Development of Route30

A User-Configurable, Client-Side Routing Library

Final Report

Advisor: Dr. Jan Oliver Wallgrün

George Adams

May 2, 2022

GEOG596B

Table of Contents

GEOG596B: Development of a User-Configurable Client-Side Routing Library

Author: George Adams

Advisor: Dr. Jan Oliver Wallgrün

Date: May 2, 2022

Subject: Final Report

Abstract: The purpose of this research was to develop a user-configurable, client-side routing library and demonstrate its functionality in an historical walking tour web app. The library and web app are written in JavaScript. They are free and open-source software (FOSS) available on GitHub (Adams, 2022a). The routing library generates close to optimal routes based on the street network and user-supplied filters and constraints. Certain algorithms (e.g., simulated annealing, A-Star) were implemented in the routing library. The efficient implementation of these algorithms forms the research element in this project.

1. Introduction

Routing solutions are frequently supported by server-side routing engines. See for example, Open Source Routing Machine (OSRM, 2021), the pgRouting library (pgRouting, 2021), and OpenRouteService (ORS, 2021). In this project, a free and open-source software (FOSS) routing library was developed as a client-side solution to the routing problem. The routing problem is often referred to as the traveling salesman problem (TSP) which is NP-hard. It has no quick solution, and its complexity increases significantly as more nodes are added (Ma, 2020). Because of this complexity, routing algorithms are designed to run within the computational power of present-day devices to find a best solution, close to optimal, for the TSP. Routing algorithms seek an optimal (e.g., shortest distance) route through a set of waypoints.

For this project, the intent of the routing library is to allow users to configure a route to meet their interests and needs. Often, tourist web apps have static, predefined routes. See, for example, walking tours for the City of McKinney, Texas (n.d.) and City of Bellingham, WA (n.d.)[1]. Also, sometimes a route is not even displayed as in City of Windsor, Colorado (n.d.)[2]. Therefore, a user-configurable routing library may be useful for the tourism sector. In this project, a tourist web app was developed to demonstrate the use of the routing library. Tourists can configure routes past structures that interest them and constrain the walking tour to more accessible[3] roadway segments. The walking tour was developed for a sample of 30 structures in Castroville, Texas.

Castroville, Texas is a small community (population around 3,000) within about 30 miles [48 km] west of San Antonio. It is one of the original settlements in Medina County, Texas, and it was settled by Alsatians immigrating to Texas in the 1840s (City of Castroville, Texas, n.d.). There are more than 70 historical buildings (mostly small homes) located in the central area of

[1] These walking tours are Esri Map Tour Story Maps (Esri, 2013).
[2] This walking tour is an Esri Shortlist Story Map (Esri, 2021).
[3] Castroville is a small town, and input datasets do not show Americans with Disabilities Act (ADA) – compliant street segments. I walked the routes to identify street segments that are more accessible to someone who may have difficulty walking.

Castroville. See the Castroville Area Chamber of Commerce (CACC) print-version historical walking tour guide (CACC, 2017, p. 38) for more information.

2. Literature Review

A literature review was conducted to identify current tourist apps and personalized routing apps and their functionality. The review focused on challenges identified and innovative concepts used in these apps. Important to this project is determining how others employed user-configurable routing.

2.1 Tourist Apps

Fino et al. (2013) describe an historical tourist app they developed for the city of San Cristobal de La Laguna. In addition, they describe the importance of tourism to UNESCO World Heritage sites (UNESCO, 2021) and the increasing interest in cultural tourism. Interestingly, they use QR codes on a tourist map to direct users to information about historical structures along two predefined walking tour routes. They also use augmented reality to make images of the buildings appear as users travel the routes. Augmented reality can enhance the user's experience, so this is a useful concept for an historical tourist app. Augmented reality was not intended for this project, but it is something that could be pursued in the future. Fino et al. (2013) predefined two possible routes in their tourist app. The research element in this project to employ user-configurable routing could enhance the work Fino et al. (2013) are doing. In the Route30 library, instead of two predefined routes, a user could instead filter POIs and develop their own route to suit their preferences.

Alamäki and Dirin (2014) describe the development of a tourist app for small tourism companies in Finland. Their app focuses on outdoor activities such as walking, hiking, biking, and kayaking. Alamäki and Dirin (2014) determined that small tourism companies have limited funding, so they currently supply hard copy maps to their customers. Their improvement was to supply an affordable, location-aware web app to these small tourism companies. They offered their app using the Software as a Service (SAAS) delivery model. Importantly, the small tourism companies wanted to edit the routes and points of interest themselves, so this capability was also provided. However, it did not appear that individual users of the app could configure the routes to suit their needs and preferences. So, the research element in this project to develop user-configurable routing could be beneficial to the work being done by Alamäki and Dirin (2014). Alamäki and Dirin (2014) describe some challenges in developing the tourist app. They found the app can be hard to see in the bright sunlight and sometimes users experience weak internet connections. They developed larger navigation controls to address challenges from bright sunlight, and to address the weak internet connections, eliminated features requiring continuous online access. For the research here involving the demo web app, the user interface was tested by walking routes in Castroville. The work from Alamäki and Dirin (2014) informed the user interface design. For example, the control buttons were made intentionally large. Also, the web app does not need to have a strong, continuous internet connection. The tourist can track their position as an option, and this works well in Castroville while walking on a tour, but it is not necessary for the app to work.

2.2 Personalized Routing Apps

Keler and Mazimpaka (2016) describe state of the art personalized routing applications that generate routes based on a person's context and environment. For example, a less experienced

driver may want a less challenging route instead of a shortest-path route. Also, a traveler may want a safer route, so Keler and Mazimpaka (2016) incorporate information such as police stations, lighting, and crimes when generating safe travel routes through Los Angeles. They developed a danger index (i.e., I_{dang}) to connect the level of danger to the costs in travelling a roadway segment. Essentially, they use a 'safety cost function'[4] in their routing algorithm to find the safest travel path through Los Angeles with higher danger areas being more costly to travel and therefore less likely to appear in a generated route. They also incorporate the idea of obstacles to routes. Obstacles are locations of crime hot spots.

The use of a specialized cost function in other routing applications is also possible. For example, Novack et al. (2018) allow pedestrians to plan their route through green areas, social places (e.g., restaurants), and along quieter (i.e., lower traffic) roadways. Key to their work is quantifying specific aspects important to pedestrians within a cost function for the routing algorithm[5]. The cost function reflects weights pedestrians assign to characterize the importance of street length, greenness, sociability, and quietness.

These personalized routing app studies reveal that when factors are important to travelers other than just the shortest path or shortest travel time, researchers integrate costs into the routing algorithm to account for these additional factors. For this project, the intent is to filter structures important to the tourist and then generate routes through this subset of structures. However, the concept of a specialized cost function or priority scheme may be useful to this project. For example, properties such as the Henri Castro homestead, which is an historical landmark [Society of Architectural Historians (SAH), 2021)], may be assigned a higher priority to ensure generated routes include this high-priority building. The Points of Interest (POI) input file includes a priority field (See Section 6.1), and in the future, a priority scheme could be implemented in Route30.

3. Accomplishments

The primary objective of this project was to develop the Route30 library, a user-configurable, client-side routing library that can be used by people developing web apps. A secondary objective was to develop a web app that demonstrates the use of this library. The demo web app is in an historical walking tour for Castroville, Texas. Figure 1 is a high-level overview of the routing library also showing the library's relationship to the web app and data files stored on the server. Three specific algorithms were developed for Route30: a routing algorithm (i.e., simulated annealing) used to generate a close-to-optimal route, a path-generation algorithm (i.e., A-Star) used to generate a distance matrix on-the-fly, and an algorithm used to limit the length of the generated route. All the algorithms were developed in JavaScript and included in the Route30 library.

[4] Keler and Mazimpaka (2016) use a safety cost function and compare their results to the more traditional shortest path strategy which uses a length (or distance) cost function.
[5] Novack et al. (2018) also describe the importance of users being able to compare the shortest and customized route. This comparison allows them to adjust their preferences, if necessary, to develop a more satisfying customized route.

