Spatial Visualization of Biodiversity

Ty Skelton, Jasper LaFortune, and Alec Shields

Abstract

The Department of Integrative Biology at Oregon State has collected a large sample of biodiversity data from various sites in the Southwest United States. Handling this data in its raw form requires certain technical knowledge of databases, as well as a bit of patience. This presents a problem for biodiversity researchers and Department of Defense land managers, who need to be able to understand and make decisions about this data easily. Our team will address this problem by creating a web interface for spatially visualizing this data. We will enable users to easily display useful graphs and maps about areas and species of interest, putting the information they care about most at their fingertips.

The Department of Defense made an investment in collecting all these samples so that it could better manage its land. However, extracting meaning from that information is challenging for land managers and researchers alike. A good solution to this problem would allow users to easily access the information that is important to them. Our solution will provide an interface that allows users to select information of interest and see it in map and graph form. Passerby at Expo will be invited to interact with our system and discover meaningful biodiversity patterns for themselves. Oregon State biodiversity researchers will provide feedback on how well our product meets their needs.

Index Terms

biodiversity, visualization

Spatial Visualization of Biodiversity

I. Project Purposes and Goals

This project aims to aid researchers and land managers in drawing meaningful conclusions from a large set of insect biodiversity data. The Department of Defense funded researchers to collect insect samples from many sites across the American Southwest over five years. The resulting dataset is too cumbersome to make sense of in raw form, especially for land managers with little formal knowledge of database operations. Our goal is to create a visualization system that allows these parties to select information of interest, display it on a map, and show useful graphs about it.

These requirements led us to a design comprised of three views. The Filter View allows users to select which data will be displayed by date, season, species, or location. The Map View shows the locations of the selected samples. The Graph View shows graphs of the data selected in the Filter View broken down by location, taxa, and time.

II. Progress

A. Base Website Setup

When we were doing our technology review in preparation for writing our design document, we decided on using Django for our web framework, Apache for our web server, Git for our version control system, MySQL for our database, Bootstrap for our front end framework, and c3/arcGIS for our graphing/map views. Weve since organized and set up our back-end web framework correctly for our web app. Since Django ships its beginning web structure in a very minimalistic state, (i.e. their templates, views, and models all in just one file with no app structure) We had to modularize those components into their own directories and files. It wasnt a long and arduous process, but it did take a bit of time learning how to go about doing that since we were all new to using Django.

Our web server configuration is fairly minimal at the moment. We are still early in development, which means were making all of our features on our local machines. Apache does not require much set up to deploy your projects to localhost. The only real challenge with getting Apache up and running was installing one dependency called MOD-WSGI which is required for Apache to serve Django content to the web, even on localhost.

Tracking our progress in development has been a breeze with Github. We use the branching feature for each of us to introduce new features to our project. Since each of us is in charge of one view (functionality) branching that is very easy and we still use it for making core changes to the framework. We also track new features and issues with Gits issue tracker and assign them all to milestones so we can properly prioritize changes by whether or not theyre in the coming release.

We all have some amount of experience interacting with MySQL, whether it be from our database class we took in pre-pro school or in the work force. That made converting our sponsors access database to MySQL much easier than it couldve been, but it still was very challenging. We have our system tied to the database perfectly and we are rendering data from it on our web page in different sections for a proof of concept.

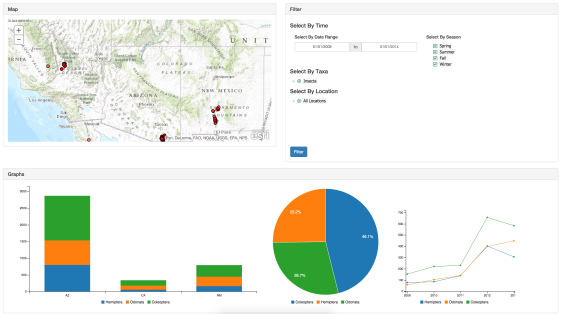
Our front end framework we chose to implement was Bootstrap. Bootstrap is a very well defined CSS/HTML framework and has seen use all kinds of applications, whatever the scale.

