CALIFORNIA STATE UNIVERSITY, NORTHRIDGE

# FUN WITH ALGORITHMS

A graduate project submitted in partial fulfillment of the requirements for
the degree of Master of Science in Computer Science

By

Armen Ourfalian

May 2018

The graduate project of Armen Ourfalian is approved:

| | |
|---|---|
| Jeff Wiegley, Ph.D | Date |
| Kyle Dewey, Ph.D | Date |
| John Noga, Ph.D, Chair | Date |

California State University, Northridge

Table of Contents

ABSTRACT

FUN WITH ALGORITHMS

By

Armen Ourfalian

Master of Science in Computer Science

Algorithm Visualizations are most effective when they are interactive. AV's are more than just videos or animations that show the runtime process of an algorithm, where videos/animations are no more effective than traditional teaching methods AV's have been shown to improve student understanding of algorithms across. But even the most effective tools are useless if they don't get used.

I set out to create a lecture-enhancing AV tool to be used by Computer Science instructors teaching Algorithms. The main objective being to create a tool that instructors will want to use. My target audience was the faculty at CSUN teaching algorithms. Having a background as a high school teacher, as well as taking multiple algorithms classes at CSUN, I have some insight into the needs of a lecturer and also the student.

I based the design of this project around my experience in Comp 496ALG (now relabeled Comp 482), covering three different algorithms taught in that class: STABLE MARRIAGE, INTERVAL SCHEDULING, and CLOSEST PAIR OF POINTS. For each topic I created an INSTANCE MAKER interface to create and edit instances of the problem, a SOLVER to perform the algorithm step-by-step, and a DISPLAY to show a visual diagram.

I demonstrated the product in front of two live classrooms, using the tool to enhance the lecture, engage the students, and pose some interesting questions. The feedback I received from these lectures was overly positive, both from the students and from the instructor.

# Chapter 1

# Introduction

## 1.1 Motivation

When I was a high school Math and Science teacher, I often tried to employ the use of technology in the classroom. I often found it to be a good way to spend less class-time on tasks that do not engage the student. An example of such a task is drawing a graphs (in Algebra class): while I was busy drawing the diagram, students would not engaged in the lesson.

So I started preparing graphs on a computer to display over a projector, and saved. valuable minutes of engaging class time, and a side benefit of using computer-generated images was that they looked far better than any graph I could have drawn by hand. The only problem with this solution was that I had to spend more of my own time outside of class to create these visuals, or find some online.

Due to my background as a teacher, I was drawn towards a thesis project where I could create an educational tool to be used in a classroom to help instructors draw diagrams and display them over a projector instead of having to draw them on the whiteboard. The overall goal of this project is to create a tool that will save time in class by reducing the teacher's non-engaging tasks, while not requiring too much of the teachers' time outside of class for preparation. For the subject matter, I chose Algorithm Visualization.

## 1.2 Background

Algorithms are a fundamental part of Computer Science education. Just about every introductory CS class discusses the various sorting algorithms, their advantages and disadvantages, followed by some video that graphically shows how each sorting algorithm is implemented

Algorithm Visualization (AV) is the use of software to create diagrams, animations, and other visual tools to facilitate the learning of an algorithm, its process (how it works and why it produces the correct result), and its complexity (runtime, required space, etc). There is an abundance of visualizations for sorting algorithms, and a decent amount for other introductory algorithms or data structures (Binary Search Trees, Linked Lists, Kruskal's Algorithm, Primm's Algorithm, etc.), but fairly few and far between for the more intermediate-level algorithms.

Studies have shown the positive effects of various AV's (see **Chapter 2 Literal Review**), and yet they are seldom used as part of the teaching process. AV can provide advantages to instructors because it allows them to display complex diagrams or data structures without having to draw them, which not only saves class time for more engaging activities, but also allows instructors to go over

examples that are much too complex to draw by hand, such as a graph with more than 20 vertices, or a 2-dimensional grid containing 500 points. Furthermore, AV is helpful to students because it gives them a chance to review the lesson outside of class, reproducing the problem on their own and giving them access to guided practice.

