Save Gas Map – An application for Google G1 dev phone on Android platform

---

CSC714 - Project Report 3

Team Members:
Raghuveer Raghavendra (rraghav2)
Prasanna Jeevan Devanur Nagabhushan (pdevanu)

1. Abstract

We have implemented “Save Gas Map” application for Google G1 dev phone on Android platform [1,2,5]. The application allows the user to enter multiple locations that the user is interested to visit and finds the optimal route between those locations and lists them on the Google Maps.

We all use Google maps to get from point A to point B. Given the address of the start location (A) and the end location (B), Google maps plots out a nice route for the trip and gives an estimate of the travel time. But if the trip includes multiple points, the user has to specify the sequence of locations himself before Google maps can give a route. It would be interesting if Google maps can give the shortest path for a set of locations.

2. Development Framework

Android is a software stack that allows developers build applications for mobile devices in the Java Programming Language. It stack includes an operating system, a middleware and a few applications. Google provides an android SDK for developing android applications. The SDK is available on all major platforms – Windows Linux and Mac OSX and is free download from [6].

We developed our application on the Android 1.1 SDK release 1. The development was done the Eclipse IDE for Java using the Android Development Tool (ADT) plug-in. Instructions for installing the SDK and ADT can be found here [7].

3. Getting Started on Android

There is a lot of help available online for developing Android applications, both from Google and third party developers. Links [6, 7, 8] help in basic setup and writing the first “Hello Android” program.

The Android platform requires all applications installed to be signed digitally with a certificate. The certificate is used to identify the authors of applications and to build trust among applications. More information on application signing can be found here [9].

Android provides a Maps API to help developers use and integrate Google Maps with their mobile applications. However, in order to use the Maps API the developers have to register with

Google Maps service and agree to certain Terms of Service. More information on registering with the Google Maps Service can be found here [10].

4. Implementation

a. GUI

We are able to create a GUI to accept inputs for multiple locations. The input location should be valid addresses. More locations can be added dynamically as and when required by the user. The ‘Add Location’ button allows adding new text boxes where the user can enter the location address. The ‘Show Map’ button displays the locations on the Google map.

Each of these addresses is converted into a pair of latitude and longitude. This is done by using the API’s getLatitude( ) and getLongitude( ) of the Android’s Location class.

Then the locations are arranged to get the optimal shortest path and the locations are displayed in order on a map. However, there is no API support to draw out a route on the Google Map in Android. Hence we just call upon the maps application on G1 to display the route. The G1’s map application cannot display a route between more than two locations, an inherent drawback of using maps on G1. So in summary, the path between the first two locations is displayed using the G1’s map application.

b. Algorithm Details

Our problem of finding the route between N locations and back to the initial location is the classical traveling salesman problem (TSP) which is NP Complete and NP Hard. Starting from a location, for covering N locations and arriving back at a starting location, there are N! possible routes. Also, in order to decide the shortest route, there needs to be square(N) calls to the Google Maps. Each call to the Google Maps system querying the distance between two locations is deemed expensive. We want to minimize this. Hence we make an assumption that for two locations that are geographically closer (in terms of their latitudes and longitudes), there exists roads connecting the locations that are proportionally closer. With this assumption, we just have to make N Google maps API calls just to query the positions of the locations (latitude, longitude).

With the obtained latitude and longitude the distance between any two different locations can be calculated using the distance formula.

$$d = \sqrt{(y2 - y1)^2 - (x2 - x1)^2}$$

Where x1, x2 represents the latitudes of two different points and y2,y1 represents their longitudes.

We have used a heuristic algorithm[3] to solve the TSP. The algorithm is called as Nearest Neighbor (NN) which is a greedy algorithm. It works by selecting the nearest unvisited location. The distance between all the locations and the initial location are calculated and the one nearest to the initial location is selected. Then the distance from the second location is calculated to all other remaining locations and the nearest location is selected and so on. NN gives path lengths of an average of 1.25*exact shortest length. There could be certain arrangement of locations which gives a longer path. We also considered other heuristic algorithms but the development complexity of NN is much less compared to others and with a reasonable performance. So we have chosen NN.

c. Major Classes

Some of the important classes that are accessed are as follows:

android.view: This class exposes the UI classes which are required to handle the screen layout of the map and the interaction that the user does with it.

com.google.android.maps: Our application displays and controls the Google map using this package. We have used MapView class - to display the map on the screen, MapController utility class - to handle the zooming and panning of the maps and GeoPoint which represents the longitude and latitude of the user.

android.location: This is used to access the location services which are used to identify the current location of the user. We have used the Location class which represents the location of the user sensed at a particular time. We use the getLatitude( ) and getLongitude( ) API's to

obtain the latitude and longitude of the users current location. The access to system location services is obtained through LocationManager class.

