Geobarra.org: A system for browsing and contextualizing data from the American Recovery and Reinvestment Act of 2009

Ben Cohen, Michael Lissner, Connor Riley

May 10, 2010

Contents

1 Introduction

This project was completed by Ben Cohen, Michael Lissner and Connor Riley as part of the requirements for the Information Visualization class at the UC Berkeley School of Information. The goal of this assignment was to "apply the theories and techniques of visualization...to a real-world product." At the outset of this project, we discussed a number of ideas varying from a visualization of traffic on BitTorrent sites to train times of the local public transportation system. Ultimately, we decided to explore new and useful visualizations of the information that is being made available as part of the American Recovery and Reinvestment act of 2009 (ARRA). Although the project was initially inspired by the Sunlight Foundation's Design for America contest,[1] we chose to pursue this project in depth for a number of other reasons as well. First, the data provided about the Recovery Act is large, complicated, messy and diverse, and so is a challenging candidate for our analytical skills. By choosing this project, we were able to demonstrate our ability to work with large datasets, a task that is difficult for both computers and humans – several times during this project, custom software had to be written for the sole purpose of processing the data, and even with custom software written, many operations we have done on the data have been time-consuming.

A second reason we chose this project is the considerable public benefit that is created through additional analysis of this data. ARRA was a landmark bill in American history, by some measures allocating as much as \$787B,[2] and the current state of analysis of the bill is not commensurate with that level of spending. In our review of information visualizations that already analyze this data, we discovered that, first, there are few substantial attempts to build solid analyses of this data, and second, of the few analyses that we discovered, each has a shortcoming in its final product. As an example, on the recovery.gov site, it is possible to view a map of every location in the United States where recovery money has been spent. Unfortunately though, this results in too many dots to be completely relevant, and filtering options are not available.

[1]http://sunlightlabs.com/contests/designforamerica/

[2]Obama signs \$787bn stimulus plan, in: BBC, Feb. 2009, URL: `http://news.bbc.co.uk/2/hi/business/7895078.stm`.

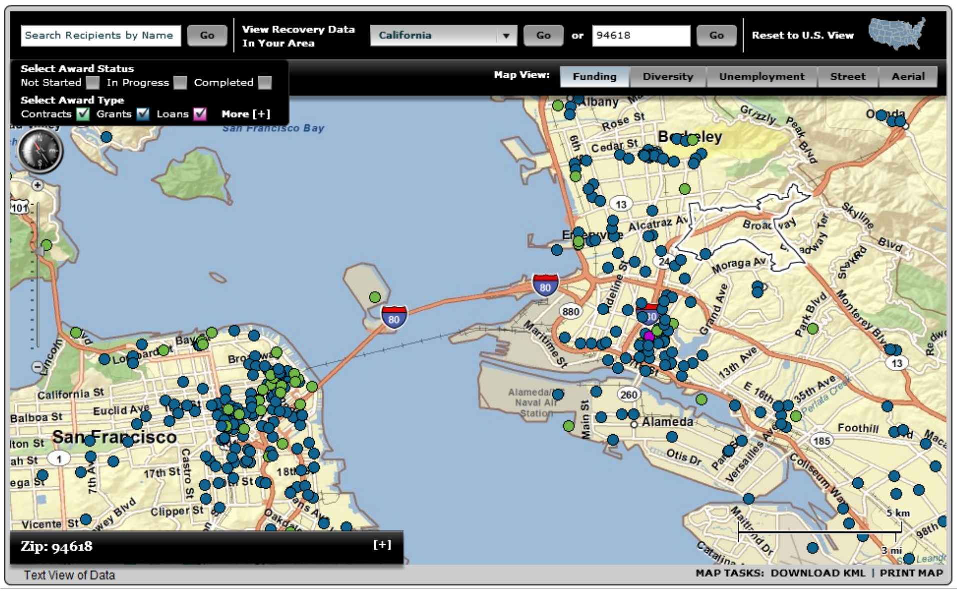


Figure 1 — A map view from the recovery.gov website

A second visualization is available from propublica.org, however while their visualizations are well-designed and aesthetically appealing, they are largely static, leaving the user with a desire for more flexibility.

