MASTERS PRESENTATION

FUN WITH ALGORITHMS

By
Armen Ourfalian

Committee Members:

John Noga, Ph.D (Chair)
Jeff Wiegley, Ph.D
Kyle Dewey, Ph.D

Date: **Friday, May 4th at 2:00 pm**
Location: **JD 3508**

ABSTRACT

I created a lecture-enhancing AV tool to be used by Computer Science instructors teaching Algorithms. The main objective of this project was to create a tool that instructors will want to use. My target audience is the faculty at CSUN teaching Algorithms.

Algorithm Visualizations (AV's) 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 multiple studies. But even the most effective tools are useless if they don't get used.

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 algorithm 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.

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

I created a lecture-enhancing AV tool to be used by Computer Science instructors teaching Algorithms. The main objective of this project was to create a tool that instructors will want to use. My target audience is the faculty at CSUN teaching Algorithms.

Algorithm Visualizations (AV's) 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 multiple studies. But even the most effective tools are useless if they don't get used.

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 algorithm 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.

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 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 graphs in an Algebra class: while I was busy drawing a diagram or graph, students would not be engaged in the lesson and would start distracting their friends and classmates.

So I started preparing graphs on a computer to display over a projector, and saved valuable minutes of class time for more engaging activities. A side benefit of using computer-generated images is that they look far better than any graph I can ever draw 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 them 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. I wanted 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 AV's are fairly few and far between for the intermediate-level algorithms.

Studies have shown the positive effects of various AV's (see **Chapter 2 Literature 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 tool will be used 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, so it does not rely on other modules. Each module is 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 [**?**]. Furthermore, I used a popular Vue library called Vuex, which is a state-management library [**?**]. I discuss Vue and Vuex more in-depth in **Chapter 8 Tools and Technologies** 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 web site:

https://funwithalgorithms.herokuapp.com

Chapter 2

Literature Review

There are many AV systems that have been created, such as ANIMAL [**?**], HalVis [**?**], or BRIDGES [**?**] that have been shown to be more effective than traditional teaching methods. And yet these systems fail to reach mainstream computer science education.

Many of these AV systems look very old compared with modern applications, and they require to be downloaded and set up on a local machine before they can be used. According to Hundhauser and Douglas [**?**], there are a number of reasons for the failure of AV's to have widespread educational use. One of those reasons is that instructors find them too difficult to use, or too time-consuming to learn, taking up more of the instructor's time in preparation than they save during class.

These observations served as the inspiration for my project, where the main goal is to create a tool that is easy enough to use that it encourages instructors to incorporate it into their curricula because it will save them time (both in the classroom and out of it) and make their lectures more engaging to the students.

Another concern some instructors have shown, according to Hundhauser and Douglas, is over the effectiveness of an AV system compared with more traditional teaching methods (such as writing on the whiteboard). Teachers are hesitant to invest their time learning a new system if they are not certain it will actually help their students learn. But according to RoBling and Naps [**?**], there are eight pedagogical requirements that make an AV system an effective learning tool. I tried to design my own project in a way that would meet most of those requirements. The requirements 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.

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 for problems and 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

   (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 algorithm 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.

Of the three data structures listed above, the first one is absolutely necessary for representing the problem instance, and one of the other two is necessary for the solver, but having all three 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 dynamic typing (almost no typing), most of the data structures consisted of either arrays or objects.

4.4 The INSTANCE MAKER

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

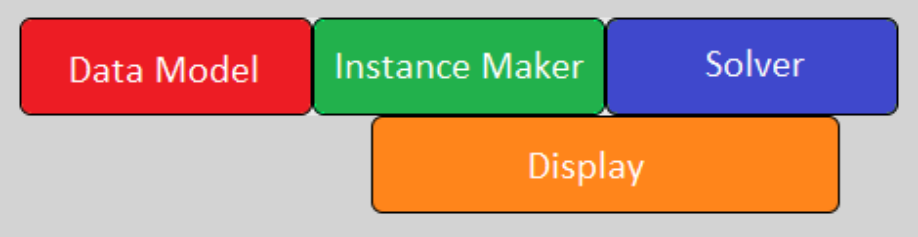
- Edit a piece of the Data Model.

  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 functionality 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 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 large would mean that the students will have a hard time understanding what's going on. An appropriate size for each step is one or two lines of the algorithm's pseudocode.

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.

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. For this reason I spent a large portion of my time validating the results of the Solver before I moved on to the next 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 you begin 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 size below $n = 7$ 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. [**?**]

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, is used as an introduction to Greedy Algorithms and makes for a good interactive lesson. During lecture, students make suggestions of possible greedy solutions to the problem (take the shortest interval first, or take the interval with the least number of overlaps first, etc.) while the instructor (usually) disproves their suggested solution. Oftentimes the instructor 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?" and 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, though I don't see any example being used with more than 50 intervals.

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 200, which means that a given problem can be divided into two subproblems no more than 7 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 instances 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 most of them had a decent understanding of the material. But more than a few of them were still confused about the problem (more than one student did not know that the problem asks for the total number of intervals). After seeing the AV perform, these students seemed to understand the problem much better.

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!"). I then used the AV to discuss with the class how an "always ahead" argument can be used to prove an algorithm's optimality.

My feeling after all these lectures was that students definitely appreciated the use of AV's in a classroom. Some of this could be attributed to the fact that it was a change of pace from their normal lectures, but I definitely think having the AV helped me as the lecturer to express my thoughts more clearly.

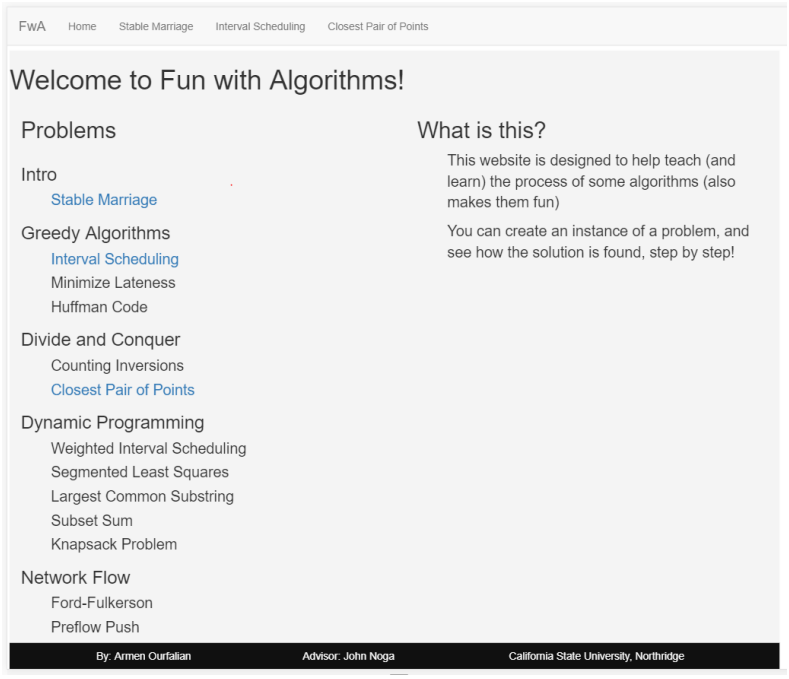
Chapter 7

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 a 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, so they 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 custom interactions that are specific to each algorithm. 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 in the live version of the app. I recommend reading this section with colors (as opposed to black an white), because some of the figures become hard to understand in black and white.

