Slide 1:
My presentation involves the development of Route30, a free and open-source software client-side routing library.

My name is George Adams. I'm completing an MGIS degree at Penn State. I'm presenting my capstone project.

I have a background in software engineering and industrial engineering. I was interested in developing a routing library because it relates to my background and experience, and I think there may be many uses for a client-side routing library.

Slide 2:
The objective of my project was to develop a user-configurable client-side routing library. It's user configurable in the sense that someone can filter points of interest and constrain the route.

In addition to the routing library, I developed an historical walking tour web app for Castroville, Texas to demonstrate how the routing library could be used. Most of the historic sites are small houses like the Henri Castro homestead I'm showing on the right.

Some characteristics of the library.
First, the library is free and open source. It's available for commercial as well as non-commercial uses.

<<No Change>> (Slide 2 Continued)
Also, we believe the library would be easy to use and may benefit organizations having limited funds and development experience.

Routing libraries are typically server-side libraries. However, Route30 is a client-side library, and I will discuss the benefits of a client-side library at the end of this presentation.

And finally, when developing the historical walking tour web app, I observed that many walking tours are static, predefined tours. The routing library allows users to dynamically generate tours that suit their preferences.

Slide 3:
First, I'm going to provide some background on routing.

NEXT:
Key to routing is using efficient algorithms to generate an optimal (or shortest) path.

Slide 4:
Next, I'll discuss the routing library design. The library has two main components: a distance matrix component and a route generation component.

Slide 5:
This is the user interface for my demo web app. You can open this app on your smart phone and do a walking tour in Castroville. The web app provides user input to the routing library and displays the route generated from the library. I'll show this functionality in this presentation.

Slide 6:
Also, I want to discuss how someone could use the library for their own applications. I'll discuss the data input format, calls to the library, and results generated for display in a web app.

Slide 7:
And finally, I'll describe the benefits to using the library. On the right, I'm showing a route generated in Castroville from a server-side solution. I'll compare this result to one generated from Route30.

Slide 8:
Let's start with some background

Slide 9:
At the core of this effort is the Traveling Salesman Problem. Given a set of points of interest (POIs) find the optimal (or shortest) path among those POIs.

Slide 10:
This would likely be fairly simple to do for a small number of POIs. For example, with 6 POIs there are only 60 distinct paths. You could use a brute-force approach and generate all 60 paths, sort them from shortest to longest, and pick the shortest one as the optimal path.

However, the number of distinct paths goes up rapidly with points of interest.

If you add one more POI, the number of distinct paths grows from 60 to 360.

Slide 11:
For even a modest application of 30 POIs, the number of distinct paths climbs to 10 to the 30th power as you can see in the upper right portion of the graphic.

NEXT:
So, a brute force solution quickly becomes infeasible.

Therefore, certain algorithms are used to find a solution that is close to the optimal route. I built the routing library to include some of these algorithms. I'll describe them when I discuss the library design.

Slide 12:
The other thing to consider when solving the Traveling Salesman Problem is that paths are not always straight lines.

NEXT:
They frequently follow a street network, so the routing library needs to have a street network as input, and it needs to be able to find shortest paths along this street network.

Slide 13:
So, the goal for the routing library is to efficiently solve the Traveling Salesman Problem and generate a route, close to optimal, through various numbers of POIs. Paths are developed along a street network subject to user constraints as well as other constraints that are part of the street network itself.

Slide 14:
Now for the design

Slide 15:
As I mentioned before, there are two components in the routing library: a distance matrix component and a route generation component. I'm going to first talk about the distance matrix component.

The goal of this component is to find the shortest path between every pair of POIs and output this information in the form of a distance matrix.

For my application, the direction of travel is not unique. So, for example, for 6 POIs, there are 15 pairs as shown in the upper shaded portion of the matrix.

In this example, each cell in the matrix displays the shortest straight-line distance between each pair of POIs.

Slide 16:
However, for my application, I'm not interested in the straight-line distance between POIs but instead the path along a street network.

I use the A-Star algorithm which is guaranteed to find the shortest path between each pair of POIs.

For the case of 6 POIs, I would invoke this algorithm 15 times.

Slide 17:
The algorithm does not look at all possible street segments but instead focuses the search on the region between the two points. It uses a heuristic to pull the solution towards the goal. This focused search allows the algorithm to execute efficiently.

