Detecting and Predicting Clusters of Evolving Binary Stars

DESIGN DOCUMENT

sdmay21-30
Goce Trajcevski (Client & Advisor)
Adam Corpstein (Report Manager),
Becker Mathie (Chief Engineer),
Ethan Vander Wiel (Test Engineer),
Joel Holm (Facilitator),
Philip Payne (Quality Assurance),
Willis Knox (Scribe)
sdmay21-30@iastate.edu
Team Website
http://sdmay21-30.sd.ece.iastate.edu/
Revised: 10/25/2020 -- 1.2

Executive Summary

Development Standards & Practices Used

- Agile development
- Project Roles
    - Scribe
    - Facilitator
    - Chief Engineer
    - Test Engineer
    - Report Manager
    - Quality Assurance
- Utilization of Azure Devops for project status and task management
- Confluence for lasting information and project record keeping
- Slack for temporary information and general communication
- Version control using Git with Branch-Review-Merge workflow

Summary of Requirements

- The product shall be a web based system accessed through an internet connection
- The product shall allow the user to enter initial conditions for the binary system and constraints on that system such as:
    - parameters of interest for clustering
    - distance functions across different attributes, with weighted combinations
    - An interval of interest, expressed as an explicit interval, or an event based interval
- When given initial conditions and constraints for a binary system, the product shall predict whether the system will merge a particular cluster of other known binaries during its evolution
- The product will maintain its own database and eventually allow for the user to input their own database and datasets.

Applicable Courses from Iowa State University Curriculum

- Com S 227/228: Introduction to OOP and Data Structures
- Com S 311: Introduction to the Design and Analysis of Algorithms
- Com S 363: Introduction to Database Management Systems
- Engl 314/309: Technical Communication/Proposal and Report Writing
- S E/Com S 309: Software Development Practices
- S E 329: Software Project Management
- S E 339: Software Architecture and Design
- S E 409: Software Requirements Engineering

New Skills/Knowledge acquired that was not taught in courses

- Angular
- PostgreSQL
- Python/Django
- Clustering Algorithms (K-Means, DBScan)
- Figma

Table of Contents

List of Figures and Tables

1 Introduction

1.1 Acknowledgement

The team would like to thank the Electrical and Computer Engineering department at Iowa State University for the resources and education we have received throughout our time spent at the university. We would also like to thank Professor Goce Trajcevski for having weekly meetings with the team to provide feedback and advice.

1.2 Problem and Project Statement

Problem Statement

As astrophysicists continue to collect data on binary star systems in our galaxy, new possibilities for research emerge. Large databases exist full of binary star data. This includes attributes such as red shift, mass ratio, hydrogen ratio, etc. Astrophysicists would like the ability to organize this data into clusters so that any two systems with similar attributes are in the same cluster and dissimilar systems are in different clusters. Astrophysicists have also computed their own data for future states of binary systems. This would add the dimension of time to the data (since the lifetime of a binary star could be millions of years, simulating future states is necessary). By organizing clusters by time, further analysis can be taken as binary stars mature.

Solution Approach

The driving force for this project is to organize data on binary star systems to aid astrophysicists in their research. To solve the problem of organizing astronomical data, we will apply pre-existing clustering algorithms to the datasets. The results of different clustering algorithms will be available on a web server. From the webpage astrophysicists will select their desired attributes and cluster weight functions to generate binary star clusters. We hope to create and maintain a web server for astrophysicists to use. Also we would like to receive feedback from them on what we can do to improve the process of generating clusters.

1.3 Operational Environment

The application will be entirely web-based, so its operational environment is the internet itself. The application will need a reliable connection to the internet for long periods of time to allow for the simulations and calculations to complete.

1.4 Requirements

Functional Requirements

- A web based application with a user interface that accepts different types of data input by users for tracking the stellar evolution of binary stars. Types of data will include but not be limited to:
    - Luminosity
    - Mass
    - Relative distance between stars
    - Helium concentration
- Users will be able to configure which parameters are of interest for clustering
- Users will be able to apply specific weights to their selected parameters that determine the level of importance of each parameter when clustering
- Users will have the ability to specify specific intervals of interest for clusters
    - This interval could either be an explicit time range or a more general event based range
- Given the user specified information, a remote server will access binary stars from a database and then compute and predict how different binary stars will cluster during their evolution

Non-Functional Requirements

- Cluster requests should not leave the user waiting for longer than 30 seconds.
- The server is equipped to handle less than 1000 simultaneous users.
- Database setup should be scalable to allow new data entries as well as the possible selection from alternate stellar databases.
- Server should be available 24/7 with weekly server resets

Environmental Requirements

- The application will be required to work only in areas with an internet connection. This connection will need to be constant and stable to allow users to access the application at any time
- The remote server will need to have the capability to connect to multiple databases at once

Economic Requirements

- Laptops or personal computers will be needed to interact with the user interface of the application

1.5 Intended Users and Uses

The end users for this project will be astrophysicists who have a need to search for similarities between binary star systems. These astrophysicists are used to working with astronomical data provided by Sloan Digital Sky Survey or the Gaia Archive. They may be familiar with the query language SQL as it’s used in these websites.

1.6 Assumptions and Limitations

We have come up with the following assumptions and limitations to refine the scope of the problem statement.

Assumptions

The user can connect to the internet

We are assuming that our intended users will have access to the internet. Without an internet connection, a user will not have the capability to access the application, and therefore they will be unable to use it.

The user has a basic understanding of clustering techniques and weighting functions

We are assuming that users have enough knowledge about the provided clustering techniques to understand how it affects the output data. Related to this, the user will need to provide their own weights in regards to each attribute. This also affects the output data, so users will need a solid grasp of how to write their own weighting functions.