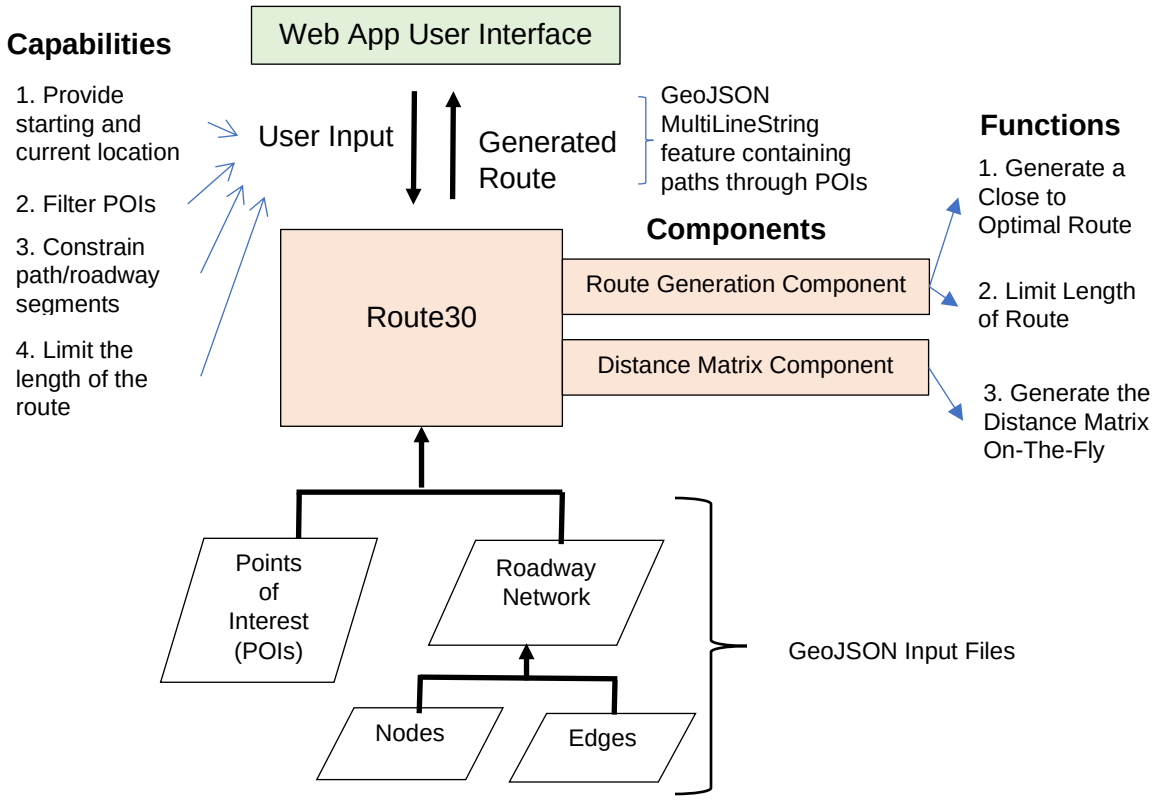


Figure 1. High-Level Overview of Route30, also Showing its Relationship to the Data Input Files and the Web App User Interface

Specific Route30 accomplishments follow:

1. **Route30, a user-configurable, free and open-source software (FOSS) routing library was developed.** This was the main goal for the project. The library and associated demo web app are freely available on GitHub (Adams, 2022a). The library generates close to optimal (or shortest path) routes through a series of POIs. The generated route is constrained by user input to include filtering POIs, constraining the path to specific types of roadway segments, and limiting the length of the optimal route. The library design is described in Section 4.
2. **The library functionality is demonstrated in a tourist web app.** A tourist web app was developed for Castroville, Texas (Adams, 2022b). This web app includes 30 historical structures as POIs. Anyone can access the web app from their smartphone and take a walking tour in Castroville. The tourist web app is described further in Section 5.
3. **The library was built to require minimal data input with data input files using a standard data format.** The library was also developed to be as general as possible. Only three input files are required: a POI file and a street network consisting of a Nodes file and an Edges file. All three files are in GeoJSON format. The only file developed from scratch was the POI file, but it is a simple point file created from QGIS. The street

network files were downloaded from Harvard Dataverse (Boeing, 2017) and edited. The format of data input files is described further in Section 6.

4. **A tuning analysis was conducted to set the simulated annealing configuration parameters. Afterwards, Route30's performance was evaluated.** The simulated annealing algorithm has two configuration parameters that can affect the quality of the output and the computation time for the algorithm. These parameters were set using a tuning analysis described in Section 7. Afterwards, Route30's performance was evaluated, and this evaluation is described in Section 8.
5. **The route generated from Route30 was compared to the route generated from a server-side solution.** The OpenRouteService (ORS) API (ORS, 2021) was used for this comparison which is described further in Section .

4. Routing Library Design

Route30 consists of five JavaScript files shown in Table 1. It has an entry-point file (route30.js) and two components: a distance matrix component (route30_Distance.js) and a route generation component (route30_Route.js). The A-Star algorithm (route30_AStar.js) is a plug-in to the distance matrix component. The simulated annealing algorithm (route30_Simulated_Annealing.js) is a plug-in to the route generation component.

Table 1. Route30 Library Files

File	Description
route30.js	Entry points to the library
route30_AStar.js	A-Star algorithm
route30_Distance.js	Distance matrix component
route30_Route.js	Route generation component
route30_Simulated_Annealing.js	Simulated annealing algorithm

Both the distance matrix component and the route generation component receive a route parameter from the web app. This parameter communicates the user's position and user-specified filters and constraints. It also passes problem-specific configuration parameters to the simulated annealing algorithm. Details of the route parameter are shown in Figure 2.

```
var routeParameter = {
        maxDistance:   3.5,
        userStart:     [-98.87934, 29.35596],
        poiFilter:     [["house", "church", "commercial"],
                        ["1800s", "1900s"]],
        edgeFilter:    [["acc_yes", "acc_no"]],
        algorithm:     [5.0, 0.9, 200, 500, 150]
}
```

Figure 2. Route Parameter

Figure 2 shows the details of the Route Parameter (i.e., routeParamter). The maximum distance field (i.e., maxDistance) constrains the route to the specified distance in kilometers. If this value

is “-1”, then no constraint exists on the distance. The user’s starting position field (i.e., userStart) specifies the coordinates for the user’s starting position. It is currently set to the location of POI 3 or the St. Louis Church. The poi filter field (i.e., poiFilter) identifies any filters on POIs. The array [“house”, “church”, “commercial”] allows houses, churches, and commercial properties to be displayed and used in the routing algorithms. The array [“1800s”, “1900s”] allows POIs from the 1800s and 1900s to be displayed and used in the routing algorithms. Only POIs matching the POI filter labels will be displayed and used in the routing algorithms. The other POIs are filtered out. The edge filter field (i.e., edgeFilter) identifies any filters on edges. The array [“acc_yes”, “acc_no”] allows all edges (or segments) to be included in the routing algorithms. Only edges matching the edge filter labels will be included in the routing algorithms. The last parameter, the algorithm parameter, passes parameters specific to the simulated annealing algorithm. It specifies the initial pseudo temperature, the temperature factor (i.e., the factor used to reduce the temperature for the next iteration), the number of temperature steps, and the configuration parameters of maximum attempts and maximum successes. So, the array [5.0, 0.9, 200, 500, and 150] corresponds to [initial pseudo temperature, temperature factor, number of temperature steps, maximum attempts, and maximum successes].

4.1 Distance Matrix Component

The distance matrix component generates a distance matrix as input to the route-generation component. Each cell of the distance matrix has the minimum distance and path between each pair of POIs. To build this matrix, the distance matrix component first generates an adjacency list and nearest-nodes list. It then invokes the A-Star algorithm passing these lists as parameters. The adjacency list, nearest-nodes list, A-Star algorithm, and output from the distance matrix component are described in the following sections.

4.1.1 *Adjacency List*

Ultimately, the nodes and edges in the path/roadway network will be used in the path-generation algorithm [e.g., the A-Star (or A*) algorithm]. The algorithm needs to know how nodes are directly connected to each other. Two nodes that are adjacent are directly connected by an edge, and the two nodes define the endpoints of that edge.

