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

(This section is in progress)

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 [8] Furthermore, I used a popular Vue library called Vuex, which is a state-management library [9] 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

(This chapter is in progress)

There are many AV systems that have been created, such s ANIMAL [6], HalVis [7], 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 [5], 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 [5] 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 Marriage
   (b) Interval Scheduling
   (c) Closest Pair of Points

5. For each algorithm, an AV will have the following:

   (a) INSTANCE MAKER: 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) SOLVER: 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) DISPLAY: 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 algorithm is performed

Chapter 4

Development Process

Since I was working by myself on this project, I did not adopt a formal development process, but my development process has been iterative, with continuous 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, published it to the live website and then I moved on to the next algorithm. The process of creating an AV for a given problem went (roughly) as follows:

4.1 Choose which Algorithms to Cover

I selected from the set of problems taught in Comp 482 at CSUN because the goal of this project is to create a tool to be used by CSUN faculty. I wanted to create an AV for at least one of each major section (Greedy, Divide and Conquer, Dynamic Programming, Network Flow) taught in that class, but due to time constraints I had to narrow it down even further. I chose the AV's based on how much it would improve a lecturer's ability to communicate the problem and its solution, how important the topic is for learning other lessons, and how tedious it is to solve the problem without the use of an AV. In **Chapter 5 The Algorithms**, I discuss in detail why I chose each of the algorithms that I created an AV for.

4.2 Set the Scope and Limitations

I was often faced with a decision between making an AV more general-purpose versus one that would accomplish a specific task very well. I kept the goal of this project in mind when making such a decision (which almost always turned out to be the less generalized option). Thus I imposed a set of restrictions in scope and problem size in order to improve the ability of each AV to accomplish its task.

These restrictions ensure that the app runs smoothly on classroom computers, fits nicely onto a web page, and does not overwhelm the viewer with too much information. However, I made sure the app would allow instances that were much larger than what a lecturer could show by hand. For example, a typical Stable Marriage problem that is demonstrated in lecture will have no more than 3 men and 3 women. Meanwhile the Stable Marriage AV in my project allows for as many as 14 (but I recommend keeping it below 7).

I also limited the scope of each problem to the simplest version that is covered in a typical class. The reason for this is to have the app be easier to use.

4.3 The Data Model

The first part of creating each AV is creating the data model. The data model consists of data structures (arrays, maps, lists, etc.) that represent an instance of the problem, and functions that mutate those data structures as an algorithms is being performed. Although most problem instances can be represented with just one or two data structures, the data model of an AV also includes other data structures that provide different perspectives of the instance and others that are necessary for the solver. For example, the Stable Marriage data model includes:

- Two $n \times n$ arrays that show preference lists,
- A list of $length \leq n$ that shows Tentative Matches,
- Two lists of $length \leq n$ that show any people who are currently unmatched.

The only *necessary* data structure in the above list is the first two, but the third one provides utility that makes the code easier to write and follow.

Since this project focuses on AV's to be used in lecture, the problem instances don't become large enough for memory management, efficiency, and response time to become an issue. And since the app is programmed in JavaScript, a language with weak typing (almost no typing), most of the data structures consisted of either arrays or objects.

4.4 The INSTANCE MAKER and Save/Load

The Instance Maker of an AV consists of all the parts of the user interface that allow the user to create and edit instances of the given problem. The first step of creating the Instance Maker is to create a textbox, even though it almost always gets removed from the final version of the Instance Maker (because typing into a textbox is very dull and prone to errors). But having a reliable way to test various inputs is useful when determining which user interface components (buttons, sliders, or custom components) to implement.

I prioritized components that are easy to use, because it is important for lecturers to use as little time as possible creating problem instances (that's one of the goals of this project after all) which will allow them to spend more time explaining the algorithm and how it works.

Depending on the specific algorithm, an Instance Maker requires one or more of the following functionalities:

- Create a new piece of data in the Data Model.

  For example: adding a new interval in Interval Scheduling

- Remove a piece of data from the Data Model.