NEXT:
So, for example, in developing the shortest path between Point 5 and Point 1, the algorithm may start searching as shown here.

NEXT:
However, during execution, it may find a better alternative route and backtrack to this alternative. This is the other characteristic of the algorithm. It has the ability to look back and find a better path.

Slide 18:
When finding paths on a street network, one problem you can run into is barriers. For example, in Castroville, one street segment runs through an elementary school campus. Any route generated would have to avoid this segment, and therefore, I identified it as a barrier in the input file.

<<And, I'm showing that over here in the right-most column.>>

Slide 19:
The other type of barrier is a node barrier.

In Castroville, Highway 90 splits the historic district. Pedestrians can only cross at one of two crosswalks. One crosswalk is in the historic district, and it's marked in green on this slide. To prevent routes from crossing Highway 90 outside of a crosswalk, I designated some nodes as barrier nodes in the input file.

Slide 20:
So, in summary, the routing library has two components, a distance matrix component and a route generation component. The distance matrix component generates a distance matrix for input to the route generation component.

NEXT:
The input to the library is a points of interest file and a street network consisting of a nodes and edges file. Although I previously showed these files in a tabular data format, they are actually GeoJSON-formatted files, and I'll describe the GeoJSON format later in this presentation.

NEXT:
And last, because routing is done on a street network, each cell of the distance matrix now contains both the distance and the shortest path between every pair of POIs.

Slide 21:
Now for the route generation component.

The purpose of this component is to find a best solution and generate a route that is close to optimal for a set of POIs.

NEXT:
To do this, I use the simulated annealing algorithm to solve the Traveling Salesman Problem. Although I will discuss this algorithm in detail, the key points to take away from this discussion are the algorithm has randomness and is configurable.

Slide 22:
The Simulated Annealing algorithm has its analogy in the field of metallurgy where metals are heated and then slow-cooled to relieve stresses. Similarly, this algorithm starts at a high pseudo temperature. An initial route is identified at random, then a series of transport or reversal processes are applied to that route while lowering the temperature.

After each process is applied, the transformed route is evaluated to see if it has a shorter total distance. If so, that new route is kept. If not, the new route may or may not be kept depending on the temperature.

As the temperature drops, fewer longer routes are accepted.

The goal of this algorithm is to find a global minimum without getting stuck in a local minimum.

| | |
|---|---|
| Slide 23:<br>So here is an example of simulated annealing.<br><br>I picked a route at random as shown by the sequence of POIs in the box. Traversing these POIs sequentially on the diagram to the right results in a total distance of 4.8 km.<br><br>Slide 24:<br>Now, I flip a coin to decide which transformation to select (reverse or transport). | Slide 25:<br>Assuming reverse is selected, I picked segment 2,1,6 at random and reverse the order of the POIs.<br><br>NEXT:<br>A shorter overall route is generated. It goes from 4.8 km to 4.2 km. Because it's shorter, the new route is kept. |
| Slide 26:<br>I flip the coin again. This time, a transport is selected.<br><br>I pick a segment at random and a location on the remaining route. In this example, segment 1,2 is picked and location zero is selected on the remaining route.<br><br>NEXT:<br>I then transport the segment to location zero. The route shown to the right is not an improvement as the total distance is now greater. It goes from 4.2 km to 4.6 km.<br><br>However, the algorithm may still accept this new route. It depends on the temperature. | Slide 27:<br>So the question that comes up is how many transport or reversal processes should be done at each temperature?<br><br>This is determined by configuration parameters. These are important parameters that affect the quality of the output and computation time for the algorithm.<br><br>As shown in this flowchart, after a transport or reversal process completes, the algorithm checks to see how many new routes were accepted at that temperature. An accepted new route is a success. If there were enough successes, then the temperature is lowered and the process is repeated. Otherwise, the algorithm checks to see if the limit on total attempts was reached. If so, the temperature may be lowered or possibly the end state has been reached. If all attempts have been tried and no better routes could be found, then the algorithm stops and returns the best solution. |

