BRIDGES: A System to Enable Creation of Engaging Data Structures Assignments with Real-World Data and Visualizations

David Burlinson, Mihai Mehedint, Chris Grafer, Kalpathi Subramanian, Jamie Payton, Paula Goolkasian
University of North Carolina at Charlotte
{dburlins, mmehedin, cgrafer1, krs, payton, pagoolka}@uncc.edu

Michael Youngblood
PARC, A Xerox Company
Michael.Youngblood@parc.com

Robert Kosara
Tableau Research
rkosara@tableau.com

ABSTRACT

Although undergraduate enrollment in Computer Science has remained strong and seen substantial increases in the past decade, retention of majors remains a significant concern, particularly for students at the freshman and sophomore level that are tackling foundational courses on algorithms and data structures. In this work, we present BRIDGES, a software infrastructure designed to enable the creation of more engaging assignments in introductory data structures courses by providing students with a simplified API that allows them to populate their own data structure implementations with live, real-world, and interesting data sets, such as those from popular social networks (e.g., Twitter, Facebook). BRIDGES also provides the ability for students to create and explore *visualizations* of the execution of the data structures that they construct in their course assignments, which can promote better understanding of the data structure and its underlying algorithms; these visualizations can be easily shared via a weblink with peers, family, and instructional staff. In this paper, we present the BRIDGES system, its design, architecture and its use in our data structures course over two semesters.

Keywords

algorithm, data structure, visualization, engagement

1. INTRODUCTION

Over the past several years, enrollment in computer science undergraduate degree programs has been increasing at rates ranging from 6-11% [9]. Despite this increase, retention of CS majors remains a critical concern, especially at the freshmen/sophomore levels, with attrition rates of up to 66% [8]. While the factors that motivate changing majors are many and varied [8], strategies to increase retention are crucial to meet the national needs for graduates that can contribute to a robust and innovative twenty-first century workforce.

Demonstrating a connection between computing and the real world has the potential to increase students' motivation and interest in computing; grounding teaching in familiar, concrete, and relevant examples has been shown to improve learning [4] and has a positive effect on the retention of computing majors, particularly for women students [7]. In practice, however, there is little support for educators who want to incorporate this kind of approach into introductory computer science courses; for most students in introductory computer science courses, there remains a disconnect between the students' own everyday experiences with existing information systems that address real-world problems and the toy examples with small, uninteresting, synthetic data sets they encounter in course assignments that are used to teach students how data structures and algorithms work.

In this paper, we introduce the **BRIDGES** (**B**ridging **R**eal-world **I**nfrastructure **D**esigned to **G**oal-align, **E**ngage, and **S**timulate) system, which (1) facilitates student access to live, real-world data sets for use in traditional data structures programming assignments and (2) makes it possible for students to view and verify (debug) *their own implementations* of data structures, by providing *visualization* capabilities. The goal is to increase the engagement of students enrolled in data structures courses in an effort to improve retention of students in the computer science major. We present the architecture of BRIDGES and sample projects that can be enabled through the use of BRIDGES in a data structures and algorithms course. We describe a study of the application of BRIDGES in two semesters of a sophomore-level course; results show that the majority of students indicate that the BRIDGES-enabled assignments are relevant to career goals and increased their interest in computing.

SIGCSE '16, March 02-05, 2016, Memphis, TN, USA

DOI: http://dx.doi.org/10.1145/2839509.2844635

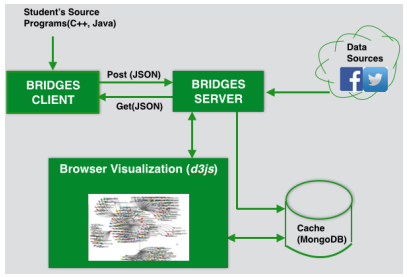


Figure 1: BRIDGES System. Based on a client-server architecture; a user constructs data structure programs using the BRIDGES client classes (Fig. 3) in Java or C++. The data structure representation is sent to the server by the client as a JSON string. After parsing and user authentication, the server stores the assignment on the MongoDB and displays the data structure on a specific web link using D3. The server also has interfaces to retrieve data from external sources upon user requests for use in their programs.

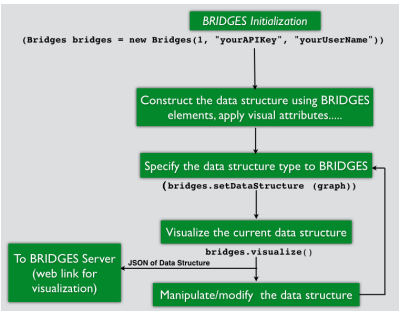


Figure 2: An Example BRIDGES program. Consists of an initialization step to specify user name and id (for authentication), assignment number. This is followed by user's construction of the data structure using BRIDGES client classes. The final 2 steps involve specifying a handle to the data structure (tree root, graph, head of a linked list, etc) and initiating the visualization modules.

2. THE BRIDGES SYSTEM

As illustrated in Fig. 1 the BRIDGES system is based on a client-server model; the client consists of the needed infrastructure for students to implement all of the basic data structures (currently Java and C++ APIs are supported). At any point, a representation of the constructed data structure can be generated and transmitted to the BRIDGES server to generate a visualization. The visualization will be displayed on a specific web link provided to the user. The BRIDGES server provides needed interfaces to data sources that make it easy for the user to make data requests. The acquired data (to be used by the user as part of course projects) is also cached on a Mongo database. Finally the server maintains a gallery of projects created by the user that can be shared. The visualization will appear on the specified web page with some capabilities for interactive manipulation. All projects are held in a database and the client will have access to an interactive gallery under their BRIDGES account.

2.1 An Example BRIDGES Program

We begin with an example template of what a BRIDGES program would look like. Fig. 2 shows the main calls made from an example program and consists of the following steps. The **Bridges** class encapsulates the details of the communication between the client and the server and uses additional helper classes in its implementation.

- **Initialization.** This step creates a BRIDGES class and takes several parameters, including an assignment number, a user id (generated by the user when he/she creates a BRIDGES account) and the corresponding user name. These parameters are used in forming a custom web link for the user's data structure.
- **Data Structure Construction.** In this step, the user constructs and manipulates any of the BRIDGES supported data structures using the BRIDGES client classes (see Section 2.2.1).
- **Specification of Data Structure Type.** This step specifies the handle to the data structure that will be transmitted to the BRIDGES server for visualization. This can be the head of a linked list, root of a tree, graph adjacency list, or array name.
- **Visualization.** This step results in the creation of a JSON string representation of previously specified data structure and its transmission to the BRIDGES server using an HTTP post request. If this is successful, a web link is returned to the user for viewing the visualization. The assignment is also stored in the MongoDB that the user can subsequently access through his BRIDGES account. This step can be repeated any number of times: the data structure can be modified, the handle respecified followed by visualization.

2.2 BRIDGES Design

2.2.1 *BRIDGES Client*

The BRIDGES client consists of a minimal set of building blocks that are needed by students in a typical sophomore level course on data structures: array, list, tree structures, graph. The class structure is illustrated in Fig. 3, roughly follows the description and implementation of Shaffer[16, 15] contains the following components:

Element. This is the foundational class in BRIDGES. Arrays, lists, tree and graph nodes are constructed using this element. Elements have a unique id, a label (used in the visualization), and visual properties (size, shape, opacity, color). Elements can also be related to another element via a *link*, as would be needed for trees, linked lists and graphs. Links have attributes: color, thickness, and opacity. Elements are declared with a generic parameter, that can be used to hold application specific data (Tweet, actor, movie, etc)

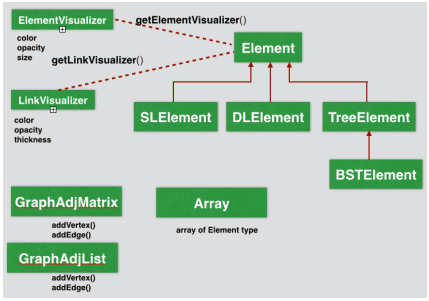


Figure 3: BRIDGES Client. Consists of the basic building blocks to construct data structures, such as arrays, linked lists, binary trees and graphs. The visualizer classes affect the visual attributes (color, opacity, size, thickness) of the elements (nodes) and links, depending on the data structure.

SLElement. Derived from Element, SLelement represents a singly linked element and has a link to the succeeding element, via a *next* pointer.

DLElement. Derived from Element, DLelement represents a doubly linked element and has links to previous and succeeding elements, via *prev* and *next* pointers.

