Geospatial Web Portal for Regional Evacuation Planning

Chee-Hung Henry Chu
Informatics Research Institute
University of Louisiana at Lafayette
Lafayette, Louisiana, USA
Email: chu@louisiana.edu

Ramesh Kolluru
School of Computing and Informatics
University of Louisiana at Lafayette
Lafayette, Louisiana, USA
Email: kolluru@louisiana.edu

***Abstract*— Evacuation of a regional population is often necessary when a disaster, such as wildfire or coastal flooding, is in the forecast. We build a map-based web portal for the planning of an evacuation of a region. A user marks a polygon on the map to indicate the extent of the region that will be evacuated. The population affected and the home value that will be impacted is calculated. The system then assigns local communities to different shelters. The evacuation plan for each local community in the region is then displayed. Underpinning the web portal is a data structure organized geographically by the U.S. postal service zip code that contains the population and a home value index. The implementation uses the Mapbox GL JS library.**

Keywords- Evacuation planning; geospatial web portal.

I. INTRODUCTION

When a major disaster due to a natural event or other causes or an emergency is imminent, it is useful to have an approximate estimate of the preliminary damage assessment and to have an evacuation plan [1-4]. In this paper, we present a web tool that assist in these two tasks.

There are different platforms to develop map-based web portals. Google Maps allow a user to display maps as images [5]. It supports JavaScript (JS) code to interact with users. Mapbox is similarly a geospatial platform. Mapbox GL is a set of libraries for different deployment platforms [6]. Mapbox GL JS is the library for Web applications. A JavaScript library for spatial analysis that works with Mapbox GL JS on the browser is turf.js [7]. These technology, when coupled with traditional map creation considerations, can bring cartography to a variety of devices [8]

A map is displayed using the Mapbox platform as an HTML file [3]. Figure 1 shows the HyperText Markup Language (HTML) code segment and the JavaScript code to display a map (Figure 2).

Our example tool operates at the level of local communities. We use the U.S. postal service zip code as the basic geographic unit. There are 508 zip codes in Louisiana that have inhabitants (vs. zip codes that are business or post office box addresses). We have the location (latitude, longitude) and population of each zip code. In Mapbox GL JS, data are organized using the GeoJSON format [9]. The set of all zip codes are collected in a FeatureCollection (Figure 3), which is an object using the JavaScript Object Notation (JSON). The features in the collection is collected as an array keyed by 'features'. Each zip code is represented as a point with the coordinates, and with "Name" and "Population" as properties. The FeatureCollection is displayed as an added layer of each zip code as a circle (Figure 4).

```
<div id='map'></div>

<script>
mapboxgl.accessToken = 'token from mapbox.com';
var map = new mapboxgl.Map({
    container: 'map',
    style: 'mapbox://styles/mapbox/streets-v9',
    center: [-92.02, 30.22], // starting position
    zoom: 7 // starting zoom level
});
</script>
```

Figure 1. HTML and JavaScript for displaying a map.

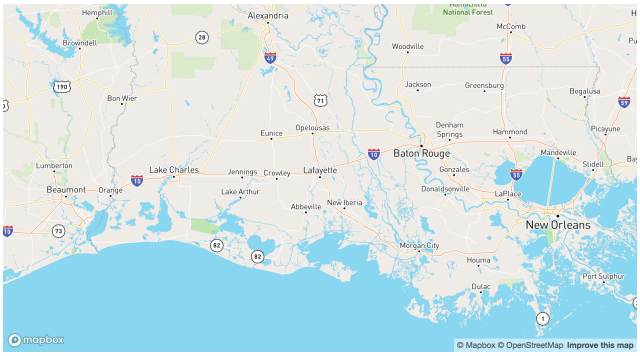

Figure 2. Map displayed based on code in Figure 1.

```
<script>
var zips={
  type: 'FeatureCollection',
  features: [
   {type:'Feature',
    properties:{Name:'70501',population:30867},
    geometry:{
      type:'Point',
      coordinates:[- 92.00959,30.2334]
    }},
    …
  ]
};
</script>
```

Figure 3. GeoJSON object to hold zip code data.

 ISBN: 978-1-61208-687-3

The rest of the paper is organized as follows. In Section II, we describe the sources of the data sets that we use. In Section III, we describe the implementation as a client (browser)-side tool using the JavaScript programming language. In Section IV, we describe the implementation as a web application, using processing at the server and client sides for better performance. We draw our conclusions in Section V.

```
// plot zip points
map.on('load', function() {
  map.addLayer({
    id: 'zips',
    type: 'symbol',
    source: {
      type: 'geojson',
      data: zips
    },
    layout: {
      'icon-image': 'circle-11'
    },
    paint: {}
  });
});
```

Figure 4. JavaScript code to display the zip code nodes as a new layer.