Limitations

The application should not overuse the client's hardware

All algorithmic work such as data normalization and star clustering will need to be completed on the server and not on the user's device. This is to ensure the application stays responsive and easy to use.

Types of clustering data is limited to what is in the pre-existing databases

The different types of data the users will be able to select for clustering will be limited to what information is stored in the databases the application will be accessing. Since we are connecting to well established, pre-existing databases, the users must restrict their clustering queries to types of data that these databases currently contain.

1.7 Expected End Product and Deliverables

Website that provides a UI to enter custom cluster queries

The main feature of this product is the website where astrophysicists will input queries to generate clusters of binary star data. The custom cluster query interface will include the following elements: selection of attributes, input of weight functions, selection of clustering techniques, and selection of the database. The attribute selections allows the user to specify which data types will be regarded in the clustering. Weight functions allow users to specify how heavily each attribute will affect the clustering. Selecting a clustering technique will change how groups of clusters are formed. Lastly, a database must be selected, from which binary star data is pulled. To be delivered April 15th, 2021.

Graphical representation of the clustering data after query

After a clustering query is processed each binary star will be assigned to a cluster. To gain insight into how clusters were created, a graphical representation of the relationship between attributes can be generated. Users will be able to select which two attributes to compare; this will generate a 2D graph showing the clusters based on the two attributes. To be delivered April 15th, 2021.

Implementation of clustering algorithms: K-means clustering and DBScan clustering

There are two clustering algorithms that will be implemented: K-means clustering and DBScan clustering. These two algorithms have their advantages. K-means clustering is an iterative algorithm that refines the cluster formations over time. The main advantage of this approach is its efficient execution time with large datasets. DBScan is a density based clustering technique. Unlike K-means, DBScan does not require a predefined number of clusters. It also is able to create flexible cluster shapes and neglects the impact of outliers. To be delivered April 15th, 2021.

User guide page

To aid users who are unfamiliar with binary star data, clustering techniques, or the process of creating a cluster query, a user guide page will be created. This page will include directions on how to interface with the UI to generate desired clustering output. To be delivered April 15th, 2021.

2 Project Plan

2.1 Task Decomposition

Iteration 1 (9/10 - 10/10):

- Familiarize with clustering algorithms
    - Kmeans & DBScan
- Consider possible software tools and platforms

Iteration 2 (10/10 - 10/25)::

- Finalize attributes and distance functions
    - algorithm and distance function research

Iteration 3 (10/25 - 11/02):

- Finalize selection of development platforms and architecture design
- Preliminary UI design

Iteration 4 (11/02/ - 11/09):

- Finalize selection of algorithm solutions
- Devise use cases and test cases
    - Use case diagram
    - Test planning

- Finalize UI functionality

Final deliverable (11/09 - 11/15):

- Prepare design presentation
- Finalize and submit design document

2.2 Risks And Risk Management/Mitigation

Below are potential risks for this project along with their respective probabilities and mitigation strategies:

- Risk: The team is unable to finish our deliverables in accordance with our timeline.
  - Probability of occurrence: 0.10
- Risk: The client requests that an additional clustering algorithm be added to the project.
  - Probability of occurrence: 0.25
- Risk: The client wants to connect to a database that was not previously provided and has unknown properties.
  - Probability of occurrence: 0.60
  - Mitigation plan:

    To handle the occurrence of this risk our team will design our application to accept various databases. We will additionally implement methods to check the compatibility of the database with our chosen clustering algorithm.
- Risk: The client requests that the cluster data be displayed with a different method that may not be compatible with how our cluster data is returned.
  - Probability of occurrence: 0.50
  - Mitigation plan:

    Similar to the mitigation plan for the previous risk, to limit the possibility of this risk our team will design the application such that it can handle displaying the data in various ways. We will also use design methods that allow for easy changes to the code that handles displaying cluster data without requiring changes to the other parts of the application.
- Risk: A team member contracts COVID-19 and is either unable to work or is only able to do limited work for a period of time.
  - Probability of occurrence: 0.50

- Mitigation plan:

  To limit the possibility of this risk the team is following recommendations for staying safe during the current pandemic. We currently hold all meetings virtually to eliminate the risk of transmission between team members. If a team member contracts COVID-19, the remaining team members will work together to carry out the sick member's responsibilities while they recover.