TreeElement. Derived from Element, TreeElement is a binary tree node, with links to its two children, via *left* and *right* pointers.

BSTElement. Derived from TreeElement, augments the TreeNode with *key* value; must be an orderable type.

GraphAdjMatrix. Implemented as a two dimensional hash table, the graph supports vertices indexed by any orderable type (int, float, string, etc).

GraphAdjList. Implemented as a hash table of vertices, with each table entry pointing a singly linked list (of type SLelement);

Array. Arrays represent a list of type Element and support the normal indexing operations of arrays as defined in programming languages; however, each element of a BRIDGES array supports visual attributes.

Implementation. Implementations using the above class structures closely follow the implementation examples in [16, 15]. However, by using the object definitions above, the basic data structure elements can now be augmented with visual attributes that can be exploited by the visualization modules. BRIDGES currently supports both Java and C++ implementations. The Java implementation uses *Java Generics*; while the C++ implementation uses *C++ templates*. Generic implementations make it possible to handle real-world datasets where indexing by string (Twitter or Facebook user name, for example) is needed for graph implementations, bypassing a remapping to integer vertex indices.

A second advantage is the ability to incorporate application specific dataset as a generic parameter to the base classes.

2.2.2 *BRIDGES Server*

The BRIDGES server has the following functions:

- **Access to Real-World Data.** As described earlier, APIs to real-world datasets such as social networks (Twitter, Facebook) can be quite complex and beyond the scope of sophomore level CS students. Thus BRIDGES provides an easy to use interface to acquire data from external data sources. To date, implemented interfaces to services include Twitter, RottenTomatoes, and IMDB, and data from Twitter follower lists and timelines from a few public accounts (ieeevis, twitterapi, wired, US geological survey, and earthquake) have been incorporated directly into course projects. Queried data from the various sources is cached in a Mongo database for the sake of efficiency and data reuse.

- **Visualization.** The BRIDGES server is responsible for receiving data structure representations (in JSON string format) from the client and generating a visual representation (students will be provided a web link). All such requests are authenticated, parsed and stored in a database (MongoDB). Uploaded assignments can be accessed with a direct url or via the user gallery and can be made public or private, allowing users to share some visualizations and hide others for later review or modification. Visualizations of arrays, linked lists, binary trees and graphs (node-link diagrams) are supported by BRIDGES.

Implementation. The server-side implementation is a Node.js application, utilizing Jade templating, a MongoDB database, Javascript, and a variety of popular Javascript libraries, such as D3[3], jQuery, and Underscore.js. The database is used to keep a record of all users, projects, and cached datasets.

2.3 BRIDGES Intervention in Data Structures Courses

In our CS program, students take the data structures course at the beginning of their sophomore year, followed by an algorithm analysis course. All BRIDGES projects were assigned to students in the data structures course. Typical enrollment in this course is around 50. The BRIDGES intervention was performed in 1 section of the data structures course in fall 2014 and Spring 2015.

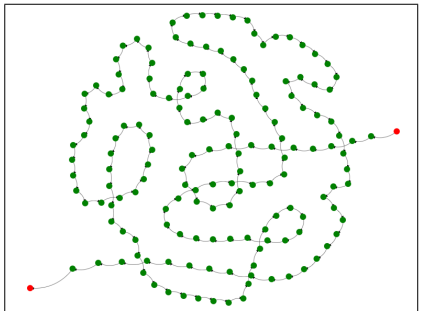
2.3.1 *Fall 2014*

Three BRIDGES projects were assigned in the Fall 2014 semester:

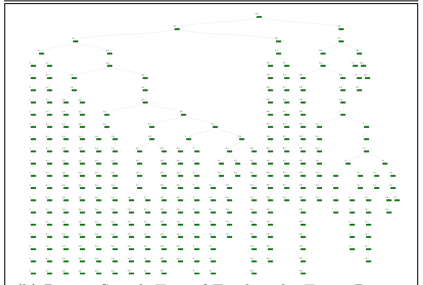
- **Queue:** In this project students to implement and test the Queue ADT, followed by using a live stream of Earthquake Tweet data (US GIS earthquake Tweets). Helper classes were provided to parse the quake data to simplify the data processing tasks. Students had to complete two tasks (a) Given a fixed size queue, incoming tweet data were enqued; if the queue became full, then the oldest tweets were dequed and snapshots of the queue visualizations were to be demonstrated, (b) Given a fixed size queue, incoming data items were to be filtered by quake magnitude prior to entering the queue. Various thresholds should be experimented with queue snapshots visualized.

- **Binary Search Tree.** This project continued to use the earthquake Tweet data, but inserted the records (quake magnitude as the search key) into a binary search tree, followed by visualization of the tree structure. Tasks on the search tree included (a) modifying the insertion algorithm to display insertion path, (b) implement the find algorithm and demonstrate (visually) searching for a quake of a particular magnitude. Fig. 4b illustrates the binary search tree sorted by earthquake magnitudes.

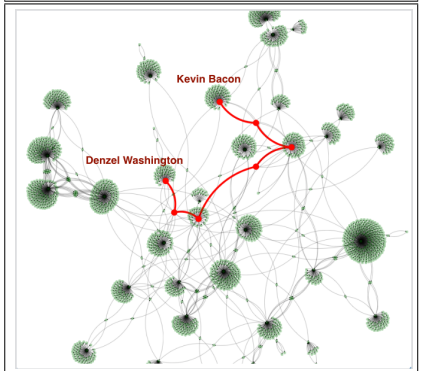
- **Graph (Bacon Number Computation).** This project involved building a graph using a reduced version of the IMDB dataset and implementing the Breadth First Search algorithm on a graph to compute the Bacon Number of an actor in an actor-movie graph[14]. The project used a curated IMDB dataset containing 1815 actor-movie pairs. Tasks involved building the actor-movie graph, determining the Bacon Number of any actor in the graph, and highlighting the path from the queried actor to the Kevin Bacon node. Fig. 4c illustrates the actor-movie graph and the path from the actor Denzel Washington to Kevin Bacon.



(a) Singly Linked List of IMDB dataset containing 1815 actor-movie pairs. Each node is a unique actor; mousing over a node (not shown) displays a list of that actor's movies.



(b) Binary Search Tree of Earthquake Tweet Dataset (US GIS), ordered by quake magnitude.



(c) Bacon Number Computation. Actor-Movie Graph of an IMDB dataset containing 1815 actor-movie pairs. Results of the Bacon number computation between Denzel Washington and Kevin Bacon. The red nodes and links display the path from Denzel Washington to Kevin Bacon.

Figure 4: BRIDGES Project Examples

2.3.2 *Spring 2015*

Four BRIDGES projects were assigned in the course:

- **Singly Linked List.** The IMDB actor-movie dataset was used in this project. Students had to read in the dataset and build a sorted linked list of the unique set of actors; the data field of each node would contain the list of movies corresponding to the actor. Tasks included finding a specific actor, followed by highlighting the node (if found), adding in new actor-movie pairs and removing an actor. Fig. 4a illustrates the singly linked list sorted by actor names.

- **Stacks: Expression Evaluation.** This project required students to build a linked list-based stack using BRIDGES elements. The stack was then used to evaluate expressions. Bridges visualizations were used to display the contents of the stack after each operation.

- **Binary Search Tree.** This project was similar to the project from the fall 2014 with slight changes in the required tasks.

- **Graph (Bacon Number Computation).** Identical to the Fall 2014 graph project.

3. EVALUATION

We have used BRIDGES in one section of our Data Structures course (ITCS 2214) for the past 2 semesters; the projects that used BRIDGES are described in Sections 2.3.1,2.3.2. The enrollment in each of these 2 sections was capped at 55; the enrollment stabilized at 36 in the Fall 2014 semester and 32 in the Spring 2015 semester. Each BRIDGES project was followed by a project survey with 15 questions relating to the concepts learned in the assignment, the required programming knowledge relative to their own experience, time required for completion of assignment, sources of support

Table 1: Responses to "The assignment was relevant to my career goals" for assignments 1-4.

	#	#Resp	SA%	A%	N%	D%	SD%
Fall 2014	1	29	0.0	48.2	20.7	31.0	0.0
	2	28	17.9	28.6	28.6	17.9	7.1
	3	19	15.8	52.6	15.8	10.5	5.3
Spring 2015	1	27	14.8	14.8	55.6	7.4	7.4
	2	24	29.1	37.5	29.1	4.1	0.0
	3	27	25.9	37.0	29.6	3.7	3.7
	4	28	35.7	25.0	25.0	7.1	7.1

Table 2: Responses to "The assignment was trivial and not essential to learning about computing." for assignments 1-4.

