Teaching and Learning Graph Algorithms Using Animation

Y. Daniel Liang
Department of Computer Science
Georgia Southern University
Savannah Campus, GA 31419
y.daniel.liang@gmail.com

ABSTRACT
Graph algorithms have many applications. Many real-world problems can be solved using graph algorithms. Graph algorithms are commonly taught in the data structures, algorithms, and discrete mathematics courses. We have created two animations to visually demonstrate the graph algorithms. The first animation is for depth-first search, breadth-first search, shortest paths, connected components, finding bipartite sets, and Hamiltonian path/cycle on unweighted graphs. The second animation is for the minimum spanning trees, shortest paths, travelling salesman problems on weighted graphs. The animations are developed using HTML, CSS, and JavaScript and are platform independent. They can be viewed from a browser on any device. The animations are useful tools for teaching and learning graph algorithms. This paper presents these animations.

Keywords
Algorithms, animation, data structures, discrete mathematics, graphs

1. INTRODUCTION
Graphs are simple mathematical structures. A graph consists of a set of vertices and a set of edges for connecting vertices. In a weighted graph, each edge is assigned with a value, called a weight. Graphs have many important applications. For example, a map can be modelled using a graph. The cities are the vertices and the roads connecting the cities are the edges, and the distances are the weights on the edges. The problem of finding the shortest distance between two cities can be solved by finding a shortest path between the two vertices in the graph. Many algorithms have been developed to solve a variety of graph problems. The common graph problems for unweighted graphs covered in the data structures, algorithms and discrete mathematics courses are depth-first search, breadth-first search, shortest paths, connected components, finding bipartite sets, and Hamiltonian path/cycle. For weighted graphs, the common problems are the minimum spanning trees, shortest paths, travelling salesman problems. We have created animations for helping instructors and students to teach and learn these algorithms.

The animations are freely accessible from https://yongdanielliang.github.io/animation/animation.html. The animations enable instructors and students to create graphs dynamically, apply the graph algorithms on graphs, and immediately see the results. The animations are effective tools for teaching and learning graph algorithms. The animations have been integrated in the Pearson's interactive REVEL™ ebooks [5, 6, 7], which have received positive reviews [9, 10]. This paper presents the graph algorithm animations for unweighted graphs and for weighted graphs, respectively.

2. SURVEYS OF GRAPH ALOGRITHM ANIMATIONS
Several graph algorithm animations are available on the Web. The most popular graph animations are accessible from http://jhave.org/ [8], https://visualgo.net/en [12] and https://www.cs.usfca.edu/~galles/visualization/Algorithms.html [3]. The first tool [8] is written in Java. Due to security restrictions on Java running on the browser, this tool cannot run from a Web browser. Neither of the tools allows you to create graphs interactively. Because the graphs are pre-created in these tools, the instructor cannot create their own graphs for class demonstration. Our graph algorithm animation tools enable instructors and students to create custom graphs dynamically and easily. Additionally, our tools combine all algorithms for unweighted graphs in one unified animation and all algorithms for weighted graphs in the other unified animation. As a result, it is simple and easy to use our graph algorithm animation tools.

3. ALGORITHM ANIMATION FOR UNWEIGHTED GRAPHS
The unweighted graph algorithm animation tool can be accessed for free from https://yongdanielliang.github.io/animation/web/GraphLearningTool.html, as shown in Figure 3.1. The process of creating a graph is simple. You can add a vertex by clicking the primary button in an open area. You can remove a vertex by clicking the vertex using the secondary button. You can add an edge between two vertices by dragging from one vertex to the other. You can also move a vertex by dragging the vertex while pressing the CTRL button.

After creating a graph, you can apply an algorithm on the graph and see the result of applying the algorithm interactively. To display a

depth-first search tree or breadth-first search tree, specify a starting vertex and click the DFS or the BFS button to see the search tree. For example, as shown in Figure 3.2, a DFS tree is displayed starting from vertex 4. A BFS tree is displayed starting from vertex 4 in Figure 3.3.

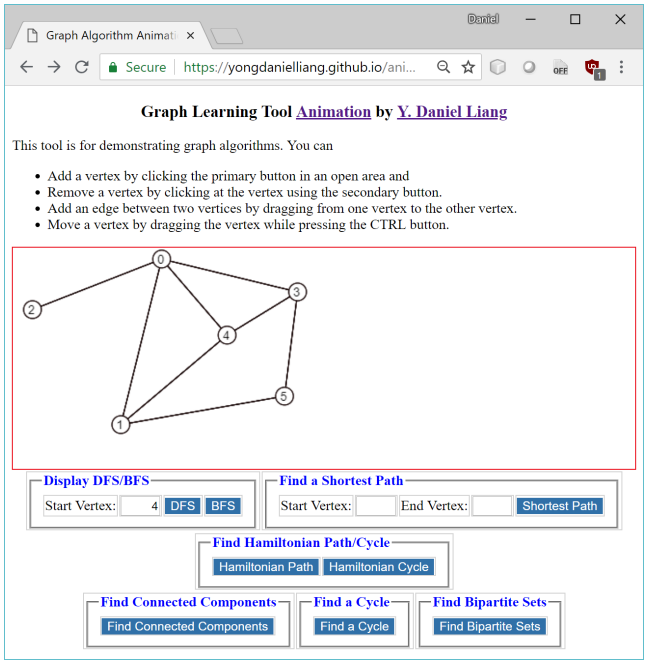

Figure 3.1: The graph algorithm animation for unweighted graphs.