II. DATA SOURCES

In this example tool, we would like to know for a given area, what is the population and what is the estimated home value. The population data for each zip code are available from the Census Bureau [10] and other web sites [11].

Getting the home value of a zip code is more challenging, unless one has access to, e.g., the tax assessor's data. We use the Zillow Home Value Index (ZHVI) [12], which as a home value index is an approximation to the median home value in a locality. To obtain the total of home values, we need to estimate the number of homes in a zip code. Since on average there are 2.8 persons in a U.S. household, we estimate the number of homes to be occupied by an average household of 2.8 persons, so that the total value V is given by

$$V = Z \times P / P_h \quad (1)$$

where Z is the ZHVI for a zip code area, P is the total population in the zip code area, and P_h is the average household size, set to 2.8 in our examples.

In order to determine the evacuation plan, we need a number of shelters. We set the number of shelters to 3 and locate them in Houston, Little Rock, and Jackson, corresponding to the west, north, and east. Considering the capacity of shelters, Houston and Jackson would have larger ones than Little Rock. We note that in practice, planning requires more local analysis for more precise decisions. Thus, decisions about choice of shelters as well as assignment of shelters form the basis of network analyses.

III. IMPLEMENTATION OF CLIENT-SIDE ONLY TOOL

A. Impacted Population and Home Value Estimate

We use the Mapbox GL JS tool to enable the user to draw a polygon (Figure 5). A function updateArea() is called when the drawn object is created, deleted, or updated. When a polygon is created, the operations we want to do are:

1. determine the enclosed area;
2. determine which of the zip codes are included;
3. determine the population of the included zip codes;
4. determine the home values of the included zip codes.

The function updateArea() calculates these and reports them to the web page.

```
var draw = new MapboxDraw({
    displayControlsDefault: false,
    controls: {
        polygon: true,  // draw polygon
        trash: true
    }
});

map.addControl(draw);  // tool control panel
map.on('draw.create', updateArea);
```

Figure 5. JavaScript code to add the drawing tool to mark a polygon.

The @turf/turf library makes these spatial analysis steps simpler. Step (1) is accomplished by the turf.area(data) call, where data is the drawn FeatureCollection object. To perform Step (2), we use the turf.pointsWithinPolygon() function, which takes a FeatureCollection (the zip codes in our example) and a polygon. It returns a FeatureCollection containing all of the points that the polygon contains. The calculations for steps (3) and (4) are straightforward.

An example output is shown in Figure 6. Only the enclosed zip codes are displayed. The information for (1), (3), and (4) are shown inside a box on the lower left of the browser.

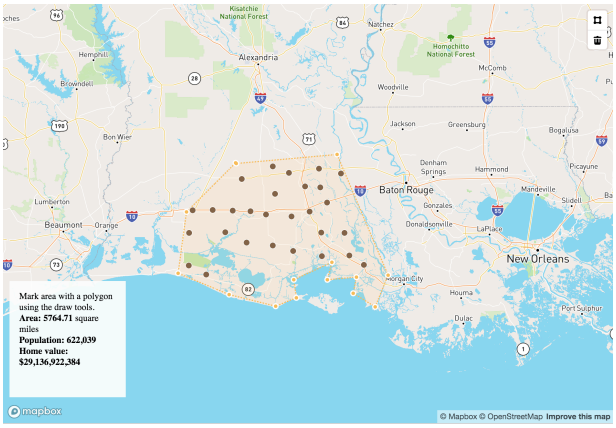

Figure 6. Map showing the zip codes within the drawn polygon (shaded).

B. *Evacuation Planning*

Given a set of zip code points, each with a population that should be evacuated. Given another set of shelter points, each with a capacity for handing evacuees. The distance between the zip code points and the shelter points and the roadway capacity between them together form the cost of transporting the evacuees to a shelter. As such, the problem can be set up as a classic transportation problem and an optimal solution using linear programming is possible. It, however, requires computational resources when the number of zip codes is large for a client-side implementation.

We might consider a heuristic solution because of the relatively similar costs due to the proximity of many of the zip codes to each other, to accommodate the limited computational resources in a client-side tool implementation. For instance, we could use a nearest neighbor approach to assign a shelter to a zip code. This is too simplistic, however, since the more capacious shelters in Houston are further away than Jackson for many zip codes.