	#	#Resp	SA%	A%	N%	D%	SD%
Fall 2014	1	29	3.5	17.2	24.1	44.8	10.3
	2	28	3.6	10.7	17.9	60.7	7.1
	3	19	5.3	10.5	15.7	42.1	26.3
Spring 2015	1	27	7.4	11.1	25.9	40.7	14.8
	2	24	0	0	25.0	41.6	33.3
	3	27	7.4	0.0	14.8	44.4	33.3
	4	28	10.7	3.6	25.0	21.4	39.2

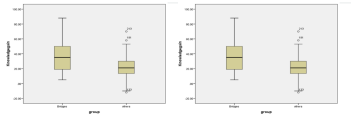
(mentors, TAs, external sources), relevance of the assignment to career goals, difficulty of the assignment, and degree to which the assignment increased their interest in computing. Responses used a 5 point Likert scale, ranging from strongly agree to strongly disagree. Tables 1,2, and 3 detail the responses to the survey questions, (1) *"The assignment was relevant to my career goals"*, (2) *"The assignment was trivial and not essential to learning about computing."*, and (3) *"The assignment increased my interest in computing"*.

- *"The assignment was relevant to my career goals"*. In the fall 2014 surveys (Table 1), a majority of students agreed on this question, on all 3 projects (48% vs. 31%, 46% vs. 25%, 68% vs. 16%) with the remainder being neutral. In the spring semester, the results were even stronger (29% vs. 14%, 66% vs. 4%, 63% vs. 7%, 61% vs. 14%), except for the first project where there were a large number of neutral responses (55%). This was a more difficult project that should have been assigned in multiple parts; the subsequent projects were corrected, so that students found the tasks more manageable.

- *"The assignment was trivial and not essential to learning about computing"*. In the fall 2014 surveys (Table 2), a majority of students disagreed (55% vs. 38%, 68% vs. 14%, 68% vs. 26%) for all projects and in the spring surveys the results were even stronger (55% vs. 18%, 75% vs. 0%, 77% vs. 7%, 61% vs. 14%).

Table 3: Responses to "The assignment increased my interest in computing" for assignments 1-4.

	#	#Resp	SA%	A%	N%	D%	SD%
Fall 2014	1	29	0.0	20.7	44.8	24.1	10.3
	2	28	10.7	25	32.1	21.4	10.7
	3	19	5.3	42.1	31.6	15.8	5.3
Spring 2015	1	27	3.7	22.2	40.7	14.8	18.5
	2	24	20.8	45.8	25.0	8.3	0.0
	3	27	14.8	48.2	25.9	7.4	3.7
	4	28	21.4	32.1	21.4	17.9	7.1



(a) Fall 2014 (b) Spring 2015

Figure 5: Knowledge Gains using BRIDGES

- *"The assignment increased my interest in computing"*. In the fall 2014 surveys (Table 3), the results were mixed, with a minority agreeing on project 1 (21% vs. 34%), slight majority on project 2 (35% vs. 32%), and a majority on project 3 (47% vs. 24%). In the spring 2015 surveys, a minority agreed on project 1 (26% vs. 33%), a majority agreed on the remaining 3 projects (66% vs. 8%, 63% vs. 11%, 53% vs. 25%). As mentioned earlier, project 1 was not modularly designed to help students to complete subtasks as part of the larger assignment, resulting in frustration for many students.

Finally, there was a major version change in BRIDGES between the two semesters. Version 2 of BRIDGES was much more robust, with an improved API (Fig. 3) that was easier to use; additionally, the documentation and tutorials were also vastly improved. Thus, the improvements in student responses across the two semesters were within our expectations.

Knowledge Gains. We also evaluated the knowledge gains of students using BRIDGES compared to students in the remaining sections of data structures. For this we created a knowledge test in data structures with contributions from all instructors teaching the course. A question bank of about 105 multiple choice questions and 38 short answer questions was created. An external project evaluator chose a subset of these questions (35 multiple choice and 6 short answers) to be used for pre/post tests, administered at the beginning and end of the semester. Students from four sections of the fall 2014 data structures course and 4 sections of the Spring 2015 underwent the the knowledge test; 1 section used BRIDGES and the other 3 did not.

Fig. 5 illustrates the knowledge gains using the pre-post scores from the knowledge test. The box plot in Fig. 5 shows knowledge gains from pre-test to post-test, with significant gains for both the BRIDGES section and the control group. It is apparent that the BRIDGES group showed larger gains ($M = 39.12$, $SD = 14.6$) than the control group ($M = 22.00$, $SD = 13.28$), $t(93) = 5.33$, $p < .001$. However, each section was taught by a different instructor. As such, the differences in knowledge gains for the groups could be explained at least partially by the differences in instructor emphasis on the knowledge tests; the BRIDGES instructor used part of the knowledge test for the final exam while the other instructors used it as a classroom exercise that did not count toward the final grade. Therefore, it is not possible to definitively pinpoint BRIDGES as a reason for increased knowledge gains.

4. RELATED WORK

Closely related to our work are the approaches in Buckley et al.'s *Socially Relevant Computing* [6] and Bart et al.'s *RealTime Web*[2]; Buckley's work incorporates real world scientific applications into both their introductory and senior capstone courses to make them more interesting and relevant. Bart's *RealTime Web* provides a set of flexible client libraries to request, parse and return real-time data from a number of web sources (Yelp, weather reports, Yahoo Finance, etc.) Our approach goes even further to make the course material relevant; in addiiton to providing easy access to real-world datasets, we provide instantaneous visualizations of the data structures that are built by the students themselves that can help to improve the understanding of the data structure and the algorithms that operate on them.

Visualizations have long been promoted as a way to improve student understanding of data structures and algorithms [1, 12, 5], and the research literature highlighting the ability of interactive visualizations to enhance discovery is vast. Previous efforts to increase engagement have shown promise for the use of visual programming (e.g., Scratch and Alice) [13, 10] 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[11] have shown increased performance and retention in the programing courses and improved attitudes toward computing, especially for at-risk students.

5. CONCLUSIONS

We have presented BRIDGES, a system that provides students with an easy-to-use API for accessing real-world datasets for use in data structures course projects and the ability to explore interactive visualizations of the data structures that are created as part of a BRIDGES-enabled assignment. Rather than look at (uninteresting) textual output of data structures assignments for verification/debugging, students can use BRIDGES to interact with visualizations of the execution of their assignments on real-world data, with the added ability to share their visualizations with peers, friends, and family. Early results of using BRIDGES over 2 semesters in 1 section of our data structures course are promising, with students reporting increased interest in computing after completing assignments.

Currently, we are extending BRIDGES on 3 fronts:

- **External Data Sources.** Ultimately, we expect that BRIDGES will provide access to large variety of interesting external data sources. We also hope to integrate the rich work of *RealTime Web*[2] to complement BRIDGES.

- **Peer Mentoring.** An important component of this project is to build strong connections within the major through peer mentoring, using BRIDGES as a shared interest; specifically, we plan to pair students in data structures courses with the students in the senior-level software development capstone course who are tackling software development projects to extend BRIDGES.

- **Extended Evaluation.** BRIDGES is open source and is available at **http://bridgesuncc.github.io/**. Java and C++ versions are available for deployment by instructors. As more instructors adopt BRIDGES, we plan to perform a larger scale evaluation that measures computing attitudes and engagement in data structures and algorithms courses across institutions.

6. ACKNOWLEDGMENTS

This work was supported by a grant from the National Science Foundation, DUE-1245841.

7. REFERENCES

[1] R. Baecker. Sorting out sorting: A case study of software visualization for teaching computer science. *Software visualization: Programming as a multimedia experience*, 1:369–381, 1998.

[2] A. C. Bart, E. Tilevich, S. Hall, T. Allevato, and C. A. Shaffer. Transforming introductory computer science projects via real-time web data. In *Proc. of the 45th ACM Technical Symposium on Computer Science Education*, pages 289–294, 2014.

[3] M. Bostock, V. Ogievetsky, and J. Heer. D3: Data-driven documents. *IEEE Trans. Visualization & Comp. Graphics*, 2011.

[4] J. D. Bransford, A. L. Brown, and R. R. Cocking. *How people learn: Brain, mind, experience and school*. National Academy Press, 1999.