The Objectives for this project were influenced by the advantages discussed in the previous paragraph. These objectives outline the goals of the project as a whole but a more detailed list of requirements can be found in **Chapter 3.1 Requirements**

### 1.3 Objectives

1. Create an AV tool to be used by CSUN Faculty teaching an intermediate Algorithms class during lecture
   (a) Algorithms covered by this project will be drawn from those taught in Comp 482
   (b) The main use case is on a projector display as part of a lecture
2. The tool must be intuitive and easy to use
   (a) Users should be able to use the tool with little or no formal training
   (b) Using the tool to create diagrams should take no more time than drawing a similar diagram by hand
3. The tool will have the following features:
   (a) Allow the user to create instances of a given problem
   (b) Allow the user to simulate the steps of an algorithm and see the solution
   (c) Display visuals to describe how the algorithm is running

### 1.4 My Approach

The ultimate goal of this project is to create a tool that will actually be used in the classroom. The target users are CSUN faculty, and the target environment is in a classroom during lecture. I kept this in mind when I was planning, designing, and testing my project. Whenever a decision had to be made between making the tool more universally applicable versus making the tool a better fit for the specific target audience, I chose the latter. For example: that is why I chose specific lessons that were covered in CSUN's Comp 482 class.

In order to make the the tool easily accessible (by faculty as well as students), it was designed as a web app, more precisely as a front-end single-page application:

- Front-end: the app runs on the client's local machine

- Single-Page application: when the user interacts with the app, it changes the page dynamically without having to reload or refresh
- Application: computer program that accomplishes some task

Each algorithm within the app is a self-contained module (ie does not rely on other modules) comprised of

- An INSTANCE MAKER: allows the user to create instances of a given problem.
- A SOLVER: allows the user to run the algorithm.
- A DISPLAY: allows the user to see the various aspects of the problem while the algorithm is being performed.

I used a JavaScript framework called Vue with an MVC approach. I chose Vue because it is a relatively new framework and its popularity is on the rise [7] Furthermore, I used a popular Vue library called Vuex, which is a state-management library [8] I hosted the app on Heroku, because they offer free hosting for hobby-level apps. The app can currently be accessed by going to the following webpage:

https://fun-with-algorithms.herokuapp.com

## Chapter 2

## Literature Review

There are many AV systems that have been created, such s ANIMAL [5], HalVis [6], or BRIDGES [3]. And yet, these systems fail to reach mainstream computer science education, according to Hundhauser and Douglas [2], for a number of reasons, the greatest of which being that instructors find them too difficult to use , to the point where these systems take up more of the instructor's time in preparation than they save during class. And according to RoBling and Naps [4], there are eight pedagogical requirements for a successful AV System, which are summed up as such:

- General-purpose system that reaches a large target audience
- Allows the user to provide input, but in a manner that is not overly burdensome
- Allows the user to go backwards and forwards to different points of the algorithm
- Allows (and encourages) the user to interact with the system, and make predictions about what the algorithm will do next.
- Provides the user with textual explanations about what's going on
- Displays changing animations so the user can detect what happened.

The observations made by [2] and [4] served as the inspiration for my project, and I set out to create a tool that would be easy to use in order to encourage instructors to incorporate it into their curricula.

# Chapter 3

# Requirements

The requirements for this project were derived from a combination of three sources. First, I relied on my personal experience as a former teacher: I have taught Math and Science at the high school level, and would often incorporate technology into my lectures partly because I'm not good enough at drawing, and also because the students responded better to anything that was on a screen.

The second source for these requirements came from Professor Noga's input since he is the faculty member who most frequently teaches Algorithms, I discussed with him how he approaches teaching each algorithm, what would be a useful feature of the app to be used during lecture, what maximum or minimum conditions need to be met by the app, and what are some interesting instances that can stimulate in-class discussions.

Finally, I incorporated as much of what I learned from my research (discussed in Chapter 2, Literature Review) to create an AV tool that would improve student learning. For example, AV's that allow the user to create their own instances and run te algorithm step by step are much more effective.