| | |
|---|---|
| Slide 28:<br>Tuning the algorithm is the process of adjusting the configuration parameters and measuring the algorithm's performance.<br><br>I ran a number of test scenarios and looked for results where the generated route distances did not appear to increase and for which the amount of time to run the algorithm showed a drop. As shown on the left, distance results are about the same up through Test E-0. Afterwards, distance begins to increase. As shown on the right, reducing the configuration parameters from 2000 attempts to 500 can lower the execution time for the algorithm from about 4 seconds to less than one second.<br><br><<And I'm showing that here.>><br><br>My final configuration parameters were 500 maximum attempts at each temperature with 150 maximum successes. | Slide 29:<br>With these configuration parameters now set, I wanted to see how the algorithm performed for different numbers of POIs.<br><br>More POIs typically means a longer route. As shown on the plot to the left, the total route length (or distance) increases as more POIs are added as expected. The time for execution would also be expected to increase because more paths have to be evaluated when increasing the number of POIs. What I found is generating the best solution among 5 random POIs takes about 200 milliseconds. The time increases to about 800 milliseconds when all 30 are included.<br><br><<And I'm showing that here with the averages once again.>><br><br>Slide 30:<br>Now for the demo web app. |
| Slide 31:<br>On the right is my user interface for an historical walking tour web app in Castroville, Texas. There are 30 POIs. Highway 90 is the wide roadway that crosses through the historic district from east to west. As shown here, a tour has already been generated (shown in orange). The tour is displayed in the list on the right.<br><br>On the left side of the screen are the control buttons.<br><br>NEXT:<br>The first thing to see is the "Build a Tour" screen. On this screen, the user can filter POIs as shown in the orange region.  Also, the user can constrain the tour as shown in the green region at the top. | Slide 32:<br>Next, the user can set their starting position manually, or the app can find the user's current position on the street network and use that.<br><br>Slide 33:<br>Finally, the user can track their position. This way the location icon (the man on the map) moves as you walk the route. |

Slide 34:
I’m going to show the app executing next.

Frequently I see walking tour web apps with static, predefined tours. However, what I’m going to show here is a web app that uses the Route30 library and allows the user to tailor a tour to suit their interests and needs.

<<Press Start Tour>>
So, the first thing that is going to happen is a route will be generated.

<<PAUSE>>
This is the route generated for 30 POIs. I’m going to show how you can select POIs from the list on the right, from icons on the map, or the navigation bar at the bottom. When you select a POI, a popup opens with more information.

(Slide 34 Continued)
Note that the route generated is over 5 km, and many users may not want to walk that far. So, I’m going to filter out some POIs, constrain the tour length, and then regenerate the route. I’m going to open the Build a Tour screen again.

<<PAUSE>>

Now the route is within 3.5 Km. Note that when limiting the length, the route generated will change if the user’s starting position changes because POIs furthest from the start will be skipped.

Finally, all of the files for the library and demo web app are in GitHub.

<<PAUSE>>
You can see the link here to the GitHub repository on the license screen. I’m going to describe the repository in the next

Slide 35:
So, how would you use the library in your own application?

Slide 36:
The link on the license screen that I’m showing here will take you to the GitHub repository. The library and demo web app are available to anyone under an MIT-type license that allows free access to the software.

NEXT:
As shown here, the files in the repository are organized in folders. To deploy the library, you transfer the folders to your own web server with the exception of the demo folder.

NEXT: (Slide 36 Continued)
The web app we were just looking at is in the demo folder. Someone could take these files and use them as a starting point. You would place these files in your root directory on the web server.

NEXT:
The five library files are here. The file names indicate if the source code is for the distance component or routing component or for one of the algorithms.

Slide 37:
There are three data files, and I describe them more completely in the user guide on GitHub, but I'm summarizing the important points here.

The first file is a Points of Interest file. I used the Castroville Area Chamber of Commerce tour guide to find many of the properties. Others I found myself while walking the tour.

The routing library requires two fields: an ID field, and a filter criteria field. You can have any number of other fields for your own application. The other fields I'm showing here are used in my demo web app.

Slide 38:
The criteria field in this file is used to specify any number of filters on POIs.

Slide 39:
I showed the data files in tabular format because they're easier to view this way, but the library actually uses them in GeoJSON format as I'm showing here.

In the code on the right, I'm specifying one POI as a feature within a collection. GeoJSON is a standard data format that many applications can read. And, it's easy to work with data in this format. For example, you can edit the file using a text editor.

For the POI on the right, I'm showing the required information in orange. In addition to what I showed previously, a geometry is also required for specifying the location of the point.

I created this file in QGIS. I first created a simple point Shapefile and then exported it to GeoJSON format.

Slide 40:
One other thing to note. When you view the POI GeoJSON file in GitHub, you can see the POIs overlaid on a map of Castroville.