[5] M. H. Brown and R. Sedgewick. *A system for algorithm animation*, volume 18. ACM, 1984.

[6] M. Buckley, J. Nordlinger, and D. Subramanian. Socially relevant computing. In *Proc. of the 39th SIGCSE Technical Symposium on Computer Science Education*, pages 347–351, 2008.

[7] J. Cohoon. Just get over it or just get on with it. In *Women and Information Technology: Research on Under-Representation*. MIT Press, 2005.

[8] J. Cohoon and L.-Y. Chen. Migrating out of computer science. *Computing Research News*, 15(2), 2003. http://archive.cra.org/CRN/articles/march03/cohoon.chen.html.

[9] Computing Research Association. CRA Taulbee Survey 2010-2011, 2010-2011.

[10] W. P. Dann, S. Cooper, and R. Pausch. *Learning to Program with Alice*. Prentice Hall, 2005.

[11] B. Moskal, D. Lurie, and S. Cooper. Evaluating the effectiveness of a new instructional approach. In *Proc. of the 35th SIGCSE Technical Symposium on Computer Science Education*, pages 75–79, 2004.

[12] W. C. Pierson and S. H. Rodger. Web-based animation of data structures using jawaa. In *ACM SIGCSE Bulletin*, volume 30, pages 267–271. ACM, 1998.

[13] M. Resnick, J. Maloney, A. Monroy-Hernández, N. Rusk, E. Eastmond, K. Brennan, A. Millner, E. Rosenbaum, J. Silver, B. Silverman, and Y. Kafai. Scratch: programming for all. *Communications of the ACM*, 52(11):60–67, 2009.

[14] R. Sedgewick and K. Wayne. *Introduction to Programming in Java*. A Case Study: Small World Phenomenon (http://introcs.cs.princeton.edu/java/home/).

[15] C. Shaffer. *Data Structures and Algorithm Analysis in C++*. Dover Publications, 2011.

[16] C. Shaffer. *Data Structures and Algorithm Analysis in Java*. Dover Publications, 2011.

Engaging Early Programming Students with Modern Assignments Using BRIDGES*

Allie Beckman[1], Matthew Mcquaigue[1], Alec Goncharow[1]
David Burlinson[1], Kalpathi Subramanian[1]
Erik Saule[1], Jamie Payton[2]
[1]Computer Science, UNC Charlotte
{abeckma2, mmcquaig, agoncha1, dburlins, krs, esaule}@uncc.edu
[2]Computer and Information Sciences, Temple University
payton@temple.edu

Abstract

Early programming courses, such as CS1, are an important time to capture the interest of the students while imparting important technical knowledge. Yet many CS1 sections use contrived assignments and activities that tend to make students uninterested and doubt the usefulness of the content. We demonstrate that one can make an interesting CS1 experience for students by coupling interesting datasets with visual representations and interactive applications. Our approach enables teaching an engaging early programming course without changing the content of that course. This approach relies on the BRIDGES system that has been under development for the past 5 years; BRIDGES provides easy access to datasets and interactive applications. The assignments we present are all scaffolded to be directly integrated into most early programming courses to make routine topics more compelling and exciting.

1 Introduction

Computational literacy and problem-solving skills are crucial facets of an increasingly tech-driven economy and world. While enrollments in computing majors have grown in recent years, particularly high attrition rates in these degree programs hamper the rate at which colleges and universities contribute to the modern workforce. Students in introductory and second-year courses are most susceptible, and much work has been done to investigate and address the factors contributing to the erosion of this student population [2].

One of the primary factors in maintaining interest among computing majors is their level of engagement with the course material. The perceived relevance of the material to the students' own lives and careers is crucial for kindling a desire to learn how to solve more complex problems down the road. Unfortunately, this is an area of weakness in many computing programs: the introductory courses are packed full of students, and it is enticing for universities to prioritize scalability over quality and rigor by relying on graduate students or more automated tools and paradigms to teach and evaluate the basic content.

Programming assignments in introductory courses have traditionally been contrived to teach basic structures like logical branching and loops using toy datasets, with basic command line input and output comprising students' interaction with the program. More engaging, modern approaches generally involve socially or culturally relevant data, simple graphical libraries, and more gamification of the material. All these features are prioritized in our work.

We present in this paper a lean educational framework that makes student engagement central to the content of introductory programming courses, without changing or compromising the rigor, content, and learning goals of these courses. Our BRIDGES framework uses a mix of real-world datasets from different domains, simple games and interactive applications as engagement tools to emphasize and reinforce core concepts in introductory programming. We present a set of assignments and their relationships to programming concepts, and we show how they were deployed in introductory CS courses and perceived by students of these classes. The framework and assignments are available online (`https://bridgesuncc.github.io/`).

2 Related Works

What CS1 typically look like. The typical first course in computer science, or CS1, introduces students to programming using a high level programming language such as Java or Python. The course goal is to introduce the basic constructs of the programming language, such as variables and expressions, control structures, functions and simple data structures like lists and strings. These courses train students to solve nontrivial problems using these constructs (e.g., see classic CS1 exemplar [11]).

There are wide variations in how the basic concepts are taught, in terms of the learning environments, tools, pedagogical approaches, and the student pop-

ulation (majors, non-majors) and demographics. Many introductory courses have begun to incorporate graphics, GUIs, and visualizations, as a creative output produced in projects [8, 4] or to illustrate key aspects of the underlying objects or algorithms [7, 3]. Furthermore, a rising awareness of the multi-disciplinary value of computing literacy has encouraged some institutions to experiment with different flavors of introductory programming courses based on game development [1], robotics [6], and data science [5].

What Makes Students Engaged. Ultimately, the goal of courses and curriculum is the overall education and academic success of the learners. To that end, materials that capture the imagination of incoming students and reinforce their interest and motivation in computing are particularly valuable. Popular assignment repositories (Nifty Assignments [14], EngageCSEdu [13], etc.) tend to include the 'fun' factor, as do game-themed assignments [16]. Usage of real-world and large datasets in course projects have also proven successful [12, 4], in contrast to using tiny, contrived or toy examples that fail to engage students.

In recent years, active learning techniques have been implemented in classrooms to promote student engagement, and include any combination of lab-based instruction, flipped classroom settings, gamification, peer-learning, and use of multimedia content [15, 9, 10].

3 The BRIDGES System

The BRIDGES system [4] is relevant to the goals of introductory CS courses such as CS1 and CS2. It has the capability to provide easy access to external datasets that can be readily used in course assignments. Secondly, it provides a 2D abstraction of a grid that can be used for games and image processing. The system provides bindings for the commonly used languages in early CS courses (Java, C++, and Python). Finally, results from assignments are highly visual and can be shared with friends and family, thanks to web-based rendering.

Dataset Access. BRIDGES provides simple APIs to access external data sources such as USGS Earthquake data, Wikidata, IMDB actor/movies, Genius' Song Lyrics, and OpenStreet Maps. A single function call returns data from a specific source, typically a list of objects, that can then be directly used. By using interesting datasets from different domains, assignments can be made more real and relevant to students.

Visualizing Bitmap Images. BRIDGES supports bitmap images through a 2D grid abstraction which can be the basis for assignments on images and

image processing. This also directly relates to 2D arrays and array addressing, which are central to CS1 and CS2.

Game API. BRIDGES supports a simple Game API that forms the basis for a number of 2D games that are readily usable and aligned with the goals of early CS courses. The game API has 4 core functions, (1) Reading Inputs, (2) Updating the game state using customized game mechanics, (3) Rendering to screen, and, (4) Maintaining a frame rate of 30 frames per second. In order to maintain simplicity of the API for new programmers and ensure smooth rendering, 2D games are implemented as 2D grids with a maximum of 1024 cells where each can be assigned a color and one of 256 predefined sprites, and 10 input keys for interaction. These constraints still enable the construction of many games and applications, and the *(student) programmer can focus on implementing the game logic and updates to the game state, while the display output and graphics are the responsibility of the system.* This lets the student focus on the course-level goals and not on the tool-level details. The user is responsible for implementing two functions: (1) initialize(), which is run once at the beginning of the game, and (2) gameLoop(), which contains all of the game logic and is called for each frame of the game.

This API is simple yet expressive enough to enable the implementation of a number of 2D games such as Bugstomp, Snake, 2048, Infinite Runner, Minesweeper, and Racing Car; as well as many of the Nifty [14] assignments such as Falling Sand, Spreading of Fire, and Hurricane Tracker. Each of these can be scaffolded to align with the learning outcomes of an early CS courses.