  For example: removing an interval from Interval Scheduling

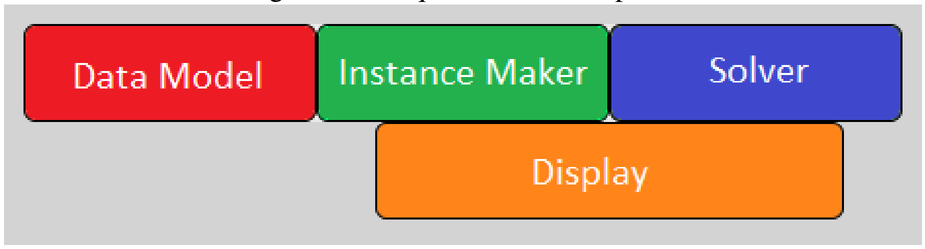
- Edit a piece of the Data Model.

  For example: reordering a person's preference list in Stable Marriage.

A few of the more specific Instance Maker components will be discussed in **Chapter 7 Guide**.

The last step in creating the Instance Maker is the Save/Load specified in Requirements 5(a)ii and 5(a)iii. The Save/Load feature converts the most important parts of the Data Model into text. The purpose of the Save/Load feature is to further reduce the amount of time lecturers spend creating instances in class by having them load up instances that they have created in advance. I chose text files to make this feature as easily accessible as possible.

Figure 4.1: Sequence of Development



4.5 The Display

The Display was often developed in parallel to the Instance Maker because the very nature of creating interactive diagrams requires the user to be able to interact with the diagrams. Oftentimes the line between the Instance Maker and the Display gets blurred, and they become a single entity. Furthermore, the Display also needs to be developed in parallel to the Solver (see below) because it needs to react to the Solver and update as an algorithm is being performed (see Figure 4.1).

Creating the Display means taking the most interesting parts of the Data Model and turning them into pretty, visual diagrams. When creating displays for the various problems I thought about the way each of these problems are solved on the whiteboard during lecture, on paper for homework, and also on paper for tests. I attempted to create visuals that matched (some of) those scenarios in the hopes that it will give students an idea of how to approach their assignments.

4.6 The Solver

The final and most difficult part of each problem is the Solver. The Solver is responsible for implementing the algorithm, but it must do so in a stepwise manner. It is important to determine what constitutes a single "step" in the Solver, where a step corresponds with a single action by the user.

Having each step be too small would force the lecturer to spend more time interacting with the Solver than their own students, while having each step be too small would give the students a hard time understanding what's going on. An appropriate size for each step is one or two lines of the algorithm's pseudocode.

The reason why the Solver is the hardest part of the development process is because its accuracy is extremely important. Having the Solver produce an incorrect result would completely negate any benefits of the entire app.

Although the actions of the Solver are represented in the Display, it is often useful for the Solver to provide a second form of feedback through messages. These messages often narrate what is being done with each user interaction, and their language is similar to the pseudocode of the algorithm.

Chapter 5

The Algorithms

5.1 Stable Marriage

5.1.1 Formal Definition

Given n men and n women, each with a preference list where they rank all the members of the opposite sex. Find a matching with no *instabilities*.

A *matching* is a list (of length n) of man-woman pairs. An *instability* occurs in a matching when a man m and a woman w both prefer each other over the person they are currently matched with.

5.1.2 Why I chose to include this Algorithm

I knew I wanted to cover Stable Marriage when I first decided on this project because it is taught during the first week of class. I gave precedence to subjects covered earlier in the semester because that is the time when students neither have a firm grasp of the subject matter nor are they used to their professor's teaching style, so they are far more likely to have trouble understanding the material than later on in the semester.

Another reason why I chose Stable Marriage was because of how tedious it is to solve this problem on paper, which (speaking from experience) involves many iterations of writing a matching and then crossing it out. Whereas having an AV tool to do the same task removes the busy work and allows the user to focus on the more important aspects of the problem.

Instances of Stable Marriage that are not trivially small ($n > 4$) tend to be too complex to solve by hand (you need at lest 2 $n \times n$ matrices to even represent the problem before even beginning to solve it). So most in-class examples are problems with size $n = 3$ or $n = 2$. Using an AV tool would allow lecturers to discuss much more interesting cases.

5.1.3 Limitations and Scope

My AV for Stable Marriage allows for problem sizes up to $n = 14$. The reason behind this limit is to ensure the web page does not become overly large or overly slow. However, I recommend keeping the problem below $n = 6$ because that is the largest problem size that will still have the preference rows fit on a single line in the web page.