2.3 Project Proposed Milestones, Metrics, and Evaluation Criteria

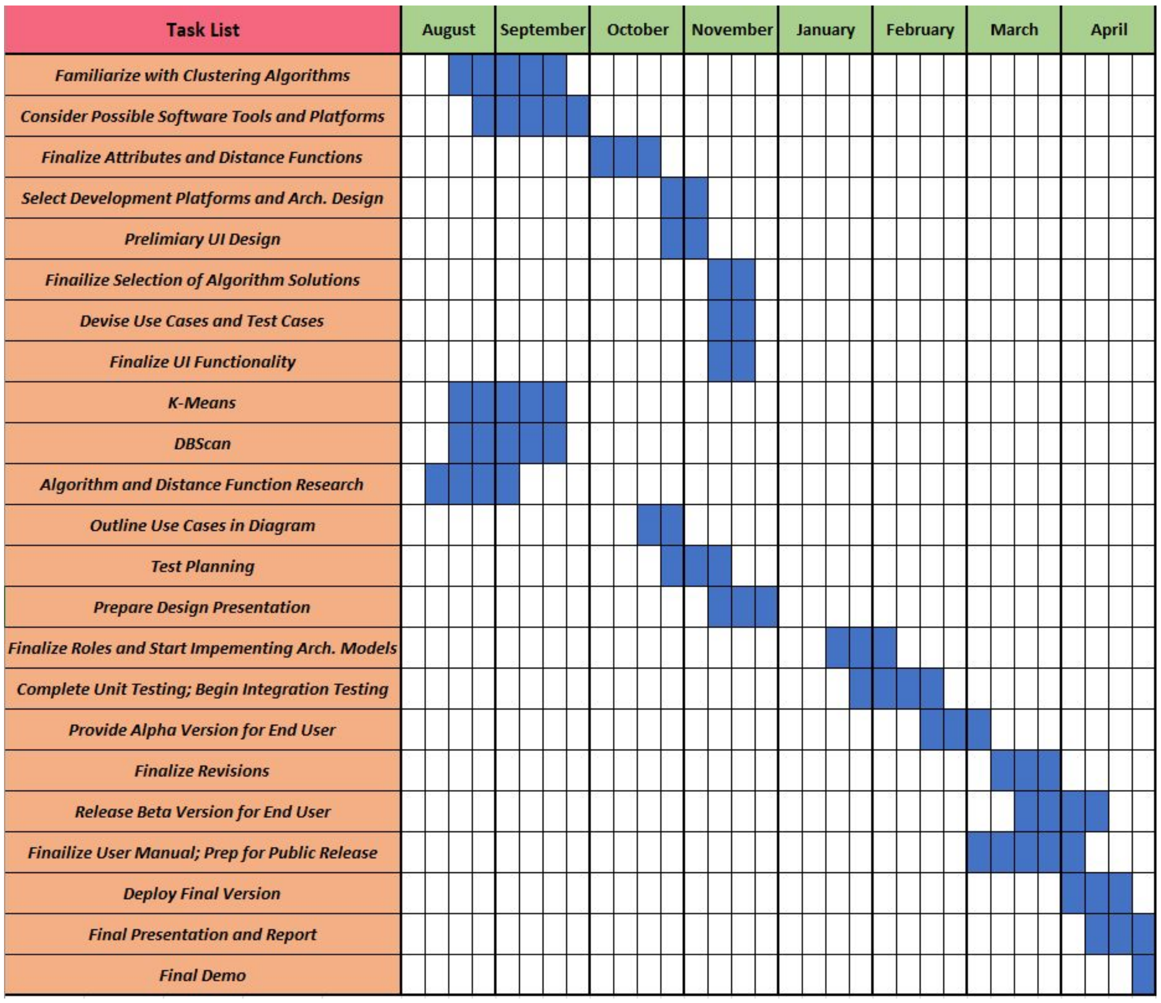
Our project follows two major milestones, the submission of our design document and the presentation of our implementation of the design. The design associated tasks have an emphasis on project requirements and use cases.

- Team understanding of binary star system clustering
- Defining functional requirements
- Outlining the best software tools/platforms for our project
- Determining project usage of algorithms and distance functions
- Defining use cases
- Designing a UI to cover use cases

The criteria above that is connected to the design phase of our project will be evaluated by how well our design covers the requirements of the original project.

The second phase of our project where we will implement our design will have tasks connected directly to the software development. Functional tests will be used to evaluate how well our implementation fulfills the design. Working towards this second milestone will also involve making improvements to non-functional requirements such as reliability, performance, and scalability.

2.4 Project Timeline/Schedule



Task List	August				September				October				November				January				February				March				April			
Familiarize with Clustering Algorithms			■	■	■	■	■																									
Consider Possible Software Tools and Platforms				■	■	■	■	■																								
Finalize Attributes and Distance Functions									■	■	■																					
Select Development Platforms and Arch. Design												■	■																			
Prelimiary UI Design												■	■																			
Finailize Selection of Algorithm Solutions														■	■																	
Devise Use Cases and Test Cases														■	■																	
Finalize UI Functionality														■	■																	
K-Means			■	■	■	■	■																									
DBScan			■	■	■	■	■																									
Algorithm and Distance Function Research		■	■	■	■																											
Outline Use Cases in Diagram											■	■																				
Test Planning												■	■	■																		
Prepare Design Presentation														■	■	■																
Finalize Roles and Start Impementing Arch. Models																			■	■	■											
Complete Unit Testing; Begin Integration Testing																				■	■	■	■									
Provide Alpha Version for End User																							■	■	■							
Finalize Revisions																										■	■	■				
Release Beta Version for End User																											■	■	■	■		
Finailize User Manual; Prep for Public Release																									■	■	■	■	■			
Deploy Final Version																													■	■	■	
Final Presentation and Report																														■	■	■
Final Demo																																■

Figure 1: Project Gantt Chart

The chart above shows the time frames associated with our tasks and deliverables for both semesters one and two. The gap indicates the winter break between the semesters. The first semester has tasks working towards our design to be implemented in the tasks layed out for the second semester.

The team member primarily responsible for each task will follow our team roles of Scribe, Facilitator, Chief Engineer, Test Engineer, and Report Manager. Our team has decided that we will be flexible in contributing outside of our roles, and will likely switch roles along the way.

2.5 Project Tracking Procedures

We plan to use standard Agile and Scrum styled boards to track our progress and current goals. We will be using Azure Devops for our project and using their built in task and story tracking. To follow Agile standards, our board will have a backlog, to-do on the current sprint, items we are currently working on, in review items, and finished columns. We will progress tasks and stories through these columns to keep track of how much work we have done and what our current priorities are.

As tasks are added from our advisor and our own grooming we will define the tasks on our backlog. Each sprint, tasks from the backlog will be moved to the to-do column and become our sprint priorities.

To communicate current progress on tasks we will discuss in Slack and weekly meetings.