Slide 41:
Next the street network which consists of nodes and edges GeoJSON files. First, I downloaded the street network shapefiles from Harvard Dataverse. I edited the shapefiles and kept the fields needed for the library. You can have extra fields; I just wanted the files as compact as possible. In both files, I added a barrier field shown in the right-most column to account for barriers on the street network as I described previously.

Each row in the Edges File is a segment of the street network with a length in meters. The segment has a starting node and ending node corresponding to the "From" and "To" fields.

Slide 42:
The thing to note is that the IDs of these "From" and "To" fields need to correspond to one of the IDs in the Nodes file. This is done for you if you download the data from Harvard Dataverse, but you would need to take this into account if you build your own street network.

Slide 43:
For the edges file, I added filter criteria to allow the user to constrain the street segments to ones that are more accessible (or easier for someone to walk along). Any amount of filter criteria could be added to this field for your own applications.

After editing the street network shapefiles, I exported them to GeoJSON format for use in the routing library.

Slide 44:
Now, with an understanding of the format and content of the data files, how do you invoke the library?

There are only two commands (or calls) to the routing library, and I'm showing them in orange here.

First the web app commands the library to initialize itself with parameters it supplies.

The library responds by initializing.

Afterwards, the web app commands the library to generate a route while passing any user-specified filters or constraints.

The library responds with its generated best solution.

I'll describe these steps in more detail next.

Slide 45:
The web app calls the first entry point, method R30LoadData, and passes the location of the three files: (POIs, Nodes, and Edges). All the entry points to the library are in the JavaScript file route30.js shown at the top of the list.

Slide 46:
When the library executes the routine, it first imports the routing library components which are the four modules shown here. These four modules include the distance matrix component, the route generation component, and their associated algorithms.

Slide 47:
Next, the routing library reads the three data files.

Slide 48:
Moving on to the next step.
When the user wants a route to be generated, the web app calls method R30Generate and passes a route parameter. This is the second entry point to the library.

Slide 49:
The route parameter is in this format.

The first entry is the maximum distance and, in this case, constrains the route to 3.5 km.

The second entry is the user's starting position.

Next are filters for POIs followed by filters on edges.

<<No Change>> (Slide 49 Continued)
And last are parameters specific to the simulated annealing algorithm. I'm not going to describe all of them here, but the last two are the configuration parameters I discussed previously.

In your own web app, you can use these values or modify them to suit your problem.

As I further develop the User Guide on GitHub, I will include a section that describes how to perform a tuning analysis to set these parameters.

Slide 50:
In the last step, the routing library responds by running the algorithms and returning its best solution in GeoJSON format.

This generated route includes three items for display on a web app:

a total route distance in kilometers,

an ordered list of POIs to be visited,

And Third, a geometry specifying the line segments making up the generated route.

Slide 51:
So, given all this information on the design, input files, and how to use the library in your own application, the last topic I want to talk about are the benefits to using the library.

Slide 52:
I wanted to do at least some comparison to routes generated by a server-side solution. So, I used the OpenRouteService API to generate a route in Castroville.

OpenRouteService is a popular server-side solution, and it’s easy to use. I can supply coordinates for a set of POIs, and it will generate a route for me.

NEXT: (Slide 52 Continued)
This is the route I generated using Route30. The two applications produce similar results, showing small differences in the route and total route length.

However, to do this comparison, I removed all the barriers from the street network because OpenRouteService does not know about these barriers, and I could not enter them into its dataset. So, in the solutions that I’m showing here the routes can cross Highway 90 anywhere.

NEXT:
The route should only cross at one location to the east.

NEXT:
Not this other one.