B. Main Layout

As dictated in the design document, the main layout of the index page of our app is split into three sections. Each section has been individually developed by each member of the team. The top right view has an interactive map in a container. In the top left there is a panel for filtering the data that is supplied to the other two views. It has drop downs and date filters for better narrowing a field of data to fit what the user is looking for. Finally, the indexs bottom container has a side-scrolling graph view that provides useful metrics on what the data consists of. Each contained view has its own template that is embedded in a primary main template in an effort to keep the main template clean and organized.

The main challenge with developing our single page web-app was providing data to the various tools were making use of in our different containers. We have written models that map objects to database entries and allow us to provide serialized data to our templates.

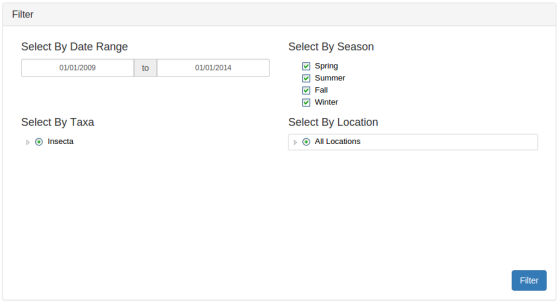
Fig. 1. Overall site layout view.



C. *Filter View*

The Filter View allows users to select information of interest by date, season, biological taxonomy, or site. Selections made in the Filter View affect what is shown in the Map and Graph Views. When the Filter View first loads, the user sees the pane shown in Figure 2. By default, all data are selected. The Date Range Filter begins populated with dates that capture the full range of data collection. The Season Filter begins with all four seasons selected. The Taxa Filter and the Site Filter begin fully collapsed, with the highest level selected.

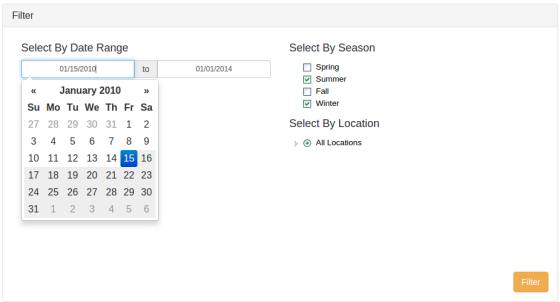
Fig. 2. The Filter view contains four tiled filters, Date, Season, Taxa, and Location. By default, all data are selected and all children collapsed.



When the user makes selections in the Filter View, the Filter button changes color, as demonstrated in Figure 3. Clicking on the Filter button applies the selections made in the Filter View to the other two views. Selections made in the Filter View persist on submission, so that the user can make selections serially.

The Date Filter allows users to view data from between a range of dates. Clicking on either date field opens up a date range picker, shown in Figure 3. Users can also type in the date manually. This causes the Graph View to only graph data collected between these two dates. The Season Filter allows users to view data collected during certain seasons across all years.

Fig. 3. When selections are made in the Filter View, the Filter Button changes color to indicate that submission is necessary. The Date Filter includes a date range picker.



The Taxa Filter, shown in Figure 4, allows users to select orders, families, gena, and species of interest hierarchically. Users can select a radio button at any level in the hierarchy. Selecting a parent node in the Taxa Filter supplies all the data from its children and none of the data from its siblings to the Graph View. The Site Filter, shown in Figure 5, allows users to select bases, drainages, sites, and samples of interest hierarchically. Selecting a parent node in the Site Filter supplies all the data from its children and none of the data from its siblings to the Graph and Map Views. Users can click on the triangle next to each parent to toggle whether its children are collapsed or expanded. Expanding or collapsing children simply determines whether or not they show on screen, and does not affect selection.

Fig. 4. The Taxa Filter allows the user to select an arbitrary level of biological taxonomy and all its children.

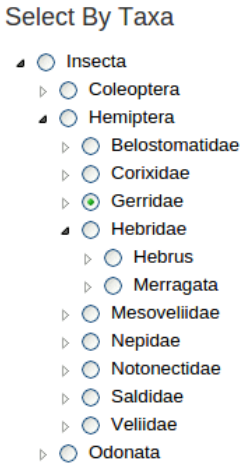
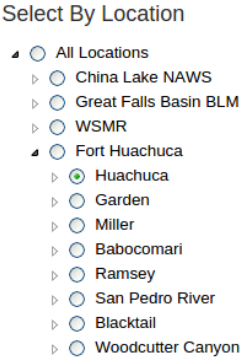


Fig. 5. The Location Filter allows the user to select an arbitrary location and all its children.



