Proximity vs. Diversity in Heterogeneous Datasets

Design Document

Group 31
Dr. Goce Trajcevski

https://sdmay21-31.sd.ece.iastate.edu/
sdmay21-31@iastate.edu

Revised November 15, 2020

Thomas Beckler
Bradley Gannon
Benjamin Huinker
Gabriel Huinker
Koushik Kumar
Cristina Marquez
Jacob Spooner
Executive Summary

In this document, we will show our design and implementation for group SDMay21-31 and the process we took to create a web application to help astrophysicists view patterns in large heterogeneous datasets of binary stellar trajectories.

Development Standards and Practices Used

Team Agile Processes:

- Agile software development practices
- Two week sprints
- Sprint planning meetings
- Daily standups
- Sprint review meetings
- Estimation of the difficulty of tasks as a team
- Allocation of responsibilities among team members

Version Control Standards:

- Group review and approval
- Store code in repository
- Git practices

Coding and Testing Standards:

- Linters
- UI development standards

Summary of Requirements

- Intuitive and responsive website to interact with a heterogeneous dataset
- Secure data storage
- Efficient and accurate data lookup
- Short server response times
- Users can easily input attributes/parameters and select proximity/diversity
- Interface is easy to control and read

Applicable Courses from Iowa State University Curriculum

- Com S 309: Agile, Git techniques
- Com S 311: Analyzing and designing efficient algorithms
- Com S 319: UML Diagrams
- Com S 329: Risk management
- Com S 339: Architecture design introduction
- Com S 363: Database Systems
• Cpr E 414: Big data analysis
• Cpr E 419: Batch & Real-time large scale data analytics
• DS 301: Machine learning
• Stat 330: Data algorithm design

New Skills/Knowledge acquired that was not taught in courses
• Web Development
• K-means, DBSCAN clustering
• Analysis of binary star data
• Remote development
# Contents

1 Introduction ................................................. 5
   1.1 Acknowledgement .......................................... 5
   1.2 Problem and Project Statement ......................... 5
   1.3 Operational Environment ................................. 5
   1.4 Requirements ........................................... 5
   1.5 Intended Users and Uses ................................. 6
   1.6 Assumptions and Limitations ............................ 6
   1.7 Expected End Product and Deliverables ............... 7

2 Project Plan ................................................. 8
   2.1 Task Decomposition ...................................... 8
   2.2 Risks And Risk Management/Mitigation................ 8
   2.3 Project Proposed Milestones, Metrics, and Evaluation Criteria ........................................ 9
   2.4 Project Timeline/Schedule ................................ 10
   2.5 Project Tracking Procedures ............................. 11
   2.6 Personnel Effort Requirements ........................... 12
   2.7 Other Resource Requirements ............................ 12
   2.8 Financial Requirements .................................. 13

3 Design ......................................................... 14
   3.1 Previous Work And Literature ......................... 14
      3.1.1 Relevant Background/Literature ..................... 14
      3.1.2 Describe what other products exist in the market ........................................ 14
      3.1.3 Differentiate your project from what is available ........................................ 14
   3.2 Design Thinking ......................................... 14
   3.3 Technology Considerations ............................... 15
      3.3.1 Backend ............................................. 16
      3.3.2 Algorithms .......................................... 16
      3.3.3 Database ............................................ 16
      3.3.4 User Interface ...................................... 16
   3.4 Proposed Design ......................................... 16
      3.4.1 Web Application Architecture ....................... 17
      3.4.2 User Interface ...................................... 18
   3.5 Design Analysis ......................................... 18
   3.6 Development Process .................................... 19
   3.7 Design Plan ............................................... 19

4 Testing and Implementation ................................. 22
   4.1 Unit Testing .............................................. 22
   4.2 Interface Testing ........................................ 22
   4.3 Acceptance Testing ....................................... 22
   4.4 Results .................................................. 23
List of Figures

1. Gantt chart ......................................................... 11
2. Broad overview of architecture ................................ 15
3. Detailed architecture design .................................... 17
4. User interface wireframe ........................................ 18
5. Design plan ....................................................... 21

List of Tables

1. Requirements ..................................................... 5
2. Task decomposition ............................................. 8
3. Risks and risk management/mitigation ....................... 9
4. Project schedule ................................................ 11
5. Personnel effort ................................................. 12
6. All considered technologies ................................... 15
1 Introduction

1.1 Acknowledgement

We are grateful for Dr. Goce Trajcevski, who meets with us every week, provides us with fundamental astrophysics knowledge, and walks us through the complicated topics our project addresses. We would also like to thank our class professor and guest speakers for taking time out of their day to teach us new skills.

1.2 Problem and Project Statement

Researchers have difficulty visualizing the large heterogeneous datasets of binary stellar trajectories. With the vast number of publicly available datasets retrieved from astronomical observations and simulation, there needs to be a tool that allows researchers to quickly search for patterns in the data. The main use case for this data is the clustering of binary stars based on the proximity and diversity of the stellar attributes across multiple distance functions. Using these distance functions, analysts would like to measure the outcome of different attributes related to the attribute’s proximity vs. diversity.

