CALIFORNIA STATE UNIVERSITY, NORTHRIDGE

A Errand Planner Application

A thesis submitted in partial fulfilment of the requirements
For the degree of
Master of Science in Computer Science

By
Nikita Rajendra Karandikar

December 2016

The thesis of Nikita Rajendra Karandikar is approved:

Dr. John Noga	Date
Dr. Adam Kaplan	Date
Dr. Robert McIlhenny, Chair	Date

California State University, Northridge

Acknowledgements

I would like to express my heartfelt gratitude to my advisor Dr. Robert McIlhenny for the continuous support of my Masters study and research, for his patience, motivation, enthusiasm, and immense knowledge.

I would also like to thank my thesis committee members Dr. John Noga and Dr. Adam Kaplan for their support and guidance.

Table of Contents

List of Figures

List of Tables

Abstract

A Errand Planner Application

By
Nikita Rajendra Karandikar
Master of Science in Computer Science

This work is an implementation of the Christofides Algorithm for the travelling salesman problem in an application. Consider, a person running errands. If they go about it in an intuitive manner, it may happen that the person may find themselves going back to the same part of town unnecessarily, missing out on certain high priority tasks or

taking too long. This application will assist with the job of planning by generating an efficient path to follow.

Considering the locations to visit as nodes and the paths between them as weighted edges, we create a graph for the Christofides algorithm to process. It uses Google API Google Distance Matrix to get the distances and durations between the nodes, i.e. the weights of the edges.

OkHttp library is used to send and receive the HTTP network requests to the Google Distance Matrix API. GSON, Google's JSON processing library is used to process the JSON response obtained from the API.

The Christofides algorithm processes this graph and generates a sequence of errands for the user to follow. The sequence is passed to the Google Maps Directions API and is shown on a Google map.

The main purpose of the application is to generate an efficient sequence of errands for the user. The application will also display the total time of the trip. The total distance travelled will also be shown.

The user will be able to input estimated time at each business and also a priority order for each node. If the estimated time of the trip is too long for the user, the application may suggest some low priority locations and/or errands that will require more time than the others. The user may elect to then drop some tasks and run the application again.

A choice of mode of transport i.e. driving, walking, bicycling is offered to the user. The user may specify restrictions such as avoid tolls or highways and these may be considered by the Google Distance Matrix API while calculating the distance between the nodes.[3]

The path to be followed is highlighted on the map as are the roads to be followed to get from one node to the next. The addresses of each node as resolved by the geocode are displayed as labels of the map marker and also in the EditText field where the user has originally entered the query.

Chapter 1 - Introduction

1.1 Introduction

This work presents the design and development of an android application implementing the Christofides algorithm for the travelling salesman problem. The Travelling salesman problem is one of the most famous problems in computer science. It is an NP Hard problem which means that there is no known way to solve it in polynomial time. Many approximation algorithms have been presented to solve this problem such as the nearest neighbor algorithm, Randomized improvement and Ant colony optimization. Christofides algorithm, developed by Nicos Christofides in 1976 is one of the algorithms. Christofides algorithm has the best worst case approximation of all known algorithms.

There are some existing applications that provide this functionality such as Route4Me, RoadWarrior and inRoute with their own proprietary algorithms. However, it is not possible to calculate the Time complexity or the Approximation ratio. These applications are targeted at businesses that need to do vehicle routing on a large scale and are paid applications.[8][9][10]

1.2 Parts of the problem:

1. The addresses provided by the user can be converted into nodes of the graph with edges showing the distance between each pair of nodes. The user does not need to provide coordinates or even perfect postal addresses although these would be processed too. There are no strict restrictions on the format and spacing between the words of the address.

2. The generated JSON responses from the Google Distance Matrix API need to be processed to extract the data of interest from it and pass it to the Christofides algorithm.
3. The generated graph is input into the Christofides algorithm implementation and the Hamiltonian path is calculated.
4. Using the Google Maps Directions API, the addresses and paths between them are shown on a Google Map.

Chapter 2 - Christofides Algorithm

2.1 Background

Given a graph, the Travelling salesman problem seeks to find the shortest possible route that visits all the nodes once and only once. This is an NP Hard problem. The Brute force algorithm will give the optimal solution but it is a function of the factorial of the number of nodes in the graph.