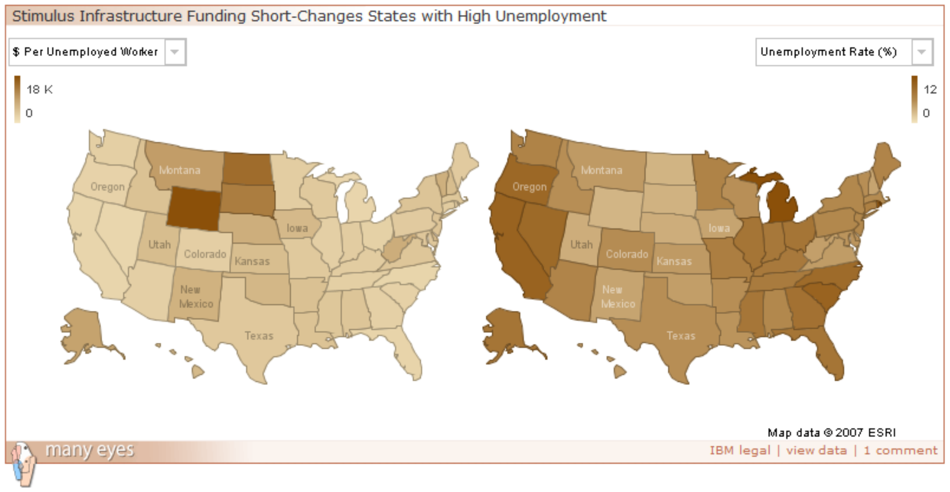


Figure 2 — Propublica's map of spending

In our project, Geobarra.org, we have made a dynamic and aesthetically pleasing visualization that can be used to query and analyze the data both numerically and geographically. We have used open-source web technologies to build our visualization, and users can start to explore the data either by choosing a geographic region, or by choosing a government agency that they find interesting, such as the Department of Homeland Security, or of Urban Development.

In building this project, we used an iterative design process. Because the Recovery data is so diverse and complicated, it was challenging to decide which parts of the data we would analyze. After interviewing an ISchool professor that has been working extensively with the Recovery Act data, Raymond Yee, we came to the conclusion that the people using this visualization would have two main goals. First, they would like to see how the Recovery money is being spent generally, and second, they would like the ability to hone in on locations relevant to them, and see how the money is being spent there. This is similar to Ben Schniederman's mantra: "Overview first, zoom and filter, then details-on-demand."[3] By using a brushed and linked interface of a choropleth, bar graphs and a table with data, we have enabled users to browse the data geographically or by filters. Once that is done, they can zoom into particular pieces of data in the table below, where they can see relevant details about an allocation.

2 Process

As mentioned previously, we used an iterative process while working on this project. A challenge we encountered, and which we mentioned previously, was the complicated nature of the published Recovery Act data. The data itself has 98 distinct fields which are explained in a PDF document. Each row of data has several unique IDs, some of which do not have meaning in isolation. As a result, the first step of our process was to attempt to understand the data, and, where unique IDs mapped to more meaningful data in other sources, we completed JOIN operations on the data, pulling various sources together into the same source. For example, each row of data in the Recovery database has a TAS number, which corresponds to a single account in the Treasury Accounting System. Although these numbers correspond to specific departments within the government, those departments are not listed in the data, and must be located elsewhere. Finding

[3]Benjamin B. Bederson/Ben Shneiderman: The craft of information visualization, 2003, p. 436.

those numbers is not trivial, and we has to process those we did find into machine-readable format.

