Indiana High School Athletic Association Project Final Report

EPICS Spring 2021

Team Members:
Dalton Morzos
Kameron Leisure
Chloe Makdad
Aleksandra Grigortsuk
Adam Crozier
Grant Hurn
Grant Bryant
Audrey Marjamaa

Table Of Contents

Introduction

The Indiana Highschool Athletics Association (IHSAA) has asked the EPICS team to create a web program that can take information about every school in the state and organize these schools into groups based on sport and based on classification. This program should also be able to take information pertaining to each school and adjust classifications.

The two main parts of the project are reclassification and realignment. Reclassification has a list of by-laws that explain the process of classifying any school in IHSAA. Our group focused on tackling the realignment piece.

Realignment has two main pieces associated, which became the main two subgroups of the project. The first is the front end web page that is displayed and handles user interactions. The second piece is the back end calculation piece that the program needs to handle in order to tackle the process of realignment.

The way in which realignment would ideally work involves the user submitting predetermined parameters into the web page. The page sends that information to a translation function that grabs the coordinates of each school in the state. After the function sends that to a program that then determines which group each school should be in. Finally, this algorithm returns its groupings back to a translation function that decodes which school goes in which group, so that the front page can display several lists.

Motivating this process is the problem of doing it by hand. In order to effectively do this process by hand, the information of more than 400 schools must be read by hand, then the process done by hand which takes a great deal of effort and mistakes are easy, so a verification process would be needed and so on. Having a computer do the process is a natural choice.

Approaching this problem in more depth, we split the group in order to better tackle the two main pieces. The web development piece and the algorithm piece. The web page needed to be developed and then also become functional in order to act as input for the user. This process was done using Angular, a framework for web development, in order to develop a page. The algorithm piece mostly consisted of determining which algorithm to use, and that is determined by what can be done with typescript and where can we find code for said algorithm. We settled on a K-Means++ derivative after validation; it gives good groupings for schools. The challenge then became how do we create an input and an output for the schools involved.

The report is organized as such: Glossary and terminology to be used throughout. Followed by a description of functional features, this also includes nonfunctional requirements and assumptions and constraints. After, there is a more detailed architecture of the current state of the project, focused more on what has been done or explored. Next, followed by a closer look at the design and its theory. Followed by some implementation of that theory, and why this design was chosen. Then, a discussion on the validity of the algorithm and some quality assessment. Then, the project organization is explored in more depth, how the work was split up. The next section is future work, the project is far from complete, so an exploration of what next is discussed. Lastly, a list of references and appendices for the different pieces we talk about throughout the report.

Glossary and terminology

- **k-means:** The k-means algorithm is an iterative algorithm for partitioning data into *n* groups or clusters
- **k-means++:** The k-means++ algorithm is a method of centroid initialization used to avoid poor k-means clustering
- **Component:** A subset of directives used by Angular to create application parts, such as drop downs or algorithms.
- **HTML:** A coding language designed for web pages, used for visual and functional work
- **CSS:** A coding language used to create stylistic choices as well as some minor functional grouping
- **Typescript:** A superset of Javascript, meant to be largely functional and has great flexibility.

Requirement Specifications

The 3 main functional pieces for the project, a part that calculates realignment, another that calculates reclassification and a web page to take input from the user. This group would create two separate pages, one for realignment and the other for reclassification. The reclassification page would then calculate, display and then also export the results as a PDF so that IHSAA can distribute and display the results elsewhere. The realignment page would take several inputs, including which sport, what classification, what year, and how many sectionals would be made. After which, this info would be compiled and sent to the background algorithm which would then return an array. The page would decode this array into a series of lists that have the name of the group, the average distance and the host school marked. The page should also access a database provided by the client and the page should be accessible from their website. Authentication is implied.

For this project, many specifications revolve around the front end application. Primarily, it would have to be done using Angular- a software framework used in developing HTML webpages. The application also needs to be easy to traverse, and pleasant on the eyes. The client would like for it to be shared online, so aesthetic is a major priority. On top of that, the application would need to accept information from the user and create a visual representation of the reclassification and realignment of Indiana schools into tournament groups. As a potential stretch goal, the client would also like some interface functions that their Tennessee counterpart deploys, such as tournament manipulation using drag and drop functionality.

For the backend, the algorithms will have to be coded using Typescript, as Angular requires it to manipulate and communicate data to the front end. It will also need to run algorithms that reclassify the schools into their respective groups, and then realign them such that users can easily identify what said groups look like. The algorithms will need to be robust, allowing short processing time and constant accurate results. They will also need to work based on user input, so the application will have to define a search result based on said input, query an Azure database to find relevant data, and parse that data so that the algorithm can function properly. The groups will need to be coordinated such that no two schools result in long distance driving, so the algorithms will need to take into consideration both geographical and geometric locations in order to prevent this.