2.2 Approximation algorithms

Several approximation algorithms have been proposed to approximately solve the Travelling salesman problem. In the approximation algorithms described below the triangle inequality is assumed to hold. The triangle inequality states that the sum of lengths two edges of a triangle will be more than the length of the third edge. In the context of the travelling salesman problem, this means that if we go from one edge A to another edge B, the edge connecting A and B is the shortest path from A to B. If an intermediate node is added, the length of the path may stay the same or increase but will not decrease. We discuss Nearest neighbour algorithm, 2- approximation algorithm, Genetic algorithm, Brute force algorithm and Christofides algorithm. Other algorithms such as Tabu search, Ant Colony and Böckenhauer are quite complex so were not considered when choosing the algorithm for the application.

2.2.1 Nearest neighbour algorithm:

Choose a starting vertex. From the starting vertex, choose the edge with the lowest weight and add that edge to the path. Continue in this manner choosing among edges that connect the current vertex to the vertices that have not yet been visited. When all the vertices have been visited, return to the starting node.

The solution found and hence the quality of the solution depends on which vertex is chosen first. Although the algorithm is easy to understand and implement, the quality of the solution is not good enough and so this algorithm is not often used in practice.

2.2.2 2-approximation algorithm

Consider a graph. Construct the minimum spanning tree of that graph. The minimum spanning tree can be constructed using either Prim’s algorithm or Kruskal’s algorithm. Now, beginning at a node, traverse along the minimum spanning tree and whenever a leaf vertex is reached traverse back along the edge and continue traversing along the minimum spanning tree. This is a depth first traversal. In this algorithm each edge is visited twice so the length of the path is twice the weight of the minimum spanning tree.

Consider an optimal path. Removing one edge of the path will give us a spanning tree. Now this spanning tree cannot have a weight less than that of the minimum spanning tree, at best it may have the same weight as the minimum spanning tree. Thus, the weight of the minimum spanning tree is not more than the weight of the optimal path. Hence, the weight of the path generated at this stage is no more than twice the weight of the optimal path.

The path may visit each node more than once so we reduce the length of the path by removing duplicate nodes, which may result in a graph with weight less than twice the weight of the optimal solution.

2.2.3 Genetic algorithm

Genetic algorithm is a strategy used to get some approximation of an exact solution when the solution space is too large to search for an optimal solution. This algorithm simulates the principles of evolutionary biology. Consider a case where there is a population of many species. Members of a population may have many similar characteristics but they also have some differences which determine their fitness. So, some members will be more fit than others. Due to all the factors that affect their fitness, some will die, some will survive and reproduce to form a new generation. A new generation will be slightly different from the previous generation. The idea is to have the more fit populations reproduce and produce new generations to hone in on the desirable characteristics. Due to this form of natural selection, a large population of varied fitness moves towards getting more and more fit solutions with each passing generation.

This concept is applied to solving the traveling salesman problem. Typically, the first population is generated consisting of random solutions. We choose the solutions that are good as calculated by a fitness function, here the length of the path, and mutate and recombine them to try and do better. We try to get closer and closer to the optimal solution. The algorithm terminates when we reach a predetermined number of generations or when the results start to asymptote or when a quality that is deemed to be sufficient is reached. The algorithm may get stuck in a local minimum as it tries to reduce the weight of the path in each generation.

2.2.4 Brute force algorithm:

This algorithm considers all possible paths in order to find the best one. This algorithm is infeasible in practice for an NP hard problem.

2.2.5 Christofides Algorithm

Consider a graph. Construct a minimum spanning tree of the graph. This can be done using either Prim’s or Kruskal’s algorithm. Perform a depth first traversal as done in the 2-approximation algorithm. As proved in the 2 approximation algorithm, the weight of the minimum spanning tree, say C(T) is less than or equal to the weight of the optimal path (H^*_G).

S is the set of odd degree vertices in the minimum spanning tree. H^*_S is the Hamiltonian path formed by only considering the vertices in S. A Hamiltonian path is the path formed by visiting all the vertices exactly once. The travelling salesman problem seeks to find the minimum weight Hamiltonian path.

From the triangle inequality described before,

$H^*_S <= H^*_G$ …………………………………………………………………………..(1)

Also, as described above,

$C(T) <= C(H^*_G)$…………………………………………………………………...(2)

A Euler path is path constructed by visiting all the vertices in a graph.