A second data manipulation that we had to complete for our project was mapping the locations of the allocations. In the Recovery data, the zip code of the allocation is provided, but the county was not. Thus, we needed another source of data in order to map and compute at that level. For this, we used the Geonames database, which provides information about every zip code, county, and state in the United States.[4]

Another tool that was invaluable to our understanding of the data was developed by Raymond Yee for his investigation of the data. Because the data is only available in a CSV file, he wrote a program that converted it to a sqlite3 database, and created a web query tool that can be used to query that database.[5] He graciously shared both of these tools with us. The program was eventually used to convert the data into a valid datamodel in the Django web framework.

After we had a grasp on what data was available for us to work with, we began sketching paper prototypes of the system. These were invaluable to us, as they allowed us to finally conceptualize the kinds of data we would need, and the tools that would be required to manipulate that data. From our paper prototypes, we created a high resolution image of how we wanted our final design to appear. Our final version is quite similar to this image. All of these sketches and images can be found in Appendix I.

After agreeing on on the overall design of the system, we divided the tasks into building the map, building the bar graphs, pulling it all together, and handing the backend issues and configuration. For the map, we have used the cartographer.js[6] library, the latest version of which contains the polylines for every county in the United States which we created for Geobarra.org.[7] For the bar graphs, we used the Flot jQuery-based charting library,[8] and for the backend and data processing, we used Apache, MySQL, Django, python, ruby and Mercurial.

After finishing our respective parts, we regrouped to determine how best to combine our disparate pieces. At this point, we had U.S. Census, Recovery and Geonames data in our database, had a functional map with county and state-level granularity, and had Javascript-based bar graphs that would dynamically update according to various triggers. In order to combine these

[4]http://geonames.org

[5]http://people.ischool.berkeley.edu/ rdhyee/s10/day15/search.html

[6]http://cartographer.visualmotive.com/

[7]http://code.google.com/p/cartographerjs/issues/detail?id=2c8

[8]http://code.google.com/p/flot/

pieces, we linked the map with the bar graph, and created an API on the database using Django views and queries.

With this functioning, we completed a final round of user testing. This round of testing consisted of interviews with three users, and click map testing using an online testing service, http://fivesecondtest.com, with an additional five users. The goal of this testing was to answer three questions: does the tool meet user needs better than existing tools, are there clear calls to action, and are the visualizations clear. For comparison, we tested users' ability to perform tasks with a ProPublica visualization of the Department of Transportation funding.[9]

Users found the combination of map and bar chart useful in understanding the funding information and comparing their local regions to the national funding. However, some users found it difficult to sort information about individual awards in the table compared to ProPublica. From the click map testing, we found that users needed more labeling of axes and units on the bar chart. Users also wanted a key for the choropleth, although when asked, all users correctly interpreted the colors used.

Most users could not identify a clear call to action when first encountering the site. After exploring the site, users understood that inputs and outputs were coincident, but most users were uncomfortable exploring initially.

In response to user testing, we are implementing axis labels, a key on the choropleth, and introductory text explaining how to explore the site. Since users enjoyed exploring the data once the controls were clear, we are pleased with the coincident inputs and outputs, and don't see a need for introducing additional control elements beyond the initial explanatory text.

3 System Description

The final version of Geobarra.org is quite similar to our original high-resolution prototype. It has bar graphs on the left, a map on the right, and a table of data on the bottom. Each of these views is brushed and linked with the others in a manner such that changes to one view also change the others. We have also used the jQuery tablesorter plugin to allow the table to show a given number of rows, and to make it sortable on one or more fields (hold shift and click).[10]

[9]http://www.propublica.org/special/states-transpo-funding-rate-416

[10]http://tablesorter.com/docs/

Our current data architecture uses a combination of cached JSON data and queried JSON data. For data that is small and does not change, such as the mapping of TAS codes to Department names, we have used cached JSON data. For larger data that requires queries, we have used Django to make an API that returns JSON data. The API accepts the following queries:

geobarra.org/stateToTasAndSum/$state/: Queries to this URL with two letter abbreviations in the last field return the a list of dictionaries containing TAS codes and their total amounts in a given state.

geobarra.org/stateAndAgencyToTasAndSum/$state/$agency/: Queries to this URL with a state abbreviation and an agency name return a dictionary containing TAS codes and their total amount for an agency.

geobarra.org/allTasAndSum/: Queries to this URL return all TAS codes and their sums for the entire United States.

geobarra.org/countyToAwards/(.*)/: Given a county name, this URL returns a list of awards.

geobarra.org/countyAndTasToAwards/(.*)/(.*)/: Given a county and a TAS code, this URL returns the details for that county and award.