For this project, we assumed much about the algorithms themselves, alongside quality of life changes. Primarily, we were given the freedom to select how the algorithm reclassifies and realigns the data, so we assumed that a k-means algorithm would be able to properly fit the constraints of the project. We also had the freedom to design the initial page, so we forgone a login page at the moment so that we may develop a working app more quickly. We also assumed that the connection form the backend to the database will have to be secure, so that the client may be able to test the project without risking loss or destruction of their private data.

Architecture

The application consists of a web page and an algorithm. In its current state, both the web page and the algorithm exist within the project files. The realignment algorithm runs and there is a visual component in displaying the output. The webpage has a drop-down menu but is non-functional.

The application allows end-users to specify the classification, gender, and the amount of sectional groupings for each sport. The algorithm runs the calculation and gives an output.

As it stands there are two components. These consist of the web page and realignment algorithm. The information taken in from the web page is designed so at some point it can be used for a database.

The web page is made to fit the type of information that runs in the algorithm. When the end-user types in the number of sectionals, that value is stored as its own variable for the amount of clusters (sectional groups) that the algorithm will generate.

The web page component handles end-user requests on the front end. The algorithm takes in the specified data and generates clusters of schools surrounded by a centroid host school on the back end.

Also within the angular project files is a reusable table component that can take a variety of specifications for the columns and data format. The component also has an input for a set of data that can then be displayed visually on the front page. The table component does not have a way to be connected to the back end at this time.

The project is a web-based application. It is accessed through the internet and everything from the end-user experience to the calculation is run on a browser.

The web page uses the Angular API. This utilizes software such as TypeScript, CSS, and HTML. It also contains a node.JSON package through JavaScript. While the algorithm runs in TypeScript, the output is then fed through Python and GeoPandas for visualization purposes.

The application also implements an Azure Database, per client request. The schema and naming conventions are provided by the client, so the database needs to be constructed to replicate their design, so that future implementation will be easy to facilitate. The database contains the following tables, with corresponding columns:

- SchoolAddresses - Id, AddressType, City, Concurrency Stamp, CountryCode, Created, LastUpdated, PostalCode, SchoolId, StateProvidenceCode, Street1, Street2
- Schools - Id, AcademicScheduleId, AthleticDepartmentFaxNumber, AthleticDepartmentTwitterId, Colors, ConcurrencyStamp, ConferenceId, Created, IhsaaSchoolNumber, IiaaaDistrictId, Inactive, LastUpdated, LogoFileId, Name, Nickname, SchoolFaxNumber, SchoolPhoneNumber, SchoolTwitterId, SchoolWebsiteUrl, NumberOfCourses, IhsaaNorthSouthDistrict, HytekCode, ArbiterID, IsEasternTime, SchoolType, InteacctCustomerID
- SchoolSportAssignments - Id, AssignmentId, ConcurrencyStamp, Created, LastUpdated, SchoolId, Classification, IsExcluded, IsOffered, ADSSubmittedSurvey, CoachSubmittedSurvey
- SportGenders - Id, ConcurrencyStamp, Created, LastUpdated, Name
- SportNameGenderAssignments - Id, ConcurrencyStamp, Created, LastUpdated, SportGenderId, SportNameId, SchoolYear, CommissionerContactId, AOAEndDate,

AOAStartDate, AppGamesDueDate, ContestSeasonStartDate, EntryDueDate, FirstPracticeDate, MtgAttendenceDueDate, PartIITestEndDate, PartIITestStartDate, RatingSurveyEndDate, RatingSurveyStartDate, RulesEndDate, RulesStartDate, SectionalStartDate, TournAppEndDate, TournAppStartDate, TournamentDrawDate, TournamentOfficialPubDate, HasCrews, MinimumVarsityEvents, RegionalPayment, SectionalPayment, SemiStatePayment, StatePayment, UnifiedTourneyApps, ClinicAttendenceEndDate, ClinicAttendenceStartDate, MtgAttendenceStartDate, VisibleOfficialAssignments, Classifications, AdminAsstContactId, AthleteEntryInstructions, PerMilePayment, EntryListReleaseDate, IntacctDepartmentNumber