The adjacency list is built from the street network GeoJSON nodes and edges input (Described in Section 6.2). The JSON structure below shows an adjacency list of two nodes (node ids of "227032018" and "227034390"). The first portion of the structure shows that node "227032018" is connected to node "227026795" by edge 9. It is also connected to that same node by edge 10.

JSON Adjacency List

```
{
    "227032018": [
        {
            "node": "227026795",
            "edge": "9"
        },
        {
            "node": "227026795",
            "edge": "10"
        }
    ],
    "227034390": [
```

```
            {
                "node": "227217926",
                "edge": "20"
            },
            {
                "node": "227034471",
                "edge": "21"
            },
            {
                "node": "227230353",
                "edge": "22"
            }
        ],

        …. Additional nodes follow the same format as above….

}
```


4.1.2 *Nearest-Nodes List*

To simplify calculations, POIs are assumed to be located at the nearest node of the street network. A nearest-nodes list specifies the relationship between each POI and its nearest street node. The JSON structure below shows four POIs (Numbered “0” to “3”). POI “0” is located nearest to street node "227166358".


```
{
    "0": "227166358",
    "1": "227166343",
    "2": "227183527",
    "3": "227183527",

    …. Additional nodes follow the same format as above….

}
```


4.1.3 *A-Star Algorithm*

The A-Star plug-in (Figure 3) is accessed from the distance matrix component. The A-Star plug-in contains interface code as well as the A-Star algorithm itself. The distance matrix component passes the adjacency and nearest-nodes lists along with the POIs and street network (nodes and edges). In addition, the distance matrix component passes a route parameter which specifies any edges to be filtered out. Each pair of POIs is processed using the A-Star algorithm. This algorithm determines the shortest path and distance for each pair. The plug-in compiles this information and returns a distance matrix consisting of path and distance structures (Section 4.1.4).

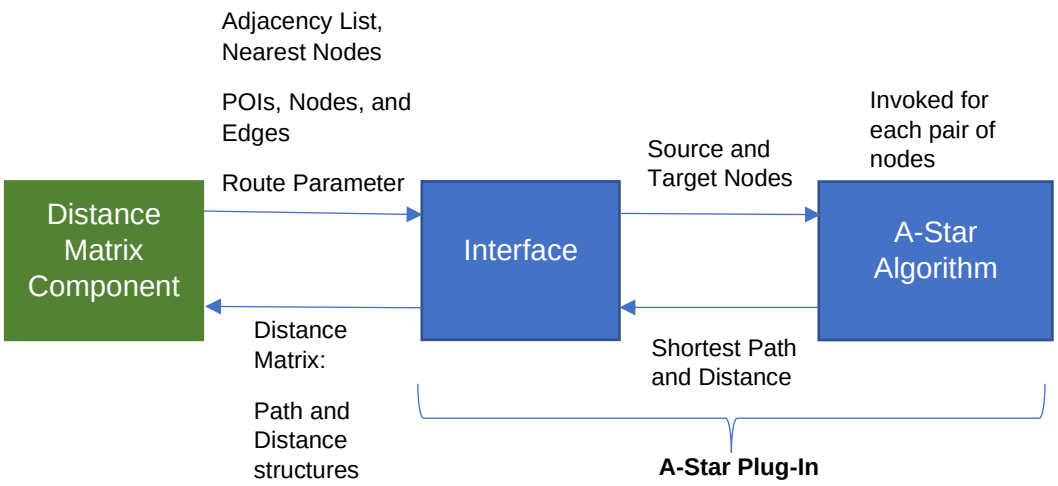


Figure 3. A-Star Plug-In showing Interface to Distance Matrix Component

There are two primary algorithms for finding the shortest path between POIs. One is Dijkstra's algorithm and the other is the A-Star (or A*) algorithm. A-Star is a modification to Dijkstra's algorithm. Whereas Dijkstra's algorithm will search out equally from the source node, A-Star prioritizes its evaluation using a heuristic[6]. It tries to identify the best next node for reaching the target. With its focused search, A-Star minimizes the number of calculations and can be faster than Dijkstra's algorithm, and therefore it was selected for Route30. Dere, Esat, and Durdu (2018) describe their use of the A-Star algorithm for transportation systems. Swift (2017), Lester (2005), and Roy (2019) describe the A-Star algorithm with pseudocode or actual code.

4.1.4 *Output from Distance Matrix Component*

The A-Star plug-in passes a distance matrix to the distance matrix component. The distance matrix component forwards the distance matrix to the route generation component (Section 4.2). The distance matrix is made of two structures: a distance structure and a path structure.

The distance structure follows and is in JSON format. This structure shows the shortest paths among POIs. For example, the shortest path from POI "0" to POI "1" is Path "227166358-227166343". The shortest path includes a distance and geometry and is characterized in a separate path structure which follows after the distance structure.

JSON Distance Structure

```
{
    "0": {
        "0": null,
        "1": "227166358-227166343",
        "2": "227166358-227183527",
        "3": "227166358-227183527",
        "4": "227166358-227154701",

        …. Additional POI-Path pairs follow the same format as above….
```

[6] The heuristic is a guess of the remaining distance from the current node being evaluated in the algorithm to the target node.

```
    },
    "1": {
        "0": "227166358-227166343",
        "1": null,
        "2": "227166343-227183527",
        "3": "227166343-227183527",
        "4": "227166343-227154701",

        …. Additional POI-Path pairs follow the same format as above….

    },

    …. Additional POIs follow the same format as above….

}
```

Paths are included in the following GeoJSON-formatted structure. Each path is a feature within a collection. As shown in the following structure, path "227166358-227166343" has distance 0.23 km. It is represented by two separate coordinate arrays (or two line segments) within a MultiLineString geometry.

Path GeoJSON Structure

```
{
    "type": "FeatureCollection",
    "features": [
        {
            "type": "Feature",
            "id": "227166358-227166343",
            "properties": {
                "distance": 0.23084899999999997,
                "endpoint_1": "227166358",
                "endpoint_2": "227166343"
            },
            "geometry": {
                "type": "MultiLineString",
                "coordinates": [
                    [
                        [
                            -98.87809,
                            29.35498
                        ],
                        [
                            -98.87724,
                            29.35573
                        ]
                    ],
                    [
                        [
                            -98.87642,
                            29.35646
                        ],
                        [
                            -98.87724,
                            29.35573
                        ]
                    ]
                ]
```

```
            }
        },
        {
            …. Additional path features follow the same format as above….
        }
    ]
}
```

4.2 Route Generation Component

The Simulated Annealing plug-in (Figure 4) is accessed from the route generation component. This plug-in contains interface code as well as the simulated annealing algorithm itself. The route generation component passes the POIs and distance matrix along with a route parameter. The route parameter specifies filters on the POIs, constraints on the route length, and parameters specific to the simulated annealing algorithm.

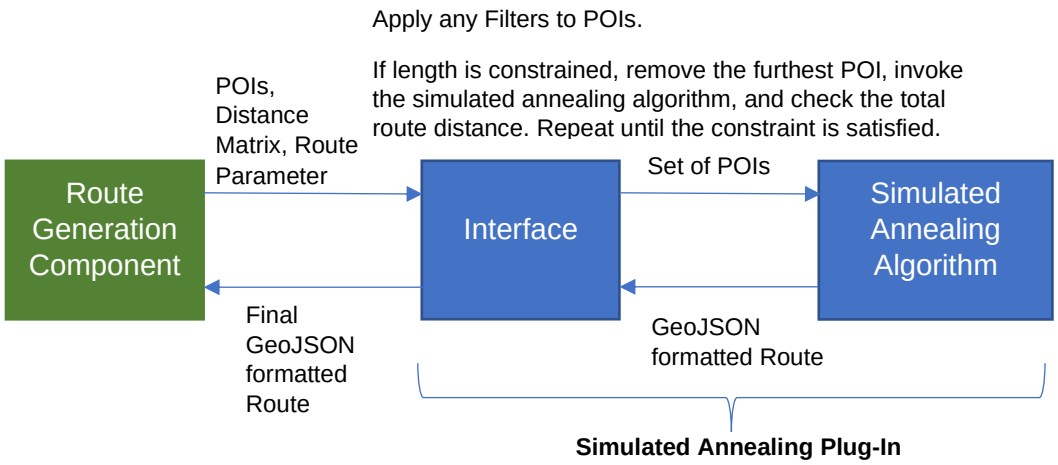


Figure 4. Route Generation Component

The simulated annealing plug-in generates a close-to-optimal route through the user's filtered set of POIs, potentially limited by a user-specified length criterion. There are two functions within the plug-in: 1) generate a close-to-optimal route and 2) limit the length of the generated route. These two functions are described in the sections that follow.