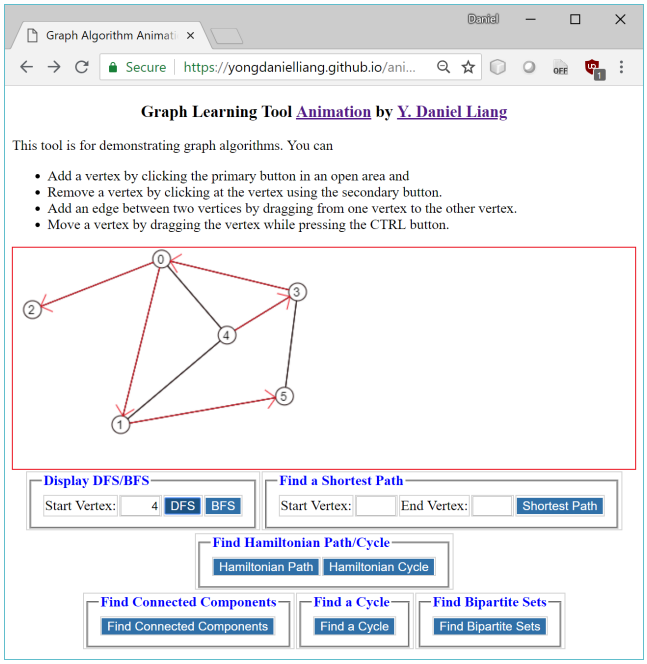

Figure 3.2: A DFS tree starting from vertex 4 is displayed.

The shortest path between two vertices in an unweighted graph can be obtained using the breadth-first search from a vertex. As shown in Figure 3.4, the user enters the starting vertex 2 and the ending vertex 5 and clicks the Shortest Path button to display a shortest path from 2 to 5.

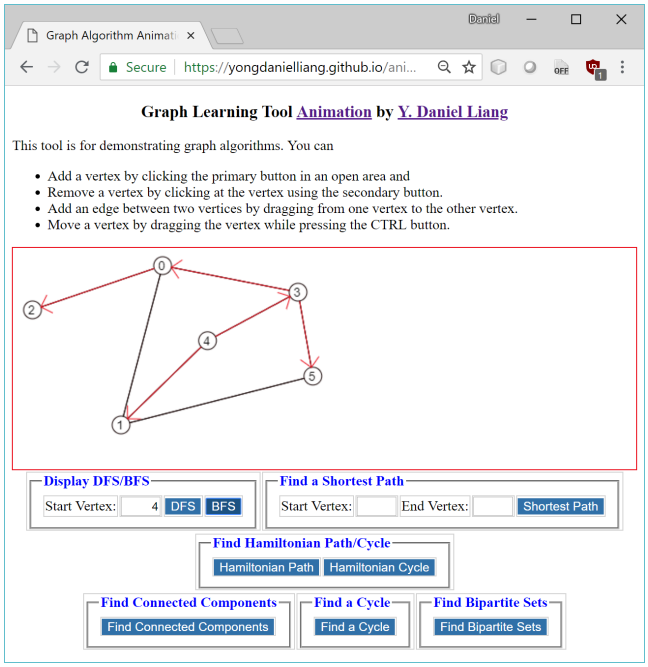

Figure 3.3: A BFS tree starting from vertex 4 is displayed.

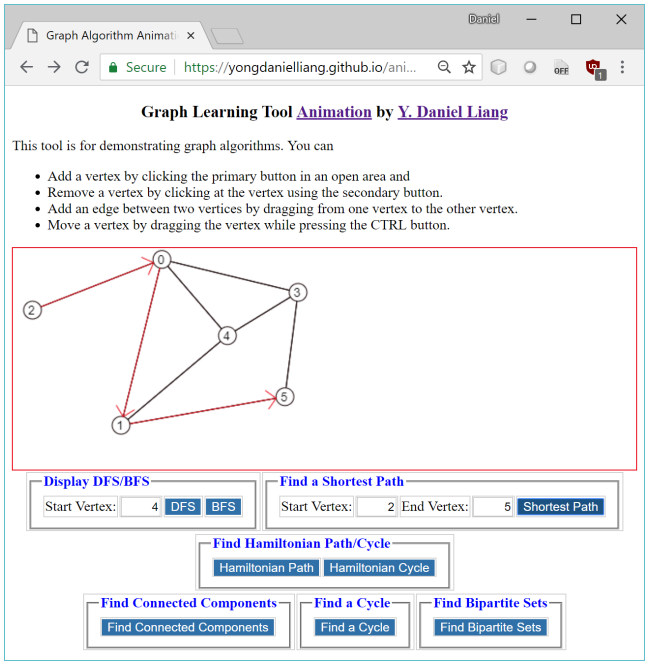

Figure 3.4: The shortest path from vertex 2 to vertex 5 is displayed.