## 3.1 Requirements

1. Create a web-app for Algorithm Visualization
   (a) The app will be a Single-Page Application
   (b) The app will run in the front-end
   (c) The app will run in the Google Chrome web browser
2. The target users of The app will be CSUN faculty who are teaching an intermediate-level algorithms class (such as Comp 482)
3. The app must be easy to use by its target users
   (a) Users should be able to interact with the app with little to no formal training
   (b) Where necessary, the app must provide instructions on any controls that do not meet the above criteria
4. The app will provide AV's for 3-5 algorithms covered in such a class, including:
   (a) Stable Matching
   (b) Interval Scheduling (unweighted)
   (c) Closest Pair of Points

5. For each algorithm, an AV will have the following:
    (a) An interface to create/modify instances of the problem
        i. User controls displayed on the webpage (text boxes, buttons, etc.)
        ii. Loading an instance from a .txt file
        iii. Saving an instance into a .txt file
        iv. Loading an instance from a database
    (b) An interface to solve the created instances (ie run the algorithm)
        i. Perform a single step of the algorithm at a time
        ii. Perform the entire algorithm automatically.
    (c) A visual diagram of an instance of the problem
        i. The diagram must be organized and easy to understand
        ii. The diagram must be appropriately sized to be displayed on a projector in front of students
        iii. The display must update as the solver is performing the algorithm

## Chapter 4

## Development Process

Since I was working by myself on this project, I did not adopt a formal development process, but it was iterative, with multiple deliveries, and continued improvement. The nature of this project is very modular. It consists of various algorithms which do not rely on one another. Thus I worked on each algorithm individually from conception to completion and published it to the live website before moving on to the next algorithm. The process of creating an AV for a given problem went (roughly) as follows:

- Decide acceptable ranges and restrictions for inputs and problem size
  - These restrictions ensure that the app will run smoothly and not slow down
  - The restrictions also make sure the visualization is easy to understand, and fits on the target screen size
  - The ranges should be larger than if the problem were being solved by hand
- Create the data model
  - What data structures are required for a given type of problem
  - What properties and methods are required to run an algorithm
  - Which parts of the data should be made available to the display
- Create the INSTANCE MAKER
  - User interface (text fields, buttons, sliders, etc.) to construct an instance manually
  - Save/Load interface through which the user can upload/download a text file to create an instance
  - Predefined example instances that can be loaded without manually being inputted (this is a nice-to-have)
  - Update the data model as the user makes changes to the problem instance
- Create the DISPLAY
  - Visual components that represent the data model
  - Must respond to changes in the data model
- Create the SOLVER
  - User interface to run the given algorithm

  - Update the data model as the algorithm is performed

- Show Professor Noga the progress
  - Discuss various decisions/restrictions/compromises made and whether they are acceptable
  - Make any necessary changes

# Chapter 5

# Tools and Technologies

I am going to assume the reader has a basic understanding of HTML, CSS, and JavaScript. When I first began working on this project, I made the decision to work in plain JavaScript, but I quickly realized that using a framework would allow me to focus on the more interesting parts of the project (how to visualize algorithms) without having to spend hours on the smaller problems (how to update dozens of elements on a page whenever a part of the data model changes). So I chose to use Vue.js for my project.

## 5.1 Vue js

Vue was released in February 2014, by Evan You [10]. It is an open source JavaScript framework for creating Single-Page Applications, but it can also be easily incorporated into an existing project. The two main features of Vue are creating reactive web pages, and components.

For a web page to be **reactive**, it needs to respond to changes. This means that various elements of the web page have to re-render themselves, whenever parts of the data model change, and conversely the data model has to update whenever a user interacts with the web page. This can be achieved in plain JavaScript by manually manipulating the elements when the data model changes and using event listeners to handle user interactions, but it is very tedious, error-prone, and inefficient. Vue utilizes a virtual DOM (Document-Object Model) to efficiently render web pages and re-render only the parts of a page that need to respond to changes in the data model, with minimal effort by the developer (me).

The other key feature provided by Vue is **components**, which are self-contained parts of an application. Components are reusable, can be nested inside one another, and inherit from one another. This makes Vue great for separating different tasks and responsibilities of a program in an organized way. Vue uses a **template** syntax for its components, where each component has an HTML, CSS, and JavaScript portion. These can all be placed in the same file (but divided into `<template>`, `<style>`, and `<script>` sections respectively), or each can go into a separate file. This template syntax was the biggest reason I chose Vue over alternative frameworks.

Below is an example of a Vue component.