4.2.1 *Generating a Close-to-Optimal Route*

The simulated annealing algorithm generates a close-to-optimal route from a distance matrix and set of POIs. 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. Lojkine (2018), Jacobson (2013), Press et al. (1992), Schneider (2014), and Walker (2018) describe the simulated annealing approach in detail.

4.2.2 *Limiting the Length of the Generated Route*

The algorithm to limit the length of the generated route was developed as part of this project's research. The limit-length algorithm determines if the user's length criterion is met. If it is not met, the algorithm identifies the POI furthest from the user's current position and removes it. It then regenerates the route through the reduced set of POIs and checks the total route length. It continues this process until the user's length criterion is met.[7] Once the route falls within the constraint, the algorithm goes back and looks at each of the POIs that were eliminated. POIs can be added back in if they do not cause the total route length to increase beyond the length constraint.

4.2.3 *Output from Route Generation Component*

The simulated annealing plug-in returns a GeoJSON route to the route generation component. This returned route is the best solution found from the simulated annealing algorithm. The route generation component forwards the route to the web app for display. Figure 5 shows the GeoJSON formatted route returned to the web app. It has three principal components: 1) a distance in kilometers, an ordered list of POIs to be visited, and 3) a geometry specifying the line segments making up the generated route.

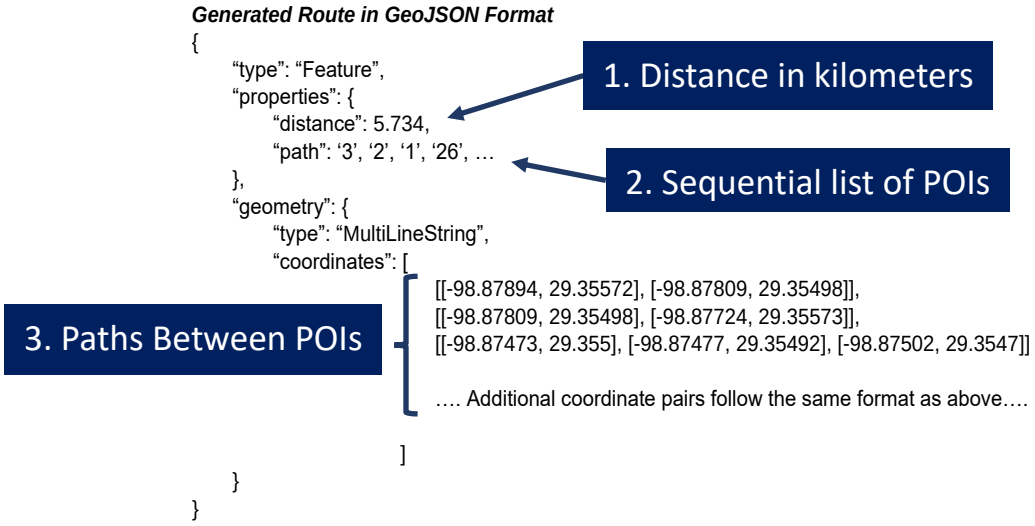


Figure 5. Generated Route in GeoJSON Format

5. Web App

Figure 6 is the user interface for the 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 in Figure 6, a tour has been generated (shown in orange). The tour is displayed in the list on the right side of the figure. On the left side of the screen are the control buttons. The functionality invoked from these buttons is described next.

[7] StackOverflow (2013) suggests calculating the Euclidean distance of each POI from the starting point and eliminating POIs that are too far away. I used this approach to limit the length of the generated route.

Figure 6. Tourist Web App

The tour button opens the Build a Tour screen shown in Figure 7. From this screen, the tourist can filter POIs and constrain the route. POIs are filtered by type of structure and era. The route is constrained to more accessible[8] segments or limited in length. After applying any desired filters and constraints, the tourist can generate a route by pressing the “Generate Tour” button at the bottom of the Build a Tour screen. A tourist could also clear the filters and constraints as well as any current tours by pressing the “Clear Tour” button.

[8]Castroville is a small town, and input datasets do not show Americans with Disabilities Act (ADA) – compliant street segments. Also, very few street segments have sidewalks. So, I walked the routes to identify street segments that may be more accessible to someone who may have difficulty walking. For example, a segment may have a steeper slope and could also be located near a busy restaurant. A segment such as this could be difficult for someone to walk along. I identified segments such as these as not accessible.

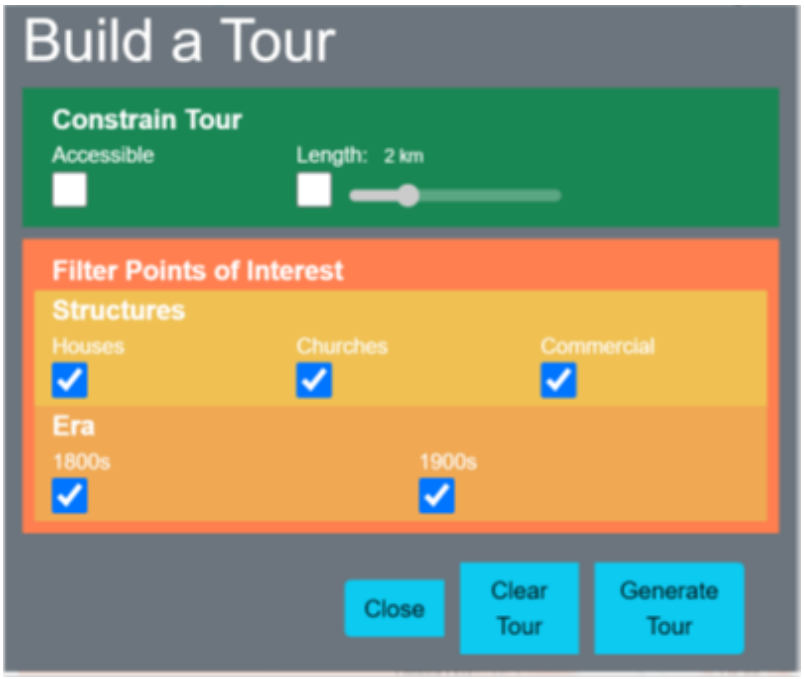


Figure 7. Build a Tour Screen

Figure 8 shows the Set Starting Position screen. The user can set their starting position manually by clicking the button on the right, or the app can find the user's current position on the street network and use that (left button).

Figure 8. Set Starting Position

Finally, Figure 9 shows the control buttons from Figure 6. The user can track their position by pressing the last button on this figure (i.e., the Start/Stop Tracking button). This way the location icon (the man on the map) moves as the tourist walks the route.

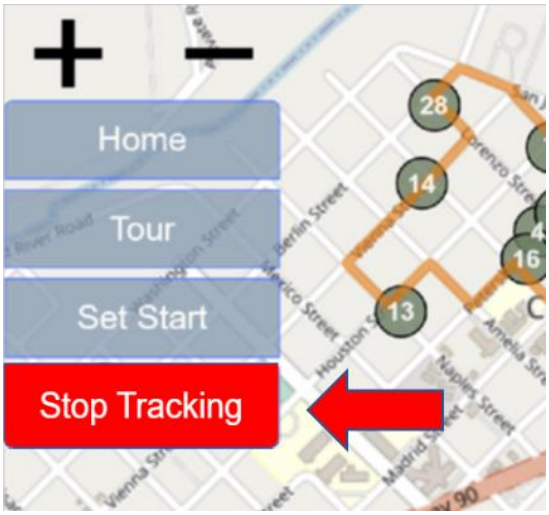


Figure 9. Start/Stop Tracking

6. Data Input Files

There are three data input files: a POI file, a Nodes file, and an Edges file. All these files are in GeoJSON format and are described in the following sections.