To solve this problem, we will develop an intuitive web-based tool that will allow users to select distance functions for different data parameters. The application will enable users to query a large heterogeneous dataset with a focus on proximity vs. diversity.

1.3 Operational Environment

For this project we are not concerned with the operational environment since the application will be hosted on university machines.

1.4 Requirements

This table contains a list of requirements with their descriptions, This table will be a work in progress throughout the semester.

<table>
<thead>
<tr>
<th>ID</th>
<th>Requirement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>A website to interact with stored data</td>
<td>The website must have tools to input constraints, which will be sent to the server to search for binary stars.</td>
</tr>
<tr>
<td>R2</td>
<td>Secure and Efficient data storage</td>
<td>Only privileged users should be able to edit data.</td>
</tr>
<tr>
<td>R3</td>
<td>Displays relevant data in a timely manner</td>
<td>Users are able to view the data that they want clustered shortly after they make a call to the database.</td>
</tr>
<tr>
<td>R4</td>
<td>Users determine which attributes require proximity/diversity</td>
<td>Users are able to assign weights to certain data points to simulate proximity and diversity within the database.</td>
</tr>
<tr>
<td>R5</td>
<td>Allow users to query heterogeneous data sets</td>
<td>Analyze the data to extract rows matching certain user-inputted measures.</td>
</tr>
<tr>
<td>R6</td>
<td>Data entry interface is intuitive to the all users</td>
<td>The user must be able to understand how to input data without any additional instructions.</td>
</tr>
</tbody>
</table>

Table 1: Requirements

Functional Requirements

- A website to interact with stored data.
Secure and Efficient data storage.
Displays relevant data in a timely manner.
Users determine which attributes require proximity/diversity
Users can change the weight of attributes
Allow users to query heterogeneous data sets

Non-Functional Requirements

- Usable by the general public
- Easy to navigate
- Simple and intuitive design

UI Requirements

- The UI works for all general client devices
- Data entry interface is intuitive to the average user

1.5 Intended Users and Uses

The intended end users are astrophysicists who will use the web application to view clustering patterns in data about binary stars. Although this application will be made specifically for researchers in this field, it will be accessible to the general public.

1.6 Assumptions and Limitations

Assumptions

- Dataset will be over 5 GB
- Primary users will be knowledgeable about the data they are viewing
- Data will be imported through CSV (Comma Separated Value) files, or will be changed in an admin interface
- A server will be provided

Limitations

- Budget: We are not given money for resources
- Time: We have two semesters to implement a web application
- COVID-19: Physical interactions between group members are discouraged during these semesters
- Computing Environment: The service must run on a university provided virtual machine
- Response Time: Server responses should be returned in a reasonable time
- Developer Experience: Field experience of developers can limit feature implementations
- Intuitiveness: Data entry interface must be intuitive to the average user
1.7 Expected End Product and Deliverables

The main deliverable for this project is a web application that will allow users to query datasets based on the proximity and diversity of various data parameters. The web tool will allow users to select distance functions and different weights of the parameters corresponding to binary stars. Weights will make it possible to assign an importance to parameters based on the users’ queries. In addition, relativized parameters will augment the functionality. Relativizing parameters will allow proximity and diversity based on relative distance, as opposed to absolute distance.

The second deliverable for this project is help documentation, which will be accessible as a page in the web application. The documentation will explain the available functionality and use cases of the web application. This deliverable will help new users understand how to use the application.
2 Project Plan

In this section we will lay out our schedule, split up tasks, delegate responsibilities, and plan our milestones for the upcoming semester.

2.1 Task Decomposition

The following table shows all the tasks that will be required to complete this project:

<table>
<thead>
<tr>
<th>ID</th>
<th>Title</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>T0</td>
<td>Design Infrastructure</td>
<td>Have frameworks and application structure formalized.</td>
</tr>
<tr>
<td>T1</td>
<td>Create local environment</td>
<td>A local environment implementing the structure from T0 is created and shared among developers.</td>
</tr>
<tr>
<td>T2</td>
<td>Initialize database</td>
<td>An SQL database is initialized with given data.</td>
</tr>
<tr>
<td>T3</td>
<td>Design UI</td>
<td>Design a web frontend that allows user selection for proximity and diversity at different weights for data attributes.</td>
</tr>
<tr>
<td>T4</td>
<td>Implement UI Design</td>
<td>Implement and develop the UI as designed in T3.</td>
</tr>
<tr>
<td>T5</td>
<td>Create queries</td>
<td>Have all the necessary queries for each function finalized.</td>
</tr>
<tr>
<td>T6</td>
<td>Implement backend business logic</td>
<td>All algorithms for parsing and representing data from T5 are implemented in the backend.</td>
</tr>
<tr>
<td>T7</td>
<td>Add UI Functionality</td>
<td>The user is able to functionally request and view data.</td>
</tr>
<tr>
<td>T8</td>
<td>Create production environment</td>
<td>Have a publicly accessible and usable environment created, and code deployed.</td>
</tr>
<tr>
<td>T9</td>
<td>Data visualization</td>
<td>Add data visualization to the frontend UI.</td>
</tr>
<tr>
<td>T10</td>
<td>Help Documentation</td>
<td>Finalize help documentation for how to use the web application.</td>
</tr>
<tr>
<td>T11</td>
<td>Presentation</td>
<td>Finish presentation and document report for project.</td>
</tr>
</tbody>
</table>