LocationOverlay: This class is used to mark the pointers on the Google map to identify the locations based upon the path we should take to reach them.

5. Testing

Firstly the GUI was tested to see if the text boxes are created dynamically and the inputs provided are read properly. Then we tested the alignment and functionality of all other GUI components such as buttons.

The algorithm was tested by providing a set of locations and checking whether a optimal route is displayed between them. We also verified that the application does not crash or became unreliable over multiple usages.

6. Results

	Locations to be visited by user	**Locations ordered by GAS MAP**
1	2313 Champion Ct, Raleigh, NC 27606 (Our house)	2313 Champion Ct, Raleigh, NC 27606
2	3201 Edwards Mill Rd # 123, Raleigh, NC 27612 (Harris Teeter)	3415 Avent Ferry Rd, Raleigh, NC 27606 (Food Lion Inc: Store Number 1454)
3	3415 Avent Ferry Rd, Raleigh, NC 27606 (Food Lion Inc: Store Number 1454)	6715 Hillsborough St, Raleigh, NC 27606 (Around the World Market)
4	6715 Hillsborough St, Raleigh, NC 27606 (Around the World Market)	3201 Edwards Mill Rd # 123, Raleigh, NC 27612 (Harris Teeter)
Total round trip distance (in miles)	**19.9**	**Total round trip distance (in miles)** **14.8**
Saved distance (in miles)	**5.1**	

Some of the possible locations visited by the students are selected in some order and the distance is calculated using the Google maps. Then these are ordered using our application and visited in the specified order. The round trip distance reduced by 5.1 miles. This clearly shows that the Save Gas Map application finds the shorter route between different locations and is useful.

7. Limitations and Future Improvements

The Google Android platform does not support APIs to obtain the path and display it between different locations. The earlier versions of Android m3/m5 supported them but they were removed in the latest versions. Hence we were not able to display the paths between all the locations. This can be overcome in future if there will be APIs in future versions of Android. Currently, we make use of the functionality of the maps application on G1 to display the path between the first two locations.

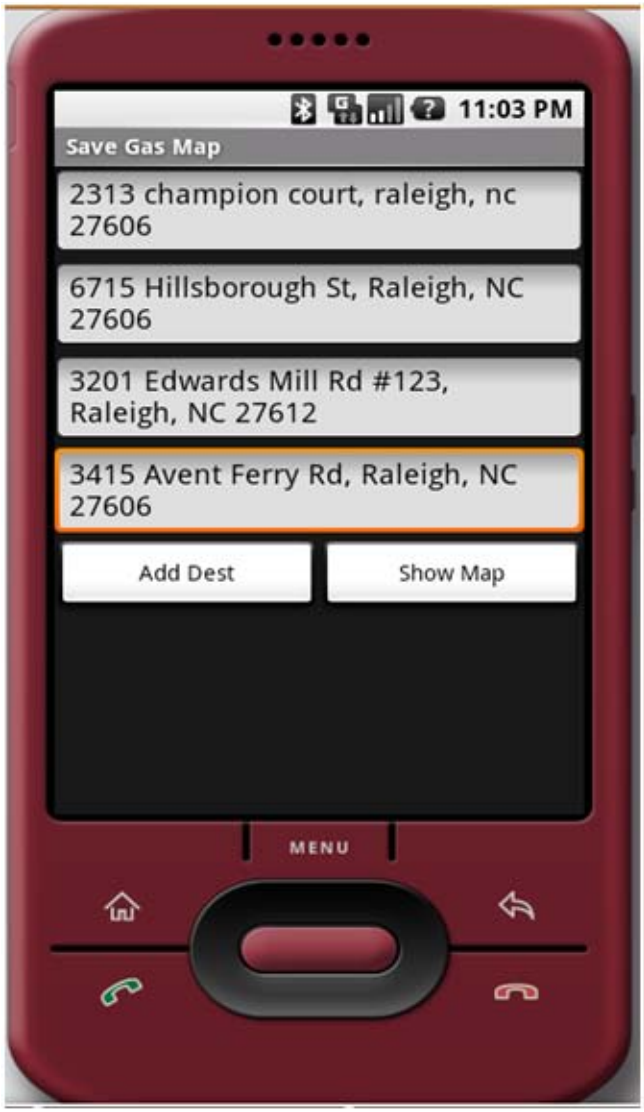
Due to the NP-Complete nature of our problem we are not able to obtain the most optimal route between different locations. The NN algorithm can be replaced by other heuristic algorithms which can provide shorter route in these cases.