Students will typically run the code in their IDE and interact with the game through a web browser. However, our system also supports exporting games as standalone Android applications.

4 A Set of Engaging CS1 Assignments

We now describe a sequence of engaging assignments using BRIDGES that can be the basis for a CS1 curriculum. Table 1 illustrates the assignments and topics these assignments are aligned with. Assignments are scaffolded (available on our website) so that the students will see the specific functions that are to be implemented, in line with the objectives of the assignment. Instructors may request solutions for planning their course.

Etch a Smile. The student is given the task of drawing a smiley face. There are two versions of this assignment, Students write single lines of code which fill specific cells on a 2D grid using x and y values with a color of their choice. Students can also draw symbols on a cell. Alternately, they can use loops to

Assignment	Topics	Engagement
Etch A Smile	Functions	creativity, fun, visual output
BugStomp	Conditionals	Game Interaction
Tic Tac Toe	Conditionals, Std. I/O	Game Interaction
Song Lyrics	2D arrays, loops, conditionals, strings	real data, explore personal choices
Image Proc.	2D arrays, loops, File I/O	real data, visual output
Mountain Paths	2D arrays, loops, greedy algorithms	real data, visual output

Table 1: Example CS1 Projects, Topic Mappings and engagement factors

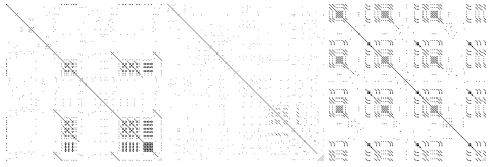
fill areas of the 2D grid with specific colors to create a face.
Where does it fit? Students will gain experience with generating a visually pleasing output using code that is more interesting than vanilla "hello world" function. They will also gain an understanding of coordinate grids from a programmers perspective. Familiarity with loops can be used to augment this assignment. Fig. 1 illustrates some examples.

Figure 1: Etch A Smile Face Example

Bugstomp. The game involves moving a sprite around a 2D grid to step on randomly generated bug sprites. The assignment involves moving around a 2D grid, generate random positions to place bugs in the grid and ensure that the player and bug are always within the grid.
Where does it fit? Moving a sprite within a 2D grid requires using input keys, updating the game board, and using conditional statements to ensure all positions stay within the grid, and checking if the character stepped on a bug.

Tic Tac Toe. An old assignment with a new look: students create a 3 x 3 board where they take turns placing their symbol and trying to get three in a row, column, or diagonal. Students can play against friends or develop a human vs computer experience. The board displays user chosen symbols for the players, and arrow keys are used to select a move.
Where does it fit? Tic-Tac-Toe is an excellent assignment for learning conditional statements, loops, and data management. Students can compare strings or values in an array to update the board. The game grid object of BRIDGES also provides options for students who have not seen arrays yet.

Analyzing Repetition in Song Lyrics. The student selects a song from an external data source (supported by BRIDGES) and parses each word. A 2D matrix is created using the words of the song that records whether the i-th word of a song is the same as the j-th word and creates unique patterns on a grid depending on the words repetitions of the selected song. See a highly repetitive song in Fig. 2c and one with little repetition in Fig. 2b.
Where does it fit? Students will have gained experience parsing and comparing strings using tokens. They will also have an improved understanding of loops, 2D array processing, conditional statements.



(a) "Radio Gaga" by Queen (b) "Guns and Ships" by L.-M. Miranda (c) "Satisfaction" by The Rolling Stones

Figure 2: Song Lyrics: Repetition Pattern for different songs

Image Processing. The student reads images given in some text format (e.g., PPM RGB images) into a 2D array and performs simple image processing operations such as image flipping, color flipping, removing a color primary. Students learn to work with images and how to manipulate colors.
Where does it fit? This is an excellent assignment for 2D array processing, understanding image structures: storage of the array data in a linear sequence, relationship between 1D and 2D addresses, and exercising loops and conditions.

Mountain Paths. This is an adaptation of a Nifty assignment [14] but it uses BRIDGES's visualization capabilities for displaying outputs. Students are provided with an elevation image of a mountain as a text file. Starting from a random pixel on the left edge of the image, the goal is to find a path that takes the least amount of effort to get to the right edge of the image. A simple greedy algorithm is used to make a local decision to move from one pixel to the next, such that the pixel with the least difference in elevation between the current position and the next position is chosen. The selected set of pixels are drawn in color to illustrate the path, as can be seen in Fig. 3.
Where does it fit? This assignment is a very nice introduction to greedy

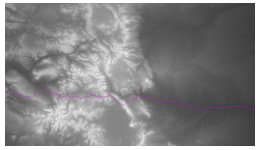


Figure 3: Path of Least Elevation Example

algorithms using a simple and highly engaging real world dataset. Students learn file I/O, implement a greedy algorithm, practice using conditionals, loops and 2D array processing.

5 Student Feedback

The projects detailed in Section 4 have been deployed in sophomore and junior level CS courses. These projects have been used often to introduce a new programming language (C++), or more frequently, as early 'warm-up' projects early in the course, or to illustrate a specific concept, such as object design. Table 2 illustrates the deployment of several of these projects over the past 3 years. We conducted project surveys and student reflections on the assignment that included the following questions:

1. What are the essential concepts that were learned by completing the assignment? [Short Answer]
2. Why is this important? [Short Answer]
3. How does the concept contribute to understanding how to program? [Short Answer]
4. The assignment was relevant to my career goals [5 point Likert scale].
5. The assignment increased my interest in computing [5 point Likert scale]

The reflection survey asked students the following questions:

1. Rate the difficulty of the module (5 point scale)
2. Roughly what percent of the module did you complete [25, 50, 75, 100]
3. Did you find the tasks engaging and meaningful? (4 point scale)
4. What did you like and not like about the assignment? [Short Answer]

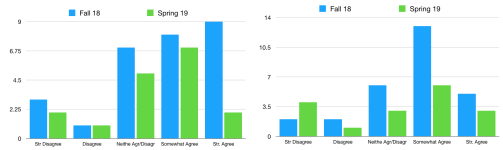
The class enrollment varied, ranging from 30-50 students across the different semesters. Participation was high (a small amount of credit was assigned for completing the surveys); however, only students who completed a substantial portion of the project were counted towards the survey participation.

Assignment	Semester/Year	Data Collection
Etch A Smile	F18,Spr.19	Student Reflection
Tic Tac Toe	F17,F18	Surveys, Student Reflection
Snakes & Ladders	Spr. 19	Student Reflection
Mountain Paths	F17,F18,Spr.19	Surveys, Student Reflection
Image Proc	F17,F18,Spr.19	Surveys, Student Reflection

Table 2: Student Feedback - Data Collection

Etch a Smile. Nearly all students found the project easy and completed it. The themes that emerged from the student reflections included the freedom to create a shape of their own choosing and play around with the color grid in creating a face. Examples of student quotes included '...allows creativity in a new way!', 'experiment and mess around with the coding', 'enjoyed the module and ... eager to learn more', 'enjoyed the assignment, it was engaging'.

Tic Tac Toe. This familiar game was also well received with over 80% completing the assignment and found it engaging. Student reflections were overall very positive, appreciated the moderate challenge, though several found it difficult as they were new to C++. Example free form responses included 'enjoyed making tic tac toe ... later plan to modify it to run on an arduino', 'game is a good way to demonstrate understanding of programming', 'liked the assignment, enjoyed creating the game'.



(a) *The assignment increased my interest in Computing.* (b) *The assignment was relevant to my career goals.*

Figure 4: Project Surveys (Fall 18 and Spring 19): Mountain Paths Assignment

Mountain Paths. This was a harder assignment, with 65-80% of the class completing 75% or more of the assignment. Almost all found the assignment engaging. The major themes from the reflection surveys were on the challenge of the assignment ('very challenging and I learned quite a bit', 'feel challenged, but also feel satisfied', 'was not prepared'), appreciation of the flexibility in the assignment ('loved the assignment allowed for different inputs', 'make a color object and amend it on each loop - it worked'), choice of the assignment

('excellent practical example of greedy algorithm') and the visualizations ('liked seeing the visualizations', 'enjoyable to finally see the best path line').

Fig. 4 shows the results of a survey of student's attitude towards computing. The results from two semesters show that the student were engaged by an assignment they perceived as relevant.