The Hamiltonian path is a path that traverses all vertices in the graph exactly once. As shown in Figure 3.5, clicking the Hamiltonian Path button displays a Hamiltonian path. The Hamiltonian cycle is a Hamiltonian path in which the starting vertex and the ending vertex are connected. As shown in Figure 3.6, clicking the Hamiltonian Cycle button displays a Hamiltonian cycle.

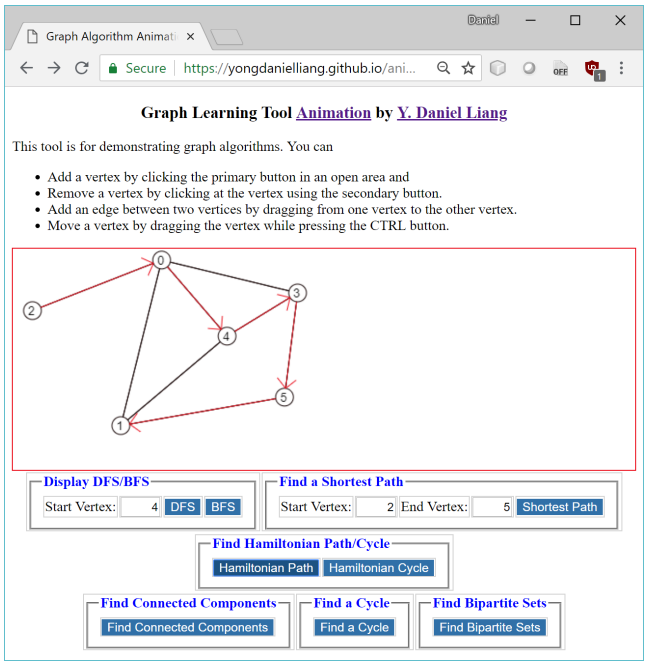

Figure 3.5: The animation displays a Hamiltonian path.

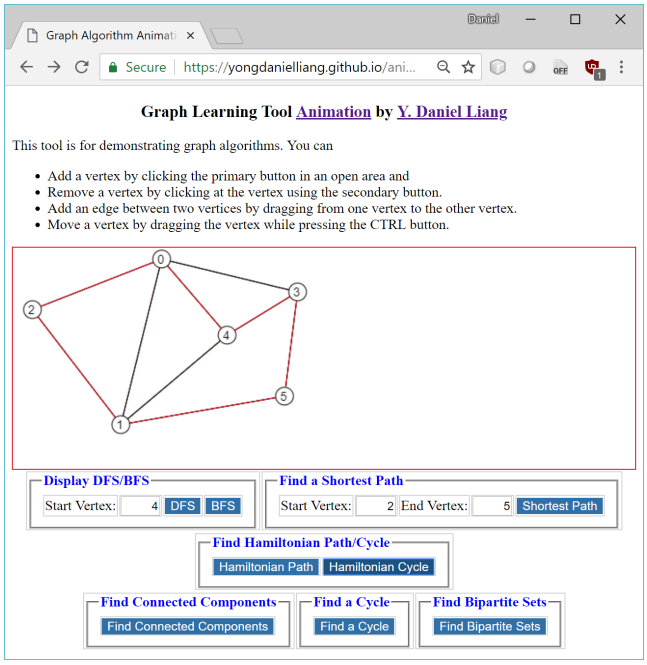

Figure 3.6: The animation displays a Hamiltonian cycle.

A connected component is a maximal connected subgraph in which every two vertices is connected by a path. You can find all connected components in the graph by clicking the Find Connected Components button as shown in Figure 3.7. Two connected components [0, 1, 4, 3, 2] and [5, 7, 6] are displayed for the graph in Figure 3.7.