Listing 5.1: Example Vue Component

```
1 <template>
2   <div>
3     <h1>Example Vue Component</h1>
4     <div>
```

```
5         <p> {{num}} </p>
6         <button @click='increment'> Plus One</button>
7         <button @click='decrement'> Minus One</button>
8       </div>
9     </div>
10  </template>
11
12  <script>
13  export default {
14    data() {
15      return {
16        num: 0,
17      };
18    },
19    methods: {
20      increment() {
21        this.num = this.num + 1;
22      },
23      decrement() {
24        this.num = this.num - 1;
25      }
26    }
27  };
28  </script>
29
30  <style scoped>
31    div {
32      border: 1px solid black;
33    }
34  </style>
```

In the above example, the code between the `<template>` tags is the HTML code, where the various elements are drawn on the web page. The only new Vue-specific concepts in this section are on lines 5, 6, and 7. On line 5, the {{ num }} inside the `<p>` tags will be replaced by value of the `num` (see next paragraph) variable associated with this Vue component. The `@click`s on lines 6 and 7 define what function is called when the user clicks on either button. The functions must be defined for this Vue component (again, see next paragraph).

The code between the `<script>` tags is the JavaScript code, where the data and functionality

is defined. Here, the value for `num` is initialized to zero (line 16), and the two functions called `increment` and `decrememnt` (lines 20 and 23) increase or decrease the value of `num` by one. When either button gets pressed, these functions are called, the value of `changes`, and the text inside the `<p>` tag is updated and re-rendered automatically.

The code in the `<style>` tags is the CSS, where the elements' style, size, and positioning are set. The `scoped` keyword tells Vue that any style rules written here will not affect any other components. This greatly reduces the complexity of styling large applications. Furthermore, when global style rules are required, they can be placed in the top-level component with the `scoped` keyword removed, and those rules will trickle down to all the nested components.

### 5.2 Alternatives to Vue

The two most popular alternatives to Vue are React and Angular [1]. Both of these frameworks are older than Vue, and they are used and maintained by tech Giants Facebook and Google respectively. In this section I will briefly compare and contrast Vue with these frameworks and discuss why I chose to use Vue over either of them.

React and Vue are very similar: they both use reusable components to separate the program into self-contained units, and they both utilize the Virtual DOM to create reactive webpages. But unlike Vue, where the HTML code is separated from the JavaScript, React combines the two together using JSX (an XML-like syntax for writing HTML code directly into JavaScript functions). The reason I chose Vue over React was because I found the JSX syntax to be much harder to follow than Vue's template syntax.

Vue was inspired by Angular, so the two frameworks have a very similar syntax. But Angular is aimed at projects with much bigger scope than mine, and has a steeper learning curve [1]. I chose Vue over Angular because it is more lightweight Angular [9].

### 5.3 Vuex

Vue allows for top-level components to send data (or bind variables) down to any components that are nested within them. However, this data binding is only one-way, so the inner components cannot send data back up to their parents. Vue provides a way to achieve this with events: child components can emit events that their parents listen for, and can respond to. Although this is functionally equivalent to having two-way data binding, it becomes much more verbose. Furthermore, the verbosity is compounded when sibling components (or worse yet, when elements that are much further apart in the tree) need to communicate with one another. This is the reason I added **Vuex** [11] to my project.

Vuex is a state-management library for Vue. It moves the responsibility of keeping track of data away from the components into a **store** whose sole purpose is to manage the data in a structured

way. A Vuex store is made up of a *state*, *mutations*, and *actions* (there are other parts to Vuex stores like *getters*, *modules*, and *plugins*, but I will not discuss them here).

The *state* is where all the data is kept for the entire application. This data can be accessed by any of the Vue components, but it can not be modified by any of the Vue components. The only way to modify any data held in the state is to use *mutations*. Mutations are functions that modify the state. *Actions* are used as an interface between Vue and Vuex. Actions are functions that are "dispatched" from Vue components, and their main task is to invoke ("commit") the mutations.