6.1 POI File (poi.geojson)

POIs are structured within a feature collection of point features. For each POI, the minimum information required for the routing library is a POI identifier, filter criteria, and a geometry specifying the POI's location. Although the priority property is not used, functionality could be implemented in the future. If an optimal route is constrained by length and cannot pass through all the available POIs, then the priority could be used to favor some POIs over others. Note that the array of filter criteria can be any length. This allows the routing library to be as general as possible so that others may use it for their own problem domain. In the example that follows, the POI is identified as a house built in the 1800s (in the "criteria" property)[9].

GeoJSON Points of Interest Feature Collection

```
{
    "type": "FeatureCollection",
    "features": [
        {
            "type":    "Feature",
            "id": "0",
            "properties": {
                "name": "Henri Castro Homestead",
                "priority": 1,
                "popup": "Henri Castro Homestead – 1845",
                "criteria": ["house", "1800s"],
                "Year": 1845,
                "Address": "1109 Fiorella Street"
            },
```

[9] POIs are filtered by type of structure (i.e., house, church, or commercial property) and era (1800s or 1900s) as shown in Figure 7.

```
            "geometry": {
                "type": "Point",
                "coordinates": [-98.882, 29.351]
            }
        },
        {
            …. Additional features follow the same format as above….
        }
    ]
}
```

6.2 Street Network

The street network consists of nodes and edges GeoJSON files. I downloaded these files from Harvard Dataverse (Boeing, 2017), edited them in QGIS to keep the fields needed for the library, and then exported them to GeoJSON format. The files can have extra fields, but they were developed to be as compact as possible. These files are described next.

6.2.1 *Nodes File (nodes.geojson)*

In the nodes file, each intersection (or node) is specified as a feature within a collection. Each node has an identifier (or id). In addition, a barrier property was added for the Route30 library to account for barriers on the street network. In Castroville, Highway 90 splits the historic district. Pedestrians can only cross at one of two crosswalks. One crosswalk is in the historic district. To prevent routes from crossing Highway 90 outside of a crosswalk, some nodes are designated as barrier nodes in the input file.

GeoJSON Nodes Feature Collection

```
{
    "type": "FeatureCollection",
    "features": [
        {
            "type": "Feature",
            "id": "226969131",
            "properties": {
                "barrier": false
            },
            "geometry": {
                "type": "Point",
                "coordinates": [-98.89888, 29.34825]
            }
        },
        {
            …. Additional features follow the same format as above….
        }
    ]
}
```

6.2.2 *Edges File (edges.geojson)*

The edges file represents the segments of a street network. Each segment has a length in meters and a starting node and ending node corresponding to the “from” and “to” properties. The identifiers in these “from” and “to” fields correspond to one of the identifiers in the nodes file. A filter criterion was added to allow the tourist to constrain street segments to ones that are

more accessible (or easier for someone to walk along). Any amount of filter criteria could be added to the “criteria” property. In addition, a barrier property was added to designate some street segments as 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, it is designated as a barrier in the edges file.

GeoJSON Edges Feature Collection


```
{
    “type”: “FeatureCollection”,
    “features”: [
        {
            “type”: “Feature”,
            “id”: “0”,
            “properties”: {
                “from”: “227297295”,
                “name”: [“United States Highway 90”],
                “to”: “226969131”,
                “length”: 33.386,
                “criteria”: [“acc_yes”],
                “barrier”: false
            },
            “geometry”: {
                “type”: “LineString”,
                “coordinates”: [[-98.8982, 29.34826], [-98.89864, 29.34825], [-98.89888, 29.34825]]
            }
        },
        {
            …. Additional features follow the same format as above….
        }
    ]
}
```


7. Tuning Analysis

Tuning the simulated annealing algorithm is the process of adjusting configuration parameters and measuring the algorithm’s performance. There are two configuration parameters used in the tuning analysis, and they set the number of transport or reversal processes conducted at each temperature step (See Section 4.2.1 for more information on the simulated annealing algorithm.).

These configuration parameters are the maximum successes and maximum attempts shown in the flowchart in Figure 10. 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.

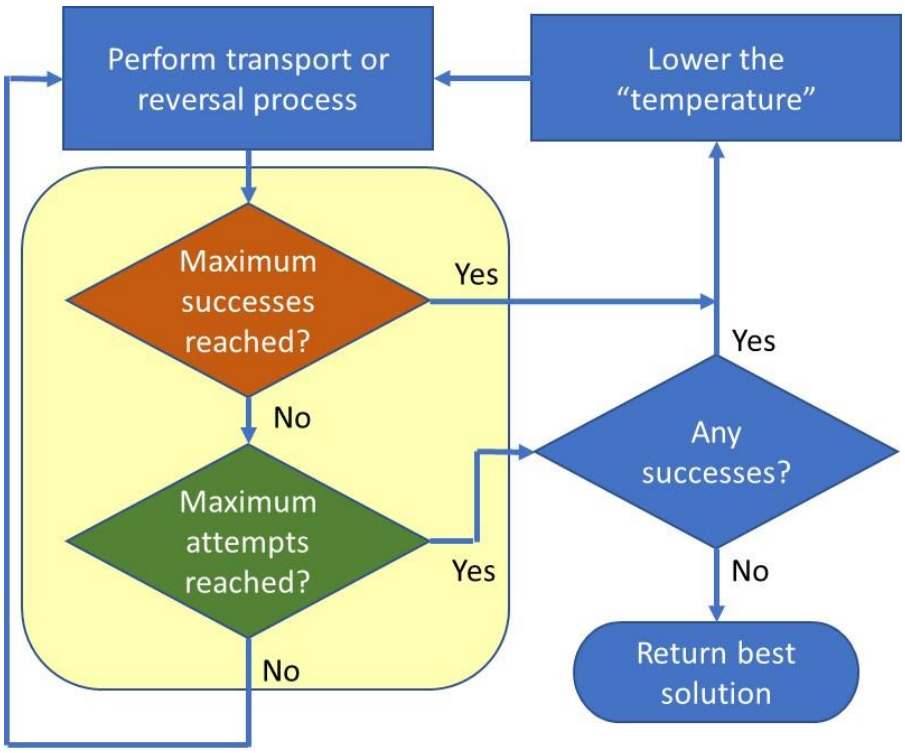


Figure 10. Flowchart showing Maximum Successes and Maximum Attempts Configuration Parameters

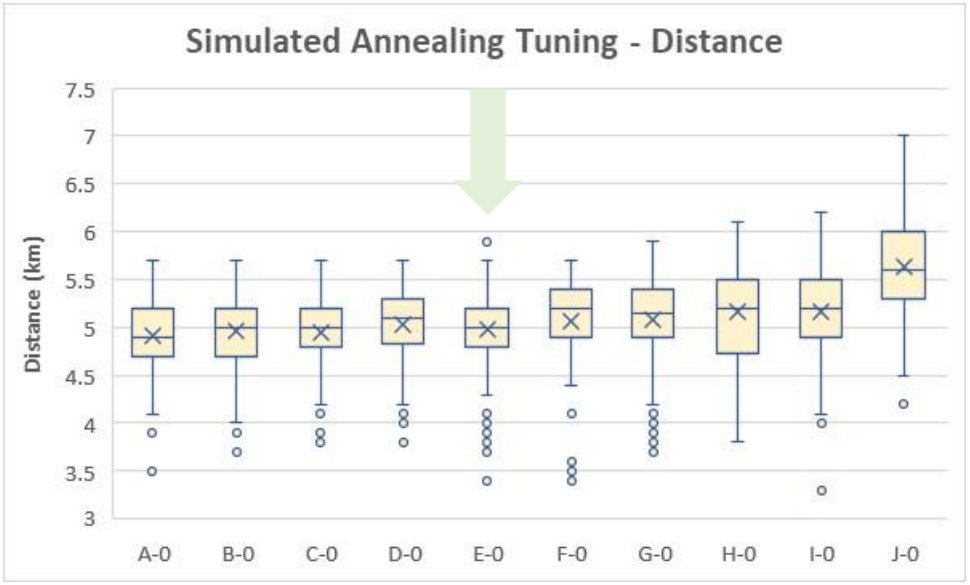
To determine the values for the configuration parameters, several test scenarios were run, and the results were evaluated for each scenario. For each test scenario, 100 routes were generated, and for each route, 20 POIs were selected at random. The goal of the tuning analysis was to look for results where generated route distances did not appear to increase and for which the amount of time to run the algorithm showed a drop.