7.1 Home Page

Figure 7.1: Home Page



The app can be found at http://funwithalgorithms.herokuapp.com/ The home page is the first page that you will see *(Figure 7.1)*. It contains a list of topics, some in black others in blue. The topics are organized by Algorithm type (Greedy, Divide and Conquer, etc.) These are a subset of algorithms covered in an intermediate Algorithms class. The topics in black are a potential list of future topics to be covered. The topics that have already been created for the app are in blue, and they are links that lead to their corresponding page. There is also a link to each topic in the navigation bar. To

return to the home screen, you can click on the far-left link in the navigation bar (FwA).

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 are shared across multiple algorithms. Reusing components across various pages gives the website a more consistent feel overall and reduces the amount of training required to use the app. I packed the majority of the reusable functionalities into the secondary navigation bar (navbar) *(Figure 7.2)* to ensure they would be easy to find no matter what page the user found themselves on.

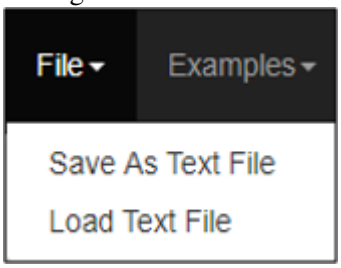
7.2.1 The Second Navbar

Figure 7.2: Second Navbar

The second navbar appears on every algorithm page, just below the regular navbar. It has a black background, which contrasts with the first navbar. While the first navbar allows navigation across the various pages of the app, the second navbar serves more as a menu bar with dropdown menus and other tools. The key features of the second navbar are discussed in the next few subsections.

7.2.2 File Menu

Figure 7.3: File Menu



A **File Menu** *(Figure 7.3)* with the options **Save as Text File** and **Load Text File**. These features should be 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 up to 200 rows, each containing two numbers $(startTime, finishTime)$. Despite these

differences, the interface for saving or loading instances is exactly the same for all problems within the entire project.

7.2.3 Saving and Loading

Figure 7.4: Stable Marriage Save and Load interfaces

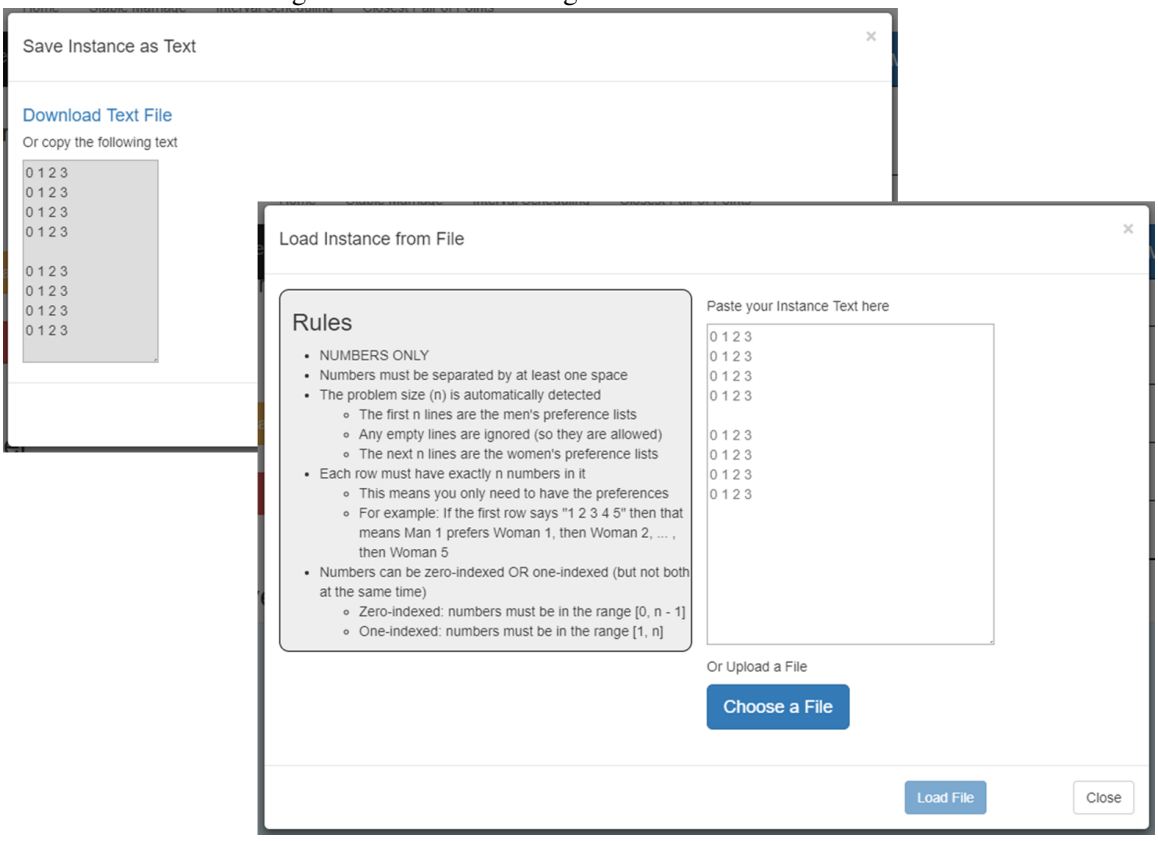


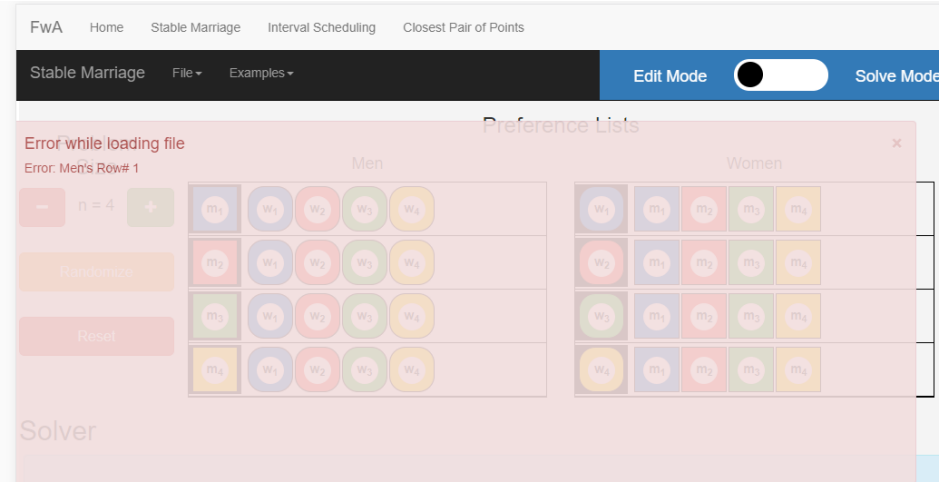
Figure 7.4 shows both the Save and Load interfaces for Stable Marriage. The Save interface is used when a user has made an instance of a problem with the app, and wants to save it as a text file on their computer. The Load interface is used when a user wants to load an instance from their computer onto the app (instead of manually creating the instance by interacting with the app).

The Save interface shows the text that represents the instance. A user can copy this text into a text file manually, or they can click the link titled “Download Text File” to download a text file directly. The text itself is meant to be a simple and straightforward representation of the instance.

The Load interface shows a set of rules that are unique to each problem. These rules are meant to guide the user on how to create files that will be accepted by the app. There is a large text box where the user can paste (or even manually type) the instance. The placeholder text (the text that is in a lighter color and usually gives hints about what kind of information should be typed into the

text box) of the text box is the same text that the Save interface shows, this is another hint that is given to the user about what the text file should look like. There is also a button labelled “Choose a File” that allows the user to upload a text file directly.