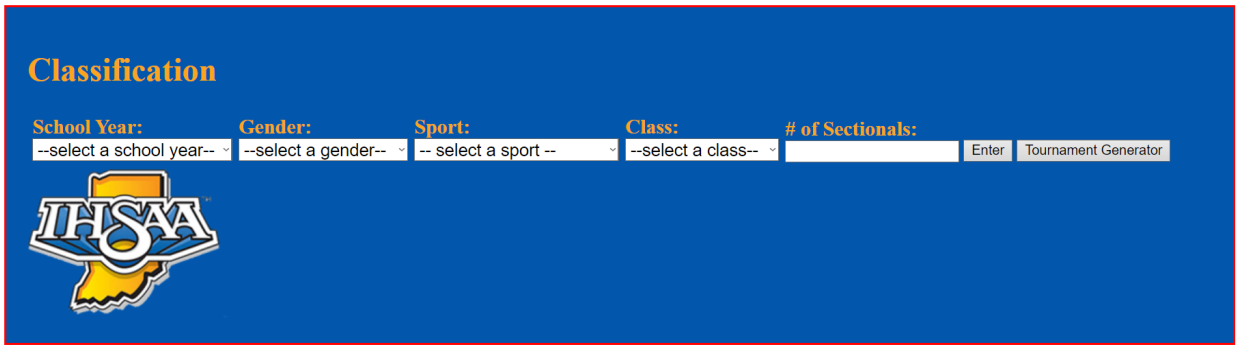
- SportNames - Id, ConcurrencyStamp, Created, LastUpdated, Name, IsTeamSport, ArbiterName
- TournamentBrackets - Id, AssignmentID, BracketNumber, Classification, ConcurrencyStamp, Created, LastUpdated, RoundNumber, OfficialCount, HostID, FinancialSchoolID, ReportID, AcctBracketNumber
- TournamentHosts - Id, AssignmentID, Classification, ConcurrencyStamp, Created, LastUpdated, RequiredGAmes, RoundNumber, SchoolID, SportGenderID
- TournamentParticipants - Id, BracketID, ConcurrencyStamp, Created, LastUpdated, PBracketID, SchoolID

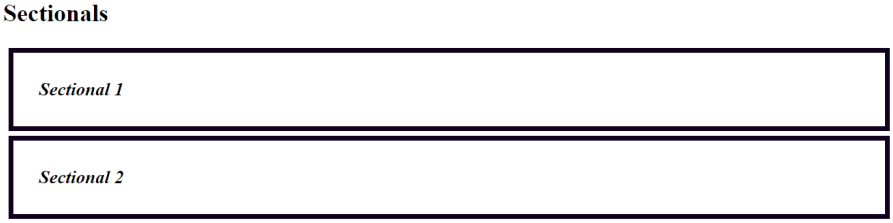
Design

The vision for the design of this project was to create a usable web page that would handle user inputted constraints and output a solution for realignment of Indiana high school sports teams.

On the front end of this project, we designed a interactable web page that has drop down menus for school year, gender, sport, and class as well as a input box for the number of sectionals. These were designed to be the user input that would be fed into the algorithm to provide an alignment for high school sports teams. Other current visual features of the web page include a logo of IHSAA that functions as a button to link the web page to the IHSAA main website.

There are also sectional boxes that are meant to function as a landing site for the output of the algorithm to be displayed. Lastly, as part of the front end is a reusable table component that can take any set of column definitions which also can specify data format for the data in that column. The table also can take any array of data that matches with the customizable column definitions. We also made the effort to match the overall color scheme and aesthetic of the organization IHSAA. Below are visuals of the current web page and an example of what a data table output could look like. Depending on the output of the algorithm, different data could be displayed on the front end web page. The design of the display can be similarly changed with the flexible table column definitions.





#	School Name	Average Distance (miles)	Host?
1	Carmel High School	34.3	No
2	North Central High School	32.1	No
3	Cathedral High School	18.2	No
4	Bishop Chatard High School	23.6	No
5	Lawrence North High School	37.0	Yes
6	Lawrence Central High School	26.9	No
7	Fishers High School	39.4	No
8	Riverside High School	27.7	No
9	Ritter High School	29.5	No
10	Pike High School	40.2	No

Implementation

The backend half of the team implemented Typescript for algorithm, Python for validation. Due to the selection of Angular to design the application, we decided to use Typescript to implement the algorithm as it is the primary language of Angular. This allows for easier integration between the frontend and backend. We would want to have the user give input parameters into the clustering algorithm, run the algorithm, and return the output in a readable format to the user. While we implemented the algorithm Typescript, should future groups decide that they would rather have the algorithm run with queries, we have included a source on how the algorithm might be run with SQL in our references section.

To validate the algorithm, we took the data output by our Typescript files and shifted that process into Python for a number of reasons. First, some members of the backend team were more familiar with handling data in Python than in Typescript. Second, Python has robust tools and packages for data visualization, manipulation, analysis that Typescript does not possess. Third, since the validation component of the project does not appear in the final application, the use of Python does not create any additional hurdles with Angular. More about this implementation and its file organization can be seen in the “Quality Assurance and Testing” and “Project Organization and Management” sections.

The frontend team implemented CSS, HTML, and Typescript to set up the webpage. Angular has things called components which nicely separates the html, css, and typescript portions. For example we gave each dropdown menu (gender, class, sport, and school year) their own component. Angular gives us the freedom to customize each certain dropdown menu to our preferred style, display, and functionality.