Table 2: Task decomposition

2.2 Risks And Risk Management/Mitigation

The following table assigns an estimated risk to each of the tasks described in section 2.1. If the estimated risk is 0.5 or greater, a possible mitigation strategy is described. The described mitigation strategy aims to provide solutions to possible issues that may arise during the product’s development.
<table>
<thead>
<tr>
<th>ID</th>
<th>Title</th>
<th>Risk</th>
<th>Reason</th>
<th>Mitigation Strategy</th>
</tr>
</thead>
<tbody>
<tr>
<td>T0</td>
<td>Design Infrastructure</td>
<td>0.1</td>
<td>Planning Stage</td>
<td>None</td>
</tr>
<tr>
<td>T1</td>
<td>Create local environment</td>
<td>0.5</td>
<td>Security Risk</td>
<td>To ensure a secure production environment, we will use environment variables laid out by the Django 12 Factor application pattern.</td>
</tr>
<tr>
<td>T2</td>
<td>Initialize database</td>
<td>0.4</td>
<td>Initialization Stage</td>
<td>None</td>
</tr>
<tr>
<td>T3</td>
<td>Design UI</td>
<td>0.1</td>
<td>Planning Stage</td>
<td>None</td>
</tr>
<tr>
<td>T4</td>
<td>Implement UI Design</td>
<td>0.1</td>
<td>Initialization Stage</td>
<td>None</td>
</tr>
<tr>
<td>T5</td>
<td>Create queries</td>
<td>0.7</td>
<td>Performance and Security risk</td>
<td>Make sure that queries meet performance levels. Given the large amount of data, timely queries are very important. Use Django QuerySets to retrieve data from the database.</td>
</tr>
<tr>
<td>T6</td>
<td>Implement backend business logic</td>
<td>0.9</td>
<td>Performance and Security Risk</td>
<td>Algorithms need to be thoroughly tested to make sure they meet performance criteria. We will use Django forms for input sanitization.</td>
</tr>
<tr>
<td>T7</td>
<td>Add UI Functionality</td>
<td>0.4</td>
<td>Performance and Security Risk</td>
<td>None</td>
</tr>
<tr>
<td>T8</td>
<td>Create production environment</td>
<td>0.9</td>
<td>Security Risk</td>
<td>The production environment must ensure no breaches or unauthorized access through any form of protocol and attack. We will go through server hardening protocols to mitigate the risk.</td>
</tr>
<tr>
<td>T9</td>
<td>Data visualization</td>
<td>0.5</td>
<td>Performance Risk</td>
<td>Possible issues with performance. Mitigation may require different tools, increased testing, and possibly no implementation.</td>
</tr>
<tr>
<td>T10</td>
<td>Help Documentation</td>
<td>0.1</td>
<td>Documentation</td>
<td>None</td>
</tr>
<tr>
<td>T11</td>
<td>Presentation</td>
<td>0.1</td>
<td>Documentation</td>
<td>None</td>
</tr>
</tbody>
</table>

Table 3: Risks and risk management/mitigation

### 2.3 Project Proposed Milestones, Metrics, and Evaluation Criteria

The following milestones are the major goals and dates for our project.

**Milestone 1: Architecture Design (Oct. 25)**

We will have a completed diagram that shows how frameworks for frontend and backend work together to create a web application. This diagram should be flexible and maintainable to allow future changes to be adopted to the infrastructure without serious change or overhaul.

**Milestone 2: Finalize Algorithm Solutions (Nov. 8)**

By this date, we will have a minimum of two algorithms to determine proximity and diversity within attributes of the dataset. The team must determine which formulas can be used to determine how "close" different attributes are to each other. The final algorithms must use formulas for all attributes of the data. The final algorithms must be able to cluster data based on these attributes.
Milestone 3: Finalize Design Document (Nov. 15)

The design document will be comprehensive and contain all information necessary for the development of the software. The document will contain all design plans for the software, how the team will work together to develop the software, and the plans for testing the quality of the software.

Milestone 4: Complete Unit Testing (Feb. 15)

Each software component will produce correct output for all unit test cases. The team will build early versions of all the software components, but they will not be fully integrated with each other yet. The software components will be tested for both valid and invalid input, and each component will be tested for vulnerabilities that could cause the entire program to crash.