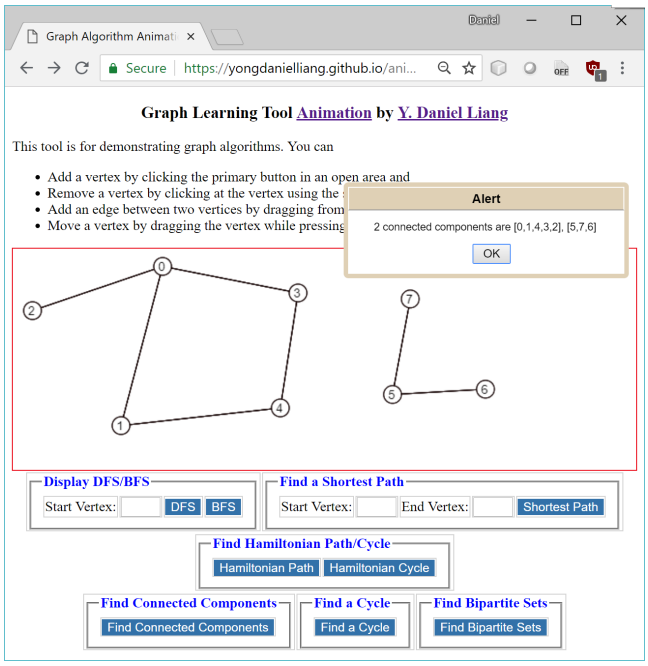

Figure 3.7: The connected components are displayed in the dialog box.

You can find a cycle in the graph by clicking the Find a Cycle button, as shown in Figure 3.8. A cycle 0, 1, 4, 3 is displayed in the dialog box.

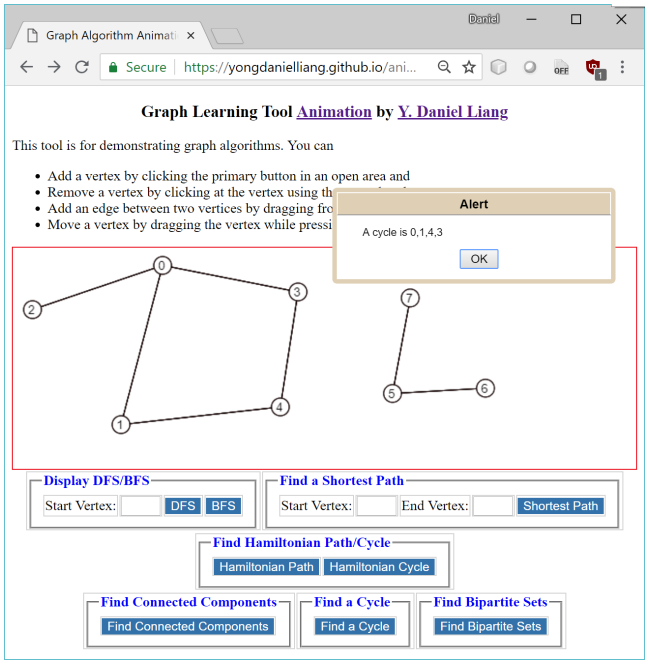

Figure 3.8: A cycle is displayed.

The bipartite sets are the two sets of vertices obtained from the graph such that no vertices in a set is connected. This type of the graph is called a bipartite graph. When you click the Find Bipartite Sets button for the graph in Figure 3.9, the animation displays that the graph is not bipartite, because no bipartite sets can be found for the graph.

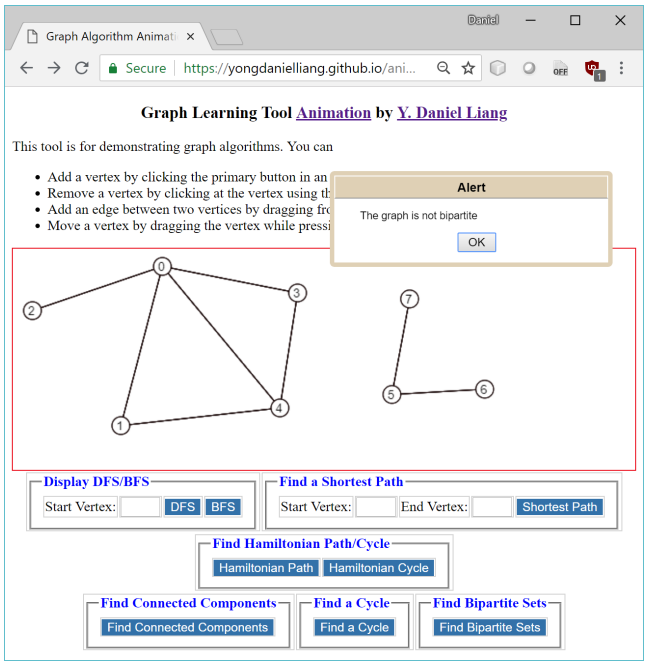

Figure 3.9: The graph is not bipartite.