For our choropleth, as was previously mentioned, we used the relatively new cartographer.js mapping library. It uses the Raphael Javascript library[11] as its backend, and allows for the efficient creation of choropleth overlays onto Google maps. When we initially approached the cartographer.js library, it had polylines for all of the States in America, but lacked county-level polylines. As part of our project, we have created these county-level polylines, and we have given them to the cartographer.js project for future developers to use. They have been recently included in version 0.4 of the library.[12]

[11]http://raphaeljs.com/

[12]To see the actual changes to the code, visit: http://code.google.com/p/cartographerjs/source/detail?r=9

4 Technologies Used

4.1 Django

Django[13] is "a high-level Python Web framework that encourages rapid development and clean, pragmatic design." We used Django as the application framework to process requests from the client, perform queries over the dataset, and return JSON to the client.

4.2 SQLite

We used this lightweight database during the application development phase to gain a deeper understanding of the Recovery data, and determine which queries were possible given the government data set, and which queries would require additional data.

4.3 Ruby and Python

Processing data from multiple sources (ProPublica, recovery.gov, the Department of the Treasury, Geonames) required us to scrape data from the web, convert PDF files to CSV, and many other tasks that were best automated by simple scripts in Ruby and Python.

4.4 Mercurial

We chose Mercurial as our version control system in order to help us manage a large amount of data and files in a central location.

4.5 cartographer.js

cartographer.js handles all of our mapping functions. The library allows us to instantiate a Google map and add graphical overlays to create a choropleth. Originally, the library only supported shapes for US states. We personally compiled the GPolyline shapes for all US counties, which have been incorporated into the cartographer.js project.

In our application, we attach click handlers to all map elements, so that when users click on a state, the map and chart can both be redrawn, showing the county shapes for the state and the state data on the chart. The map is also automatically redrawn to show all states when a user zooms out.

[13]http://djangoproject.com

4.6 jQuery

The jQuery library for JavaScript lets us take advantage of several shortcuts for user interaction with the different pieces of the visualization. jQuery's support for AJAX queries makes it very simple to call the various Django queries and use the JSON data in the query response.

4.6.1 Flot

Flot is a robust jQuery charting library. Flot allows us to chart several data series side-by-side, as well as capture the relevant data points when users click and hover over the graph. In our application, interaction with the chart is handled through a single method, which is called whenever the user clicks on the map or changes the options for the displayed agencies.

4.6.2 TableSorter

The TableSorter jQuery library automatically handles user interaction with our data table. The library take an HTMl table element and makes each column header into a control element for sorting the column. It handles text, numbers, monetary units, and percentages with no customization. Additionally, the library handles table paging, letting users browse through the table without scrolling up and down the page.

4.7 JSON

We use JSON data extensively, both as the response format from server queries and as static variables for some of the chart elements. JSON is easy to generate with Python, and jQuery includes several shortcuts for requesting and processing JSON data.