The tuning analysis was performed in two steps. First, the value for maximum attempts was determined. Afterwards, the value for maximum successes was determined. These two steps are described next.

7.1 Step 1: Set the Maximum Attempts Parameter

As shown in Figure 11, the first parameter tuned was the maximum attempts at each temperature. The maximum successes parameter was set equal to maximum attempts which effectively removes it from evaluation in this first step of the tuning analysis. The upper plot in Figure 11 shows the change in generated route distance as maximum attempts is lowered from 2,000 down to 50. As shown in the upper plot, distance results are about the same up through Test E-0. Afterwards, distance begins to increase. As shown in the lower plot, reducing the configuration parameters from 2,000 attempts to 500 can lower the execution time for the algorithm from about 4 seconds to less than one second. So, in this first step of the analysis, Test E-0 was identified as having the best value for maximum attempts because distance results remained about the same (starting from Test A-0 and going through Test E-0), but the computation time showed a noticeable drop.

Max Attempts	2000	1500	1000	750	500	250	200	150	100	50
Max Successes	2000	1500	1000	750	500	250	200	150	100	50
	A-0	B-0	C-0	D-0	E-0	F-0	G-0	H-0	I-0	J-0



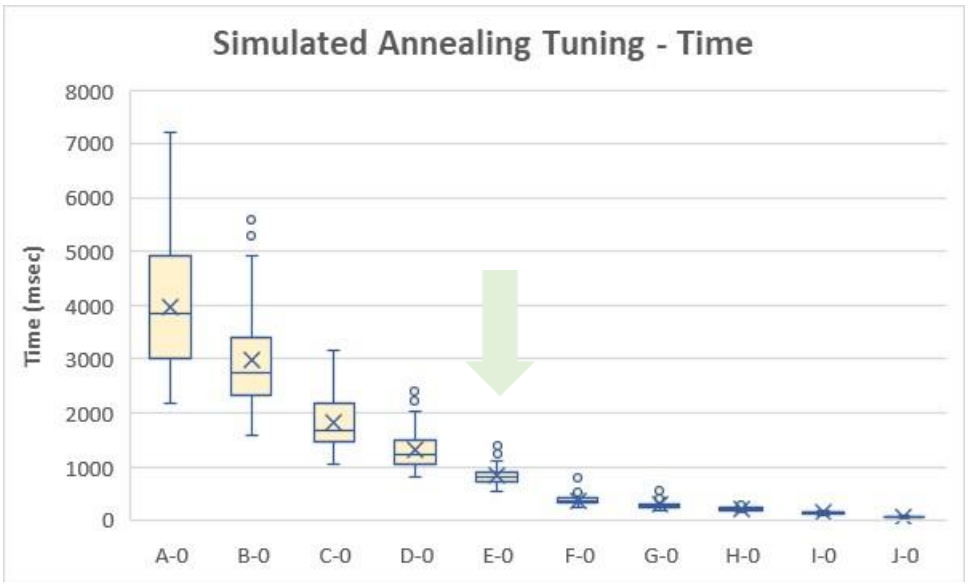


Figure 11. Tuning Maximum Attempts: Distance versus Scenario (Upper Figure), Computation Time versus Scenario (Lower Figure). The "X" marks the average for each scenario. The arrows mark the test scenario used to set the maximum attempts parameter. Selected 20 random POIs out of 30 for each repetition – 100 repetitions total for each test case.

7.2 Step 2: Set the Maximum Successes Parameter

With maximum attempts fixed at 500 from step one of the tuning analysis, the maximum successes parameter was lowered from 500 to 50 as shown in Figure 12. The upper plot shows little difference as the maximum successes parameter is lowered. Test E-7 was picked with maximum successes set to 150. This test case appeared to be the best choice because

afterwards, Test E-8 shows the average distance increasing and Test E-9 shows a wider boxplot. In the lower plot of Figure 12, computation time shows a noticeable drop as maximum successes is lowered. Changing maximum successes from 500 (i.e., Test E-0) to 150 (i.e., Test E-7) results in a drop in average computation time from just over 800 msec to just under 600 msec.

Max Attempts	500	500	500	500	500	500	500	500	500	500
Max Successes	500	450	400	350	300	250	200	150	100	50
	E-0	E-1	E-2	E-3	E-4	E-5	E-6	E-7	E-8	E--9

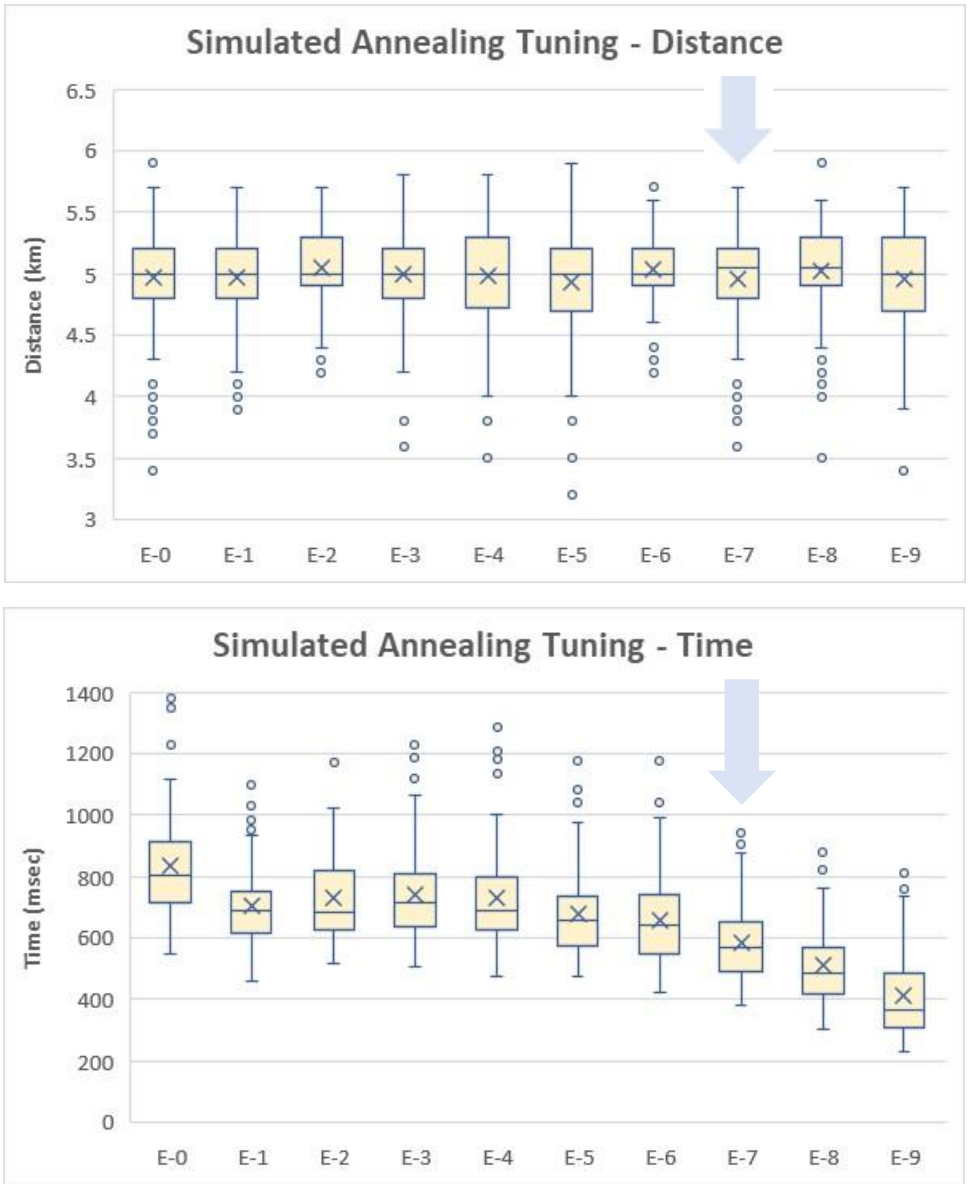


Figure 12. Tuning Maximum Successes: Distance versus Scenario (Upper Figure), Computation Time versus Scenario (Lower Figure). The "X" marks the average for each scenario. The arrows mark the test scenario used to set the maximum successes parameter. Selected 20 random POIs out of 30 for each repetition – 100 repetitions total for each test case.