The database did not reach the implementation stage in the end. The process for connecting the database to the application took too long to conceive, and thus attention was turned elsewhere, such that more tangible work can be seen by the client. In light of this, A document was created to be given to future groups such that implementation of their own database can be quickly achieved. This document explores: how to set up an Azure Subscription, how to create a Azure database, importing data, having multiple users access the data, and concerns about owning an Azure database. This should allow future database administrators to setup in half the time required by this team.

Quality Assurance & Testing

Collecting Comparable Data

To start the process of realignment, grouping schools into sectionals, it was necessary to collect a set of data points to be the factors in determining these groupings. The first idea was to collect the distances between all schools using Google Maps API Javascript. While this provided accurate results, this meant a total of 85,078 data points. Not only was this a large amount of data, it was also not possible to gain all this information easily because the Google Maps API is a paid service that limits the amount of calls for data we could do a day unless the user pays to exceed this limit. The total cost would be hundreds of dollars, which made this not a possible option.

But the Google Maps API also gave us the ability to receive another type of data points: longitude and latitude. This was only 413 calls for data, which did not exceed the limit of the API. From this, we have comparable data to use in the k-means++ clustering algorithm. We saved this data in a Google Sheets file. This should be implemented into the database.

Algorithm Output

Once we determined our approach to clustering the high schools in Indiana through k-means++ and found Github repositories which we could employ, we needed to test the output of the algorithm. But before this could be done, it was necessary to organize the output so it could be usable.

GoldinGuy/K-Means-TS's output was the number of iterations, number of clusters, array of indexes, and an array of centroids. The index array was an array the size of the input array. It contained numbers 1 to the number of clusters. The index of the index array corresponds to the index of the input array, and the value of the index array is the number of clusters which the input array value belongs to. We create an array that is the size of the number of clusters. Each index of this array will contain an array of the schools (coordinates) which are in the cluster of the index number.

The second type of output we want is connected with the array of centroids GoldinGuy's algorithm provides. These centroids are coordinates that are not from our input array of school coordinates, and these centroids act as the host school of each sectional. We create an array that is the size of the array of centroids, and run an algorithm that finds the school closest to their cluster's centroid.

Jalarrea/kmeans-same-size has a unique output as well. This algorithm, while it provides more even sized clusters than GoldinGuy/K-Means-TS's algorithm, it does not calculate centroids. It provides an array with the value in the format **{x: -1.1, y: -1.1, k:5}**. This is a more difficult output to organize than the other k-means++ algorithm because it has a combination of strings and numbers. The output is the longitude, latitude, and cluster number. We create a new array similar to one we created from GoldinGuy/K-Means-TS's output, but we need to account for the differences in format.

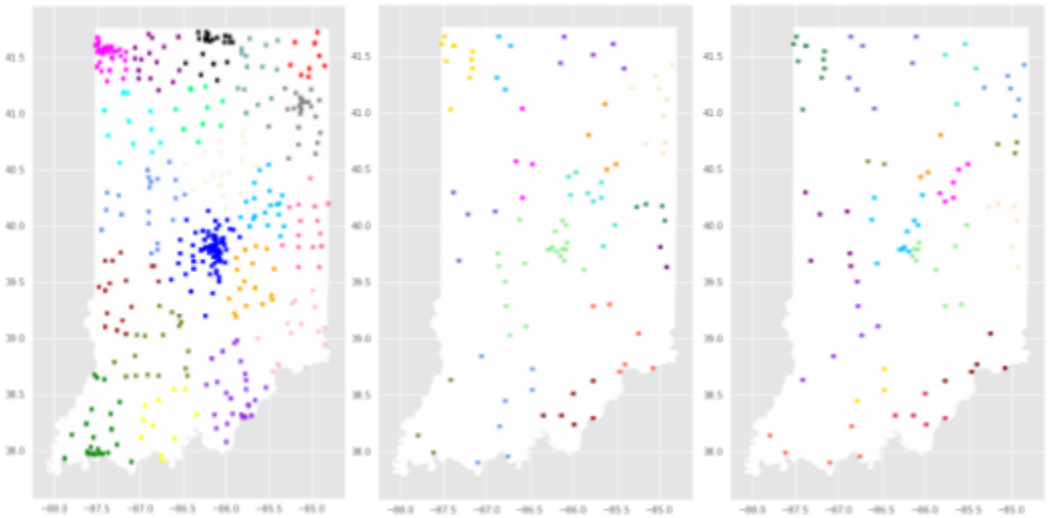
We need output from both of these algorithms. GoldinGuy/K-Means-TS gives us our centroids, and Jalarrea/kmeans-same-size gives us the even sized clusters. With the organized outputs from the algorithms, we can test the validity of these clusters and centroids.

