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

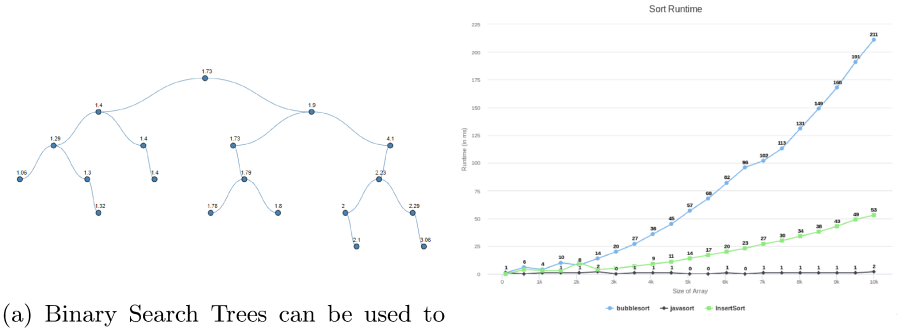
Table 1: A set of assignments using real data, real problems to teach algorithms.

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).

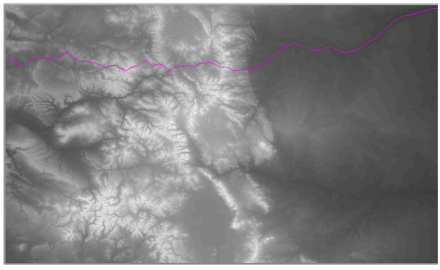


Figure 2: Path of Least Elevation (Greedy)

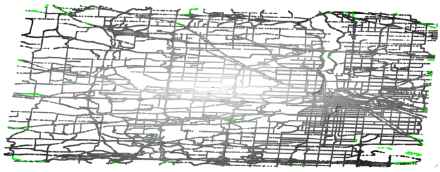


Figure 3: Map of Minneapolis, MN colored by distances to the center of the map.

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.

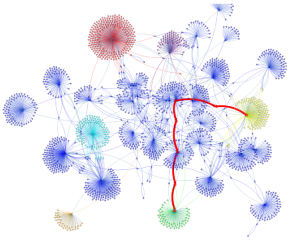


Figure 4: Bacon Number

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.