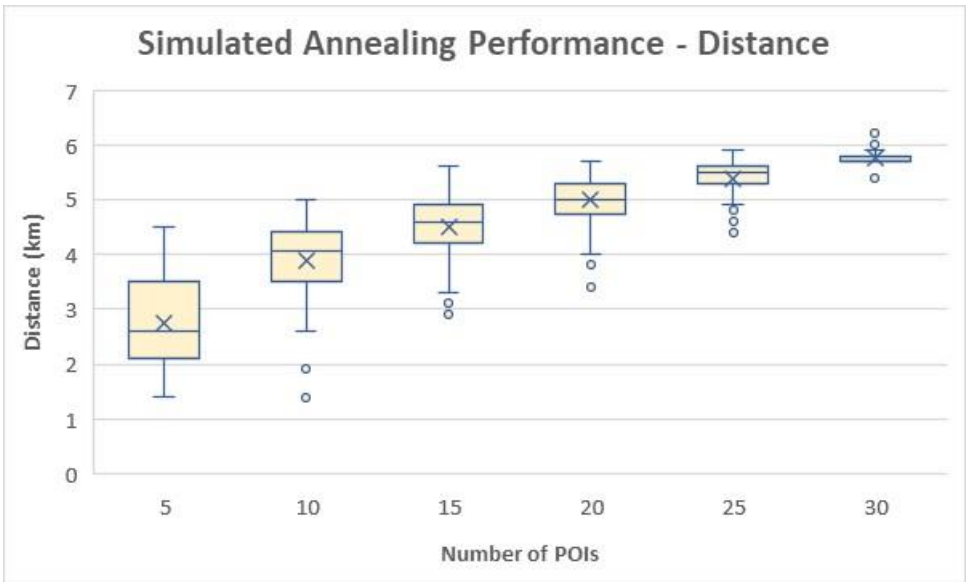
The final configuration parameters were 500 maximum attempts at each temperature with 150 maximum successes.

8. Performance Evaluation

With the configuration parameters now set (i.e., maximum attempts = 500, maximum successes = 150), it is important to evaluate the simulated annealing algorithm's performance for different numbers of POIs.

More POIs typically means a longer route. As shown on the upper plot in Figure 13, the total route length (or distance) increases as more POIs are added as expected. The computation time for execution would also be expected to increase because more paths have to be evaluated when increasing the number of POIs. As shown on the lower plot in Figure 13, generating a route among 5 random POIs takes about 200 milliseconds. The time increases to about 800 milliseconds when all 30 are included.

Number of POIs	5	10	15	20	25	30



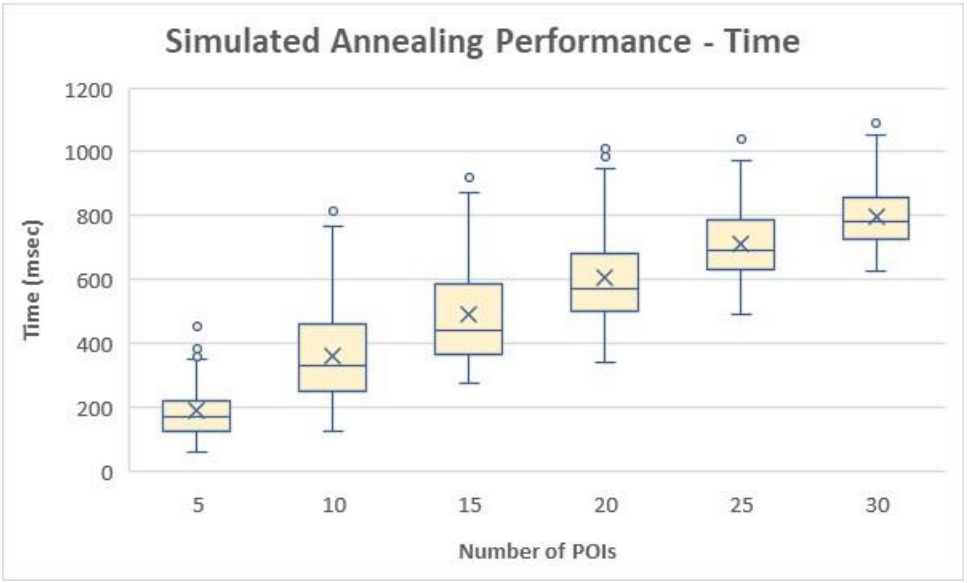
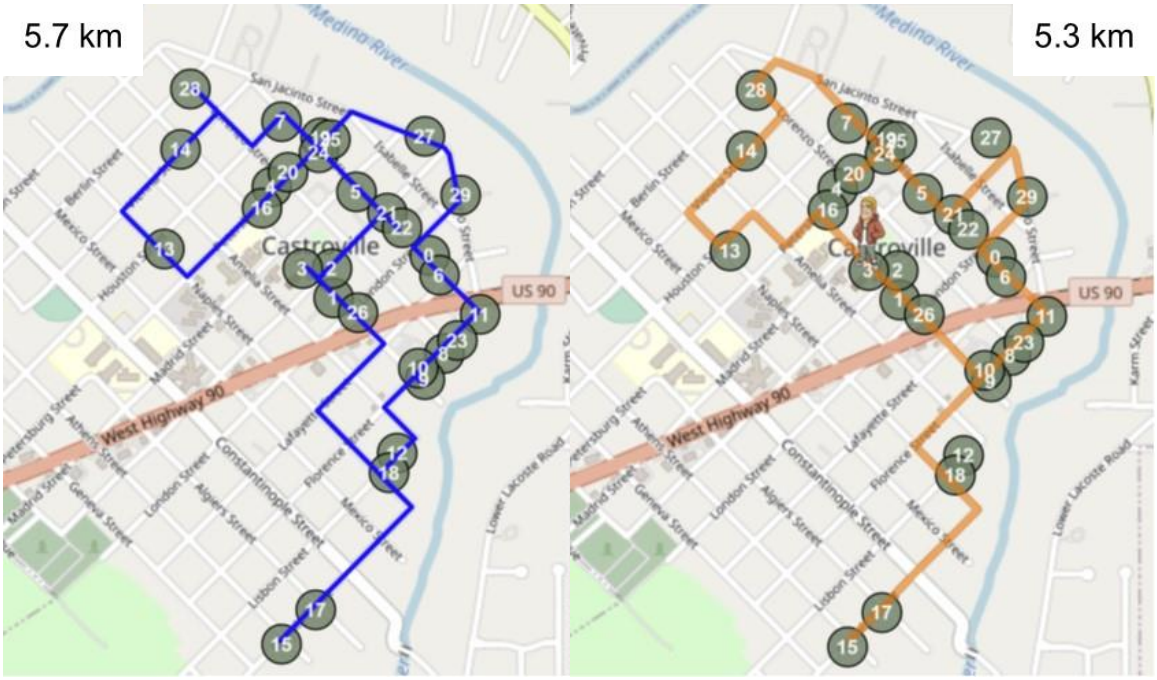


Figure 13. Performance Evaluation of Simulated Annealing Algorithm. Each test case was run with a random number of POIs selected out of 30 POIs total. Selected POIs randomly out of 30 for each repetition – 100 repetitions total for each test case.

