann, Stephen Cooper, and Randy Pausch. 2005. *Learning to Program with Alice*. Prentice Hall.
[17] Alex Edgcomb, Frank Vahid, Roman Lysecky, and Susan Lysecky. 2017. Getting Students to Earnestly Do Reading, Studying, and Homework in an Introductory Programming Class. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education (SIGCSE '17)*. ACM, New York, NY, USA, 171–176.
[18] Anthony Estey, Hieke Keuning, and Yvonne Coady. 2017. Automatically Classifying Students in Need of Support by Detecting Changes in Programming Behaviour. In *Proceedings of ACM SIGCSE 2017*. New York, NY, USA, 189–194.
[19] Eric Fouh, Ville Karavirta, Daniel A. Breakiron, Sally Hamouda, Simin Hall, Thomas L. Naps, and Clifford A. Shaffer. 2014. Design and architecture of an interactive eTextbook - The OpenDSA system. *Science of Computer Programming* 88 (2014), 22 – 40.
[20] Mark Guzdial. 2003. A Media Computation Course for Non-majors. In *Proceedings of the ITICSE 2003*. ACM, New York, NY, USA, 104–108.
[21] J. Heinrich, R. Seifert, M. Burch, and D. Weiskopf. 2011. BiCluster viewer: a visualization tool for analyzing gene expression data. In *Proceedings of the Conference on Advances in Visual Computing (ISVC '11)*. 641–652.
[22] HighCharts. 2017. (2017). https://www.highcharts.com/
[23] Diane Horton, Michelle Craig, Jennifer Campbell, Paul Gries, and Daniel Zingaro. 2014. Comparing Outcomes in Inverted and Traditional CS1. In *Proceedings of the ITICSE 2014*. ACM, New York, NY, USA, 261–266.
[24] JSAV. 2017. (2017). http://jsav.io/
[25] Ville Karavirta and Clifford A Shaffer. 2016. Creating engaging online learning material with the JSAV javascript algorithm visualization library. *IEEE Transactions on Learning Technologies* 9, 2 (2016), 171–183.
[26] Hassan Khosravi and Kendra M.L. Cooper. 2017. Using Learning Analytics to Investigate Patterns of Performance and Engagement in Large Classes. In *Proceedings of the ACM SIGCSE 2017*. ACM, New York, NY, USA, 309–314.
[27] S. Kotian, T. Banerjee, A. Lockhart, K. Huang, U. V. Catalyurek, and J. D. Parvin. 2014. NUSAP1 influences the DNA damage response by controlling BRCA1 protein levels. *Cancer Biology and Therapy* 15, 5 (2014), 533–543.
[28] Celine Latulipe, N. Bruce Long, and Carlos E. Seminario. 2015. Structuring Flipped Classes with Lightweight Teams and Gamification. In *Proceedings of the ACM SIGCSE 2015*. ACM, New York, NY, USA, 392–397.
[29] Linxiao Ma, John Ferguson, Marc Roper, and Murray Wood. 2007. Investigating the Viability of Mental Models Held by Novice Programmers. *SIGCSE Bulletin* 39, 1 (March 2007), 499–503.
[30] J.H. Morris, L. Apeltsin, A.M. Newman, J. Baumbach, T. Wittkop, G. Su, G.D. Bader, and T.E. Ferrin. 2011. clusterMaker: a multi-algorithm clustering plugin for cytoscape. *BMC Bioinformatics* 12, 1 (2011).
[31] Barbara Moskal, Deborah Lurie, and Stephen Cooper. 2004. Evaluating the effectiveness of a new instructional approach. In *Proceedings of the ACM SIGCSE 2004*. ACM, 75–79.
[32] Willard C Pierson and Susan H Rodger. 1998. Web-based animation of data structures using JAWAA. In *ACM SIGCSE Bulletin*, Vol. 30. ACM, 267–271.
[33] Johanna Pirker, Maria Riffnaller-Schiefer, and Christian Gütl. 2014. Motivational Active Learning: Engaging University Students in Computer Science Education. In *Proceedings of ITICSE 2014*. ACM, New York, NY, USA, 297–302.
[34] Mitchel Resnick, John Maloney, Andrés Monroy-Hernández, Natalie Rusk, Evelyn Eastmond, Karen Brennan, Amon Millner, Eric Rosenbaum, Jay Silver, Brian Silverman, and Yasmin Kafai. 2009. Scratch: programming for all. *Commun. ACM* 52, 11 (2009), 60–67.
[35] Clifford A Shaffer, Ville Karavirta, Ari Korhonen, and Thomas L Naps. 2011. Opendsa: beginning a community active-ebook project. In *Proceedings of the 11th Koli Calling International Conference on Computing Education Research*. ACM, 112–117.
[36] Marc Streit, Samuel Gratzl, Michael Gillhofer, Andreas Mayr, Andreas Mitterecker, and Sepp Hochreiter. 2014. Furby: fuzzy force-directed bicluster visualization. *BMC Bioinformatics* 15 (suppl 6), S4 (2014).
[37] Frank Vahid and Alex Edgcomb. 2016. New CollegeLevel Interactive STEM Learning Material: Findings and Directions. In *EnFUSE Symposium, Washington, D.C., April 27-29, 2016*.
[38] Frank Vahid, Alex Edgcomb, Susan Lysecky, and Roman Lysecky. 2016. *New web-based interactive learning material for digital design*. Vol. 2016-June. American Society for Engineering Education.
[39] S. Zweben and B. Bizot. 2016. 2015 Taulbee Survey. *Computing Research News* 28, 5 (may 2016). Computing Research Association.
[40] Zybooks. 2016. (2016). www.zybooks.com Last Access: July 2017.