There are many variations of the Stable Marriage problem, such as:

- The number of men does not need to equal the number of women.
- Some people can choose to stay unmarried if they are unhappy with their current matching.
- The pairs in a matching don't need to be man-woman pairs, they can be pairs of any two people.

My project only covers the basic version of the Stable Marriage problem, where there are an equal number of men and women, and every person must be married to a member of the opposite sex. [4]

5.2 Interval Scheduling

5.2.1 Formal Definition

Given a set of intervals $(startTime, finishTime)$, find the maximum number of intervals that can be scheduled without having any two intervals *overlap*.

Two intervals are said to *overlap* if one interval starts after the other starts, but before the other interval is finished. For example, the two intervals $p_A = (start_A, finish_A)$ and $p_B = (start_B, finish_B)$ are said to *overlap* if:

$$start_A < start_B < finish_A$$

5.2.2 Why I chose to include this Algorithm

Interval Scheduling is also taught very early on in the semester and is used as an introduction to Greedy Algorithm and makes for a good interactive lesson. During lecture, students make suggestions of possible greedy solutions to the problem (take the interval with the least number of overlaps first, etc.) while the professor (usually) disproves their suggested solution. Oftentimes the professor has to use a counterexample to do this, and that is exactly where an AV tool such as my project would be useful.

Another reason I chose Interval Scheduling was because students often misunderstand the actual problem they are trying to solve. The Interval Scheduling problem is usually introduced as ”imagine you have a resource that many people want to use, such as a basketball or an opera house. How can you make the largest number of people happy?” so many students intuitively think that they must fill up as much of the resource's time as possible in order to make efficient use of the resource. Meanwhile, other students think they must fill up as little time as possible to avoid wear-and-tear on the resource. This creates a situation where the teacher correcting the first group's misunderstanding only reinforces the belief of the second group and vice versa, but having an AV tool would better allow the teacher to demonstrate what a proper solution looks like and minimize this sort of mass confusion.

5.2.3 Limitations and Scope

My AV for Interval Scheduling limits the intervals' start and finish times to any integer between $t = 0$ and $t = 30$ in order to ensure that the display fits on the screen. I also impose a maximum of $n = 200$ intervals to ensure the app does not slow down.

My project only covers Unweighted Interval Scheduling, but some variations of this problem are:

- Weighted Interval Scheduling - each interval is assigned a value, and the goal is to find a set of intervals that either maximizes or minimizes the sum of all its values
- Channel Allocation - all of the given intervals must be scheduled, but intervals that overlap must be scheduled on different channels. The goal is to use as few channels as possible.

5.3 Closest Pair of Points

5.3.1 Formal Definition

Given a set of points (x, y), find the minimum distance between any pair.

5.3.2 Why I chose to include this Algorithm

When discussing with my advisor about potential Divide and Conquer algorithms, he suggested Closest Pair of Points. This problem is easy to understand (even kids can understand what is being asked), but difficult to demonstrate to a class because drawing points on a whiteboard is both tedious and inaccurate.

Closest Pair of Points is not a problem given on exams because its instances are either trivially easy or too time-consuming and filled with repetitive calculations. However, it is a very useful topic to cover in lecture because it provides a gateway to other Divide and Conquer algorithms. There are many general concepts that are taught when teaching about Divide and Conquer algorithms in general, for example:

- When to stop dividing a problem into subproblems and instead just solve it.
- How many times can a single problem be divided.
- Why is the time complexity of two smaller problems smaller than the time complexity of the original problem.
- What is the Time complexity of the recombining step.

An AV for an easy problem such as Closest Pair of Points will give professors the ability to explain these concepts with an example that is simple to understand.

5.3.3 Limitations and Scope

Variations on this problem come by either changing the space in which the points exist, or the definition of distance (or both). In order to keep the problem simple and easy to understand, I used Euclidean distnace to measure the distance between two points, because it is the most intuitive.

$$d(A,B) = \sqrt{(A_x - B_x)^2 + (A_y - B_y)^2}$$

I also limited this problem to a 2-dimensional x-y plane, with coordinates ranging from 0 to 500 (a possible alternative was to make the range between -250 and 250). The maximum number of points allowed in a given instance is 500, which means that a given problem can be divided into two subproblems no more than 8 times.