2.6 Personnel Effort Requirements

Task	Iteration Length	Person Hour Estimate	Team Members Needed	Total
Familiarize with clustering algorithms	30 Days	5 each	6	30 Hours
Consider possible software tools and platforms	30 Days	5 each	6	30 Hours
Finalize attributes and distance functions	15 Days	10 each	6	60 Hours
Initial UI Design	16 Days	5 each	3	15 Hours
Finalize selection of development platforms and architecture design	16 Days	10 each	4	40 Hours
Finalize selection of algorithm solutions	10 Days	5 each	3	15 Hours
Devise use cases and test cases	10 Days	5 each	3	15 Hours
Finalize UI functionality	10 Days	5 each	3	15 Hours
Prepare design presentation	10 Days	4 each	6	24 Hours
Finalize and submit design document	10 Days	3 each	6	16 Hours
Finalize roles and start implementing arch. models	21 Days	15 each	6	90 Hours
Complete unit testing and begin integration testing	28 days	25 each	2	50 Hours
Provide alpha version	21 days	10 each	6	60 Hours
Finalize Revisions	21 days	15 each	6	90 Hours
Release beta	28 days	15 each	6	90 Hours
Finalize user manual/documentation	35 days	10 each	2	20 Hours
Release final version	21 days	15 each	6	90 Hours
Final project and presentation	21 days	10 each	6	60 Hours
Demo	7 days	4 each	6	24 Hours

Table 1: Personnel Effort Table

Above we can see each task broken down into a more specific estimate of hours needed to complete. We followed the class recommendation of about 30 hours a week of work while creating our iteration lengths. Each task does not require all members to participate, but in the planning phase many of them do.

For each task, not only did we estimate the amount of hours needed but also the amount of people that would participate in each task. Later in the project, tasks can more easily be broken down into front end and back end and the team members who are assigned to each domain will be more focused on those tasks.

Starting with the “Finalize roles and start implementing arch. models” task, the estimates get much broader and the tasks less decomposed. This is the beginning of next semester and thus it is harder to estimate accurately the amount of work for each task.

Our personal effort chart does not schedule out when the tasks will be completed and also does not include in class assignments that do not contribute to the project.

2.7 Other Resource Requirements

For our project, there are only a few resources required. Aside from developmental resources, like computers, we will need server space to host our backend and database. The space the database occupies will be fixed as initially we are only hosting the data we already have. As a stretch goal, we have the opportunity to include more data and that will contribute to the amount of space we need. Other than that, the backend isn't hosting much more than a python script that will run calculations and be available for REST queries.

We also may need to host our front end using a hoster like Vercel, AWS, or GCP. For proof of concept projects, these sites rarely need to be paid for, but there are cases where we would need to purchase hosting credits.

2.8 Financial Requirements

The only financial requirements needed for our project is if we need to pay to host our front end, as mentioned above. During the course of our project, it is doubtful we will hit the page enough for the sites to demand payment, however it could become necessary.

For AWS Lambda, the pricing is $0.20 for 1 million requests or $0.0000166667 for every GB-second.

There is no financial requirement for our backend as Iowa State has agreed to give us server space for free.

3 Design

3.1 Previous Work And Literature

Clustering Algorithms

K-means

- One of the clustering algorithms we're using is K-means. This is a well known process and has many descriptions/tutorials on the internet. The main reference point for our implementation of K-means is derived from the article *K-means: A Complete Introduction* from the website towardsdatascience.com.

DBScan

- A second clustering algorithm is DBScan. This algorithm is also well known and documented on the website towardsdatascience.com under an article titled *DBSCAN Clustering — Explained*.

Research Papers

The following research papers introduce similar problems that are solved via clustering algorithms. Concepts and implementation details are pertinent to our clustering of binary stars through time.

Computing Longest Duration Flocks in Trajectory Data

- This paper presents the concept of monitoring flocks and meetings during a period of time. The concepts of flocks and meetings are expanded to include the specification varying or fixed; this is all a variation of our notion of clusters. Algorithms are provided for each cluster type and efficiency is discussed.

Discovery of Convoys in Trajectory Databases

- This paper discusses three algorithms for finding clusters during a period of time. Each tracted node moves along a "trajectory", computing which trajectories are closest to each other is the problem. The paper presents a density based clustering algorithm and adds optimizations to increase efficiency.

Difference from Similar Products

Our project incorporates technologies such as clustering algorithms and machine learning that are widely used. However, our product stands on its own as a unique tool for astrophysicists to organize queried data of binary stars. Already in this field databases are available through SQL queries. One such example is through http://skyserver.sdss.org/. This simple SQL query tool has no concept of clustering, so received data is hard to work with. Our product will organize the queried data into clusters based on weighted attribute functions. It will also include the clustering of data through time. Most of this data is numerical, so we will provide a visual interface to display the data in an intuitive way.

3.2 Design Thinking

For the front-end, we want to create a powerful but usable design. User experience plays a big part in all software and any application should consider it. Our project has a very specific user in mind, but there are still a wide range of accessibility and usability to consider.

For scientific applications, it can be difficult to create a design that is clean but also powerful enough to satisfy the scientist's needs. Our user forms and simulation output needs to be quickly understandable. We want to enable a scientist to adjust any part of the simulation they want to, but also correctly infer preferences where they might not.

For our entire system, we want to create components that interact efficiently with each other. Our backend needs to be able to process a lot of complicated data at once and send it all to the front end, which requires us to consider the fastest way to communicate between components. Efficiency and cleanliness of communication really drives the design of our system and it is something we are always considering while choosing technologies.