I utilize Vuex for my project by creating a different store for each algorithm: the *state* holds all the information about the problem instance, the *mutations* holds all the operations to edit the instance as well as all the operations that will be done to run the algorithm, and the *actions* correspond to various tasks the user may wish to perform. As an example, the Vuex store for STABLE MARRIAGE has the following:

- state
  - Two $n \times n$ arrays where the preferences of the men and women are kept
  - An $n \times n$ array of rejections (which women have rejected which men)
  - A list ($0 \leq length \leq n$) to keep track of what tentative matchings currently exist.
- mutations
  - `swapPreferenceBoxes`: reorder the preference array
  - `propose`: a man makes a proposal to a woman
  - `acceptProposal`: the woman accepts the proposal
  - `rejectProposal`: the woman rejects the proposal
  - `resetSolver`: reset the algorithm to start from the beginning
- actions
  - `proposeDispose`: This is called when the user clicks a particular button in the solver to run a single step of the Gayle-Shapely algorithm.
  - `loadFile`: This is called when the user loads a text file instead of manually editing the problem instance.

Using Vuex to handle state management removes that responsibility from the Vue components, making their sole responsibility to create a display. This separation of responsibilities not only makes the Vue components more reusable it also makes the runtime process of each algorithm easier to implement, debug, and test.

# Chapter 6

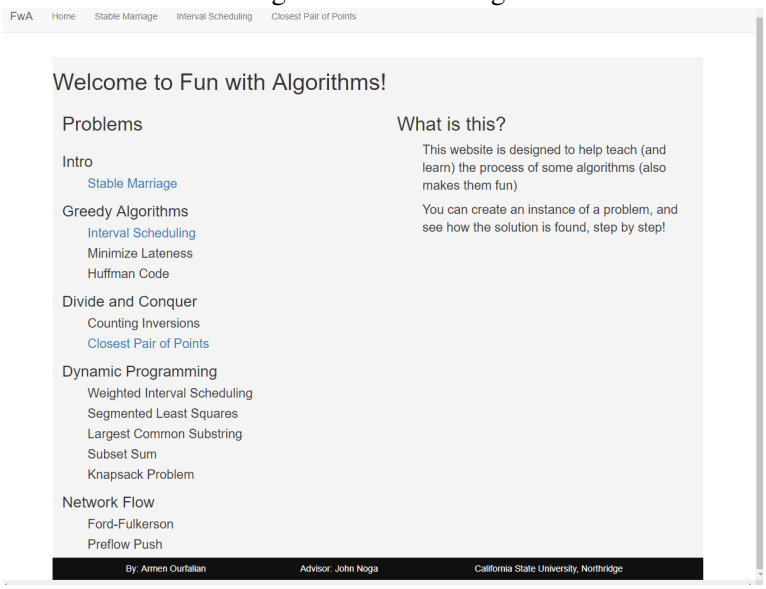
# Guide

In this section I am going to discuss the various pages of the app, I am also going to give a brief description of the various components in each page, and how the user can interact with them. Note that some of the details (and images) in this section are subject to minor changes.

## 6.1 Home Screen

The home page of this app contains a list of topics in black, organized by type of algorithm. These topics are among those covered in an intermediate Algorithms class. The topics that have been covered are in blue instead of black, and they are links that lead to their corresponding page. The purpose of this page is to show a list of potential future additions to this project. The navigation bar also has links to each problem that is covered by the project.

Figure 6.1: Home Page




## 6.2 Reusable Components

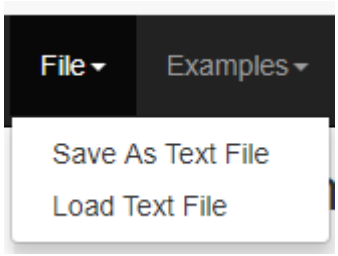
As mentioned in **Chapter 5 Tools and Technologies**, a key feature of Vue is reusable components. So a number of components in this app were created with reusability in mind. Reusing the same components across various pages gives the website a more consistent interface overall. This also lessens the learning curve required to use this app. I packed the majority of the reausable functionalities into the secondary navigation bar to ensure they would be easy to find no matter what page the user found themselves on. The key features of the second navbar are the following:

Figure 6.2: Second Navbar


A **File Menu** with the options *Save as Text* and *Load as Text*. These features are familiar to the modern user. Although the interface to save or load an instance is exactly the same For each problem, the save and load functionalities are customized for each problem. This is because each problem requires a different format for its input data.