Image Processing This assignment continued from the Mountain Paths assignment. Students overall found the assignment challenging (roughly half in Fall 2018, 70% in Spring 19 found it difficult), and about 60-70% completed 75% or more of the assignment. Over 90% found the assignment engaging. Students appreciated the similarity of this assignment to the previous assignment ('last assignment was necessary to prepare me'), appreciation for the assignment ('good assignment with fairly clear instructions'), enjoyment ('liked the image processing portion', 'really liked seeing how easily implement some simple image processing with this assignment', 'liked manipulation of the image'), assessing success/failure ('my procrastination bit me...')

6 Conclusions

We have presented a set of highly engaging assignments that meet the principal goals of a typical introductory programming course like CS1. The assignments contain important elements of engagement, such as the use of real-world data, visualizations to see the final outputs, and interactive games and applications. We deployed and tested these assignments and collected student feedback. Overall, the student responses have been very positive: They find the assignments interesting, fun, and yet challenging.

Although these assignments have not yet been deployed in CS1, the topics and the assignments are at the level of a CS1. We have begun working with CS1/CS2 instructors using BRIDGES as part of their course. It would be interesting to do a comparison of the student responses with those deployed in a dedicated CS1 course using these engagement principles.

Acknowledgment

This material is based upon work supported by the National Science Foundation under grant no. DUE-1726809 and CCF-1652442.

References

[1] Jessica D Bayliss and Sean Strout. Games as a flavor of CS1. In *ACM SIGCSE Bulletin*, volume 38, pages 500–504, 2006.

[2] Theresa Beaubouef and John Mason. Why the high attrition rate for computer science students: some thoughts and observations. *ACM SIGCSE Bulletin*, 37(2):103–106, 2005.

[3] Sarah Buchanan, Brandon Ochs, and Joseph J LaViola Jr. CSTutor: a pen-based tutor for data structure visualization. In *Proc. ACM SIGCSE*, pages 565–570, 2012.

[4] David Burlinson, Mihai Mehedint, Chris Grafer, Kalpathi Subramanian, Jamie Payton, Paula Goolkasian, Michael Youngblood, and Robert Kosara. Bridges: A system to enable creation of engaging data structures assignments with real-world data and visualizations. In *Proc. ACM SIGCSE*, pages 18–23, 2016.

[5] Sarah Dahlby Albright, Titus H Klinge, and Samuel A Rebelsky. A functional approach to data science in CS1. In *Proc. ACM SIGCSE*, pages 1035–1040, 2018.

[6] Amy Delman, Adiba Ishak, Lawrence Goetz, Mikhail Kunin, Yedidyah Langsam, and Theodore Raphan. Development of a system for teaching CS1 in C/C++ with lego NXT robots. In *FECS*, pages 396–400, 2010.

[7] Prasun Dewan. How a language-based GUI generator can influence the teaching of object-oriented programming. In *Proc. ACM SIGCSE*, pages 69–74, 2012.

[8] Ira Greenberg, Deepak Kumar, and Dianna Xu. Creative coding and visual portfolios for CS1. In *Proc. ACM SIGCSE*, pages 247–252, 2012.

[9] Mark Guzdial. A media computation course for non-majors. In *Proc. ITICSE*, pages 104–108, 2003.

[10] Diane Horton, Michelle Craig, Jennifer Campbell, Paul Gries, and Daniel Zingaro. Comparing outcomes in inverted and traditional CS1. In *Proc. ITICSE*, pages 261–266, 2014.

[11] Joint Taskforce on ACM Curricula. *Computer Science Curricula 2013: Curriculum Guidelines for Undergraduate Degree Programs in Computer Science*. ACM/IEEE Computer Society, 2013.

[12] Lucas Layman, Laurie Williams, and Kelli Slaten. Note to self: Make assignments meaningful. In *Proc. ACM SIGCSE*, SIGCSE '07, pages 459–463, 2007.

[13] Alvaro Monge, Beth A. Quinn, and Cameron L. Fadjo. EngageCSEdu: CS1 and CS2 materials for engaging and retaining undergraduate CS students. In *Proc. ACM SIGCSE*, pages 271–271, 2015.

[14] Nick Parlante. Nifty assignments, 2018.

[15] Johanna Pirker, Maria Riffnaller-Schiefer, and Christian Gütl. Motivational active learning: Engaging university students in computer science education. In *Proc. ITICSE*, pages 297–302, 2014.

[16] Kelvin Sung, Rebecca Rosenberg, Michael Panitz, and Ruth Anderson. Assessing game-themed programming assignments for CS1/2 courses. In *Proc. of GDCSE*, GDCSE '08, pages 51–55, 2008.

Real-World Assignments at Scale to Reinforce the Importance of Algorithms and Complexity*

Jason Strahler[1], Matthew Mcquaigue[1], Alec Goncharow[1]
David Burlinson[1], Kalpathi Subramanian[1]
Erik Saule[1], Jamie Payton[2]
[1]Computer Science, UNC Charlotte
{jstrahl1, mmcquaig, agoncha1, dburlins, krs, esaule}@uncc.edu
[2]Computer and Information Sciences, Temple University
payton@temple.edu

Abstract

Computer Science students in algorithm courses often drop out and feel that what they are learning is disconnected from real life programming. Instructors, on the other hand, feel that algorithmic content is foundational for the long term development of students. The disconnect seems to stem from students not perceiving the importance of algorithmic paradigms, and how they impact performance in applications.

We present the point of view that by solving real-world problems where algorithmic paradigms and complexity matter, students will become more engaged with the course and appreciate its importance. Our approach relies on a lean educational framework that provides simplified access to real life datasets and benchmarking features. The assignments we present are all scaffolded, and easily integrated into most algorithms courses. Feedback from using some of the assignments in various courses is presented to argue for the validity of the approach.

1 Introduction

Algorithms courses are generally a cornerstone of computer science bachelor's degree programs. They introduce concepts critical to CS students' theoretical foundations, and are a significant part of the core curriculum recommended by ACM in their 2013 CS guidelines [9]. Meanwhile, students often see these classes as being overly theoretical; they express frustration that calculating complexity, proving correctness, and studying obscure paradigms are the focus of the entire class. While some mathematically-minded students will appreciate the content, most perceive the class as unimportant. As such, the failure rate in algorithms classes is often high, and they are known in many universities for being the weed-out class.

Runtime efficiency of algorithms is a hard topic to teach. Gal-Ezer and Zur showed that students have many misconceptions about runtime efficiency, and wrote "Concepts like big-O..., are known to be difficult to conceive and difficult to teach and learn" [6]. Misconceptions in algorithms courses are common and have received some academic attention [4]. While some efforts were made to design tools to help students understand algorithms [11, 7], little effort has focused on getting students engaged in the topic.

We present extensions to the BRIDGES framework which are designed to help engage students with the content in algorithms courses. In particular, BRIDGES provides access to real-world data which can be processed by algorithms to solve real-life problems. Algorithms can be visualized in multiple ways to improve students' understanding of how a particular algorithm works. Moreover, our framework provides larger scale instances which enable performing run-time benchmarking of algorithms. These different features enable students to develop a keener understanding of why algorithmic content is relevant to them and their career. The BRIDGES framework and scaffolded assignments are available online (`http://bridgesuncc.github.io/`).

2 Related Works

What Makes Students Engaged. Strategies for engagement seem to come from two complementary approaches. **Teaching style engagement strategies** focuses on how a course is taught and managed. Active learning techniques have become popular in recent years to promote student engagement and can include any combination of lab-based instruction, flipped classrooms, gamification, peer-learning, and use of multimedia content [8]. **Content based engagement strategies** present the topics of the course using learning materials that capture the interest of the students. This is a common thread in the popular assignment repositories such as Nifty Assignments [14], EngageCSEdu [12],

and game-themed assignments [3]. Real-world and large datasets in course projects have been demonstrated to successfully engage students [1], in contrast to tiny contrived examples.

What Algorithms Courses Typically Look Like. Looking at topics and learning outcomes in curriculum guidelines [9], algorithms textbooks [2], and concept inventories of algorithms courses [4] paints a picture of a typical algorithms course. Algorithms courses typically start with a discussion of runtime estimations, using complexity notations like Big Oh and Big Theta, followed by methods for proving algorithm correctness, and computing runtime complexity. Courses may look at advanced data structures such as trees, graphs, or hash tables to define problems, or as a way to solve problems in the class. Algorithm design techniques such as brute force, divide and conquer, greedy algorithms and dynamic programming are introduced to solve a variety of problems. Advanced courses may contain discussions of calculability, NP-Completeness and methods such as Branch and Bound methods, and approximation algorithms.