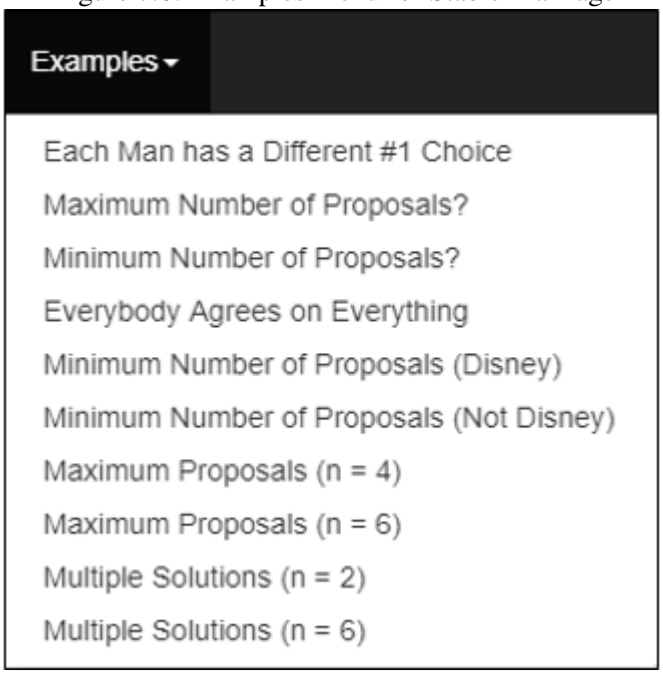
Figure 7.5: Stable Marriage Error Message



Upon clicking the button labelled “Load File” If there are any problems with the file, the app will display a message in red specifying which line contained the error, as seen in *Figure 7.5*. If there are errors in the file, the load may or may not fail completely, depending on the algorithm: Both Stable Marriage and Interval Scheduling only load a file if there are no errors, whereas Closest Pair of Points will still load the file, but ignores any lines that had errors in them.

7.2.4 Examples Menu

Figure 7.6: Examples Menu for Stable Marriage



Next to the **File Menu** is an **Examples Menu** *(Figure 7.6)* that contains a list of premade instances for each problem. These instances are interesting cases that are often brought up in lecture. When a user selects one of the items from this menu, that instance is automatically loaded into the current page.

7.2.5 Edit Mode and Solve Mode

Figure 7.7: Edit Mode

Figure 7.8: Solve Mode

On the far-right side of the second navbar is a switch *(Figures 7.7 and 7.8)* 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 algorithms: the problem instance can only be changed (**edited**) when the page is on Edit Mode, and algorithm can only be performed (**solved**) 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 in the worst case, and be confusing in the best case. 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.

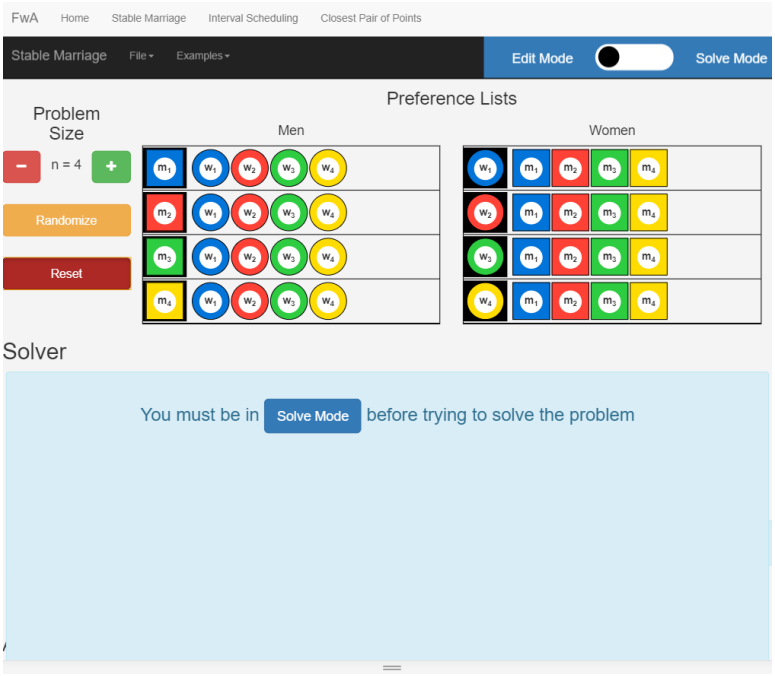
7.2.6 Automator

Figure 7.9: Automator Component

Included in some pages is an **Automator** component. *Figure 7.9* shows what this component looks like. The **Automator** will run the algorithm automatically at set time intervals. The user can increase or decrease the speed of the Automator within some preset limiits. They can also pause the Automator as it is running if they wish to continue manually from any point. The Automator will be disabled if the page is in **Edit Mode** or if the algorithm has already completed.

7.3 Stable Marriage

Figure 7.10: Stable Marriage in Edit Mode



Upon first visiting the Stable Marriage page, the user sees a page similar to *Figure 7.10*. By default, the page is loaded in **Edit Mode**, with a default instance of size $n = 4$.

In the center of the page are the two preference lists, each with 4 rows (the number of rows is always equal to the Problem Size). Each row has five (Problem Size $+1$) colorful boxes (squares and circles are both referred to as boxes)

The first box in each row has a thick border, and represents a person (a man or a woman), and the rest of the boxes represent that person's rankings of the members of the opposite sex. The further left a person is, the higher they are ranked.

Figure 7.11: Example Preference Row

As an example: in *Figure 7.11* we can see that m_2 ranks w_4 as his most preferred choice, then w_1, then w_2, and w_3 is his least preferred choice.

The colors correspond to numbers (1 is blue, 2 is red, etc.). The purpose of the colors is to create an easier way for students to differentiate the various people. Note that the fact that the colors are the same for men and women holds no significance (ie there is no inherent relationship between m_1 and w_1, they just both happen to be blue). The boxes for men are squares and the boxes for women are circles. This is to allow students to easily differentiate between each gender.

7.3.1 Instance Maker

On the left-hand side of the page are four buttons:

- $-$ **and** $+$**:** These buttons decrease or increase the problem size, respectively
- **Randomize:** This button will shuffle the preference lists to create a random instance, without changing the problem size
- **Reset:** This button will reset the preference lists to their default position, which is in increasing numerical order

Aside from the buttons, the main user interaction with the INSTANCE MAKER happens directly on the preference lists. To edit the preference lists, the user has to reorder the preferences of each

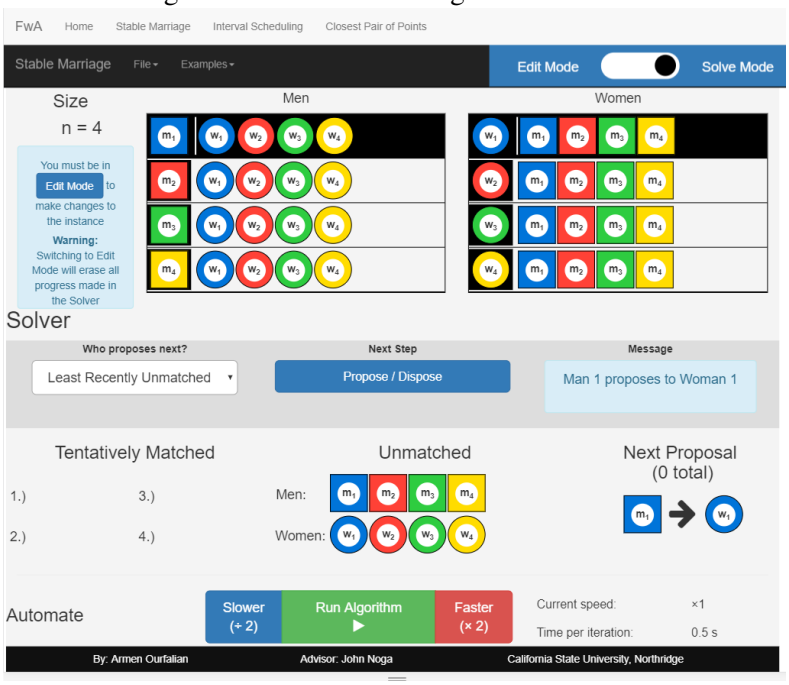
person. This can be done by clicking on any box inside of a preference row (except the box on the far-left with the thick border) and dragging that box horizontally left and right. This will swap that box with its neighbor.