Chapter 6

Testing and Validation

The two main goals of this project are to create an AV tool, and to create a tool that would be used by lecturers in an Algorithms classroom. To test for the first goal, I performed unit testing on the Data Models for each of the implemented algorithms. To test for the second goal, I did live demonstrations of the app in front of live classrooms.

6.1 Unit Testing

Unit tests were performed on the Data Models of each algorithm to ensure that the algorithms were being implemented correctly. The Vuex Store is where the Data Model is held, so each component of the Vuex Store was unit tested. Each mutation and action in the store was tested with correct inputs as well as incorrect inputs to make sure that the system would handle errors correctly.

Since all the functionality was implemented in the Data Model, and the Components simply display the data that is held in the Vuex Store, no automated tests were done for the user interface, but I tested each Vue Component manually to ensure it demonstrated the correct behavior. This manual testing was done on the Google Chrome browser, though I do not anticipate any problems arising if Mozilla Firefox is used instead.

6.2 In-Class Presentations

6.2.1 Stable Marriage

I demonstrated the Stable Marriage AV to two different sections of Professor Noga's Comp 482 class. The class had already been lectured about the Stable Marriage problem during the previous week, so they already had a solid understanding of the problem beforehand. I had a few problem instances that I had saved on my computer in advance, I used those instances to discuss some subtle intricacies of the Stable Marriage problem, such as:

- In some instances the women's preference lists have no effect on the outcome
- Some instance can have multiple stable matchings
- Does the order in which the men propose affect the outcome?
- What is the lowest number of proposals that can happen? What instance can produce that result?
- What is the highest number of proposals that can happen? What instance can produce that result?

Using the tool allowed me to discuss these ideas with the students, show them why some properties hold true while others don't. And on more than one occasion I repeated an example when I noticed that some students were still confused.

6.2.2 Interval Scheduling

Similar to the Stable Marriage demonstration, I performed a lecture to two sections of the Comp 482 class the week after they had learned Interval Scheduling, so they had a decent understanding of the material. However, I would like to note that there were still multiple students who were confused about the problem (more than one student did not know that the problem asks for the total number of intervals).

I used the app to perform a discussion-heavy lecture on various possible greedy solutions to the problem (take the shortest interval, or the interval with the least number of overlaps), and why many of those solutions would not produce the optimal result ("Because here's a counterexample!")

Chapter 7

Guide

(This chapter is in progress)

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. Most parts of the user interface, such as buttons, text fields, and menu items are meant to be obvious and intuitive. These do not require any sort of special training for users to learn how they work, and I will discuss them only briefly here. But the user interface of this app also contains a few interactions that are specific to the problem being solved. The purpose of this section is to outline those interactions. Note that some of the details (and images) in this section are subject to minor changes.

7.1 Home Screen

The home page of this app contains a list of topics in black, organized by type of algorithm. *(Figure 10.1)* 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.

7.2 Reusable Components

As mentioned in **Chapter 8 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 reusable functionalities into the secondary navigation bar *(Figure 10.2)* 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:

A **File Menu** *(Figure 10.3)* 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.

Next to the **File Menu** is an **Examples Menu** *(Figure 10.4)* that contains a list of pre-made instances for each problem that are loaded from a database. When a user selects one of the items from this menu, that instance is automatically loaded into the current page.

On the far-right side of the second navbar is a switch *(Figures 10.5 and 10.6)* that goes 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.

7.3 Stable Marriage

Like all algorithm pages, the Stable Marriage page has an INSTANCE MAKER, a DISPLAY and a SOLVER. The DISPLAY is always visible on the page, and the user can swap between the other two by choosing Edit Mode or Solve Mode on the secondary navbar. When first navigating to this page, the user is presented with the INSTANCE MAKER *(Figure 10.7)*

7.4 Interval Scheduling

7.5 Closest Pair of Points

Chapter 8

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.