9. Comparison of Route30 to a Server-Side Solution

The OpenRouteService (ORS) API (ORS, 2021) was used to generate a route through the same 30 POIs as Route30. The comparison of the route generated from ORS and Route30 is shown in Figure 14. To do this comparison, I had to remove all the barriers from the street network because ORS does not know about these barriers, and I could not enter them into its dataset. Therefore, the solutions in Figure 14 allow the routes to cross Highway 90 anywhere.

As shown in Figure 14, the two applications produce similar results, showing small differences in the route and total route length. A crosswalk does not exist near POI 26, but both solutions show the routes crossing Highway 90 there in these unconstrained (i.e., no barrier) solutions. Highway 90 is a busy roadway, and it is not possible to cross near POI 26. Because Route30 allows direct access to the input datasets, it may be the best option and possibly the only option for generating a feasible solution.



OpenRouteService Route30

Figure 14. Comparison of Route Generated from a Server-Side Solution and Route30

10. Summary

In summary, Route30, a free and open-source software client-side routing library was developed along with a demo web app. The demo web app may be a good starting point for someone who wants to use the library because it provides an example of how to initialize the library, generate routes from the library, and display the close-to-optimal route generated from the library.

Route30 has certain benefits. First, the library provides direct access to inputs which allows the developer of a web app to specify constraints on the street network such as barriers or filters. This can help ensure that routes generated are feasible so that someone can safely walk on the generated route and use available crosswalks. Second, the library requires minimal data input. Only three files are required (i.e., a POI file and a street network consisting of a nodes and edges file). The input data files require only a minimal amount of data and use a standard GeoJSON format. The benefit to using a standard GeoJSON format is that other applications can read the files or create the files as well. Third, the simulated annealing algorithm and A-Star algorithm were developed as plug-ins and can be replaced without changing other portions of the distance matrix component or route generation component. This allows anyone to develop and test new variations of these algorithms or completely new algorithms. Fourth, the library and

demo web app are free for anyone to use. Also, because Route30 is a client-side library, there is no expense associated with a routing server such as the need for addition skilled staff to maintain, manage, or upgrade the server. Fifth, the route generation algorithm can be tuned to the specific problem which can help it generate solutions as accurately and efficiently as possible. And finally, Route30 is expandable. There are other routing scenarios that are possible. The 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. However, another possible routing scenario is the user may want to find just the shortest path between two points. Other routing scenarios can be implemented in the future by adding more entry points to the library.

In this project, the usefulness of Route30 was demonstrated by an historical tour web app in Castroville, Texas. There are probably many other uses for Route30. If a street network does not exist, then one could be built from scratch. 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. Similarly, the library could be used for museums or warehouse routing. 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. Last, 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.

11. References

Adams, G. (2022a). Route30 GitHub Repository. Accessed February 8, 2022, from Link.

Adams, G. (2022b). Route30 Demo Tour app. Accessed February 8, 2022, from Link.

Alamäki, A. & Dirin, A. (2014). Designing mobile guide service for small tourism companies using user centered design principle. In *Proceedings of the International Conference on Computer Science, Computer Engineering, and Social Media*, Thessaloniki, Greece (Vol. 2014, pp. 47-58). Accessed August 12, 2021, from Link.

Boeing, G. (2017). *U.S. Street Network Shapefiles, Node/Edge Lists, and GraphML Files.* https://doi.org/10.7910/DVN/CUWWYJ, Harvard Dataverse, V2.

Castroville Area Chamber of Commerce (CACC). (2017). *Wilkomme…Your guide to Castroville.* Accessed August 10, 2021, from Link.

City of Bellingham, WA. (n.d.). *Downtown Historic Walking Tour*. Accessed August 10, 2021, from Link.

City of Castroville, Texas. (n.d.). *Historic preservation.* Accessed August 10, 2021, from Link.

City of McKinney, Texas. (n.d.). *McKinney historic downtown building tour*. Accessed August 10, 2021, from Link.

City of Windsor, Colorado. (n.d.). *Windsor historical walking tours*. Accessed August 10, 2021, from Link.

County Commissioners Association of Pennsylvania (CCAP). (2021). *PA GIS Conference.* Accessed October 11, 2021, from Link.

Dere, Esat & Durdu, Akif. (2018). *Usage of the A* Algorithm to Find the Shortest Path in Transportation Systems*. Accessed September 4, 2021, from Link.

Fino, E. R., Martín-Gutiérrez, J., Fernández, M. D. M., & Davara, E. A. (2013). Interactive tourist guide: connecting web 2.0, augmented reality and qr codes. *Procedia Computer Science*, *25*, 338-344. Accessed August 11, 2021, from Link.

Hamdi, M. (2019). Converting map image to graph representation for a map helper app. Capstone Report. School of Science & Engineering – Al Akhawayn University. Accessed September 1, 2021, from Link.

Internet Engineering Task Force (IETF). (2016). RFC 7946 – The GeoJSON format. Accessed October 11, 2021, from Link.

Jacobson, L. (2013). *Simulated annealing for beginners.* Accessed August 24, 2021, from Link.

Keler, A., & Mazimpaka, J. D. (2016). Safety-aware routing for motorised tourists based on open data and VGI. *Journal of location Based services, 10*(1), 64-77. Accessed August 17, 2021, from Link.

Lester, P. (2005). *A* pathfinding for beginners*. Accessed September 4, 2021, from Link.

Lojkine, O. (2018). *Salesman*. Accessed August 24, 2021, from Link.

Ma, S. (2020). Unders*tanding the travelling salesman problem (TSP).* Routific. Accessed August 23, from Link.

Novack, T., Wang, Z., & Zipf, A. (2018). A system for generating customized pleasant pedestrian routes based on OpenStreetMap data. *Sensors, 18*(11), 3794. Accessed August 17, 2021, from Link.

OpenRouteService (ORS). (2021). *Openrouteservice: Openrouteservice API services*. The Heidelberg Institute for Geoinformation Technology. Accessed August 15, 2021, from Link.

Open Source Routing Machine (OSRM). (2021). *OSRM: Modern C++ routing engine for shortest paths in road.* Accessed August 15, 2021, from Link.

pgRouting. (2021). *pgRouting Project*. Accessed August 15, 2021, from Link.

Press, W. H., Teukolsky, S.A, Vetterling, W.T., & Flannery, B.P. (1992). *Numerical recipes in C (2nd ed.).* Cambridge, UK: Cambridge University Press.

Roy, B. (2019). *A-Star (A*) search algorithm*. Accessed September 4, 2021, from Link.

Schneider, T. (2014). *The traveling salesman with simulated annealing, R, and Shiny*. Accessed August 24, 2021, from Link.

Society of Architectural Historians (SAH). (2021). *Henri Castro homestead*. Accessed August 19, 2021, from Link.

Sokkappa, P. (1990). *The cost-constrained traveling salesman problem*. Lawrence Livermore National Laboratory. UCRL-LR—105145, DE91 004943. Accessed September 14, 2021, from Link.

StackOverflow. (2013). *Travelling salesman with time limit: Response from Zim-Zam O'Pootertoot.* Accessed September 14, 2021, from Link.

Swift, N. (2017). *Easy A* (star) pathfinding*. Accessed September 4, 2021, from Link.

Techie Delight. (2021a). *Single-source shortest paths – Dijkstra's algorithm*. Accessed September 14, 2021, from Link.

Techie Delight. (2021b). All-pairs shortest paths – Floyd Warshall algorithm. Accessed September 15, 2021, from Link.

TheChalkface.net. (n.d.). *The travelling salesman problem*. Accessed September 14, 2021, from Link.

TutorialsPoint. (2021). *All-pairs shortest paths*. Accessed September 14, 2021, from Link.

Tyagi, N. (2020). *Dijkstra's algorithm: The shortest path algorithm.* AnalyticSteps. Accessed September 15, 2021, from Link.

United Nations Educational, Scientific, and Cultural Organization (UNESCO). (2021). *World Heritage List*. Accessed August 11, 2021, from Link.

Walker, J. (2018). *Simulated annealing – The travelling salesman problem*. Accessed August 24, 2021, from Link.