Milestone 5: Complete Integration Testing (Feb. 26)

Each component will be tested to ensure that they are compatible with each other. Each component will also be tested for vulnerabilities caused by component interactions. The team will integrate all of the software components together, and all components of the architecture will work with each other to form the software as a whole.

Milestone 6: Alpha Release (Mar. 5)

A publicly available web application that allows users to view data for at least a single distance function based on the attributes the users select.

Milestone 7: Beta Release (Mar. 20)

The software will meet at least 90% of the requirements specified in the design document. The software will be carefully tested to ensure that it is easily usable by the intended user. The application will be responsive on all devices.

Milestone 8: Final Software Version (Apr. 15)

The final version of our application will be on our production server. The software will meet all of the requirements specified in the design document, and the software will also be tested thoroughly to ensure that it contains no major bugs. The help documentation will be created for the final version of the software, and the intended users will be able to use the software for performing work. The software will be in its finished state, and no further changes will be required.

2.4 Project Timeline/Schedule
Above is our project’s preliminary Gantt chart. The following table will elaborate on the tasks in the Gantt chart, and the dates in which these tasks are to be completed.

<table>
<thead>
<tr>
<th>ID</th>
<th>Task</th>
<th>Start Date</th>
<th>End Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>T0</td>
<td>Design Infrastructure</td>
<td>10/7/20</td>
<td>10/27/20</td>
</tr>
<tr>
<td>T3</td>
<td>Design UI</td>
<td>10/28/20</td>
<td>11/17/20</td>
</tr>
<tr>
<td>T1</td>
<td>Create Local Environment</td>
<td>2/1/21</td>
<td>2/5/21</td>
</tr>
<tr>
<td>T2</td>
<td>Initialize Database</td>
<td>2/8/21</td>
<td>2/10/21</td>
</tr>
<tr>
<td>T4</td>
<td>Implement UI Design</td>
<td>2/11/21</td>
<td>3/5/21</td>
</tr>
<tr>
<td>T5</td>
<td>Create Queries</td>
<td>2/11/21</td>
<td>2/18/21</td>
</tr>
<tr>
<td>T6</td>
<td>Implement Backend Logic</td>
<td>2/19/21</td>
<td>3/5/21</td>
</tr>
<tr>
<td>T7</td>
<td>Add UI Functionality</td>
<td>3/15/21</td>
<td>4/2/21</td>
</tr>
<tr>
<td>T8</td>
<td>Create Production Environment</td>
<td>4/5/21</td>
<td>4/13/21</td>
</tr>
<tr>
<td>T9</td>
<td>Data Visualization</td>
<td>4/7/21</td>
<td>4/16/21</td>
</tr>
<tr>
<td>T10</td>
<td>Help Documentation</td>
<td>4/19/21</td>
<td>4/26/21</td>
</tr>
<tr>
<td>T11</td>
<td>Presentation</td>
<td>4/27/21</td>
<td>5/5/21</td>
</tr>
</tbody>
</table>

Table 4: Project schedule

2.5 Project Tracking Procedures

Our group will track progress through the use of GitHub, Slack, and Google Drive. We will use GitHub to store the codebase. We will also use a GitHub project board as a Kanban board for members to create, assign, review, and complete tasks. The GitHub project board will be our primary way to track progress throughout the development of our project. Slack will be used as a message board for group members to ask and answer questions. Finally, all of our documentation will be located in Google Drive. This will grant all group members access to the documentation and allow all members to work on the documentation synchronously. Together, these tools will assist our team in the completion of the project by May 2021. To ensure progress is made each week, we will hold a meeting once a week to discuss the progress we have made and the current tasks we are working on.
2.6 Personnel Effort Requirements

The following table outlines the estimated hours required of each task in the project. Additionally, a short explanation of each task is provided in the rightmost column. In total, this project will require approximately 138 hours to complete.