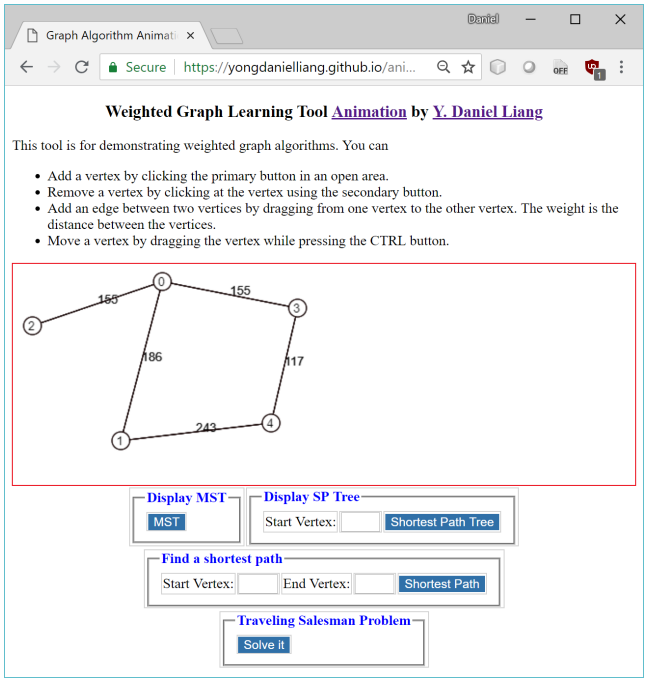

Figure 4.1: The graph algorithm animation for weighted graphs.

4. ALGORITHM ANIMATION FOR WEIGHTED GRAPHS

The weighted graph algorithm animation can be accessed from https://yongdanielliang.github.io/animation/web/WeightedGraphLearningTool.html, as shown in Figure 4.1. The process of creating a weighted graph is similar to creating an unweighted graph. You can add/remove a vertex in the same way as in the unweighted graph algorithm animation. You can add an edge by dragging from one vertex to the other. The weight of the edge is the distance between the two vertices.

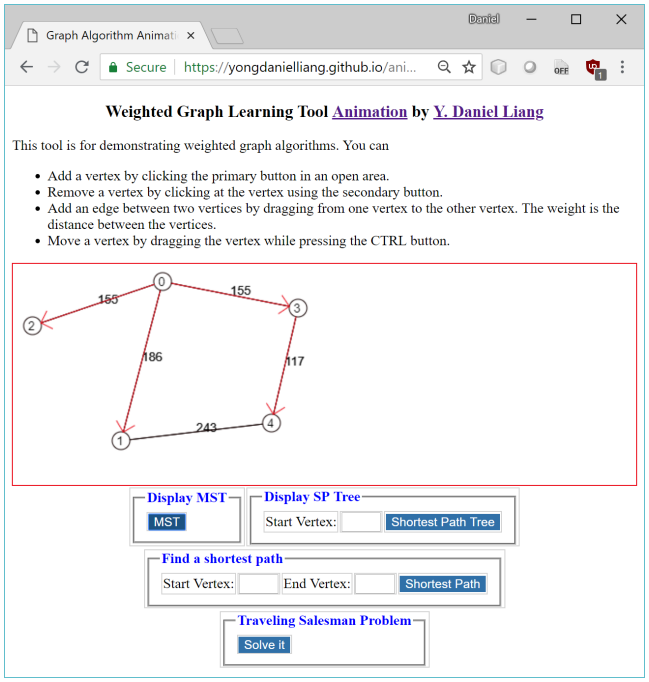

Figure 4.2: A minimum spanning tree is displayed.

A spanning tree of a graph is a connected subgraph that contains all the vertices in the graph and the subgraph is a tree. A minimum spanning tree of a graph is a spanning tree with the minimum total weights. You can obtain the minimum spanning tree by clicking the MST button, as shown in Figure 4.2.

A shortest path tree can be found using the Dijkstra's algorithm. The tree represents a single source all shortest paths. For example, Figure 4.3 shows the shortest path tree starting from vertex 1.

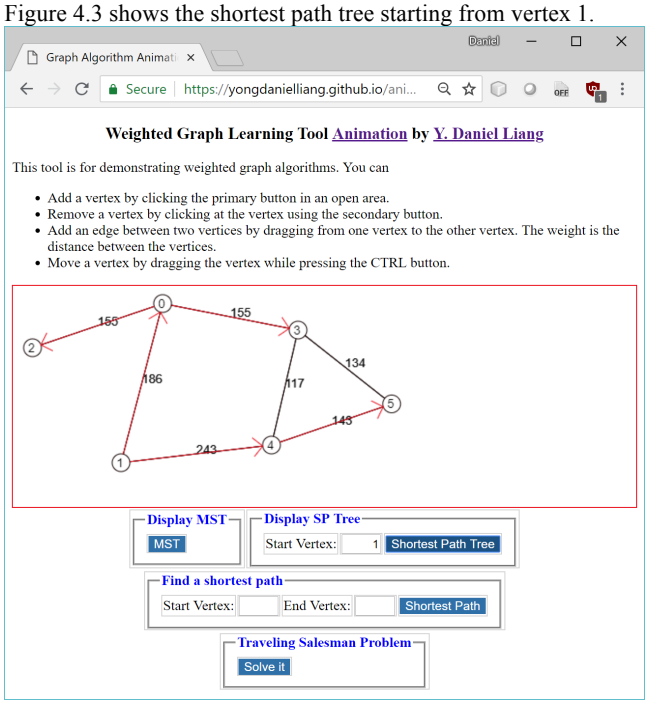

Figure 4.3: A shortest path tree starts from vertex 1.