Existing Educational Efforts in Algorithms Courses have been aimed at visualizing what a data structure or algorithm looks like [13, 10]. These efforts have focused on creating a visually interactive way to show and teach algorithms to students, and was shown to be effective at learning algorithmic concepts [5]. There have also been efforts to bring real world map data into algorithm courses [15], for use in sequential search, graph traversal, Dijkstra's algorithm, and convex hulls. This effort shows an interest in bringing real world data into algorithm courses instead of using synthetic or contrived data.

3 The BRIDGES system

The BRIDGES system enables assignments relevant to the goals of introductory CS courses, including algorithms courses. The system provides bindings for Java, C++, and Python based on a commonly used object hierarchy. Output from assignments are highly visual and can easily be shared with friends and family.

Scalable Dataset Access. A simple API provides access to external data sources such as USGS Earthquake data, Wiki Data, IMDB actor/movies, Genius' Song Lyrics, and OpenStreet Maps. Usually a single function call returns a set of easy to understand objects. Assignments that leverage these interesting datasets seem more real and relevant. When possible, these data are accessed live. BRIDGES plays the intermediary, dealing with credential issues, access policies, etc. Because the data is live and real, the datasets can be as large as

Assignment	Topics	Engagement
Plotting Complexity	Order of Growth	Visual Output, use own machine spec
Sorting Implementation	Order of Growth, Divide/Conquer, Decrease/Conquer, Heaps	Visual Output, Real Data
Book Distance	Order of Growth, Trees, Hash Maps	Visual Output, Real Data Analysis, Interest:literature
Mountain Path	Order of Growth, Optimization, Dynamic Programming, Shortest Path, Greedy Algorithms, Problem Modeling	Visual Output, Real Data, Real Problem, Interest:hiking
Routing in a City	Order of Growth, Shortest Path	Visual Output, Real Data, Real Problem, Choose own city, Interest:social
Hollywood Analysis	Order of Growth, Graphs, BFS, Graph Construction	Visual Output, Real Data Analysis, Interest:entertainment, Fun

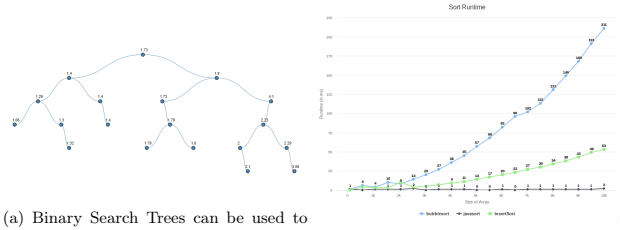
Table 1: A set of assignments using real data, real problems to teach algorithms.

they need to be. One could access a map of every street in the United States from Open Street Map or get information on every single movie ever released. To minimize network transfers and computational costs, the data is cached, both within the BRIDGES infrastructure and on the student's machine. The expectation is that having data at that scale will let students explore different problems, see practical applications as part of their studies, and realize that problems can be at large scale without appearing to be made up.

Performance and Benchmarking Features. Algorithm teaching tools typically lack elements to understand questions related to the runtime of algorithms. BRIDGES provides features to plot performance charts. Elaborate visualizations of complexity [16] have been designed for trained algorithm experts, but BRIDGES uses classic runtime plots where algorithms are associated with pairs $(size, time)$ displayed using a classic line chart (See Figure 1b for a sample output). Secondly, BRIDGES enables automatic benchmarking of particular problems. Take sorting of an array as an example. The student-implemented sorting function is passed to a benchmarking object that will run the algorithm at different sizes to extract performance information. When possible, the benchmarks are run using real-world data, to avoid students' feeling that the problems have been scaled up for their own sake.

4 A Set of Engaging and Scalable Algorithm Assignments

Next we present a set of assignments leveraging BRIDGES which are appropriate for an algorithms course. Table 1 presents concisely the assignments, the topics they map to, and their engagement characteristics. The assignments cover most topics in an algorithm class, often use real data, perform a real analysis, or solve a real life problem. The assignments are linked to areas of interest usually popular with students. Assignments scaffolded for C++, Java, and Python are accessible online (`http://bridgesuncc.github.io/newassignments.html`).



(a) Binary Search Trees can be used to sort

(b) Timing experiment

Figure 1: Sorting in BRIDGES

The Classic Plotting of Complexity consists in plotting the cost of a few algorithms given their precise instruction count for given machines and shows that an algorithm with higher complexity running on a much faster machine will still be slower than an algorithm with a better complexity running on a smaller machine. For instance, the runtime an algorithm using $10^4 n$ operations and an other one using $5*10^4 n$ operations on a machine at 1MHz would be much faster than a machine at 100MHz running an algorithm taking $10^2 n^2$ operations. Engagement comes from the visual output, which is easy to understand and from personalizing the assignment, by using a student's own machine.

The Classic Sorting Assignment consists in implementing classic sorting algorithms. Insertion sort and selection sort are quadratic decrease-and-conquer algorithms often covered when talking about arrays or correctness. While Merge sort and Quick sort are often used to show an $\Theta(n \log n)$ algorithm using a divide-and-conquer approach, sorting by leveraging a tree data structure such as a Heap or a Binary Search Tree can also be used.

BRIDGES can be used to understand how the sorting happens across different sorting algorithms by visualizing the current state of the algorithm. Figure 1a shows a Binary Search Tree that can be then in-order traversed to extract a sorted array. BRIDGES provides unit testing and benchmarking of the algorithms. Figure 1b shows the performance of insertion sort and bubble sort compared to Java's standard sorting function.

Engagement comes primarily from the visualization of the algorithm and of the runtimes. Real data can be used for what is being sorted: Figure 1a, for instance, shows sorting using the magnitude of recent Earthquakes. But without a more precise scaffold, it can appear artificial to students.

Computing Book Distance is inspired by Natural Language Processing. The idea is to compare how close books are using a bag-of-word model. For a particular book, one can compute how many times each word appears and normalize that count to form a vector representation of the book (maybe the word "dog" appears 1% of the time.) These frequency vectors can be leveraged to compare books, for instance, by using the L1 distance or a cosine similarity function. This analysis will highlight that books from a particular author are more similar than books from different authors.

The core of the assignment is the implementation of a Dictionary which can be done using different data structures: array, linked list, binary search tree or a hash map. The application in the assignment is counting words and computing distance between sparse vectors. BRIDGES provides access to data for this assignment through Project Gutenberg. Using small Shakespearean poems is very good for debugging. But computing the distance between larger books will drive home the importance of complexity. A comparison of all the works of Shakespeare against all the works of Mark Twain will take hours if one uses a $\Theta(n)$ implementation of Dictionary such as an unsorted array while it will take only seconds using an $O(1)$ hash maps.

Engagment. The Dictionary can be visualized and debugged using BRIDGES. The application is real: this type of analysis was conducted to identify who wrote the Federalist Papers. Students with interests in literature will appreciate this assignment and plug in their favorite classic authors.

Optimizing Path through a Mountain. The mountain path assignment first appeared as a Nifty assignment [14]. Given an elevation map (image) the task is to find the lowest cost path, from one end of the image to the other, where the cost is defined as the sum of difference in elevation between consecutive pixels in the path. A simple greedy approach is used to make local decisions and the selected path of pixels is drawn in color (shown in Figure 2).

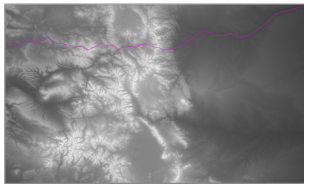


Figure 2: Path of Least Elevation (Greedy)

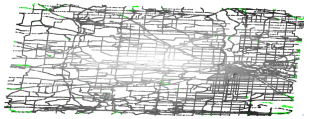


Figure 3: Map of Minneapolis, MN colored by distances to the center of the map.

This problem from Nifty is particularly good to teach algorithms for optimization problems. The algorithm is a greedy heuristic. However, one can easily build a variant where you always have to go right, but are allowed to choose whether to go up-right, straight-right, or down-right by looking at the entire map. This variant can be solved using dynamic programming or a brute force method. With a large map, the difference in computation time between the greedy heuristic, and the optimal algorithm will be obvious and enable discussions of time-quality tradeoff. A second variant is to go from a pixel to any of the 8 neighbors and the problem becomes a classic shortest path problem.
Engagement. The output is visual, and addresses a real problem using real data. It will echo well in students who love hiking. History-buffs may be interested in figuring out where Hannibal should have crossed the Alps, or whether Xerxes the Great had to fight the Greek army at Thermopylae. Finally, BRIDGES lets students choose the elevation map they will use in the assignment, for instance, their own campus, city, or a favorite hiking area.