Algorithm Testing

The ultimate goal of realignment is to create new sectionals, which should be evenly-sized and contain schools that are geographically close in terms of both physical and driving distance. To test whether our algorithm could successfully do this, we put it through the following validation process.

Though there are numerous algorithms and methods for clustering data, we ultimately chose to test two algorithms on our data: k-means++ and graph spectral clustering. Given the pre-existing code libraries implementing k-means and k-means++ in Typescript, we started by testing and validating those algorithms. Our first test was simply to run k-means++ on the entire data set, meaning every school in the IHSAA system with no regard for sport, gender, or classification. Since the only purpose of this test was to verify that the algorithm could indeed cluster the schools, we arbitrarily chose to create twenty clusters. Once we had the output from the Typescript programs, we transitioned from using Typescript to using Python. The results of this first test can be seen in the below figure as “Validation Test 1.” From this test, we saw that the k-means++ algorithm could cluster data and was worth further pursuing.

Once we knew that the algorithm could work, we began testing it on realistic data. After collecting the 2A Boys Baseball data, we fed that data to the k-means++ algorithm implemented in the GoldinGuy repository, which can be found in our references. Our client had previously indicated that IHSAA prefers to have eight or sixteen sectionals per sport and classification for the purposes of tournaments. Given the number of schools in this classification, we chose to have the algorithm output sixteen sectionals. We then took the output from Typescript and fed it to our Python files for validation. The output of our first test can be seen in the figure below as “Validation Test 2.” Though the data was clustered, we were not satisfied with the variance of the cluster sizes. In these tests, the sizes ranged from as small as two to as large as thirteen on the same map, which is far too large of a discrepancy for a successful realignment tool.



Validation Test 1: Standard k-means++ on the entire data set

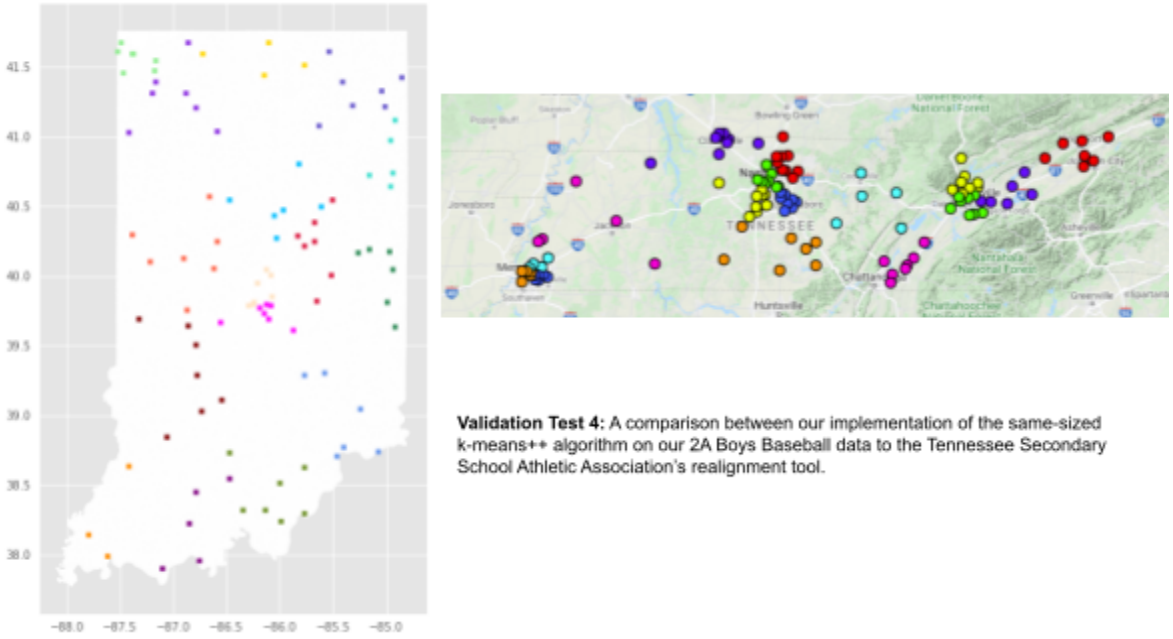
Validation Test 2: Standard k-means++ on 2A boys baseball data with no adjustments for cluster size

Validation Test 3: Graph spectral clustering on 2A Boys Baseball data

In response, we decided to test whether graph spectral clustering was worth pursuing. Because of the extensive linear algebra required to run this algorithm, we used some

pre-existing Python code to implement it. Once again, we sought to create sixteen sectionals with the 2A Boys Baseball data. After feeding our data into the graph spectral clustering files, we got the output seen in the figure below under “Validation Test 3.” We determined that the improvements, if any, were not significant enough to use graph spectral clustering over k-means++ because of the potential difficulties that could arise when trying to implement this algorithm in Typescript.