The addresses cannot be invalid since the nearest address which suits will be selected without prompting to the user. This can be changed to allow the user to select from a list of addresses. The auto fill feature can be added for the address boxes so that user does not have to type the whole address.

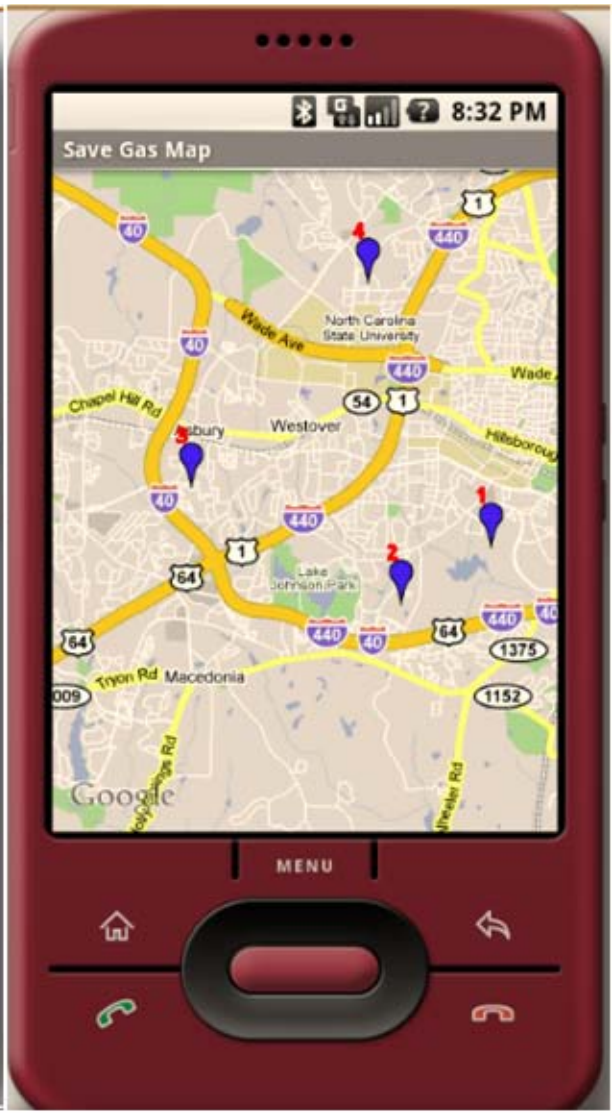
8. Schedule

Activity	Date	Team member	Status
Project Proposal	03-16-2009	Both	Done
Website	03-16-2009	Jeevan	Done
Initial Setup (Linux)	03-24-2009	Raghu	Done
Initial Setup (Windows)	03-25-2009	Jeevan	Done
Literature survey of Android API's	04-12-2009	Both	Done
Basic Implementation (View/Maps/Location)	04-01-2009	Both	Done
Implementation of required GUI	04-05-2009	Raghu	Done
Implementation of the algorithm	04-05-2009	Jeevan	Done
Integration of GUI and algorithm	04-06-2009	Raghu	Done
Setup on the G1 Phone	04-08-2009	Jeevan	Done
Unit Testing & Bug fixing	04-12-2009	Both	Done
System Testing & Bug fixing	04-20-2009	Both	Done