<table>
<thead>
<tr>
<th>ID</th>
<th>Task</th>
<th>Hours</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>T0</td>
<td>Design Infrastructure</td>
<td>5</td>
<td>The frameworks and application structure form the foundation on which application is built.</td>
</tr>
<tr>
<td>T1</td>
<td>Create local environment</td>
<td>5</td>
<td>A local environment is necessary to develop in and verify proper functionality of application before deployment.</td>
</tr>
<tr>
<td>T2</td>
<td>Initialize database</td>
<td>5</td>
<td>A SQL database with the given data will be utilized for proximity and diversity algorithms in the back-end.</td>
</tr>
<tr>
<td>T3</td>
<td>Design UI</td>
<td>3</td>
<td>Create visual design for the frontend UI.</td>
</tr>
<tr>
<td>T4</td>
<td>Implement UI Design</td>
<td>20</td>
<td>The UI forms a large portion of this project. This is the medium in which the users of the app will interact with the data.</td>
</tr>
<tr>
<td>T5</td>
<td>Create queries</td>
<td>20</td>
<td>The queries used with the SQL database will be somewhat complex to account for proximity and diversity weighting.</td>
</tr>
<tr>
<td>T6</td>
<td>Implement backend business logic</td>
<td>25</td>
<td>Implementation of the SQL queries into the back-end is necessary before interaction with the frontend.</td>
</tr>
<tr>
<td>T7</td>
<td>Add UI functionality</td>
<td>10</td>
<td>Fully functional ability of frontend to communicate with backend and data requests.</td>
</tr>
<tr>
<td>T8</td>
<td>Create production environment</td>
<td>10</td>
<td>The app is now functional and can be deployed to the public user.</td>
</tr>
<tr>
<td>T9</td>
<td>Data Visualization</td>
<td>15</td>
<td>Utilization of data visualization techniques in the frontend would add a nice feature to the app.</td>
</tr>
<tr>
<td>T10</td>
<td>Help Documentation</td>
<td>10</td>
<td>A guide on how to use the application will be helpful for the user.</td>
</tr>
<tr>
<td>T11</td>
<td>Presentation</td>
<td>10</td>
<td>An overview of the project in written and oral form.</td>
</tr>
</tbody>
</table>

Table 5: Personnel effort

2.7 Other Resource Requirements

This project necessitates the use of computing and storage resources, which the university is providing. During the development and testing stages, our team will generally require one server and a database instance. If the purpose of the tests is to test something like basic functionality, a local server and database can be used. In the case of testing a high number of users or large amounts of data being continuously sent, we may need more or higher powered servers. To account for multiple test cases, the server should have at least 1 vCPU, 4 GiB of memory, and 50 GiB SSD.

An ideal production architecture will include several servers and a load balancer and queue for handling multiple concurrent requests. However, for the purposes of this project and developing what resembles a proof-of-concept, that level of complexity may not be necessary. Since we are using ISU servers, we may have to develop these tools ourselves.
A more applicable production architecture will include between one and three servers and a database with multiple read replicas. Since the application will not be creating, updating, or deleting data, our focus will not have to include dealing with concurrent writes to the database.

To improve runtimes and response times for some algorithms, our team may use the Apache Hadoop library for large-scale data analysis. In this case, we need a cluster of servers for distributed computing. ISU has an on-site server cluster that will accomplish our goals.

2.8 Financial Requirements

Our project will be hosted on university machines; therefore, there will be no cost associated with this project.
3 Design

In this section we will discuss the design of our project and how the individual components of our design will interact together.

3.1 Previous Work And Literature

In order to accurately design and implement the clients vision we had to research existing implementations of clustering and algorithms that manage similar data.

3.1.1 Relevant Background/Literature

Our two main clustering algorithms we found were K-means and DBSCAN. These two algorithms will provide a crucial foundation to the algorithms for our data.

K-means clustering is a clustering algorithm that aims to partition data into k clusters in a way that data points in the same cluster are similar and data points in different clusters are farther apart. This will allow us to group data together based on a defined distance function (Euclidean distance) [1].

DBSCAN Clustering stands for Density-Based Spatial Clustering of Applications with Noise. This type of clustering is able to find arbitrary shaped clusters and clusters with noise. The main limiter of DBSCAN is that a point belongs to a cluster if it is close to many points from that cluster. There are 2 key parameters of DBSCAN: eps and mindPts. Eps is defined as the distance that specifies neighbors. If the distance between 2 points is less than or equal to eps then they are classified as neighbors. MinPts is the minimum number of data points that define a cluster [2,3].

3.1.2 Describe what other products exist in the market

The other products that exist in the market are similar databases with different types of visualizations. Since our goal isn’t visualization the existing products are useful for getting a better grasp of the database queries we will use. In these tools all of them have some type of “query database” section that shows the database tables based on some user query. They also all have some type of visualization that may help the user better understand the information in the tables [4,5,6].

3.1.3 Differentiate your project from what is available

Our project is taking our database and performing certain clustering algorithms based on user input and returning a graphical representation of the determined user inputs. This is quite different from the links provided above since they chose to visualize their data within the scope of the universe. We will visualize our data using specific clustering algorithms in a graphical representation.

3.2 Design Thinking

In designing our project, we have identified and defined needs which will be addressed in our final product. Our users need a way to query and visualize data regarding binary stellar trajectories, in order to facilitate their research. The astrophysicists for which this product is intended need to identify patterns in the given heterogeneous data sets. These needs drive our design, and inform our choices. We know that our solution will involve a web application, which further limits our possible solutions. In the following figure we illustrate our high level architecture, based on our ideation of our product.
Web Server and Algorithms (Backend framework):  
In order to properly and efficiently parse and format data we will need a server to run algorithms on large bits of data. For this we will need a backend framework and subsequent language specific packages to help create the algorithms.

Store Data (SQL Database):  
Our data sets are very large, and in order to efficiently query and run our algorithms, we need a SQL database or distributed database.