Instead, we used a slightly different version of the k-means++ algorithm that was designed with the goal of creating clusters of similar sizes. Applying this to our 2A Boys Baseball data, we finally got clusters that were in a reasonable size range. Additionally, we compared our output to the output of the Tennessee Secondary School Athletic Association’s realignment tool, which inspired the IHSAA to create their own. This comparison can be seen below. We determined that the outputs were reasonably similar, with similar clustering patterns occurring in both urban and rural areas.



Validation Test 4: A comparison between our implementation of the same-sized k-means++ algorithm on our 2A Boys Baseball data to the Tennessee Secondary School Athletic Association’s realignment tool.

Now that we had the algorithm outputting evenly-sized sectionals, we looked to see whether the intracluster distances were reasonable for practical use. Once again using Python and the OpenStreetMap API, we found that the average intracluster driving distance was approximately fifty minutes.The output of this test can be seen in the table below.

Number of Schools	Number of Clusters	Cluster/Section Sizes	Avg. Driving Time
98	16	3-7	50 minutes

Project Organization & Management

The team's organizational structure was broken down simply into two different sections to keep things organized with a front and back end team. There was a front end team that included Grant Hurn, Grant Bryan and Audrey Marjamaa who worked on more of the front end web development side of the project. This group focused on web development through Angular which consists of HTML, CSS, and TypeScript. The back end team included Dalton Morzos, Kameron Leisure, Adam Crozier, Chloe Makdad, and Aleksandra Grigortsuk. The role of the back end team was to create the reclassification and realignment of schools using k-means++. Realignment led to creating an algorithm to cluster Indiana high schools into sectionals using k-means++, using databases given to us by IHSAA, and tying in the algorithm to the front end website. Dalton was primarily on the back end but was our team leader so he moved around on both ends to make sure everything was flowing smoothly.

Kameron Leisure's initial role was the job of Database Admin. He was tasked with the creation and management of the Azure database, and to research into establishing a connection to the app. He successfully duplicated the clients database structure and data composition in the first two sprints, but elected to abandon the database afterwards. The establishment of a HTTPS connection from the app to the database took a considerable amount of time, and he claimed that it would have been preferable to refocus effort elsewhere- as the next team could start this portion of the project with a clean slate. In response to this decision he created a Azure database creation and management guide to help future groups get set up with an Azure subscription quickly. After his work with the database, he continued to work on the back end and attempted to find a way to get the front end to communicate with the algorithm component. This was partially successful, as he could not show if the application shared the information correctly.

Chloe Makdad worked on the back end portion of the project, specifically on the data science aspect of the project. She developed an approach to the realignment portion of the project, identified the appropriate algorithms to examine, and worked to assess the efficacy of these algorithms on real IHSAA data using Typescript and Python. Along with Aleksandra Grigortsuk and Adam Crozier, she also began work on formatting the output of the algorithm for use on the front end of the project.

Aleksandra Grigortsuk also worked on the back end portion of the project, mainly on implementation of Google Maps Javascript API and k-means++ algorithms. She researched Google Maps API for calculating the coordinates of each school and their driving distances. She implemented k-means++ and same size k-means++ algorithms.

Adam Crozier worked on the back end portion of this project. This included examining code already available on Github that would fit the needs for this project. From this, he helped the team hit the ground running on the k-means++ algorithm. He also assisted with finding a way to read in data from end-users and using that in tandem with the algorithm.

Grant Bryant worked in the front end portion of this project. He helped develop the initial main page using HTML and CSS. This means the dropdown menus, a link to the main IHSAA website, the sectionals boxes, as well as the color scheme was designed in order to make the process simple for whoever runs the application. This was coded inside Angular which uses the Typescript, HTML, and CSS components all together.

Grant Hurn worked on the front end side of this project. He helped initialize foundational code through HTML to get things rolling so that other front end team members could add on. He helped where he could in regard to the coding in CSS and TypeScript which all correlated together through Angular. He learned Angular throughout the semester in order to contribute back to the team. He also helped out on some non-technical assignments such as the EPICS team website and the final report poster.

Audrey Marjamaa also worked on the front end for this project. Her initial focus was on learning how to utilize angular and how to code in TypeScript, CSS, and HTML in order to begin creating a basis for the web page. Her specific focus shifted to the display of the output of the algorithm onto the front end. She utilized the features of angular, specifically the material module, to design functional data tables where the data will be displayed.

The work was divided up clearly between both the front end team and back end team. To reiterate, our team leader was Dalton Morzos who worked primarily on the back end team but also supervised the front end team as well. The work on the front end team to develop the website was dispersed evenly through the use of GitHub. The three team members on the front end team worked through Angular in order to set the foundation through HTML, CSS, and TypeScript. There was a huge learning curve on the front end team to learn Angular and understand how all the components work together. The front end team used GitHub to update the code and add additional work while keeping it organized. The back end team work was divided evenly and started using the databases to create a realignment algorithm. This was dispersed between the 5 group members and was worked on throughout the semester. Once the algorithm was created, all the team members worked together to attempt the link between the algorithm and front end website.