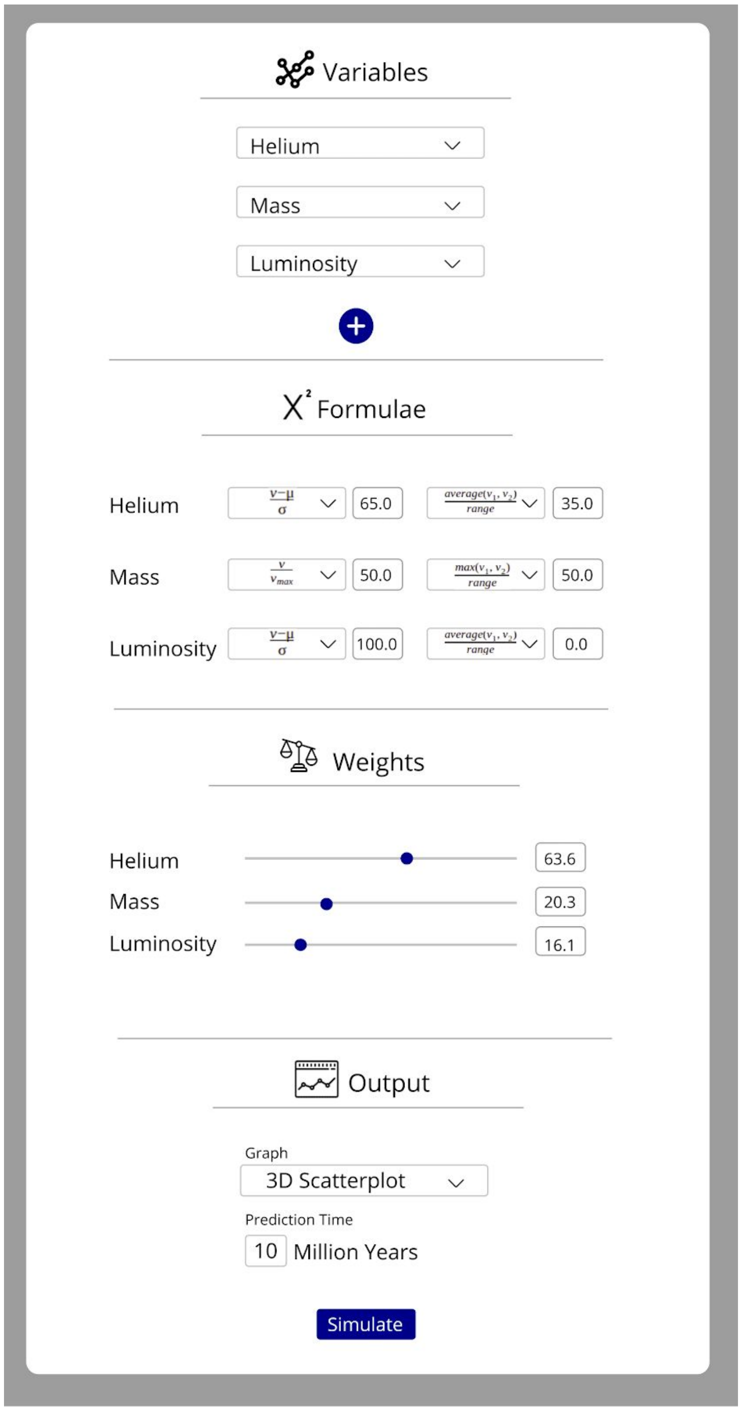
Lastly, while choosing a method of storing and retrieving data we had to consider what our data would most commonly look like. There are a lot of different types of databases to choose from and it is important to know what kind of data you'll want to store. Our data is mostly simple relational data that are easily formed into rows and columns and so we immediately thought of a relational database like PostgreSQL.

3.3 Proposed Design

Frontend

On the frontend, we have decided to go with the javascript framework Angular. It is important that we have a frontend framework that will make our lives easier while designing the user interfaces of our application. Using base html, css, and javascript works for simple projects, but we need a framework that communicates with a backend, handles forms, and outputs complicated data with ease. Our front end will include different user forms that allow a user to tweak and edit the algorithms and data to give them a simulation that reflects their desired output, fulfilling the functional requirements of our application. The interfaces need to be fast and easy to understand.

The following is an example of what our user would see when filling out a form to submit a query.



Variables

Helium

Mass

Luminosity

X^2 Formulae

Helium	$\frac{v-\mu}{\sigma}$	65.0	$\frac{average(v_1, v_2)}{range}$	35.0
Mass	$\frac{v}{v_{max}}$	50.0	$\frac{max(v_1, v_2)}{range}$	50.0
Luminosity	$\frac{v-\mu}{\sigma}$	100.0	$\frac{average(v_1, v_2)}{range}$	0.0

Weights

Helium	63.6
Mass	20.3
Luminosity	16.1

Output

Graph: 3D Scatterplot

Prediction Time: 10 Million Years

Simulate

Figure 2: Prototype User Interface for Querying

Web App and REST API

We have decided to go with a web application for our project. With a web app, we don't have to worry about storing data locally on a scientist's computer, which would dramatically increase the size of our application. We also can ensure a quick communication between the frontend and backend using REST APIs that serve json.

REST APIs are great for larger projects as they allow team members to work separately on a project and follow a defined contract. The contract defines what data needs to be transferred and what that data will look like. JSON is a clean and readable data format that helps to define the contract between parties. We also know that the technologies we chose work well together when using REST APIs.

Our frontend will be a relatively small load on a computer as all of the heavy lifting will be done by our server on the backend. This will enable any scientist to use our application and expect fast responses no matter their computer's power. We will also be able to control the functionality of our backend much easier as it will all be hosted on a server we create. This fulfills our nonfunctional requirements of working with an internet connection on a personal computer or tablet.

Backend and Database

For the backend, we are using the Python web framework Django. As discussed in section 3.4, Django is a powerful tool that allows us to process a lot of information on a web server. This web server will get information from the database using a SQL connection, run calculations with specifications chosen by the user, and send the finalized data to the frontend. With a web server, we can have the powerful tools within preexisting python libraries handle a lot of the calculations.