Jasper was assigned to developing the Filter View. The main obstacle he faced was not any of the filters themselves, but rather the learning curve for the development workflow. His experience using git for collaborative projects was limited, so on multiple occasions he called for help from the others. His experience with web development was also limited, so tasks such as setting up a server required assistance.

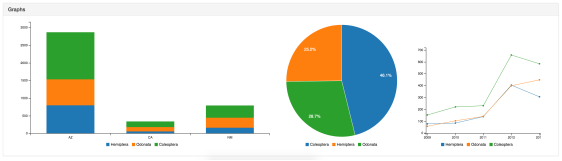
When it came to writing code, the Filter View presented two main challenges. The first was extracting meaningul information from the data posted by the filters. To do this, we created a Filter Helper class that populates a dictionary with all the relevant fields for refining queries. The second major challenge was making filter selections persistent. To do this, we had to embed code in our html template to populate the fields from the data in the Filter Helper dictionary. As such, data flows from the input forms in the Filter View, to the Filter Helper dictionary, back to the input forms.

D. Map View

E. Graph View

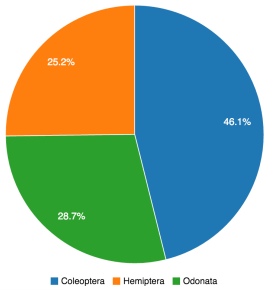
The graph view portion of our web tool is alpha-level ready. We have inter-actable example charts already deployed that render static data. A couple of them even have data being dynamically loaded into them from the database, but that wasnt an alpha-level goal for us, so we are focusing on that for the beta release. The particular statistics of interest havent been nailed-down yet, so our sponsor wanted us to have a proof-of-concept with just displaying how example graphs would look. We are able to customize every aspect of them in regards to color, spacing, font, layout, etc. An example of what the graph-portion looks like is shown above. When the page is resized to be smaller, the graph window view will display a horizontal scroll bar so that the user can still see the metrics theyre interested in without having to scrunch them up.

Fig. 6. Graph View



One of the graphs that we were successful in dynamically rendering data into was the pie-chart that displayed the number of biodiversity readings within a specific order. Since this was just an attempt to render data from the database on-load, its not actually a graph that we will show when in production, because it could be specified for genus or species. We just wanted to have the biodiversity distribution pie chart available for alpha along with some way to prove we could get data dynamically from the database.

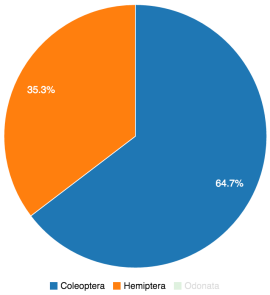
Fig. 7. Pie chart showing order distribution



The graph below displays a functionality in c3 that is really useful and is one of the primary reasons that we chose c3 as our graphing library for development. If the user is to click (or touch on a touchpad) the label for a particular dataset within a graph, then the graph will hide that dataset and focus on only the remaining. This is a very important functionality, because when taking biodiversity readings there could be an under-represented species or genus that only shares a small percentage of the dataset. There could even be multiple that are dominated by one large collection. If the user wants to zoom in on those particular small readings, they could just hide the dominating dataset by clicking its label.

During development we encountered some varyingly-sized problems that slowed us down, but we were able to overcome them before alpha release. Even though providing data to the graph view dynamically from the database was not a primary goal for the alpha, it was a very challenging process. None of us have used Django in the past and its ORM (object relational mapping) model structure, for constructing and querying objects from the database, was completely foreign to us. Once we were able to learn how Django does it s aggregate queries to the database it became a little bit more clear how to perform our queries, but it still proves to be challenging. We were able to get the correct data to render in one or two graphs, but the process of getting exactly the data we need along with massaging it into the format required by c3 (and arcGIS, filter, etc.) is a challenge in itself as well. Come beta release, we will be providing dynamic data to each template and hopefully filtering/interacting between them.

Fig. 8. Showing the interactive pie chart



F. Database and Preparing Data

G. Requests Between Views

H. Filter to Graph

I. Filter to Map

J. What the Graph View Displays