User Interface (Client web browser):  
We need to implement a simple, intuitive user interface that facilitates use of our web application. To facilitate this implementation, we intend to use a client-side framework.

3.3 Technology Considerations

In order to complete all the components mentioned in the previous section, we had to consider 4 main areas of technologies to fulfill our design: Backend, Algorithms, Database, and User Interface. Our considered frameworks are shown below in the following table, with an explanation in each relevant subsection.

<table>
<thead>
<tr>
<th>Backend</th>
<th>Packages/Languages</th>
<th>Database</th>
<th>User Interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>Django</td>
<td>Python</td>
<td>MySQL</td>
<td>HTML/CSS/JS</td>
</tr>
<tr>
<td>Flask</td>
<td>PHP</td>
<td>PostgreSQL</td>
<td>Tailwind CSS</td>
</tr>
<tr>
<td>Node.js</td>
<td>Java</td>
<td>HDFS</td>
<td>Bootstrap</td>
</tr>
<tr>
<td>Laravel</td>
<td>JavaScript</td>
<td></td>
<td>SCSS</td>
</tr>
<tr>
<td>Springboot</td>
<td>scikit-learn</td>
<td></td>
<td>ReactJS</td>
</tr>
<tr>
<td></td>
<td>NumPy</td>
<td></td>
<td>Vue.js</td>
</tr>
<tr>
<td></td>
<td>pandas</td>
<td></td>
<td>AngularJS</td>
</tr>
<tr>
<td></td>
<td>Django REST</td>
<td></td>
<td>D3.js</td>
</tr>
<tr>
<td></td>
<td>Express.js</td>
<td></td>
<td>Chart.js</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Vega</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>three.js</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>P5.js</td>
</tr>
</tbody>
</table>

Table 6: All considered technologies
3.3.1 Backend

Our first, and most important, module we had to consider was our Backend framework and language. Because 6 out of 7 team members had developed in Python, we narrowed the selection down to Python, and Django or Flask. From these two frameworks, we decided on Django over Flask, because Django has a larger community, more built-in tools, and a gradual learning curve.

3.3.2 Algorithms

To analyze the data extracted from the database, we need to perform some data analytics prior to the server sending a response to the client. We plan to use Python and some of its libraries because it is the most widely-used language most used for data analytics. In addition, writing PySpark code in future iterations for distributed data analytics will interface well with the Django framework we plan to use for a server.

As described in Section 2.7, we plan to use PySpark in future iterations to speed up transaction processing.

3.3.3 Database

Because our data was given in SQL, the experience of our team, and because Django supports these frameworks out of the box we had to decide between MySQL and PostgreSQL. From the advice of our advisor Goce, and because one of our datasets was given as a PostgreSQL database, we chose PostgreSQL.

With the anticipation of processing large amounts of data, in future iterations, if we use Apache Spark, we also plan to use the Hadoop Distributed File System (HDFS) to store the data in a distributed file system.

3.3.4 User Interface

Since our goal is to create a website to act as an interface to the backend, we will use the fundamental HTML/CSS/JS stack along with some frameworks. After designing an initial UI, shown in the following section, we decided this project did not need a frontend Javascript framework like ReactJS, Vue.js, or AngularJS. For quicker development and more appealing aesthetics we had to choose between the CSS frameworks Bootstrap and Tailwind CSS. Bootstrap is a commonly used CSS framework that comes with a plethora of UI components and utility classes. On the other hand Tailwind CSS is a newer, lightweight framework focused solely on its versatile and easy-to-use classes based on its utility first principles. Since our UI is designed with a focus on simplicity, Tailwind CSS's lighter size will make our project easier to develop and faster to render.

Our data visualization will only need to render 2D charts, for this reason we were considering D3.js or Chart.js, which is built on top of D3.js. Chart.js provides a simpler and cleaner usage than D3.js specifically for charts, so we chose Chart.js as our primary data visualization tool.

3.4 Proposed Design

From our chosen frameworks in the previous section and the higher level design in section 3.2, the following lays out a detailed map of our proposed design. First we show how each of our frameworks interact with each other, and then our initial User Interface design.
3.4.1 Web Application Architecture

The following sections will show how these components interact with each other to create a cohesive, scalable, and maintainable web application.

**Client Side:**
Our client side will consist of a webpage that will allow users to select their desired distance functions and set parameter weights. Chart.js will be used to render the data output as a chart, which will facilitate the visualization of data. Tailwind CSS will provide a fast and easy styling to our webpage with minimal CSS code provided.

**Virtual Machine:**
On our virtual machine our primary systems are nginx, Gunicorn, PostgreSQL, and Redis. nginx will be our primary web server, and will be our initial interaction to incoming requests on port 80. nginx will spawn a new process that will tell Gunicorn to run a WSGI server for our Django instance. The Django instance will also interact with a PostgreSQL database for data storage, as well as a redis cache for caching of data to speedup response times.