On a touchscreen device (such as an iPad), this dragging function does not work. Instead the user must touch on one box (the box will be highlighted in dark grey) and then touch another box in the same row to swap them.

In both cases, when the boxes are swapped, they will both rotate 360^o in place to indicate that they have been changed.

7.3.2 Solver

Figure 7.12: Stable Marriage in Solve Mode



When the page is changed to **Solve Mode**, the user will see a page similar to *Figure 7.12*. In this mode, the buttons of the INSTANCE MAKER have been removed, but a new set of controls has been made available just below the preference lists:

- **Who Proposes Next:** This is a dropdown menu to determine which man is proposes next. The available options are:
  - **Least Recently Unmatched:** The man that has gone longest without proposing is the next one to propose. At the beginning when no men have proposed yet, the men propose in increasing order of their number (m_1 goes first, then m_2, and so on).

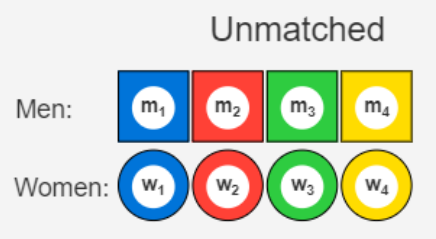
- **Most Recently Unmatched:** The opposite of the previous option, the man who has most recently proposed goes next. At the beginning, when no men have proposed yet, the men propose in decreasing order of their number (m_4 goes first, then m_3 and so on)
- **Choose By Clicking:** When this option is selected, the user decides which man proposes next by clicking on the man's box in the Unmatched component. The first click makes the man propose, and the second click makes the woman respond. After the second click, the user may click on another man (or they may click on the same man again).

  Note that if this option is selected, the **Automator** will be disabled.
- **Random:** In this case, the man who proposes next will be chosen randomly.

- **Next Step:** This button is clicked to perform the next step of the algorithm
- **Message:** The message is a type of feedback that either gives instructions to the user (such as "Click the blue button to perform the next step of the algorithm") or describes which step of the algorithm is being perform (such as "Man 1 proposes to Woman 1");

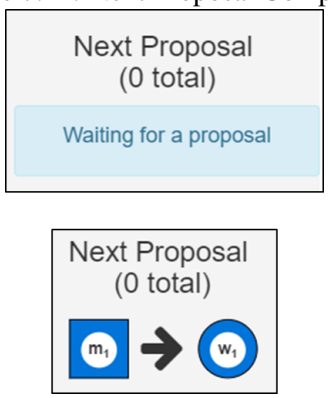
Below the controls are three components that provide useful information as the algorithm is being performed.

Figure 7.13: Unmatched Component



In the middle is the list of people who are currently **Unmatched**. This list is useful for discussing which man should propose next, and if a woman is unmatched, then she won't reject a proposal from any man. Also, if the "Choose By Clicking" option is selected from the **Who Proposes Next**, then the user must click on the men's boxes in the **Unmatched** component.

Figure 7.14: Next Proposal Component



To the right is the **Next Proposal** component, which is a graphical representation of a proposal. *Figure 7.14* shows some examples of the **Next Proposal** component

Figure 7.15: Tentatively Matched Component

Figure 7.15 shows some examples of the **Tentatively Matched** component. This component is the complement of the **Unmatched component** in that it only shows the people who are currently matched.

Although the three components described in this section convey a lot of information, the user needs to constantly look at the preference lists to determine what the algorithm will do next. 1 Since it can be hard to focus on multiple sources of information back and forth, the preference lists also communicate much of the same information as these sections (albeit more symbolically)

Figure 7.16: What the Preference Lists look like in the middle of the Algorithm being performed

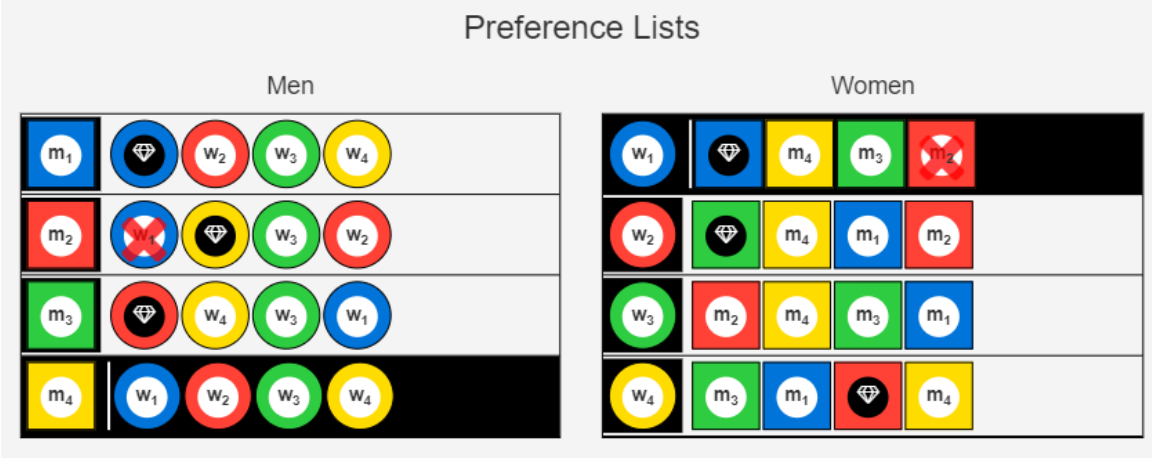


Figure 7.16 is an example of what the preference lists may look like in the middle of the algorithm. If a person is part of a tentatively matched couple, their partner's number is replaced with a white diamond (in this case m_1 and w_1 are tentatively matched, m_2 and w_4 are matched, and m_3 and w_2 are matched). If a man has been rejected by a woman, that woman's box is crossed out with a red X. And if a man is proposing to a woman, their preference rows are highlighted in black.

7.4 Interval Scheduling

Figure 7.17: Interval Scheduling in Edit Mode

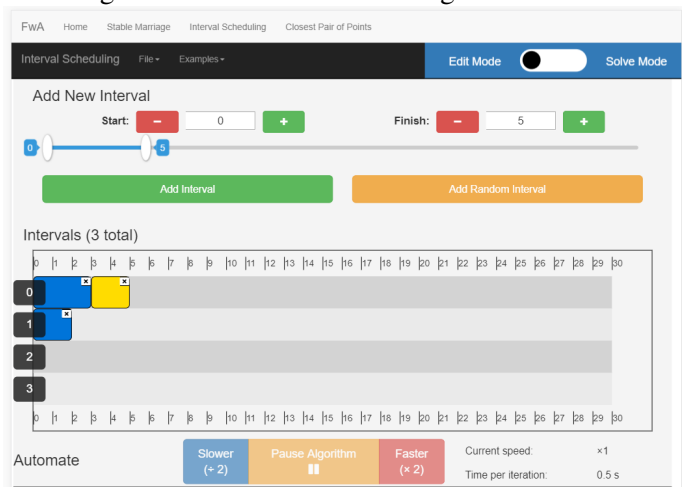


Figure 7.17 shows the Interval Scheduling page upon first visiting it. The top portion of the page represents the INSTANCE MAKER and the bottom portion represents the DISPLAY.