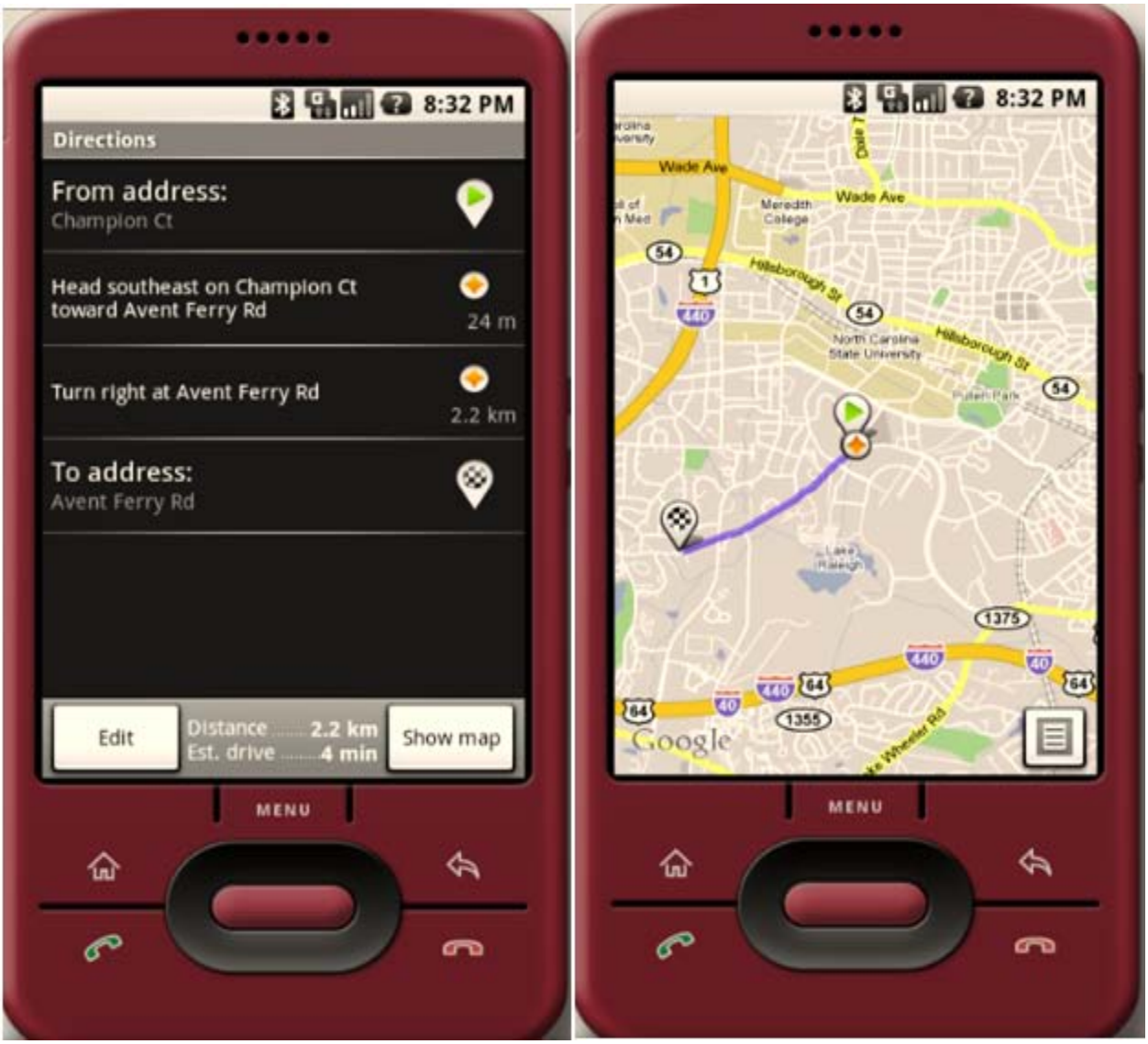
9. Snapshots



1. **Screen to enter addresses**



2. **Map showing locations in order**



3. Screens displaying the routes

10. Project web page

http://www4.ncsu.edu/~pdevanu/Site/Project.htm

11. References

1. http://source.android.com/
2. http://developer.android.com/
3. http://en.wikipedia.org/wiki/Traveling_salesman_problem#Heuristic_and_approximation_algorithms
4. http://iphoneapplicationlist.com/
5. http://androidcommunity.com/
6. http://developer.android.com/sdk/1.1_r1/index.html
7. http://developer.android.com/sdk/1.1_r1/installing.html
8. http://developer.android.com/guide/tutorials/hello-world.html
9. http://developer.android.com/guide/publishing/app-signing.html
10. http://developer.android.com/guide/topics/location/geo/mapkey.html