We propose a compromise solution between ease of calculation and realistic performance. We map the bounding boxes of the shelters and the impacted zip codes. We would like to "embed" the shelters inside the polygon and then use the nearest assignment method. For each shelter, after using bilinear scaling to map it to the bounding box of the impacted zip codes, we use a scale factor to bring it to the interior of the bounding box. The more capacious the shelter is, the smaller the scale factor we assign, so that the shelter will be closer to the center of the zip code bounding box.

The turf library functions bbox() and nearest() respectively perform the bounding box and nearest point calculations.

An example is shown in Figure 7, where the enclosed zip code points are color coded, depending on which shelter it is assigned to. Two other examples (Figure 8 and Figure 9) with different shapes and at different locations are shown to illustrate that the shelter assignments are reasonably robust. In these examples, nodes marked in gold are assigned to the shelter in the west ("Houston"); those in dark blue are assigned to the shelter in the north ("Little Rock"); and the rest are assigned to the northeast ("Jackson").

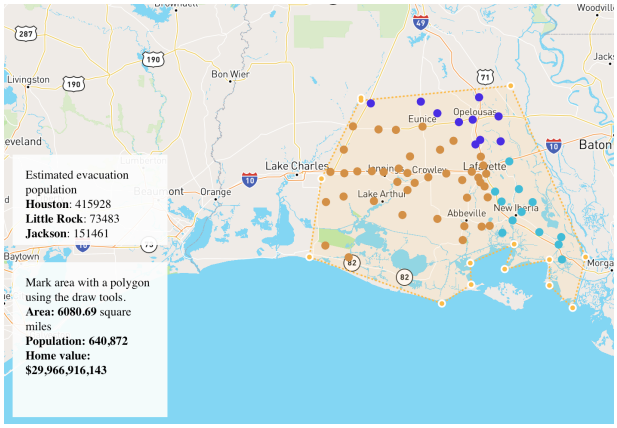

Figure 7. Map showing the zip codes color coded to indicate which shelter the population should head to.

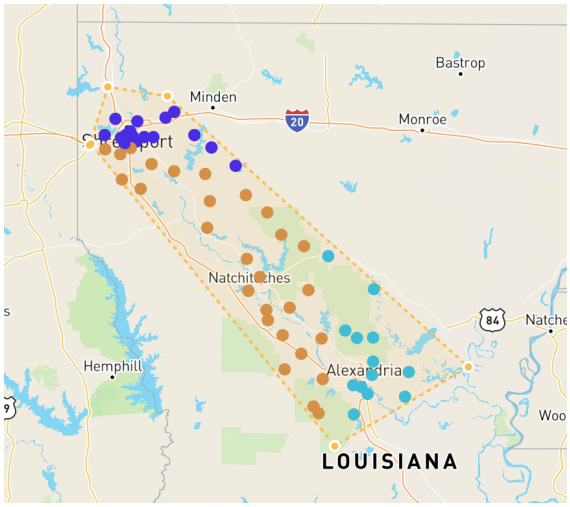

Figure 8. The zip codes in the northwestern corner of the state.

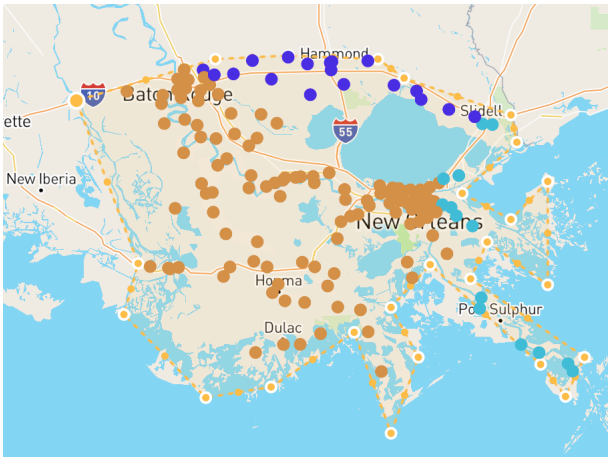

Figure 9. The zip codes in the southeastern corner of the state.

IV. WEB APPLICATION IMPLEMENTATION

The foregoing discussion assumes the portal does all of the processing on the client-side, using the rich functionalities enabled by the @turf.turf library. An advantage of having a client-side web page is that there is minimal communication between the browser and the server. The disadvantage is that all the data have to be loaded, whether they are needed or not. As an example, the data associated with all 508 zip codes in the State of Louisiana have to be pre-loaded, resulting in an HTML file that is 73K bytes. A more serious challenge to client-side processing is that some more sophisticated shelter assignment algorithms may not be able to be ported to the client-side. An example is an algorithm that requires linear programming optimization.