7.4.1 Instance Maker

The INSTANCE MAKER for Interval Scheduling consists of an interface to add and delete intervals:

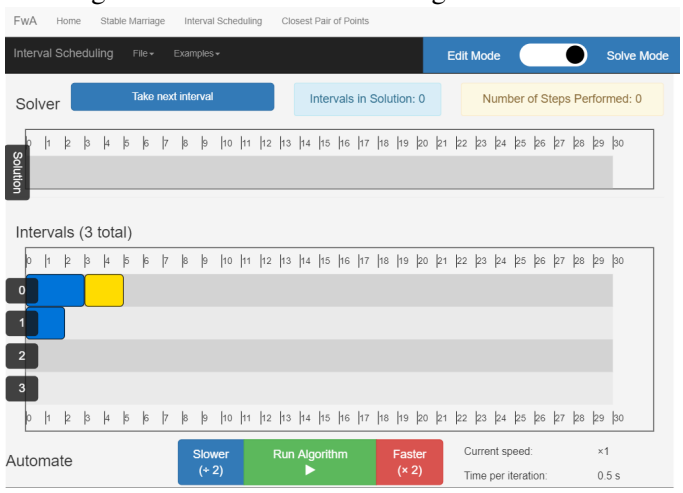
To delete an interval, the user must click on the small X in the top-right corner of the interval in the DISPLAY.

To add an interval $(startTime, finishTime)$, the user can use the $-$ and $+$ buttons, the dual sliders, or even type into the text boxes for **Start** and **Finish**. All three of these methods are equivalent, and the user may even mix-and-match between the three methods. Also, the "Add Random Interval" button will add a new interval with random start and finish times.

When the user adds an interval, it gets added to the least-occupied row in the display, as long as it doesn't overlap with any other intervals already in that row. If no such rows exist, then a new row is created and the interval is added to that row.

7.4.2 Solver

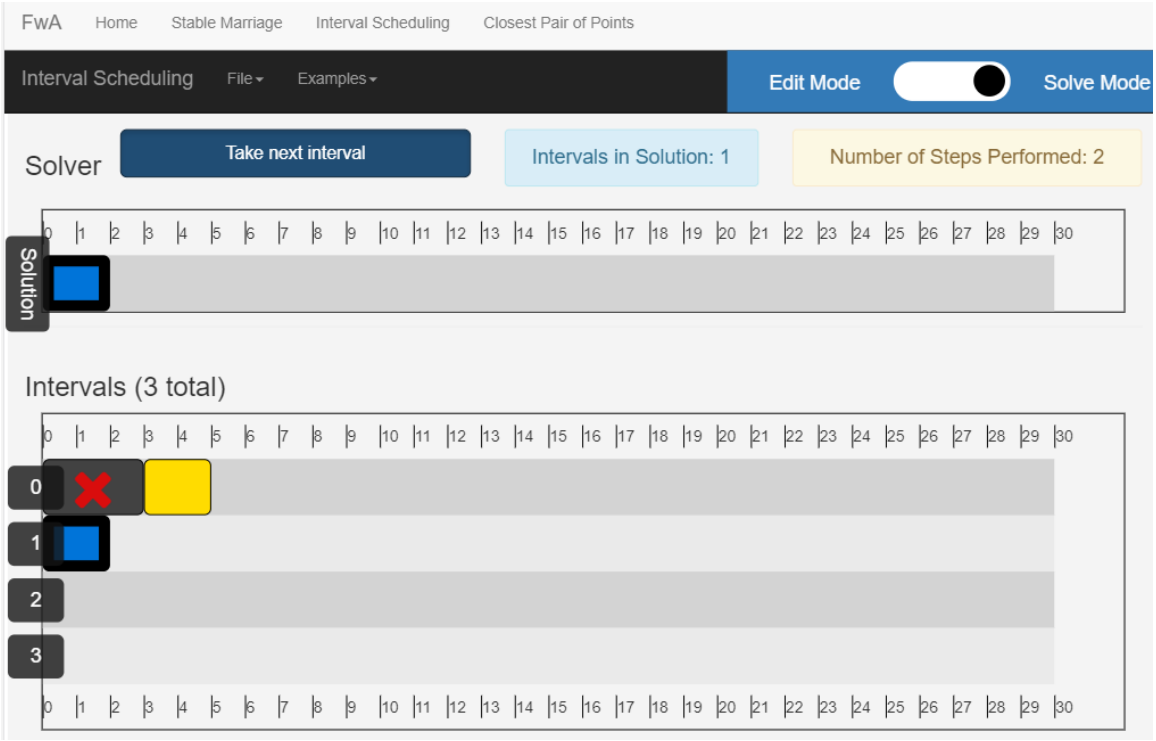
Figure 7.18: Interval Scheduling in Solve Mode



Switching the page to **Solve Mode** will remove the controls to add and delete intervals, and bring out a new set of controls (which consist of a single button). The blue button alternates between saying "Take Next Interval" and "Remove any intervals that overlap". The yellow counter on the far right will increase by 1 each time the blue button is clicked. In the middle, the blue counter keeps track of how many intervals are in the solution set. This is the number that an instructor needs to point to in front of a confused class and explain that it's the number of intervals that is important, and not some other property of the intervals in the solution set. The row labeled "Solution" shows which intervals have been added to the solution.

As the algorithms is being performed, the interval with the earliest finish time gets added to the Solution row, and any intervals that overlap with it are marked with a red X, as seen in *Figure 7.19*