The teams management process was kept simple through team meetings during our EPICS class meetings on Mondays/Wednesdays from 3:50-5:05. During this time, each team member would answer 3 questions: what have you been working on, what will you work on now, and tell any challenges you faced. This allowed for the team to plan around challenges and helped each other where it was needed. We did not conduct outside of the classroom meetings because we would clear up any issues during class. We used Github and Angular to upload new and edited code to see the updates on both the front end and back end. The Github application was perfect for keeping everyone's code organized throughout the semester. Lastly, our main source of communication came through a Discord server where we had text channels, file-upload channels, and voice channels to divide up the front end and back end teams. Overall, this was performed by Dalton Morzos to get everything organized for the team and he did a great job over the course of the semester.

In the first two sprints of the project, an Azure database and subscription were used to house data provided by the client. The goal of this was to provide a way for the group to test and manipulate real data to meet the needs of the project without harming the client’s current database. It replicated the structure of the clients database and contained all of the original information, which then was used to extrapolate and store information needed by other algorithms.

How to Run the Web Page

On the front end of our angular application, we utilized a built in angular material table module to make the data tables. This module comes from the angular library, @angular/material as can be seen in the file app.module. The purpose of using this is to provide a more user friendly display of the data. To make a specific table, you can provide column definitions and configurations in the app.component.ts file. There is an example of this formatting in the file currently. There is also an example of a data input as it should be formatted in the same file. This data input is where the back-end will connect back to the front-end display. The basic reusable data-table component design exists within the app-table files. To get a table to display on the web page, you type a line within the app.component.html file. There is an example of this currently in that file that is commented out.

How to Run the Algorithm and Validation Files

Currently, this project uses the k-means++ algorithm to group, or cluster, the schools. The majority of the code relating to the algorithm can be found in the path IHSAA/K-Means-TS-master/runningKMeans/ of the GitHub repository. Important files in the runningKMeans folder include:

- **dataCoordsArr.ts**: A file that takes in latitude and longitude coordinates from a .txt file and outputs them into an array that can be used by the kMeansTesting2.ts file
- **kMeansTesting2.ts**: A file that takes in a data array from dataCoordsArr.ts and outputs files containing both the centroids and the clusters/sectionals
- **findingHosts.ts**: A file that is a prototype for a host school selection component of the app. It takes output from the kMeansTesting2.ts file and outputs an array with the longitude and latitude coordinates of the potential host schools
- **sameSizeDataArray.ts**: A file takes in data and outputs a text file that contains data in a form that will run in sameSize.ts
- **sameSize.ts**: A file that should take in output from sameSizeDataArray.ts and run the same-size k-means++ algorithm on it. Currently, the file has the output from the .txt file from sameSizeDataArray.ts simply copied and pasted rather than reading from the file. However, it does successfully run and output clustered data
- **ClusteringDemo.ipynb**: A Jupyter Notebook that contains our validation testing for the k-means++ algorithm
- **SpectralClusteringDemo.ipynb**: A Jupyter Notebook that contains our validation testing for the graph spectral clustering algorithm
- **kmeans.py**: An implementation of the kmeans++ algorithm in Python which is used in SpectralClusteringDemo.ipynb
- **spectral_clustering.py**: An implementation of the graph spectral clustering algorithm in Python which is used in SpectralClusteringDemo.ipynb
- Additionally, there are numerous .txt files that serve as input and output to many of the files mentioned above

The algorithm was implemented in Typescript using code originating from two GitHub repositories: GoldinGuy/K-Means-TS and jalarrea/kmeans-same-size. In addition to the

documentation we provide, there is additional documentation in each of those repositories that can provide additional detail on how the k-means++ algorithm works in Typescript.

Though we were unable to connect the algorithm so that it works with user input from the front end, we were able to validate the algorithm, and we'll now detail how to replicate our validation process. The data used in our validation is generated by two files. The first is kMeansTesting2.ts, which takes in the coordinates of every school in Indiana and produces clusters and centroids. We stored these arrays in a .txt file called groups.txt and centroids.txt. We also tested the algorithm on a specific classification and sport. For this part of the experiment, we used a file called sameSize.ts, and the output is stored in a file called sameSizeClustered.txt.

There were multiple ways we validated that the k-means++ algorithm is suitable for use in this project. This validation can be found in the file ClusteringDemo.ipynb, which is a Jupyter Notebook. To run this file, you will need to install some programs, packages, and dependencies. First, you will want to install the latest version of Python 3, probably Python 3.8. Then, you'll need to install the latest version of Anaconda. Once you've done that, you'll need to install the following packages in Python: numpy, pandas, matplotlib, descartes, geopandas, shapley, requests, and json. The process for installing geopandas can be quite difficult, but there is a resource that should be helpful in that process listed under "How to install geopandas and dependencies" in the references section.