8.1 Vue js

Vue was released in February 2014, by Evan You [11]. 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 8.1: Example Vue Component

```
<template>
  <div>
    <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` variable associated with this Vue component (see next paragraph). The `@clicks` 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 are 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 `num` 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 other components. This greatly reduces the complexity of styling large applications because the style rules for each component is located within the component itself. And 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.

8.2 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 a 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 main reason I chose Vue over React was because I found using JSX 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 .or example: there is a command in Angular called `ng-if`, and its counterpart in Vue is called `v-if`. 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 than Angular [10].

8.3 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** [12] 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 user interface. 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 9

Conclusions and Future Work

(This chapter is in progress)

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.

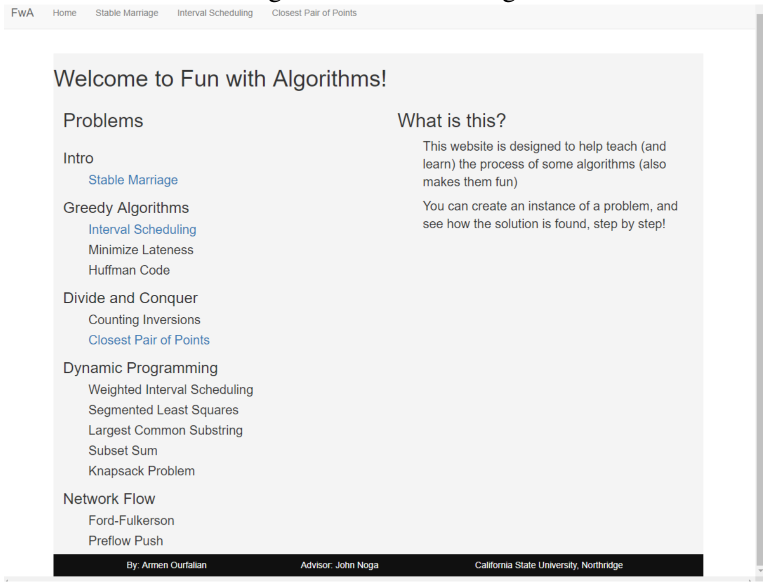
Chapter 10

Appendices

(This chapter is in progress)

10.1 Figures

Figure 10.1: Home Page



FwA Home Stable Marriage Interval Scheduling Closest Pair of Points

Welcome to Fun with Algorithms!

Problems

Intro

Stable Marriage

Greedy Algorithms

Interval Scheduling

Minimize Lateness

Huffman Code

Divide and Conquer

Counting Inversions

Closest Pair of Points

Dynamic Programming