**Django Instance:**
The Django instance will house the logic and functionality of our project, as well as the algorithms that will interpret our data.
3.4.2 User Interface

Using the frontend tools we chose in the previous section, our UI will be built with three main parts: a data rendering section, a distance function selection, and an attributes weight section. The data rendering will contain a graph and data table showing users their results. From the distance drop down a user can change their preferred distance function. Finally, users can adjust the weights of attributes in the final section. When a user would like to see their resulting data they can click the submit button to render the new data.

3.5 Design Analysis

The strength of our design plan lies in the fact that all members of the team are experienced with at least one of the frameworks, and all of the frameworks are well known and reliable. Using reliable frameworks is helpful as there are resources available to overcome many of the issues that may be encountered during the development process. Using the combined resources of each team members knowledge and outside documentation will prove invaluable to our team.

Each framework provides a unique solution to a problem, and when frameworks are combined then many of our issues are solved. However, unforeseen problems may arise when implementing possible solutions. Some possible issues include developer knowledge of proposed frameworks, and other unknown factors.
Pros and cons of each technology were discussed in Section 3.4, so they will not be discussed here. Instead possible modifications will be discussed and evaluated. Some easy modifications to be made are different implementations of the User Interface. As the UI is the key component of our product that our users will see, it is imperative that it is smooth and easy to understand. Modifying our UI is simple to do and does not create deep-seated issues in development. Modifying does not seem to be needed. Each backend framework has been proven to be useful in many other completed well received products and our team does not currently see any modifications to be made besides how each framework interacts with the others.

Overall, the proposed design is solid and well thought out and there are not many complex modifications to be made.

3.6 Development Process

For the development process, we will be following an Agile methodology with a close implementation of Scrum. Given that the project size is relatively small, the waterfall model could have been considered. However, this methodology would quickly fail due to the constant updating of the project and the receival of new information that changes aspects of the design. Agile and Scrum allow for quick adaptation to these changes in a way that will allow for success.

3.7 Design Plan

Our design plan is not a perfect reflection of the structure of our architecture. Rather, it is an abstraction of how we are breaking down our project into manageable components/teams.

The next part will show the work done and plan relative to the use cases and requirements that component is responsible for.

Algorithms
What we have done:
- Research on how to find proximity and diversity of data
- Research on possible algorithms to find proximity and diversity

What we are planning to do:
- Decide on which existing algorithms to use in our algorithm
- Decide how to implement the overall algorithm
- Implement the algorithm in program code

How this fulfills use-cases / requirements:
- R3: The algorithm displays results in a timely manner
- R4: Users determine which attributes require proximity/diversity
  - Users can select different weights for attributes

Backend
What we have done:
- Set up server
- R2: Secure and efficient data storage
- R6: Data entry
- Setup initial infrastructure and database
What we are planning to do:

- Create the viewpoints
- Create the interface between algorithm and frontend

How this fulfills use-cases / requirements:

- Web application
- Publicly available

**Frontend**

What we have done:

- Created the design mockup

What we are doing:

- Finalizing design

What we are planning to do:

- Setting up the wireframe with no CSS and minimal JS
- Styling the wireframe to make the user interface more visually appealing
- Create Chart.js JSON objects

How this fulfills use-cases / requirements:

- Users need to visualize data
- need to input desired query parameters
- what distance function is used

Our Design Plan is shown in Figure 5 with the 5 main steps: Empathize, Define, Ideate, Prototype, and Test. After we talked with our clients we gathered information on the project and defined our requirements as mentioned in Section 1.4. With these requirements we created our solution to the infrastructure and design as shown in Section 3.4. In our second semester we will use an Agile process to Prototype and Test our application.
Figure 5: Design plan

- **Empathize**
  - Gather general information on proposed project
  - Empathize with project lead to understand what the true problem is
  - Ask questions when unclear on certain material, ask "why."

- **Define**
  - Define tasks for sprint
  - Create requirements for sprint
  - Understand the problem to be solved

- **Ideate**
  - Brainstorm solutions and responsibilities for issues
  - Create diagrams / UI Docs

- **Prototype**
  - Write quick and easy to test code based on the proposed solution
  - Push code to staging
  - Write test cases

- **Test**
  - Show work done to client
  - Determine that solutions are satisfactory
  - Verify test cases succeed
4 Testing and Implementation

In order to make sure the project fulfills all requirements, and allows for better maintainability, testing will be an important part of development.

4.1 Unit Testing

Unit testing is the smallest form of testing that will make sure individual components of our project run correctly.

Algorithms
For each distance function, unit tests will cover the following conditions:

- No inputs return empty set
- Invalid inputs raise a ValueError
- Valid inputs and data return a response dataset that can be verified with a known correct dataset.

Business Logic
The main purpose of our business logic is to validate and sanitize user input. To test this for each user input field, we will test the following conditions:

- If the user inputs an invalid value, the system will replace it with a default value.
- Value must be an integer, if not replace with default value

Database queries
For each query we create will write tests for the following conditions:

- No data in the database returns empty queryset
- Invalid inputs raise a ValueError
- Valid inputs and data return a response dataset that can be verified with a known correct dataset.