Figure 7.19: Interval Scheduling Algorithm



7.5 Closest Pair of Points

Figure 7.20: Closest Pair of Points in Edit Mode

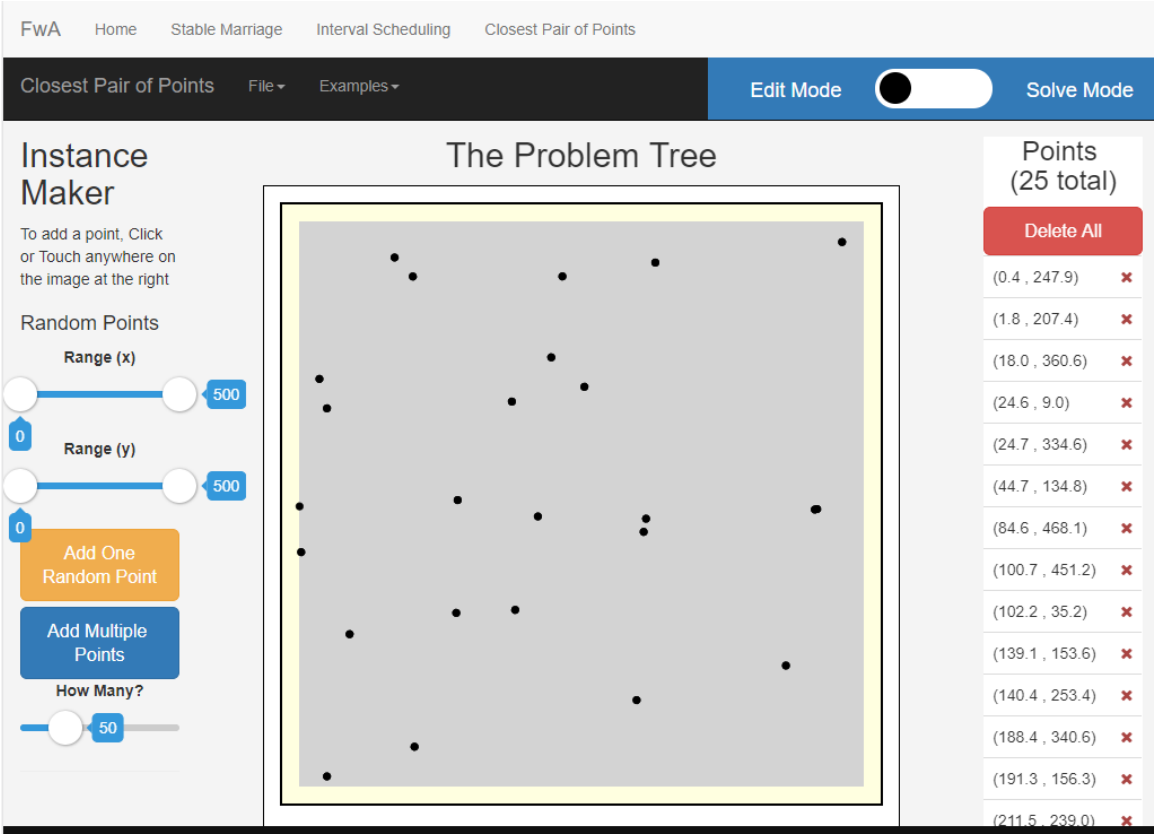


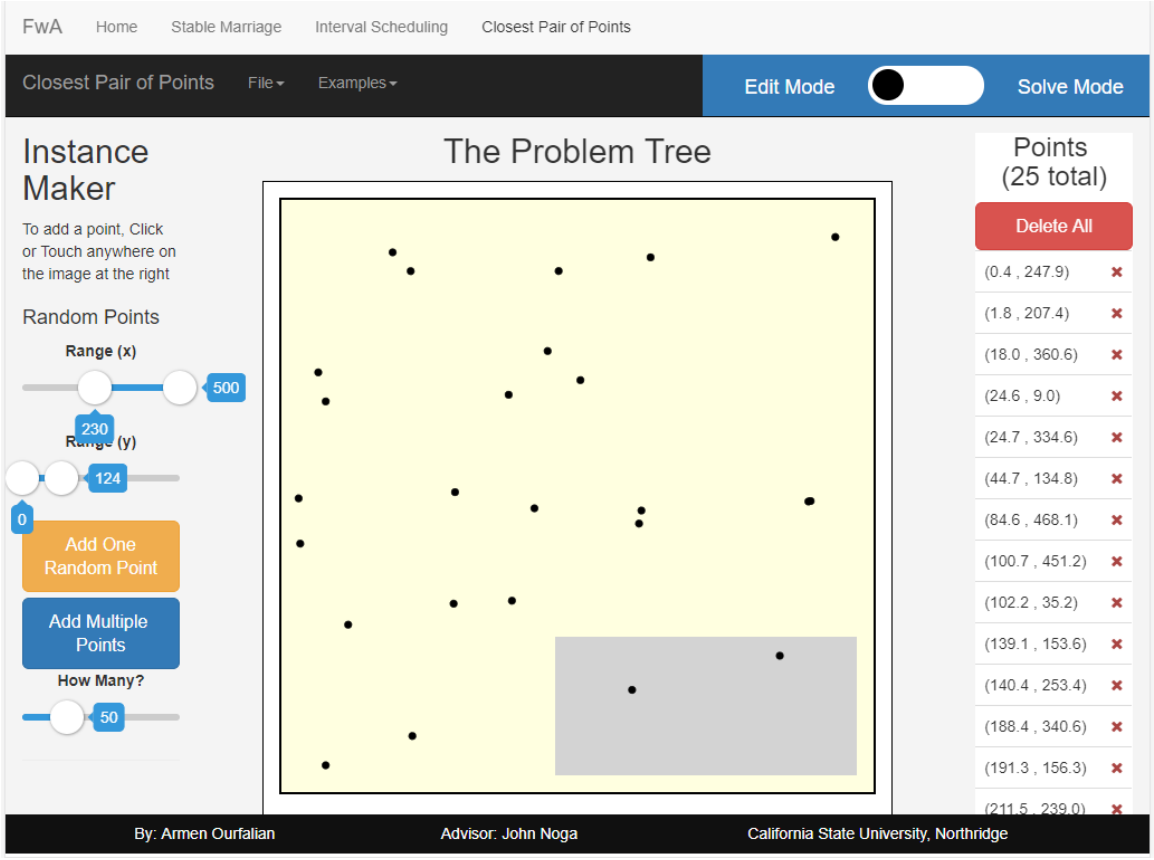
Figure 7.20 shows the Closest Pair of Points page upon first visiting it. In the center of the page is the DISPLAY: a square grid with 25 random points in it. On the left is the INSTANCE MAKER and on the right is a table of coordinates for all the points in the grid.

7.5.1 Instance Maker

The INSTANCE MAKER consists of an interface to add points to the grid and delete points from the grid. Too add a point, the simplest way is to click on the grid, and a point will be created at that location. An alternative method of adding points is by clicking one of the two available buttons. These buttons add a random point to the grid. The number of points added by the "Add Multiple Points" button is determined by the slider below it.

The user can specify the area where they want the random points to be in by changing the two sliders above the points. When these sliders are moved, the gray box inside the grid will change its size and location to reflect the values in the sliders (see *Figure 7.21*)

Figure 7.21: Changing the Range for Random Points



To delete points, the user can click on the red X next to the coordinates of the points they wan to delete. For the sake of clarity, when they move the mouse (or tap on) the coordinates, the point will change its color to red to identify itself. Alternatively, if the user wishes to delete all the points at once, they may do so by clicking the “Delete All” button above the list of coordinates.