A shortest path between two vertices can be found after a shortest path tree is constructed. To find a shortest path from vertex u to v, first create a shortest path tree starting from vertex u. A path from u to v in the tree is the shortest path from u to v. For example, Figure 4.4 shows the shortest path from vertex 1 to vertex 5.

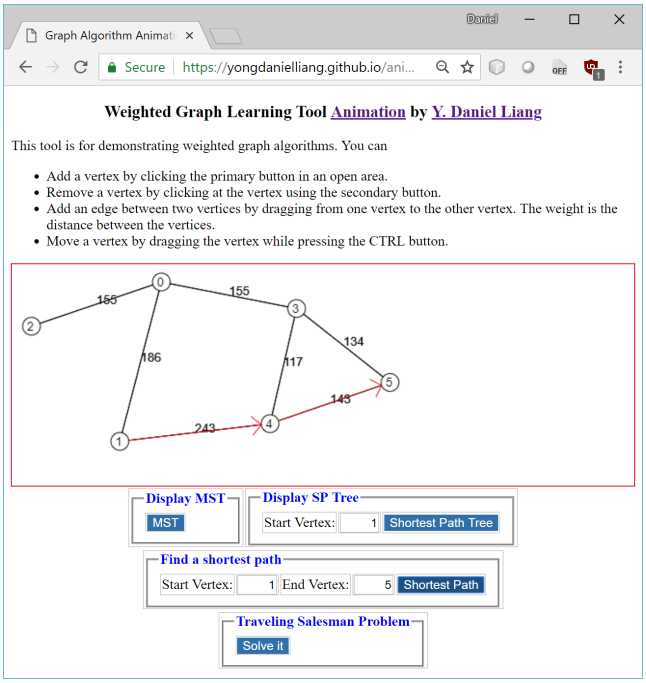

Figure 4.4: A shortest path from vertex 1 to vertex 5 is displayed.

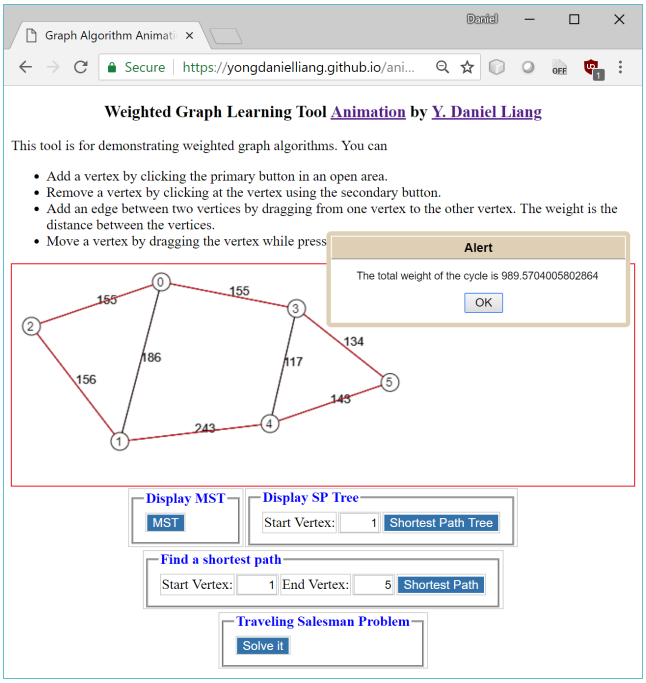

Figure 4.5: A solution for the travelling salesman problem is found with the total weights displayed in the dialog box.

The travelling salesman problem is to find a shortest path that starts from a vertex and visits each vertex exactly once and returns back to the original vertex. Figure 4.5 shows a solution to the problem for the graph in the figure.

5. BENEFITS OF USING GRAPH ALGORITHM ANIMATION

Here are the major benefits for instructors and students to use our tool.

Benefit 1: In a typical lecture for the graph algorithms, the instructor draws various types of graphs on the board and shows the result of applying the algorithms by hand. This is a tedious and time-consuming process. This tool enables the instructor to create a graph dynamically and show the results of applying the algorithm spontaneously. The instructor can draw any type of graph using this tool. The instructor can create vertices anywhere on the screen and can move it to a new location after it is created. The vertices can also be deleted. By dragging the mouse from one vertex to another, an edge between the two vertices can be created. All these interactive features also work on mobile devices.