4.2 Interface Testing

We will use Django Test Cases to make sure the data returned from our algorithms and database render to correct HTML. For all the edge cases and distance functions we will write a Django Test Case that will test if a valid HTML response is returned. The acceptance case for these tests will be a response with a 200 status code and valid HTML. Any other response will be considered a failed test.

4.3 Acceptance Testing

Acceptance testing is an important part of development, for both Agile and for making sure we are developing the product the client wants. Before interacting with the clients we will make sure that all requirements listed in Section 1.4 are fulfilled.

While asking for feedback from the client, we will have them use the web application to evaluate the following areas for correctness and intuitiveness:

- Distance functions
- User Interface
- Visualizations
- Assigning weights of attributes
- Results returned in timely manner
4.4 Results

We have explored our designs with our client to verify the intuitive approach we will use for our user interface. From these results we added in data visualization in the form of a Chart.js graph that will allow users to more easily perceive and understand the data.
5 Implementation:

During this semester, the focus was on the design document and only preliminary setup was done for implementation. To get our project ready for next semester we created a minimal Django code base and set up an Iowa State VM to serve our application. We also seeded our server database with the dataset we will be using for the project, and we integrated all the internal connections to access our database and display data in our web application.

Going into next semester, we will implement five core tasks: algorithms, user input, frontend design, data visualization, and testing. The user input, frontend, testing, and data visualization have excellent documentation that will integrate well with our current infrastructure. On top of the well documented code base, many of our team members have experience in one of these areas. Our algorithm implementation will be our most complicated section, because of the lack of direction and experience our team has in that domain.

Algorithms:
The core of our project is our algorithms, and how we parse our data. Our main goal while implementing the algorithms is learning and narrowing down the types of clustering that we will use. Since most of our team is new to the idea of clustering algorithms, this aspect of our project will require the most time to understand and implement. However, several Python libraries like scikit-learn have clustering algorithms that we can utilize, that will make this problem feasible and be developed quicker.

Frontend:
The frontend's primary focus is the style and display of our user interface. Since we will use Tailwind CSS for styling, it will allow our frontend team to create styles faster and more easily because we will not have to worry about creating custom CSS classes.

User Input:
To make user input simple we will use Django's built in Form class that will handle input verification validation and form building for us. Combining this Form class with the template tags that Django provides will make form implementation easy to manage and test.

Data Visualization:
For this project, data visualization will take the form of a multi-line chart that contains the data corresponding to the parameters specified by the end user. We will be using Chart.js to accomplish this task. Chart.js is a well-known and widely-used library that has a large community of developers and thorough documentation. We will be using those to our advantage to easily create a functional chart for data visualization in a short period of time. This aspect of the project is highly feasible for the reasons mentioned above and will be implemented in our project.

Testing:
As a part of our CI/CD we will run both our unit and interface tests before new code is pushed to our production environment. This will ensure that all code that is running on the server will pass every test we created. We will also have two demos with our client before our final deadline that will give us our acceptance testing for our project.
6 Closing Material

6.1 Conclusion

Our goal is to create an easy way for researchers and astrophysicists to visualize or view large heterogeneous datasets of binary stellar trajectories in meaningful ways or patterns. We will create a web-application which will have a focus on proximity vs. diversity and contain data previously collected from astronomical observations. When our product is complete, our users will be able to select distance functions for different data parameters, as well as the weight associated with each data parameter.

This semester we have focused primarily on creating a design plan, infrastructure design, researching algorithms, and getting a server setup for next semester. To make sure we are prepared to hit the ground running next semester, we got an initial code base and a simple landing page ready and accessible to the public.

Going into next semester we will dive into implementation using Agile as our guiding method. Our three teams will work to finish their responsibilities mentioned in section 3.7. The primary goals will be to finish our algorithms, creating an elegant user interface, and combining the two with an efficient backend interface.

As mentioned in section 3.3, we feel that Python, Django, HTML/CSS/JavaScript, and PostgreSQL are the best languages and frameworks for this web application due to reliability and developer familiarity. We believe these tools, accompanied by our design plan our group created, will provide the best end product for our client. When we bring together our technologies, team experience, and Agile-enabled fast-paced work environment, we will be set on a path to create a web application that will exceed the expectations of our clients with a maintainable and scalable infrastructure.
6.2 References


6.3 Appendices

Django - https://www.djangoproject.com/
Python - https://www.python.org/
Tailwind CSS -https://tailwindcss.com/
Postgres - https://www.postgresql.org/
Redis - https://redis.io/
Gunicorn - https://gunicorn.org/
nginx - https://www.nginx.com/
Chart.js - https://www.chartjs.org/
pandas - https://pandas.pydata.org/
NumPy - https://numpy.org/