Once you have everything installed, open an Anaconda window, navigate to the directory containing ClusteringDemo.ipynb, and type "jupyter notebook". This should open a tab in your browser, and you should see the file ClusteringDemo.ipynb. Then, you can run each block using shift+ENTER.

However, before you can successfully run the validation file, you'll need to download the Indiana shape file we used to your local machine. Unfortunately, the file was too large to upload to our GitHub repository, but you can find it in the link provided in the references section titled "Indiana shape file." Download it and save it in a folder named indianaShapeFile in the same directory as ClusteringDemo.ipynb. This should be all you need to run and validate the k-means++ algorithm on the IHSAA data.

The validation runs in two parts. The first part is purely visual. We take the schools or a subset of the schools and cluster them. Then, we plot those clusters on a map of Indiana and see whether they look reasonable. The second part computes the intra-cluster distances. Coupled with the visual, this should give you enough information to determine whether the k-means++ algorithm is sufficient.

In addition to our validation of the k-means++, we also briefly examined the efficacy of graph spectral clustering on the data. While we ultimately decided not to use graph spectral clustering in our implementation, there is a file called SpectralClusteringDemo.ipynb, where you can find more details about graph spectral clustering and see it in action on our data.

Future Work

For a future database connection, the next team will have to set up a Azure subscription and database. A document to help facilitate this is provided. Furthermore, the team will need to establish a connection from said database to the application. Research showed that one possible option would be to set up logic gates through the Azure portal. These can be used to establish an HTTPS connection and query the database using SQL, which would then return information to the app and be stored.

The next step for displaying data of clusters and their centroid would be to find a way to calculate average driving times between distances. Avoid Google Maps API since it is a paid service and can only do a certain amount of calls. Of course, it can be used for validation. Open Source Routing Machine (OSRM) for the backend requires a docker and has a complicated set up; we'd prefer to avoid this method as well.

Interconnect the web page, algorithm, and database. Have the web page read end-user input and store that input as variables: gender/classification/sport/number of sectionals/average sectional size. Once the parameters are specified, store those as corresponding variables and pass them through to the database. Have a filter for input variables so the project does not grab schools (coordinates) which are not relevant to the request i.e. “Girls 3A soccer, 5 sectionals,” and pass the relevant data to the backend algorithm.

Then display the output of the algorithm in a readable tabular form which displays the sectional groupings, the centroid, and the driving distance between the school and its cluster’s centroid. The user should then be able to save the output in some format, such as png, to their device.

The future of frontend work will be checking for accurate and neatly displayed sectional tables so there won't be as much scrolling to see every table for desired calculations. Also going forward, you should be able to drag a specific school from one table to another table. This will then have the sectional table adjust size accordingly as well as show the new computed drive time, new average amount of miles to drive, and new value for teams inside sectional.

Other future front end work will be adjusting the Graphical User Interface to make things look more like the product owner wants it to. This includes presenting an interactive Indiana map similar to the Tennessee map mentioned in the Quality Assurance and Testing chapter.

References

Learning TypeScript and references: https://www.typescriptlang.org/docs/handbook/
Code examples https://angular.io/start
Google Maps Javascript API: https://developers.google.com/maps/documentation
Google Maps Formulas in Google Sheets: https://www.labnol.org/google-maps-sheets-200817
OSRM: https://www.npmjs.com/package/osrm
K means++ same size Github: https://github.com/jalarrea/kmeans-same-size
K means++ Github: https://github.com/GoldinGuy/K-Means-TS
Explaining coding same size k means++: https://elki-project.github.io/tutorial
K means implementation with SQL queries:
http://www2.cs.uh.edu/~ordonez/pdfwww/w-2004-KDD-sqlkm.pdf
How to install geopandas and dependencies: https://towardsdatascience.com/geopandas
Indiana shape file: https://www.census.gov/geographies/mapping-files
Approximate driving distances using Python with OpenStreetMap:
https://towardsdatascience.com/driving-distance-between-two-or-more-places-in-python-89779d691def#1c17
Introduction to Angular Material Tables: https://material.angular.io/components/table/overview &
https://blog.angular-university.io/angular-material-data-table/
Setting Up Angular Material:
https://www.techiediaries.com/angular-material-table/#:~:text=Angular%20Material%20data%20tables%20provide,like%20pagination%2C%20filtering%20and%20ordering.&text=We'll%20also%20see%20how,component%20to%20display%20contact%20details.
Improvement to Data Tables:
https://medium.com/@bouidiaarahmene/angular-make-your-mat-table-more-dynamic-and-reusable-80a379a1b31e