Benefit 2: Once a graph is created, the tool can show the result of applying the algorithm on the graph interactively. After a graph is modified, with a click of button, the tool can show the new result of applying the algorithm on the new graph. This is tremendously helpful to show students different scenarios.

Benefit 3: A picture is worth a thousand words. An interactive animation is worth more than pictures. The interactive animation not only catches student attention in the class, it also engages the student with visual interaction. Students can use the tool to study before and after the lectures to see how an algorithm works on different graphs.

Benefit 4: Our animation also serves as examples for students to write their own programs to visualize the graph algorithms. This gives students the opportunity to get deeper into the algorithms and see how the algorithms work in their own animation. In our data structures and algorithms courses, we assign projects for students to write their own code for graph algorithm animation. Students like the algorithm animation projects. As supported in [11], students learn better, when they actually implement the algorithms using animation.

6. EVALUATION

Many algorithm animation tools are available. It is safe to say that algorithm animation assists instruction, but whether it helps students to learn is a mixed bag. Some experiments show positive student outcome [1, 8], while others say there are no significant difference to students whether animations are used or not [2]. An experiment conducted at George Washington University [4] showed that the students who used an interactive version of courseware spent more time and performed worse overall than those who used the non-interactive version of the courseware. The reason behind this is that the tools are ineffective and difficult to use. Our goal is to develop a simple tool that is effective and easy to use. First, our tool is free and directly accessible on the Web and

can run on any device from a Web browser. There is no need to install any software. Second, our tool is intuitive. It has only four lines of instructions on how to use it. Third, we combined all the algorithms for unweighted graphs into one application and all the algorithms for the weighted graph into another application, rather to have a separate application for each algorithm. Once a graph is created, the user can apply different algorithms on the same graph.

We use the animation in our data structures and algorithm course in Java. The course covers recursion, Java generics, use of Java collections framework for array lists, linked lists, stacks, queues, priority queues, sets, and maps, implementation of array lists, linked lists, stacks, queues, and priority queues, binary search trees, AVL trees, hashing, and graphs applications for unweighted graphs and weighted graphs. The graph algorithms is a small part in the course, which is covered at the end of the semester. The course is offered every semester.

In the spring of 2015, we conducted a survey for a class of 26 students. The survey has many questions. Two questions related to the graph algorithm animation are the following:

1. Does the graph algorithm animation help you learn graph algorithms? 20 answered yes, 2 answered no, and 4 answered "not sure".
2. Is the graph algorithm animation intuitive and easy to use? All answered yes.

In the fall of 2015, we conducted a second survey for a class of 22 students. This time, we used a scale of 1 to 10 for answers, where 1 is poor and 10 is excellent. The result is as follows:

1. Does the graph algorithm animation help you learn graph algorithms? The average answer is 7.4.
2. Is the graph algorithm animation intuitive and easy to use? The average answer is 9.1.

The survey strongly suggests that the tool is easy to use and helps students learn graph algorithms.

7. IMPLEMENTATION OF THE ANIMATION

The animations are implemented using HTML, CSS, and JavaScript. The user interface is created using HTML. The style is defined in CSS. The user interaction and algorithms are implemented using JavaScript. We define the classes Graph and WeightedGraph to model unweighted graphs and weighted graphs. WeightedGraph is a subtype of Graph. The algorithms such as DFS, BFS, minimum spanning tree, and shortest path are implemented in these classes. The complete code for these classes can be obtained from https://yongdanielliang.github.io/animation/web/Graph.js and https://yongdanielliang.github.io/animation/web/WeightedGraph.js. When the user clicks the right mouse button on the canvas, a new vertex is created. The addVertex method in the Graph class is invoked to add the vertex to the graph. When the user clicks the left mouse button on a vertex, the vertex is removed. The removeVertex method in the Graph class is invoked to remove the vertex from the graph. When the user drags the mouse button from one vertex to another, an edge between the two vertices is created. The program invokes the addEdge method in the Graph class to add the edge to the graph. Each button on the user interface corresponds to an algorithm. For example, when the user clicks the DFS button in Figure 3.2, the program invokes the dfs method in the Graph class to find a depth-first search tree. The tree is then displayed on the canvas in the user interface.

The system is built using a modular approach. An animation for a new algorithm can be easily added by creating a button in the user interface and implementing the algorithm in the Graph class for unweighted graphs or in the WeightedGraph class for weighted graphs.

The source code (HTML, CSS, JavaScript) for the animations can be viewed using the "view page source" function in the browser. With the knowledge of HTML, CSS, JavaScript, and graph algorithms, one can modify the code to add new animations for custom algorithms.