The backend will also easily receive data from the frontend using REST. Our REST API will be set up to handle different specifications and requests by the frontend and indiscriminately respond to each.

Our database will be in SQL to maintain the relational qualities of our pre-existing data. The types of requests we will be sending to our database are easily handled by SQL. The connection between our Python backend and SQL database will be fast and efficient, handling the nonfunctional requirement of an efficient processing of our complicated data. SQL is suited to store large amounts of relational data like the data we have already.

System Summary

To start, our system will handle events on the frontend. Here, a user will be able to choose different algorithms, weights of data, and output styles they want. The frontend compiles this information and sends it to the backend via a REST request. The backend will process the received information, query for data using an SQL connection, and run the simulation. After the simulation is completed, the backend will send the information back to the frontend as a REST response and the frontend will display it. This process will be quick and efficient, but powerful enough to handle almost any desired simulation from the user.

Requirements

For functional requirements, we mentioned we need to have a user specify many attributes to adjust the simulation. Our frontend will have those displayed via web forms and our backend will have an endpoint set up to handle the specifications. We also require the user to have an internet connection and the system needs to run on a personal computer, which is fulfilled by our choice of a web application.

For nonfunctional requirements, we need the system to be able to process data quickly. Our application will not be usable if it takes too long or is too slow while simulating. For instance, no cluster query should take longer than 30 seconds to process. We believe that a python backend that serves information via REST will be efficient to handle the type of data we have.

Current Implementation

So far, we have created basic designs for our frontend as well as implemented an initial database. Our frontend designs are clean and allow the user to easily adjust parameters as they want. Our database can already handle the large amounts of data that was given to us and we can quickly query that data.

3.4 Technology Considerations

Frontend

For the frontend of our project, the team decided right away to use Javascript as the language. We then considered both the Angular framework and the React library.

Angular

Angular is a component based javascript framework by Google. Angular takes UI features and breaks them up into components. These components are usually small files considered to be a small subset of your application. Angular uses a real document model instead of a virtual document model, which makes it easier to use 3rd party libraries. Also, Angular provides an extensive development experience by providing a powerful command line interface. Although most members on our team haven’t used Angular before, we will have to learn, but we found out that Angular is very similar to React.

React

React is a Javascript library made by Facebook. Most of our team members have had experience with React. React is similar to Angular in more ways than not, but some of the biggest differences between Angular and react is that react uses a virtual document model, which is known to be faster, but dealing with 3rd party libraries can be troublesome such as graphing libraries. React doesn’t have two-way data binding for UI components. This allows you to bind form inputs to values in code which allows us to easily manipulate and send requests to our backend.

Backend

We mostly had to decide between which language to use on our backend.

Java

Java is a language every member of the team knew prior to starting this project. It is widely used and has the powerful Spring framework that makes creating web applications like this project much more approachable. Most members of the team have also used the Spring framework, and it would allow for us to start implementing the backend of our application immediately.

A drawback of using Java would be the lack of support for a simulation application like ours. Finding and using libraries that implement clustering algorithms is extremely difficult, and implementing our own versions of these algorithms would take considerable time and be prone to many errors that would need to be thoroughly tested.

Python

Python is a very approachable language that is relatively easy to pick up. About half of the team members have used Python before all of the others are willing to learn it for the project. It also has its own powerful web framework in Django. None of the team members have used the Django framework before, and we would have to spend considerable time learning it.

A major benefit of using Python is the readily available modules that implement both the K-Means and DBscan clustering algorithms that we will be using for our project. These clustering algorithms are quite complex in nature and require a very strong mathematical background, so modules available that correctly implement them would be a major benefit to our project. Another major benefit of Python is the **NumPy** module. This module provides extensive mathematical tools that would allow us to perform intense calculations, which is required for our project as we are going to be simulating the evolution of binary stars for potentially millions of years.

With these considerations in mind, our group decided to use Python as our backend language. Many members have an interest in learning Python, and the benefits it will provide to our application outweigh the time investment that will be spent learning the language.

3.5 Design Analysis

Our design has not been implemented yet, so this section will address observations, thoughts, and ideas to modify or iterate over the design. First of all, our proposed design could use more specification in terms of requirements. The more we specify, the more we will realize what we do and don't want. Second, iterations over the design every few weeks will strengthen it as we become more and more familiar with the technologies. Third, possible additions may be made to the design documents as our team encounters new technologies that will be convenient to achieving our requirements.

3.6 Development Process

We have decided to go with an Agile development process for our design. Software development is nearly impossible to design all at once and thus it is important to be continuously designing while developing. Breaking a project into sprints allows for careful implementation as well as reflection. We can reflect on the implementation of our design as we are creating it.

It is guaranteed that you will run into hiccups and problems throughout the implementation of a software application. At the beginning of each sprint, we will be able to reflect on those problems and create a plan to adjust the design of our system.

Agile development also allows us to receive feedback from our advisor and client while implementing our application. Especially in an era of remote work, it is important that our team understands the exact desires of our client. At the end of each sprint, we will be able to show our client the progress we have made and receive feedback that will make our project more successful.

As a team, we are all comfortable with Agile development as we have all used it before. Since we all follow agile methodologies, we will be on the same page with the continuous design and implementation of our system.