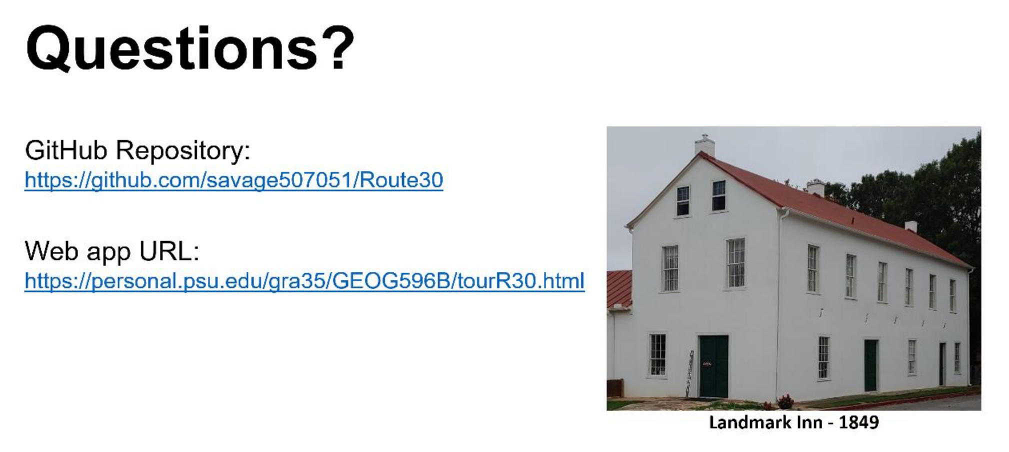
Slide 53:
Therefore, one of the advantages to using the library is you have direct control over the inputs. For example, you can specify barriers and filter criteria on the street network.

2. Second, the library requires minimal data input, only three files, and the data that is input uses a standard GeoJSON format.

3. Third, the algorithms were developed as plug-ins and can be replaced. This would allow anyone to test new variations or test completely new algorithms.

4. Fourth, the library and demo web app are free for anyone to use. I believe it would also be easy to implement a new solution by starting from the demo web app. Also, because the library is a client-side library, you do not have the expense of a routing server or need additional skilled staff to maintain, manage, or upgrade a routing server.

| | |
|---|---|
| <<No Change>> (Slide 53 Continued)<br>5. Fifth, you can tune the algorithm to your own problem which can allow you to generate solutions as accurately and efficiently as possible.<br><br>6. And last, the library is expandable. There are other routing scenarios that are possible. My work here is limited to finding an optimal path through a set of POIs where the user starts at one point, travels through the other POIs and returns to their starting position.<br><br>However, another possible routing scenario is the user may want to find just the shortest path between two points. When I implement other routing scenarios in the future, I can add more entry points to the library. | Slide 54:<br>Possible future uses, and I would like to have input from you on any additional you may see:<br><br>1. In addition to a tourist app, we believe the library may be useful to public works departments and could help them when sending people to the field to survey equipment.<br><br>2. Second, If the nodes and edges network files do not exist, you can build your own. So, for example, a network could be built for trails in a natural area and the library could be used to provide routes through that natural area.<br><br>3. Similarly, the library could be used for museums or warehouse routing. |
| <<No Change>> (Slide 54 Continued)<br>4. The library may also be useful for small business delivery operations. For example, an auto parts store could setup a web app to deliver parts to its customers.<br><br>5. Last, we believe the library could be used as an experimental platform to test new or modified routing algorithms for example by changing out the A-Star and Simulated Annealing algorithms. | Slide 55:<br>In summary, I described Route30, a free and open-source software client-side routing library and its structure.<br><br>I demoed a web app that uses the library.<br><br>Third, I described how you can find the library on GitHub, setup the input, and call the library from your own web app to retrieve a GeoJSON formatted route.<br><br>Last, I described what I see as benefits to the library. In particular, you may not be able to generate a feasible route without direct access to the inputs. So, the client-side library may be your best option and possibly your only option for generating a route depending on your problem. |

| | |
|---|---|
| <<No Change>> (Slide 55 Continued)<br>Also, the library is an affordable solution, and we believe it would be easy to implement by people who may not have much experience developing web apps.<br><br>The image on the right is a house that I ran across while working on the web app. Some houses that appear to be historic properties are not recorded anywhere. So, a web app like the one I developed could be used to identify properties that may be historic. | Slide 56:<br>I would like to acknowledge Dr. Jan Oliver Wallgrun at Penn State who was my advisor throughout this project. His help has been greatly appreciated.<br><br>I would also like to acknowledge:<br>• Ms. Adrienne Goldsberry who is here from Penn State helping me out today and<br>• Ms. Lora Robbins, City of Hondo, TX who reviewed the web app and provided me feedback.<br><br>Slide 57:<br>Any questions. |
|  **Questions?**<br><br>GitHub Repository:<br>https://github.com/savage507051/Route30<br><br>Web app URL:<br>https://personal.psu.edu/gra35/GEOG596B/tourR30.html<br><br> <br>**Landmark Inn - 1849** | |