8. LESSONS LEARNED

We started the project to develop the animations for graph algorithms in 2008. Over the years, we have created the animations for many graph algorithms and continuously improved the animation based on the feedback from the students and instructors. There are several lessons learned from developing the animations and from using the animations in classrooms.

- The first lesson learned is to make the animation easy to access. We initially developed the animation using Java applets. Due to security restrictions, many users cannot access the animation. We recreated the animation using HTML, CSS, and JavaScript. The animations now can be viewed anywhere from a browser on a computer and on a mobile device.

- The second lesson learned is to make the animation simple to deploy. We initially developed the animation for each algorithm. We had a total of thirteen animations: an animation for DFS, an animation for BFS, an animation for finding a shortest path, etc. With so many animations, it is difficult to deploy and the user has to click many links to access the animation. Later we combined all the animations into two animations: one for unweighted graph algorithms and the other for the weighted graph algorithms. Now we just need to deploy two animations rather than thirteen separate animations.

- The third lesson learned is to make the animation easy to use. In the early version, the animation lets the user enter the coordinates for each vertex and specify the edges in text boxes in order to create a graph. This proved to be difficult and time-consuming for the user to create a graph. Later we improved it by letting the user use the mouse gestures to add and remove vertices and create the edges. With the mouse gestures, the user can create a graph quickly and easily.

9. FUTURE WORK

We have improved the tool over the years. At present, the tool enables the user to create a graph, apply the algorithm on the graph,

and show the result of applying the algorithm. However, it does not show the intermediate steps to obtain the result. The future work is to expand the animation to show the user the step-by-step procedure for obtaining the results while still retaining the tool's simplicity.

10. CONCLUSIONS

This paper presented graph algorithm animation that is a useful tool for teaching and learning graph algorithms. It enables instructors and students to create custom graphs and see how the graph algorithms work. We developed two animations: one for the unweighted graphs and the other for the weighted graphs. For the unweighted graph, our algorithm animation supports the depth-first search, breadth-first search, shortest path, Hamiltonian path/cycle, finding connected components, finding a cycle, and finding bipartite sets. For the weighted graph, our animation supports minimum spanning trees, shortest path trees, shortest path, and travelling salesman problem. The animations are freely accessible from https://yongdanielliang.github.io/animation/animation.html.

11. ACKNOWLEDGEMENTS

Many thanks to Dr. Steven Gordon and the anonymous reviewers for their careful reading and constructive suggestions for improving the presentation of the paper.

12. REFERENCES

[1] Bazik, J., Tamassia, R., Reiss, S.P., van Dam A., "Software Visualization in Teaching at Brown University," in Software Visualization: Programming as a Multimedia Experience, The MIT Press, pp. 383-398, 1998.

[2] Byrne, M.D., Catrambone, R. and Stasko, J.T., "Do Algorithm Animations Aid Learning?", Tech. Rep. No. GIT-GVU-96-18, Georgia Tech Graphics, Visualization, and Usability Center, 1996.

[3] David Galles, Data Structure Visualizations, https://www.cs.usfca.edu/~galles/visualization/Algorithms.html. Computer Science Department, University of San Francisco.

[4] Jarc, D.J., M.B. Feldman, and R.S. Heller, "Accessing the Benefits of Interactive Prediction Using Web-based Algorithm Animation Courseware," in Proceedings of the ACM SIGCSE Technical Session, Austin, Texas, March 2000.

[5] Liang, Y. Daniel, REVEL™ for Introduction to Java Programming and Data Structures. ISBN-13: 978-0134167008. Pearson Education, 2016.

[6] Liang, Y. Daniel, REVEL™ for Introduction to C++ Programming and Data Structures. ISBN-13: 978-0134669854. Pearson Education, 2018.

[7] Liang, Y. Daniel, REVEL™ for Introduction to Python Programming and Data Structures. ISBN-13: 978-0135187753. Pearson Education, 2018.

[8] Naps, T., Eagan, J, and Norton, L. 2000. "JHAVÉ – An Environment to Actively Engage Students in Web-based Algorithm Visualizations", in Proceedings of the SIGCSE Session, ACM Meetings, Austin, Texas.Visualgo.net, Algorithm and Data Structure Animations, visualgo.net/en.

[9] REVEL™ educator study observes homework and exam grades at University of Louisiana, Spring 2016, http://www.pearsoned.com/results/revel-educator-study-observes-homework-exam-grades-university-louisiana/.

[10] REVEL educator study assesses quiz, exam, and final course grades at Central Michigan University, Fall 2015, http://www.pearsoned.com/results/revel-educator-study-assesses-quiz-exam-final-course-grades-central-michigan-university/.

[11] Stasko, John, "Using Student-Built Algorithm Animations as Learning Aids" in Proceedings of the ACM SIGCSE Technical Session, San Jose, CA., February, 1997.

[12] Visualgo.net, Algorithm and Data Structure Animations, visualgo.net/en.