7.5.2 Solver

Figure 7.22: Closest Pair of Points in Solve Mode

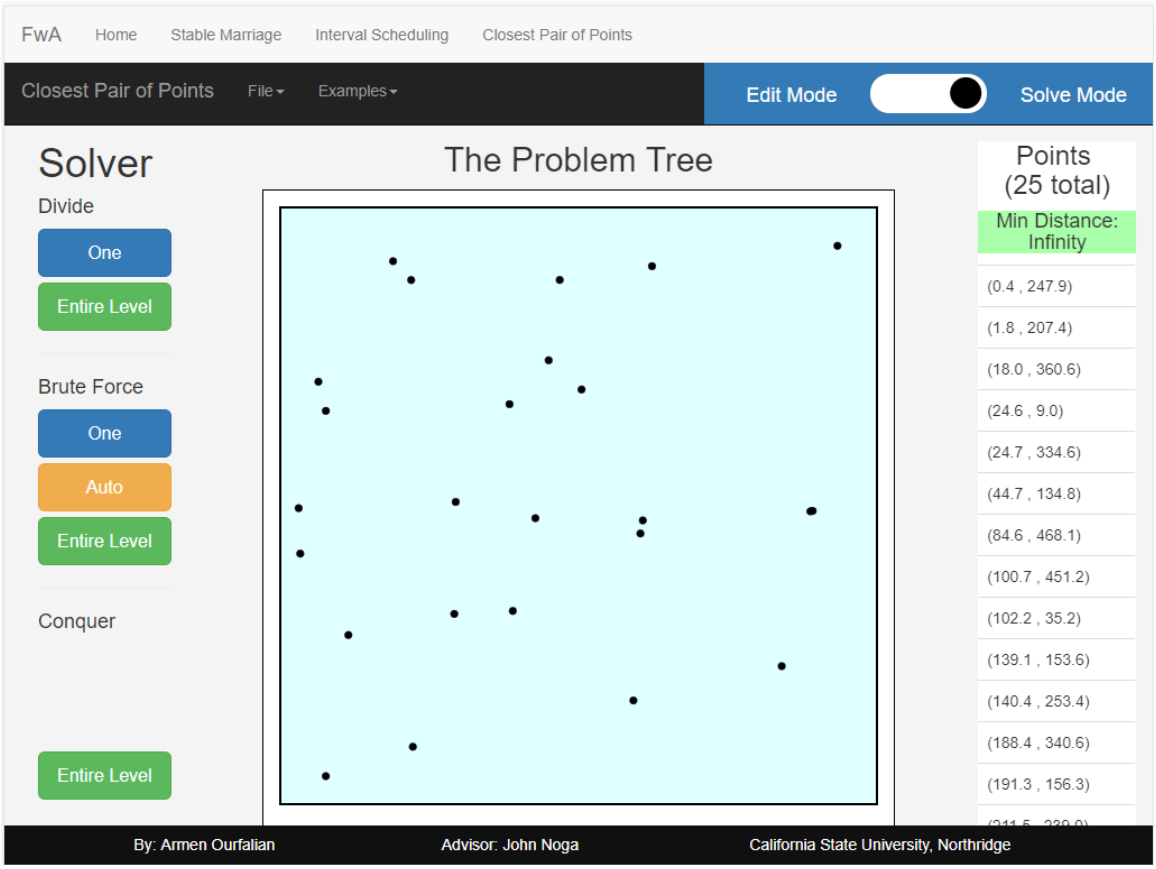


Figure 7.22 shows the Closest Pair of Points page in Solve mode. The buttons and sliders to add or delete points have disappeared, and in their place are a new set of buttons on the left side. These buttons provide three functions: Divide, Brute Force, and Conquer. Furthermore, each function comes in up to three varieties: **One** (blue), **Auto** (yellow), and **Entire Level** (green). Some of these buttons disappear if they are disabled, and reappear when they may be pressed. Also, the list of coordinates now also specifies what the current minimum distance is for any two points on the grid. When Solve Mode is first entered, this distance is Infinity (because no distances have been checked yet).

7.5.2.1 Divide

Figure 7.23: Dividing a Grid

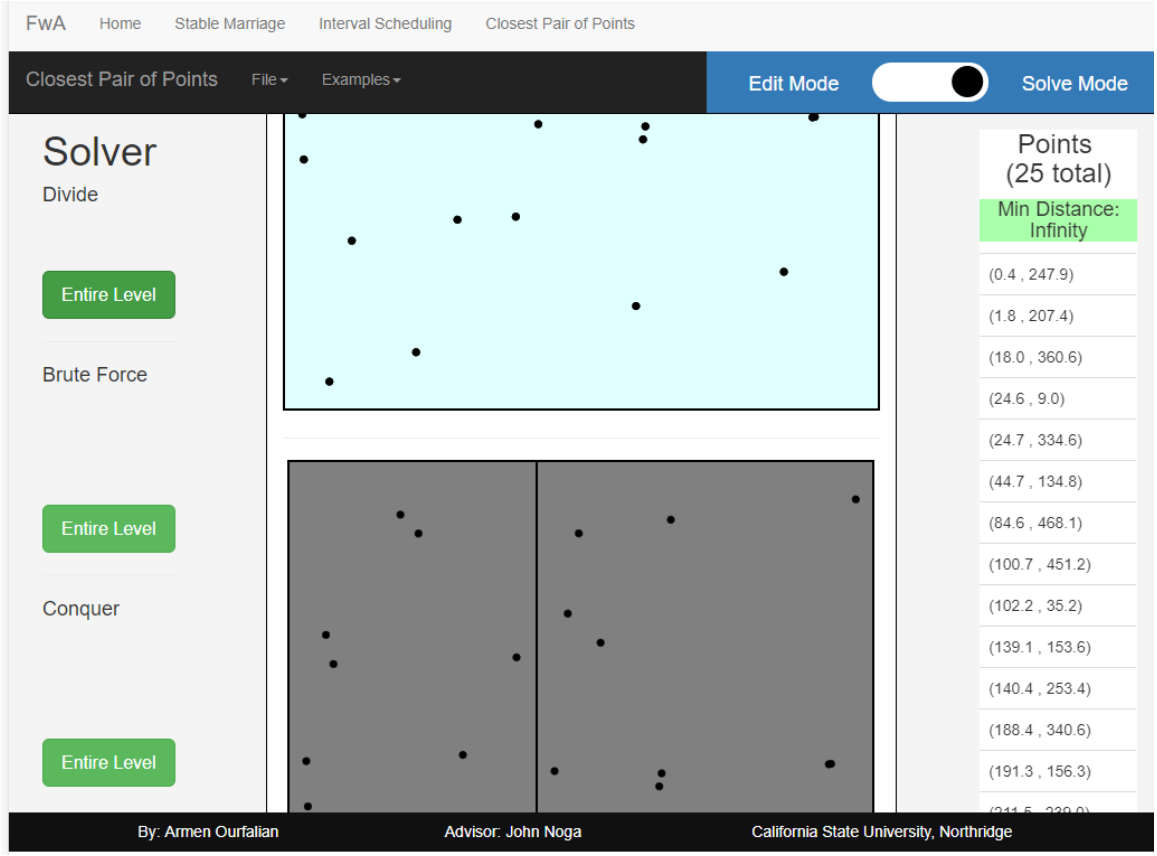


Figure 7.23 shows what happens after either of the "Divide" buttons are clicked. The grid separates into two smaller grids. Each smaller grid has half the number of points as the original grid, either the left half or the right half. Now that there is more than one grid on the page, the user can select a grid by clicking on it. The selected grid will be a light blue color whereas the other grids will be a dark gray. Clicking a button will only affect the selected grid, with the exception of the "Entire Level" buttons. "Divide Entire Level" will perform the Divide function on every single grid that is at the same horizontal level as the selected grid. A grid can only be divided if it has more than three points.

7.5.2.2 Brute Force

Figure 7.24: Brute Force

When a grid has three or fewer points in it, the only function that a user can perform on it is Brute Force. To Brute Force a problem means to try every possible result and take the best one. In this case, the Brute Force function will check the distance between every pair of points, and take the minimum of those.

Clicking the "Brute Force One" button will turn one of the points in the grid blue (the leftmost point). Each successive click will turn the next point in the list yellow, signifying the distance between that point and the blue point has been checked. When all points have been turned yellow, the blue point becomes orange, and the first yellow point becomes the new blue point (See *Figure 7.24*. This button serves as an example of why brute forcing is not a good option for most problems; any time the number of points is larger than 9 or 10, this process gets very tiresome, tedious, and boring.