3.7 Design Plan

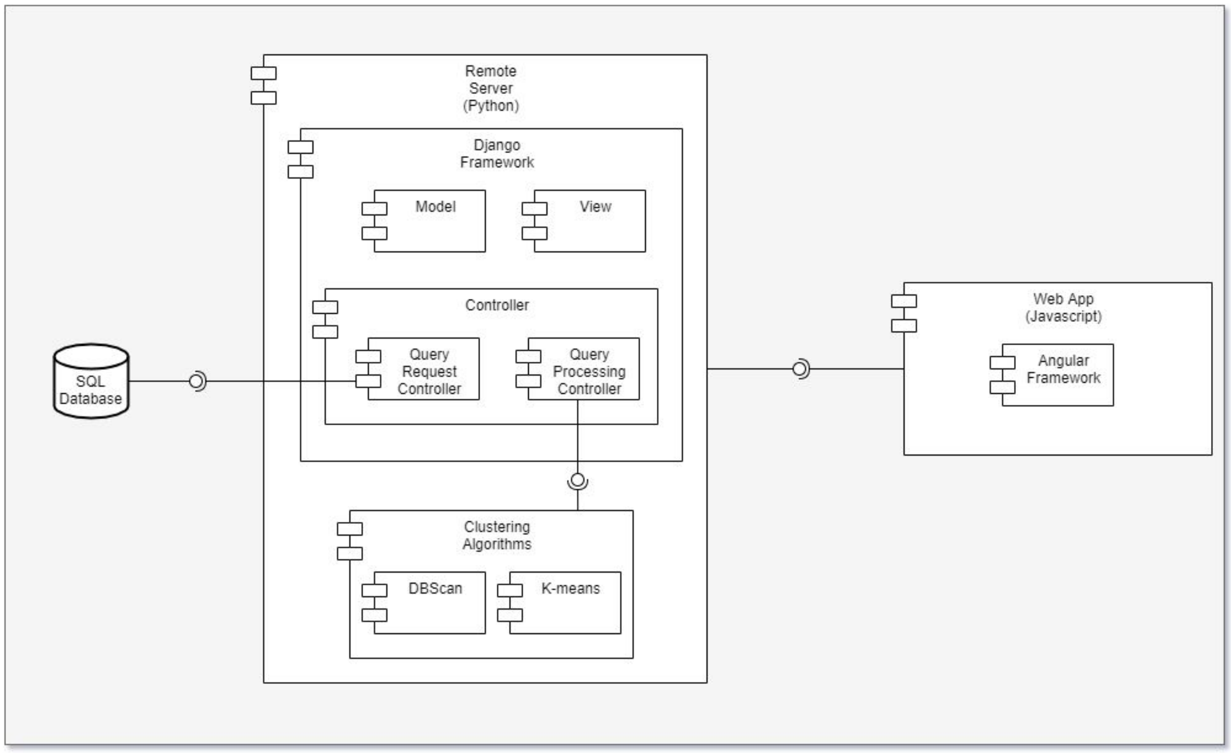


Figure 3: Component Diagram

Above is a diagram defining the design of our application and its various components.

4 Testing

4.1 Unit Testing

Our project testing will be all software unit testing on both our front end and back end

The main units that will tested in isolation are the following:

- User Interface
  - Our project needs to verify that the components in our user interface our both easy to use and correctly take input from the user to then pass to the back end
  - Testing: Our front end built with javascript using the modern framework has access to many convenient testing libraries. We will likely use Jasmine and Karma (built into Angular) for unit tests and end to end tests for our user interface. Filling in sample inputs and running snapshot tests will verify that our user interface is gathering input to send to the backend correctly and reliably
- Database Access
  - We will need to verify that the input being sent to our backend is processed and the correct result is returned to be displayed by the front end.
  - Testing: to test that our input is being correctly passed to the backend via our API calls we will simulate a number of API calls and unit test the responses received.
- Data Normalization/Comparison
  - Before we place the data queried from the database into the clustering algorithm, we will need to normalize the different types of data so they can be compared to each other. We plan on being able to cluster binary stars by up to five different types of data at once (for example, cluster by mass ratio, luminosity, and volume). These different values can not be immediately thrown into the clustering algorithm without being normalized so they can be fairly compared to each other.
  - We have come up with multiple formulae that will normalize our data to each other so clustering can occur. See Appendix A for the table of all of the relevant formulae.
  - To test this unit, we will need to simulate a variety of different data combinations to be normalized. This will allow us to verify that the data is being appropriately normalized so it can be used for the clustering algorithm.
- Clustering Algorithm Execution
  - We are using a premade python library to execute the K-means clustering algorithm we are using for our project. While this has certainly been tested in production of the library, we will need to perform tests to verify it is working as intended in the context of our project.
- Front End Graphing Displays
  - Once the results of the simulation are complete we need to verify that the graphics we use to display the results are being displayed correctly in an understandable way

4.2 Interface Testing

The following are the two major units of our project that will need to be extensively tested to ensure that the application works as intended and provides a suitable user interface.

1. User Interface
   - In order to test that the user's input is correctly taken in by the user interface and passed to the backend. Our JavaScript front end making use of Angular the modern web framework can make use of Jest, a popular JavaScript testing library

    that can model snapshots of our UI components and test that the values they are given match the values that were inputted by the user.
    - We can also make use of a forms library to help handle problems related to form state, submission and validation
2. Backend Producing Correct Clustering Results
    - From the front end's perspective this interfacing will be tested by simulating API calls to the backend and testing the results against expected values again using something like jest.
    - The backend will have various unit tests not entirely worked out yet that will be designed to verify correct execution of the clustering algorithms and correctness of our distance functions.

Other relevant interfaces include:

- Database returning relevant data based on user input
- Backend data normalization calculation
- Correct graphing display based on clustering results

4.3 Acceptance Testing

Our non-functional requirements testing will be designed to verify the environmental and economic requirements for our project. Our main environmental requirement involves a secure internet connection in order to use our application. Measuring network performance and issuing notifications indicating no/poor connection will improve and test the user experience. Other nonfunctional tests related to performance can be performed through google open source lighthouse service. This is an easy way to get reliable statistics on site performance and accessibility. We also plan to get input from our project advisor as well as input from the members of the Iowa State physics department that will be using our application on the overall user experience.