Weighted Interval Scheduling

Segmented Least Squares

Largest Common Substring

Subset Sum

Knapsack Problem

Network Flow

Ford-Fulkerson

Preflow Push

What is this?

This website is designed to help teach (and learn) the process of some algorithms (also makes them fun)

You can create an instance of a problem, and see how the solution is found, step by step!

By: Armen Ourfalian Advisor: John Noga California State University, Northridge

Figure 10.2: Second Navbar

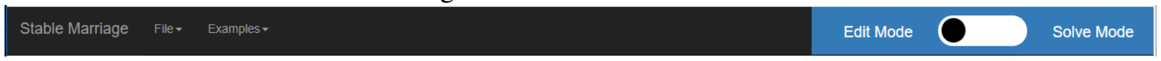


Figure 10.3: File Menu

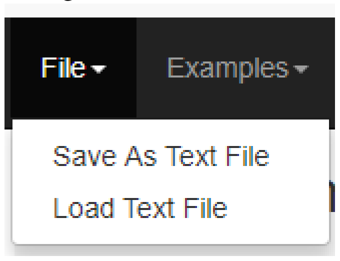


Figure 10.4: Examples Menu for Stable Marriage

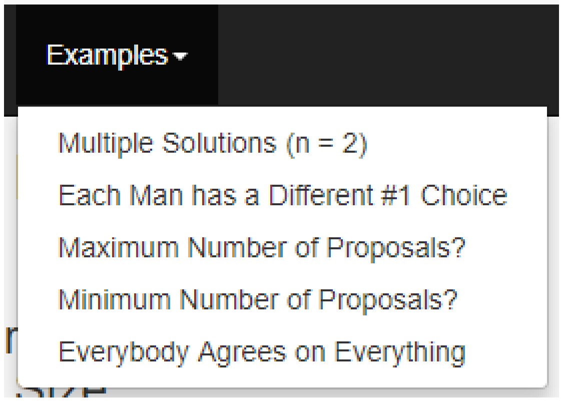

Examples ▾
Multiple Solutions (n = 2)
Each Man has a Different #1 Choice
Maximum Number of Proposals?
Minimum Number of Proposals?
Everybody Agrees on Everything

Figure 10.5: Edit Mode

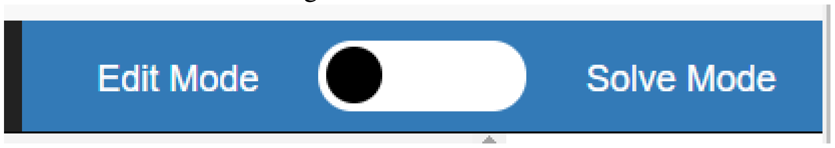


Figure 10.6: Solve Mode

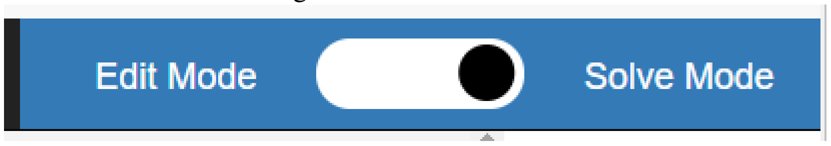
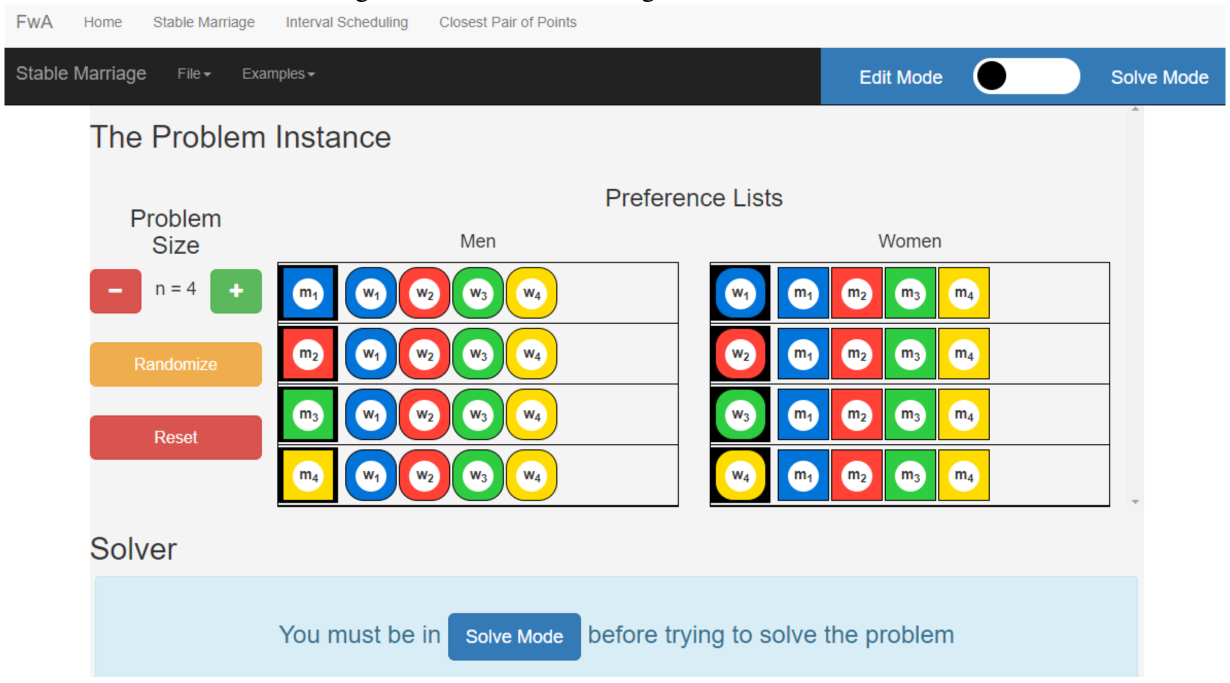


Figure 10.7: Stable Marriage in Edit Mode



You must be in Solve Mode before trying to solve the problem

Men				
m_1	w_1	w_2	w_3	w_4
m_2	w_1	w_2	w_3	w_4
m_3	w_1	w_2	w_3	w_4
m_4	w_1	w_2	w_3	w_4

Women				
w_1	m_1	m_2	m_3	m_4
w_2	m_1	m_2	m_3	m_4
w_3	m_1	m_2	m_3	m_4
w_4	m_1	m_2	m_3	m_4

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] Eva Tardos Jon Kleinberk. *Algorithm Design*. Pearson, 2014.

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

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

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

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

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

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

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

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