The "Auto" option for Brute Force will perform the entire brute force operation in one click, and the "Entire Level" option will perform it on every grid at the same horizontal level as the selected grid. Although these two buttons go against good AV practices, their purpose is to speed up a process that

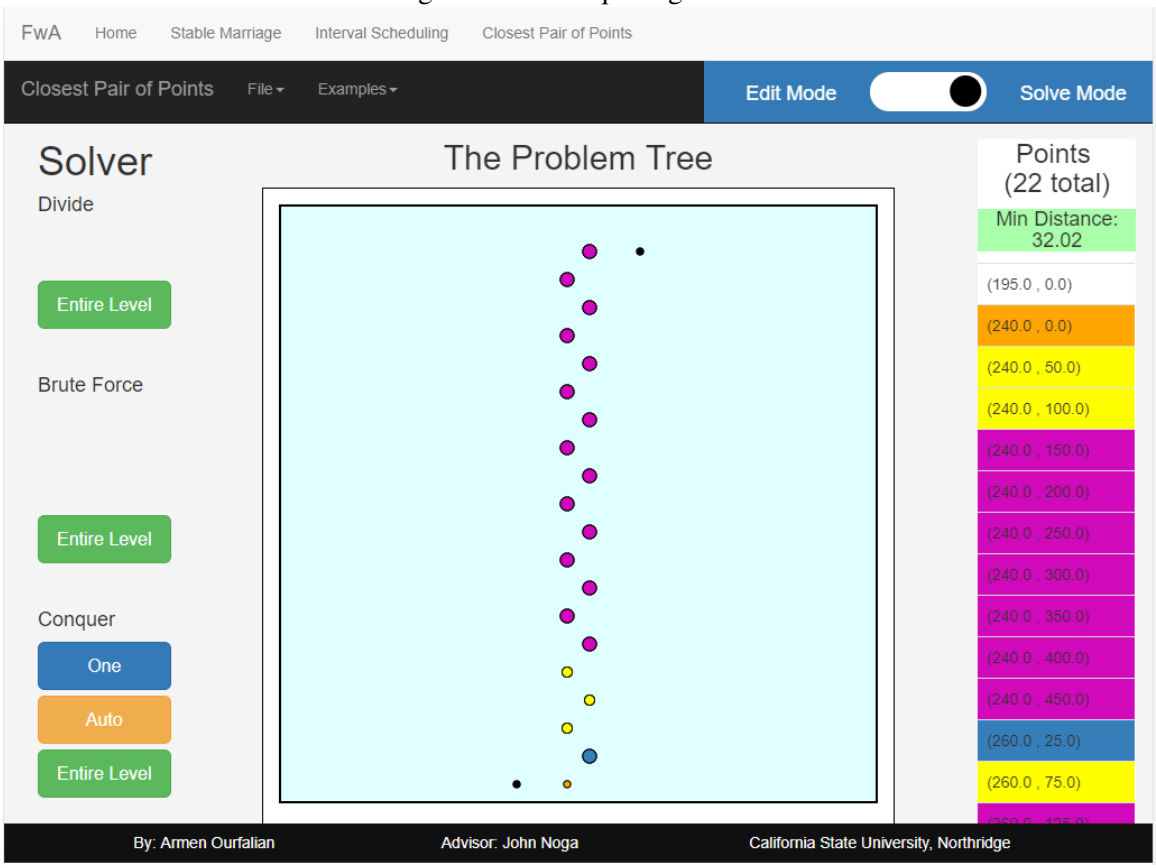
could take up a lot of time. By using these buttons a lecturer can skip ahead to the more interesting parts of the discussion

7.5.2.3 Conquer

Conquering is the complement to Dividing. Two smaller grids that have both already been solved are combined back into the original, larger grid. The new minimum distances is the smaller of their minimum distances. However, there is a strip in the middle of the two grids where the distance between the points has not been checked. The conquer function serves to check the points that are in this strip to see if any two points are closer than the minimum distance.

The conquer function is similar to the Brute Force function in many ways: they both iterate through a set of points, changing the colors of points. However, the conquer function has a much smaller time complexity because each point is being checked with a finite number of other points.

Figure 7.25: Conquering a Grid



Clicking on the “Conquer One” button will turn all of the points inside of the strip to a purple color. Then, each successive click will behave exactly like the Brute Force function. The bottom-most point in the strip turns blue, and its distance is checked with the next 8 points (that are also in the strip). Afterwards, this point is turned orange, the next point is turned blue, and the process is repeated (see *Figure 7.25*).

Since each point is being checked with only the next 8, this process is much less time-consuming than the brute force function. Yet it can still take a very long time to complete. For that reason, the “Conquer Auto” button will automate this process and only show the results. Similarly, the “Conquer Entire Level” button will perform the Conquer Auto function on each grid at the same horizontal level as the selected grid. Note that a grid can only be conquered if it has been divided and if both of its sub-grids have been fully solved (either through brute force or through conquering).

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 [**?**]. 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

```
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` 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 [**?**]. 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 [**?**]. I chose Vue over Angular because it is more lightweight than Angular [**?**].

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** [**?**] 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

Algorithm Visualization is an educationally effective method of teaching how algorithms work. But educators tend to stick with the method they're used to: drawing diagrams on the whiteboard. Using AV's would save in-class time for discussion, produces higher-quality diagrams, and is reproducible outside of the classroom.

The goal of this project was to create an Algorithm Visualization tool to be used by lecturers to simplify the task of drawing complex diagrams. The tool is front-end web application that can be accessed by going to https://www.funwithalgorithms.com/

I chose three algorithms taught in CSUN's Comp 482 class: Stable Marriage, Interval Scheduling, and Closest Pair of Points. Each AV consists of three main components: an INSTANCE MAKER, a DISPLAY, and a SOLVER. The INSTANCE MAKER allows users to create instances of a given problem by interacting with the page. The DISPLAY shows diagrams depicting the various properties of the instance. The SOLVER runs the algorithm step by step, modifying the DISPLAY in the process.

One of the biggest challenges of this project was separating responsibilities among the various components in an organized manner. Vue.js was used to keep isolate responsibilities into components, and Vuex was used to create an organized way for the components to communicate with each other.

There are plenty of opportunities to build upon this project, the most obvious of which is to add more AV's for different algorithms (the home page of the app lists quite a few good candidates). In an ideal scenario, the app would have an AV for each major algorithm covered in an entire course.

Aside from adding more AV, this project can be built-upon by adding more features to each AV itself for either functionality or to create a more fluid user interface. For example, the Stable Marriage AV could have a section where the user (after creating the instance) checks to see whether a matching is stable. The Interval Scheduling could allow the user to reorder the intervals into different rows. The Closest Pair of Points could allow the user to zoom in or out of the problem tree, and have a mini map to show them which part of the tree they are looking at.

Appendix A

Glossary

- **Algorithm** $\rightarrow$ A series of steps to solve a problem. An algorithm must always terminate (it cannot run forever).
- **Algorithm Visualization (AV)** $\rightarrow$ Software that displays diagrams, animations, and other interactive tools to facilitate the learning of an algorithm. An AV is more interactive than an animation or a video.
- **App / Page / Program / Tool / Web Page** $\rightarrow$ I use these terms interchangeably to refer to the program that I have created for this project.
- **Front-End** $\rightarrow$ A front-end web application runs on the client's machine instead of a server.
- **Instructor / Lecturer / Professor /User** $\rightarrow$ I use these terms interchangeably to refer to the person that is using the app. This is not to say that student (or other person with no relation to the class) cannot use the app (it is available on the Internet for all to enjoy!)
- **Problem** $\rightarrow$ The word problem is used in this document in the context of a *Mathematical Problem*. Problems are different than algorithms, but students often use the two words interchangeably.

  As an example: Interval Scheduling is a **Problem**, Earliest-Finishing-Time is an **Algorithm** that can be used to solve the Problem.
- **Problem Instance** $\rightarrow$ A specific example of a problem with an input. Homework questions and test questions often describe a problem instance and ask the student to find the solution to that instance.