5 Appendix I

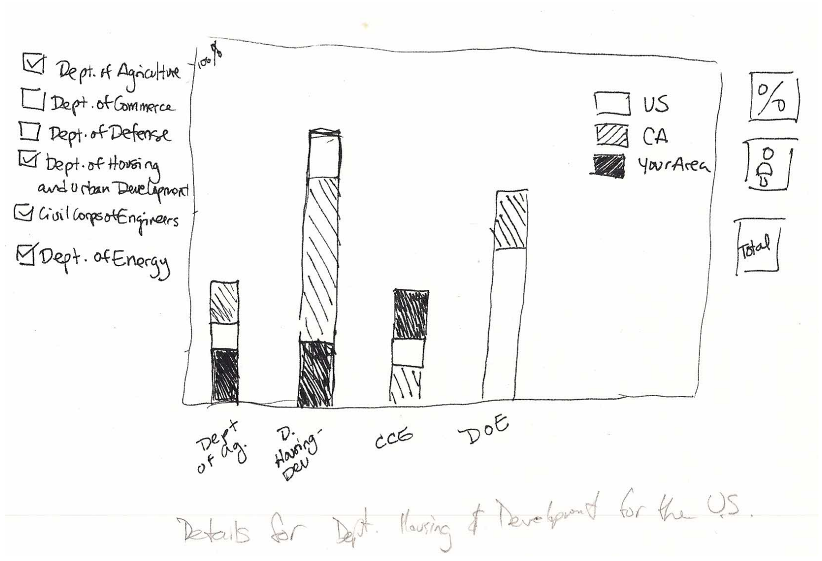


Figure 3 — Our first paper prototype, demonstrating the data we would place into bar graphs.

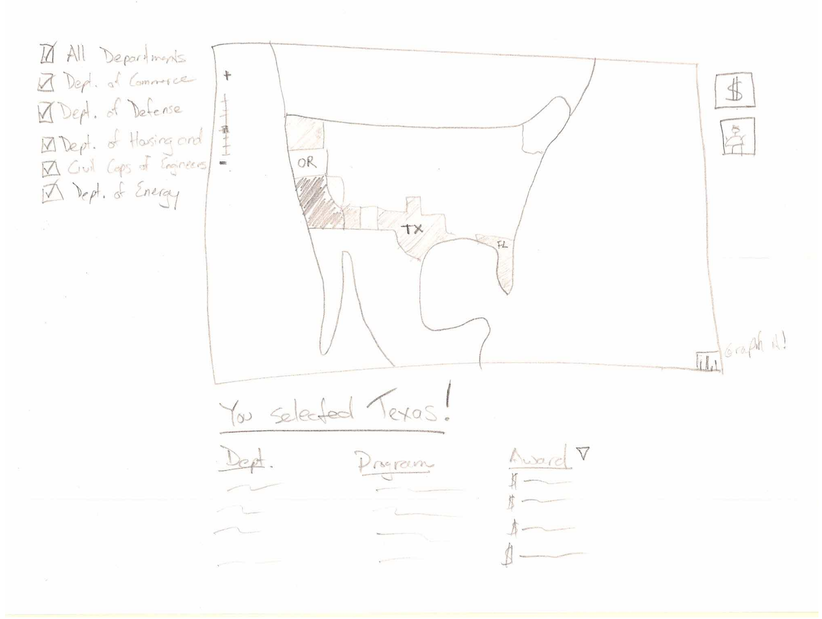
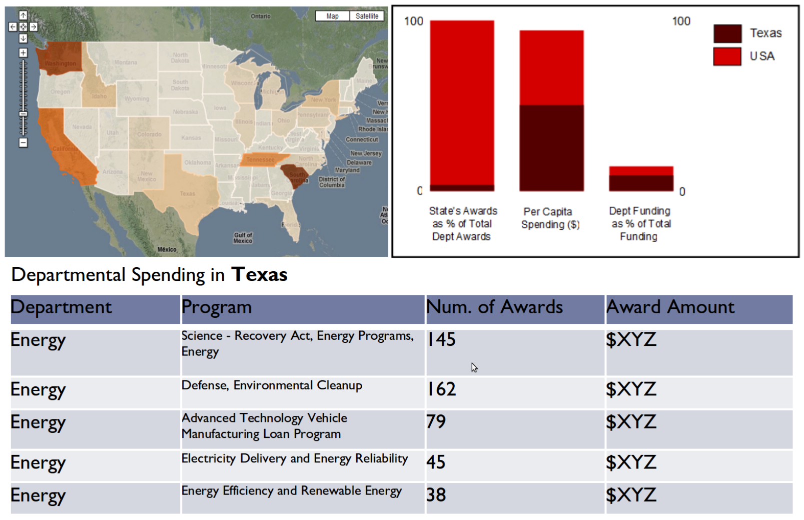