To ensure that our functional requirements are being met we will run end-to-end tests. This confirms that the user can specify the parameters of interest and that those values are correctly stored within the application to be applied to the clustering algorithm performed on the backend. The front end then needs to correctly interpret the results returned from the backend and display the graph representing the evolution of the binary star system with the corresponding information on whether or not the system will merge.

4.4 Results

So far, we have not tested much of our system as we are still within the design phase of our project. However, we have done some basic testing of our database. We imported a large amount of data into our database and ran some basic queries. We ran into some issues while importing as the current version of PostgreSQL does not allow for more than 1 GB of data to be imported at a time. However, we found a work around by breaking up the data into chunks and importing that way. We hope to automate that process in the future. Our basic queries are working great and we have already confirmed our hopes that PostgreSQL will work quickly on our large amount of relational data.

5 Implementation

Frontend

The front end can be designed and mostly implemented without communication to the backend. Using angular, we can implement the initial design components and get a good prototype quickly which will leave us time to implement the more complicated data rendering and backend communicating parts of the frontend.

The most difficult part of the frontend will be the data rendering. The data coming from our backend will be complex and need to be organized properly for effective visualization. We will also need to properly research the best graphing techniques to render clustering data.

Communication with the backend will not be difficult as all modern javascript frameworks come ready to communicate with REST APIs.

REST API Webapp

Creating the REST webapp will be one of the first milestones of our project in the spring. Getting a working communication with the frontend will help the front end devs not get stuck during implementation. Luckily, creating a REST API will not be difficult using Django. We will also need to use this time to deploy the backend on the server space we request. Having the backend on a running server will allow all members to work on the backend continuously. This will be needed for everyone to efficiently work on our project.

Backend & Database

The database is already in an initial running state, so our priority for the backend will be connecting the database and begin running basic clustering algorithms. Once we know what the clustering algorithms will look like, the frontend can begin their work on data rendering. Connecting the backend to the database will also be fairly easy as python can easily connect to most modern databases.

Testing

As for testing, we will need to quickly get CI/CD and basic unit tests working towards the beginning of our implementation. Ensuring that updates to the system don't collapse our backend is a requirement of any software development team. Unit tests will help maintain a consistent goal of the system despite any changes.

System Priority Overview

Frontend: Begin implementing preliminary UI and experimenting with data rendering once available.

Backend: Create REST connection and begin communicating with DB. Also, start experimenting with clustering algorithms.

Testing: Implement CI/CD and basic unit tests to ensure system compliance.

6 Closing Material

6.1 Conclusion

In conclusion, our team has completed an initial set of milestones. First, our team has organized itself so that communication is efficient and expectations are shared. We've determined individual roles, shared opinions on preferred development work, and provided peer feedback. Next, the team has laid out comprehensive requirements for this application. This includes a developed and expanded list of project tasks and associated risks. At the end of fall semester, we've completed an initial design including the project architecture, technology selection, and testing plan.

Moving forward, our team anticipates completing the following goals. First, we need to achieve proficiency with the technologies selected. This includes, at the least, knowledge of python, javascript and the angular framework, SQL with postgres, and REST API technology. Second, the deployment of the backend and functioning REST API responses will enable the frontend to begin development. Third, the frontend will set up an interactive UI and basic use cases. This includes an initial tool for visualizing data. Fourth, development of clustering algorithms on the backend will give users pertinent clustering data output. And lastly, we will add the variable of distance functions to clustering algorithms giving users even more preference in how clusters are formed.

Using this design document as a guide, our team plans to move forward developing this application. We believe that the plan described in this document is the best plan of action. Following an Agile development methodology gives the team flexibility to iteratively apply changes to the project. The technologies are well chosen due to their compatibility and ease of use. Lastly, the testing plan will verify functionality with respect to the requirements presented in the design document.

6.2 References

Gudmundsson, Joachim, and Marc van Kreveld. (2006, November). *Computing Longest Duration Flocks in Trajectory Data*. Proceedings of the 14th Annual ACM International Symposium on Advances in Geographic Information Systems, 35-42.

Jeung, Hoyoung, and Man Lung Yiu, Xiaofang Zhou, Christian S. Jensen, Heng Tao Shen. *Discovery of Convoys in Trajectory Databases*. https://doi.org/10.14778/1453856.1453971, 2010

Jeffares, Alan. *K-means: A Complete Introduction*. https://towardsdatascience.com/k-means-a-complete-introduction-1702af9cd8c, 2019.

Yildirim, Soner. *DBSCAN Clustering — Explained*. https://towardsdatascience.com/dbscan-clustering-explained-97556a2ad556, 2020.

6.3 Appendices

Appendix A

The following is a table containing the formulae our group has come up with so the different types of data can be clustered. The user will have the option to select which equation they want to use to preprocess the data before the clustering algorithm is applied.

Amplitude Scaling	**Offset Translation**	**Divide by Maximum Value**	**Minmax Normalization**	**Ratio Between Stars**	**Average Over Range**	**Difference Over Range**
$\frac{v-\mu}{\sigma}$	$v - \mu$	$\frac{v}{v_{max}}$	$\frac{v - min}{max - min}$	$\frac{v_1}{v_2}$	$\frac{average(v_1, v_2)}{range}$	$\left\lvert \frac{v_1 - v_2}{range} \right\rvert$
Normalizing	Normalizing	Normalizing	Normalizing	Normalizing	Non-normalizing	Non-normalizing

Table 2: Clustering Formulae Table

Handling Multiple Data Points Per Value

- $v = average(v_1, v_2)$
- $v = max(v_1, v_2)$
- $v = min(v_1, v_2)$
- $v = max(v_1, v_2) - min(v_1, v_2)$