We refactored our application to a web application that uses the node.js framework. We divide the data, the presentation, and logic control in the model-view-controller

 ISBN: 978-1-61208-687-3

pattern. The zip code (population, home value, location) data and the shelter data are stored on the server side. The map display and the polygon drawing interface are shown on the browser to interact with the user. Once the user has drawn the polygon on the map, the polygon data are sent to the server as a geoJSON "featureCollection" object. The server app contains the logic to find enclosed area, the zip codes enclosed, etc., i.e. the steps referenced in the updateArea() function as mentioned in III.A. The output is then generated as a map drawing HTML file and sent to the browser for the user.

The key challenge is to send the polygon data from the client to the server. Because MapBoxDraw() returns a geoJSON object containing the polygon coordinates, a straightforward way to send it to the server is to use an XMLHttpRequest(), as shown in Figure 10. Upon creation of a polygon, updateArea() is executed and the polygon geoJSON is sent to the server. The difference between the implementation here and the one in Section III is that updateArea() now does not do any further processing. On the server side, the app receives the JSON data as the request body (Figure 12); other processing steps are similar to the discussion in III.A by using the @turf.turf node.js module (Figure 11).

```
function updateArea() {
    var polygon = draw.getAll();
    // send to server
    var xhttp = new XMLHttpRequest();
  xhttp.open('POST', {{{url}}} , true);
    xhttp.setRequestHeader('Content-Type',
'application/json');
    xhttp.send(JSON.stringify(polygon));

    return;
}
```

Figure 10. JavaScript code to send the drawn polygon data to the server from the client-side.

```
// holds polygon data from browser
var polygon={};

app.post('/', jsonParser, function(req,res){
   polygon = req.body;
   res.send('received data');
});
```

Figure 11. JavaScript code in the server app that receives the polygon geoJSON data and holds the data as a geoJSON ("polygon") to be passed to other routes.

V. CONCLUSIONS

Web-based map tools open the geospatial information display and manipulation door to more data scientists. We demonstrate an example tool that determines the estimates of population affected, impacted home values, and assigned shelter locations based on a user marking the potential disaster area. We further refactor the HTML and client side JavaScript code to a JavaScript node.js server side app.

Ongoing work includes taking into account the impacted public facilities, the road capacity and fuel availability on the evacuation paths, size and locations of temporary populations such as tourists and temporary workers.

ACKNOWLEDGMENT

The authors gratefully acknowledge their colleague Dr. Michael Dunaway, Director of the National Incident Management Systems and Advanced Technologies Institute, for valuable discussions on disaster management. They further thank the anonymous reviewers whose comments help us to improve the manuscript.

REFERENCES

[1] S. M. Rahat Rahman, M. S. Mamun, M. A. Basit, and M. M. Rahman, "Evacuation plan for the solution to disaster management for the coastal region of Bangladesh: A review," International Conference on Engineering Research, Innovation and Education, 2017, Sylhet, Bangladesh, 6 pages.

[2] R. Alsnih and P. Stopher, "A review of the procedures associated with devising emergency evacuation plans," Transportation Research Record, vol. 1865, no. 1, pp. 89-97, 2004.

[3] G. Ayfadopoulou, I. Stamos, E. Mitsakis, and J.M.S. Grau, "Dynamic traffic assignment based evacuation planning for CBD areas," Procedia – Social Behavioral Sciences, vol. 48, pp. 1078-1087, 2012.

[4] G. Li, L. Zhang, and Z. Wang, "Optimization and planning of emergency evacuation routes considering traffic control," Scientific World Journal, 15 pages, 2014.

[5] Google Maps Platform, https://cloud.google.com/maps-platform/maps/ [Accessed: January 2019]

[6] Application programming interface specifications of the Mapbox GL JS library, https://www.mapbox.com/mapbox-gl-js/api/ [Accessed: January 2019]

[7] Turf.js: A JavaScript library for spatial and statistical analysis, https://www.mapbox.com/help/analysis-with-turf/ [Accessed: January 2019]

[8] I. Muelenhaus, Web Cartography: Map Design for Interactive and Mobile Devices, CRC Press, Boca Raton, Fla., 2014.

[9] The GeoJSON format for geospatial data interchange, https://tools.ietf.org/html/rfc7946 [Accessed: January 2019]

[10] ZIP code tabulation areas, https://www.census.gov/geo/reference/zctas.html [Accessed: January 2019]

[11] United States population by ZIP code, https://www.kaggle.com/census/us-population-by-zip-code [Accessed: January 2019]

[12] Zillow home value index: Methodology, https://www.zillow.com/research/zhvi-methodology-6032/ [Created: January 2014; Accessed: November 2018]

 ISBN: 978-1-61208-687-3