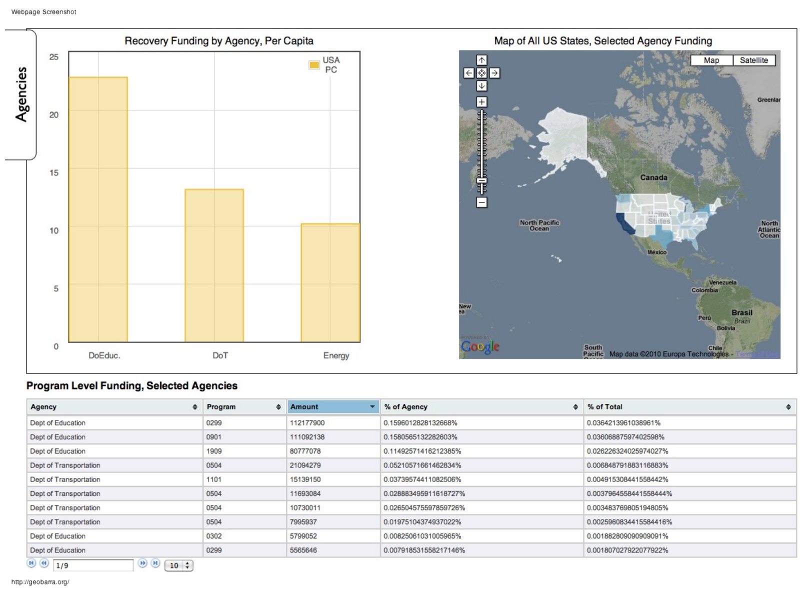
Figure 4 — Our second paper prototype, demonstrating the data we would display with a choropleth.


Map
Satellite
100
100
0
0
Texas
USA
State's Awards as % of Total Dept Awards
Per Capita Spending ($)
Dept Funding as % of Total Funding

Departmental Spending in **Texas**

Department	Program	Num. of Awards	Award Amount
Energy	Science - Recovery Act, Energy Programs, Energy	145	$XYZ
Energy	Defense, Environmental Cleanup	162	$XYZ
Energy	Advanced Technology Vehicle Manufacturing Loan Program	79	$XYZ
Energy	Electricity Delivery and Energy Reliability	45	$XYZ
Energy	Energy Efficiency and Renewable Energy	38	$XYZ

Figure 5 — Our third prototype, combining the the first two and including tabular data.



Program Level Funding, Selected Agencies

Agency	Program	Amount	% of Agency	% of Total
Dept of Education	0299	112177900	0.1596012828132668%	0.0364213961038961%
Dept of Education	0901	111092138	0.1580565132282603%	0.03606887597402598%
Dept of Education	1909	80777078	0.11492571416212385%	0.026226324025974027%
Dept of Transportation	0504	21094279	0.05210571661462834%	0.006848791883116883%
Dept of Transportation	1101	15139150	0.03739574411082506%	0.004915308441558442%
Dept of Transportation	0504	11693084	0.028883495911618727%	0.0037964558441558444%
Dept of Transportation	0504	10730011	0.026504575597859726%	0.003483769805194805%
Dept of Transportation	0504	7995937	0.01975104374937022%	0.0025960834415584416%
Dept of Education	0302	5799052	0.00825061031005965%	0.001882809090909091%
Dept of Education	0299	5565646	0.007918531558217146%	0.001807027922077922%

1/9 10

http://geobarra.org/

Figure 6 — The interactive prototype on which we performed our user tests.