Scalable Routing through a City. Implementing Dijkstra's algorithm with best case complexity is difficult because it relies on a Fibonacci Heap [2]. Often implementations seen in algorithms courses use a regular Binary Heap which has a higher complexity. This is not a major problem when the map is small as the difference is hard to notice. BRIDGES provides access to Open Street Map data and enables students to access maps of the continental US for any latitude/longitude bounding box at different resolutions (the graph of Minneapolis is shown in Figure 3). This enables BRIDGES to benchmark automatically Dijkstra's implementation.
Engagement. Students get to choose the map they will use, can relate to the need for routing in a city, and get the feeling they are solving a real problem.

Analysis of Hollywood Movies. Computing the Bacon Number of an actor consists in figuring out how many movies away an actor is from Kevin Bacon by only going from actor to one of his/her movie and from a movie to one of the actors that played in it. Listing the actors and movies in the chain is a game known as the "Six Degrees of Kevin Bacon". Provided a graph composed of movies and actor with edges if the actor played in the movie, computing the Bacon number of an actor is achieved with a BFS traversal.

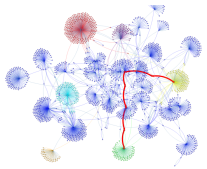


Figure 4: Bacon Number

BRIDGES enables accessing a toy actor-movie graph from IMDB with about 2000 edges. This graph can be styled to highlight the shortest path between an actor and Kevin Bacon (see Figure 4).

It is also possible to access all the data of all English movies since the 1910s. The students will build graphs of about 1M edges and BRIDGES enables to easily query particular time intervals. This makes the dataset very versatile, permitting students to study who was the most central actor for a particular decade; this calculation requires computing a BFS from each actor in the dataset. While a single BFS computation takes about 0.5 sec. on a 1M edge graph, calculating BFS from all vertices takes on the order of a day, thus emphasizing that calculation time compounds quickly at real world scales.
Engagement. The analysis that students perform is real and a big data problem, and a dataset students are often interested in is explored visually.

5 Results: Student Reactions

We deployed several of the above projects in courses on algorithms, data structures and object oriented systems. We obtained student feedback on the projects through reflection and project surveys, performed after each project. The reflection survey gives students the chance to describe their learning experiences and specifically what they liked and did not liked about the assignment. Other questions focused on engagement, completion time, preparedness, assignment difficulty, and if the assignment increased their interest in computing.

Book Distance Project. The project was assigned in a data structures class in Fall 15 as a four-part project. Student started by writing the book distance comparison using a provided Dictionary implementation using unsorted arrays, then they implemented a Dictionary object using sorted arrays, binary search trees, and hash maps. They computed distance between books of different sizes. No formal reflection or engagement quiz was given. The following comments come from the notes of the instructor (an author of this paper).

Students initially felt that the implementing the application was difficult, but eventually appreciated seeing distances between books. Computing the distances using sorted arrays was very slow and students computed only some of the distances. Using the BST, students were able to compute all the pairwise comparisons. Many students were surprised at the speed of hash maps, and commented on how they "killed their laptop" on computation that was unnecessary with better data structures.

Mountain Path. We only considered the greedy algorithm. BRIDGES was used to display the input elevation image and the final path. The project was given in Fall 18, Spring 19 and Spring 20 in an object oriented programming course. Students found the project highly engaging (96%, 100%,85%) and

difficult (85%, 74%, 78%). Short answer responses also indicated difficulties with programming concepts, clarity of instructions, and also satisfaction with the assignment challenges. Students really seemed to enjoy the project, with remarks such as 'excellent practical example of the greedy algorithm', 'very challenging and learned quite a bit', 'at first ... intimidated, ... after thoroughly reading... I felt a bit more confident', 'liked the assignment challenged my programming ability'

Bacon Number. The project was given to an algorithms course in Spring 16 and Fall 17 to an algorithms course. Only the first part of the project was given to the students, to implement the Bacon Number project on the 2000 node graph. Students found the project to be difficult (57% vs. 30%, 45% vs. 9%) but increased their interest in computing (51% vs 30%), and felt it was relevant to their career goals (48% vs. 16%, 54% vs. 18%)

Students were excited about the visual output "This is the first time in ALL of comp sci, that we could actually visually see the data structures", interest in lowering runtime: "an individual [may be] required to work with a large data structure and ... to effectively traverse [it] in a reasonable computation time", liked working with graphs, "graphs are more fun/interesting" and found them relevant, "Bfs is an extremely useful and popular algorithm and graphs are used frequently in computer science and tech industry".

6 Conclusion

This paper presents strategies to engage students with the content of typical algorithms courses. The main strategy is to assign course projects that are or look like real-life problems and rooted in domains students care about and use visualization. The data used in the projects are extracted from live sources, grounding the projects in reality. We presented 6 assignments/projects that cover most of the content of a typical algorithms course. Three of the projects were assigned to students in various courses. The projects were deemed hard but were eventually perceived positively by the students. A threat to the validity of this work is that the projects were not all in a single algorithms course, and one probably one would not assign all of them in such a course. We will address that in the future by performing such an intervention and conducting formal studies.

Acknowledgment

This material is based upon work supported by the National Science Foundation under grant no. DUE-1726809 and CCF-1652442.

References

[1] David Burlinson, Mihai Mehedint, Chris Grafer, Kalpathi Subramanian, Jamie Payton, Paula Goolkasian, Michael Youngblood, and Robert Kosara. BRIDGES: A system to enable creation of engaging data structures assignments with real-world data and visualizations. In *Proc. ACM SIGCSE 2016*, pages 18–23, 2016.

[2] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT Press, third edition edition, 2009.

[3] Peter Drake and Kelvin Sung. Teaching introductory programming with popular board games. In *Proc. of ACM SIGCSE*, pages 619–624, 2011.

[4] Mohammed F. Farghally, Kyu Han Koh, Jeremy V. Ernst, and Clifford A. Shaffer. Towards a concept inventory for algorithm analysis topics. In *Proc. SIGCSE*, pages 207–212, 2017.

[5] Mohammed F. Farghally, Kyu Han Koh, Hossameldin Shahin, and Clifford A. Shaffer. Evaluating the effectiveness of algorithm analysis visualizations. In *Proc. SIGCSE*, pages 201–206, 2017.

[6] Judith Gal-Ezer and Ela Zur. The efficiency of algorithms—misconceptions. *Computers and Education*, 42(3):215 – 226, 2004.

[7] Michael T. Goodrich and Roberto Tamassia. Teaching the analysis of algorithms with visual proofs. *SIGCSE Bull.*, 30(1):207–211, March 1998.

[8] Mark Guzdial. A media computation course for non-majors. In *Proceedings of the ITICSE 2003*, pages 104–108, 2003.

[9] Joint Taskforce on ACM Curricula. *Computer Science Curricula 2013: Curriculum Guidelines for Undergraduate Degree Programs in Computer Science*. ACM/IEEE Computer Society, 2013.

[10] Jeff Lucas, Thomas L. Naps, and Guido Rößling. Visualgraph: A graph class designed for both undergraduate students and educators. *SIGCSE Bull.*, 35(1):167–171, January 2003.

[11] Joan M. Lucas. Illustrating the interaction of algorithms and data structures using the matching problem. In *Proc. SIGCSE*, pages 247–252, 2015.

[12] Alvaro Monge, Beth A. Quinn, and Cameron L. Fadjo. EngageCSEdu: CS1 and CS2 materials for engaging and retaining undergraduate CS students. In *Proc. of ACM SIGCSE*, pages 271–271, 2015.

[13] Thomas L. Naps, James R. Eagan, and Laura L. Norton. JHAVÉ - an environment to actively engage students in web-based algorithm visualizations. In *Proc. SIGCSE*, pages 109–113, 2000.

[14] Nick Parlante. Nifty assignments, 2018.

[15] James D. Teresco, Razieh Fathi, Lukasz Ziarek, MariaRose Bamundo, Arjol Pengu, and Clarice F. Tarbay. Map-based algorithm visualization with METAL highway data. In *Proc. SIGCSE*, pages 550–555, 2018.

[16] Jeyarajan Thiyagalingam, Simon Walton, Brian Duffy, Anne Trefethen, and Min Chen. Complexity plots. In *Proc. EuroVis*, pages 111–120, 2013.