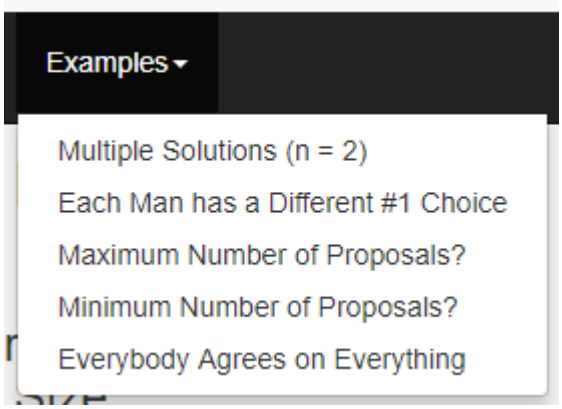
For example: STABLE MARRIAGE requires an input of exactly two $n \times n$ lists of numbers, where the numbers must be in the range $[0, n-1]$ or $[1, n]$, whereas the INTERVAL SCHEDULING requires an input of any number of rows, each containing two numbers. Despite these differences, the interface for saving or loading instances is exactly the same for all problems within the entire project.

Figure 6.3: File Menu




An **Examples Menu** that contains a list of pre-made instances for each problem that are loaded from a database.

Figure 6.4: Examples Menu

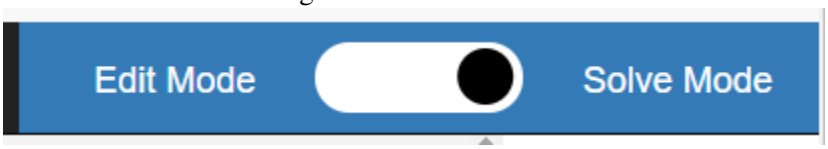



A switch to go between **Edit Mode** and **Solve Mode**. Edit Mode will display the INSTANCE MAKER, whereas Solve Mode will display the SOLVER. In both modes, the DISPLAY will also

be visible. The functionality of this switch is the same across all problems within the app: the problem instance can only be changed when the page is on Edit Mode, and algorithm can only be performed while in Solve Mode. The reason for this is because giving the user the ability to change the problem instance mid-way through the algorithm would cause unexpected problems such as infinite loops or divide-by-zero errors. For this reason, whenever the user switches from Solve Mode into Edit Mode, the SOLVER is completely reset to its initial state, and any progress made in the algorithm is lost.

Figure 6.5: Edit Mode


Figure 6.6: Solve Mode



# Chapter 7

## Conclusions and Future Work

I am overall pleased with the result of this project, though it is far from complete. For future work, I would like to implement more problems, as well as create more generic tools that can be useful in the classroom. One such tool would be a graph builder: a tool that would help users draw graphs (directed or undirected) in an effortless way.

## References


[1] Angular vs. React vs. Vue: A 2017 comparison . https://medium.com/unicorn-supplies/angular-vs-react-vs-vue-a-2017-comparison-c5c52d620176.

[2] Sarah Douglas Christopher Hundhausen and John Stasko. A Meta-Study of Algorithm Visualization Effectiveness . *Journal of Visual Languages and Computing*, 2002.

[3] David Burlinson et. al. Bridges: A System to Enable Creation of Engaging Data Structures Assignments with Real-World Data and Visualizations . *SIGCSE*, 2016.

[4] Guido Rling and Thomas L. Naps. A Testbed for Pedagogical Requirements in Algorithm Visualization . *ITiCSE*, 2002.

[5] Guido Rssling and Bernd Freisleben. ANIMAL: A System for Supporting Multiple Roles in Algorithm Animation . *Journal of Visual Languages and Computing*, 2002.

[6] N. Hari Narayanan Steven Hansen and Mary Hegarty. Designing Educationally Effective Algorithm Visualizations . *Journal of Visual Languages and Computing*, 2002.

[7] Top JavaScript Libraries & Tech to Learn in 2018 . https://medium.com/javascript-scene/top-javascript-libraries-tech-to-learn-in-2018-c38028e028e6.

[8] Vue.js Documentation . https://vuejs.org/.

[9] Vue Comparison with Other Frameworks . https://vuejs.org/v2/guide/comparison.html/.

[10] First Week of Launching Vue.js . http://blog.evanyou.me/2014/02/11/first-week-of-launching-an-oss-project/.

[11] Vuex Documentation . https://vuex.vuejs.org/.