Identify the odd degree vertices S in the minimum spanning tree T. Construct a perfect matching of the odd degree vertices and add the edges identified in the perfect matching to T. Construct a Euler path C of the graph formed. There are always an even number of odd degree vertices in a minimum spanning tree. In a graph, the total degree of the graph is calculated by adding the degrees of all the vertices in the graph. The total degree of a graph is twice the number of edges in the graph as each edge has two vertices at its ends. Thus, the total degree of a graph is always even. The total degree of the graph is the sum of the total degree of the even number vertices and of the total degree of the odd number vertices. Now, the total degree of the even number vertices is even. So the total degree of the odd number vertices is also even. Thus, there are an even number of odd vertices. Let

$C(M)$ be the cost of the edges in the perfect matching and let $C(C)$ be the cost of the Euler path obtained:

$C(C) = C(T)+C(M)$ …………………………………………………………………...(3)

From (2) and (3),

$C(C) <=C(H^*_G)+C(M)$………………………………………………………..…...(4)

Consider H^*_S and construct a matching by taking every other edge, splitting the graph into two groups of matchings M_1 and M_2. Now, the minimum matching is M thus,

$C(M)<=C(M_1)$…………………………………………………………………...(5)

$C(M)<=C(M_2)$…………………………………………………………………...(6)

From (5) and (6),

$C(M)<= (½)(C(M_1)+C(M_2))$…………………………………………….……(7)

As the Hamiltonian cycle is constructed from M_1 and M_2,

$C(H^*_S) = (C(M_1)+C(M_2))$ ……………………………………….….……...…(8)

From (1) and (8),

$C(H^*_{G>})>= (C(M_1)+C(M_2))$...………………………………………….……..….(9)

From (7) and (9)

$C(M)<=(½)C(H^*_G)$……………………………………………….………………(10)

From (6) and (10)

$C(C) <=(3/2)C(H^*_G)$…………………………………………………………………(11)

Thus, the Christofides algorithm generates a solution that is a 1.5 approximation of the optimal in the worst case.

The Christofides Algorithm was developed by Nicos Christofides and published in 1976. It is has the best worst case approximation for solving the Travelling salesman problem. Its time complexity is $O(n^3)$.

2.3 Implementation and comparison of algorithms

Test data was obtained from http://www.math.uwaterloo.ca/tsp/

The following Test data was used:

dj38.tsp: Dataset with 38 vertices

lu980.tsp: Dataset with 980 vertices

nu3496.tsp: Dataset with 3496 vertices

ca4663.tsp: Dataset with 4663 vertices

Nearest neighbor algorithm:

	First node	Weight of path	Time
dj38.tsp	38	7941	0.0s
lu980.tsp	327	14590	0.03s
nu3496.tsp	2334	119866	0.49s
ca4663.tsp	4663	1633565	0.95s

Table 2.3.1 Nearest neighbor algorithm implementation

2-Approximation algorithm

	Weight of path	Time
dj38.tsp	8552	0.0s
lu980.tsp	17452	0.16s
nu3496.tsp	144521	4.34s
ca4663.tsp	1758540	10.64s

Table 2.3.2 2-Approximation algorithm implementation

Christofides Algorithm

	Weight of path	Time
dj38.tsp	6770	0.01s
lu980.tsp	13754	8.78s
nu3496.tsp	115010	47m 39s
ca4663.tsp	1512161	16m 14s

Table 2.3.3 Christofides algorithm implementation

Genetic Algorithm

	Weight of path	Time
dj38.tsp	7337	2s
lu980.tsp	265992	23m 5s
nu3496.tsp	4910493	304m 6s
ca4663.tsp	11868417	507m 3s

Table 2.3.4 Genetic algorithm implementation

Comparison of the implementations

	dj38.tsp	lu980.tsp	nu3496.tsp	ca4663.tsp
Nearest neighbor	7941	14590	119866	1633565
2-Approx	8552	17452	144521	1758540
Christofides	6770	13754	115010	1512161
Genetic	7337	265992	4910493	11868417
Optimum	6656	11340	96132	1290319

Table 2.3.5 Comparison of the implementations

The algorithms were implemented on a Dell Inspiron Laptop with the following particulars:

Processor: Intel Core i5

RAM: 4 GB

Operating System: Windows 7 64 bit

After studying the results of this implementation, the Christofides Algorithm was chosen for the application.

2.4 Algorithm

The steps are as follows[1][2]

- Construct the Minimum Spanning Tree T of the given graph G.
- Identify the set of odd degree vertices O. As described in 2.2.5 T has an even number of vertices.
- Find the minimum weight perfect matching M of the odd degree vertices.
- Combine M and T to form a multigraph H where every vertex has an even degree.
- Form an Euler circuit E from H.
- Make the Euler circuit into a Hamiltonian circuit by shortcutting through repeated vertices.

2.5 Example

Consider the following completely connected graph shown in Figure:2.5.1

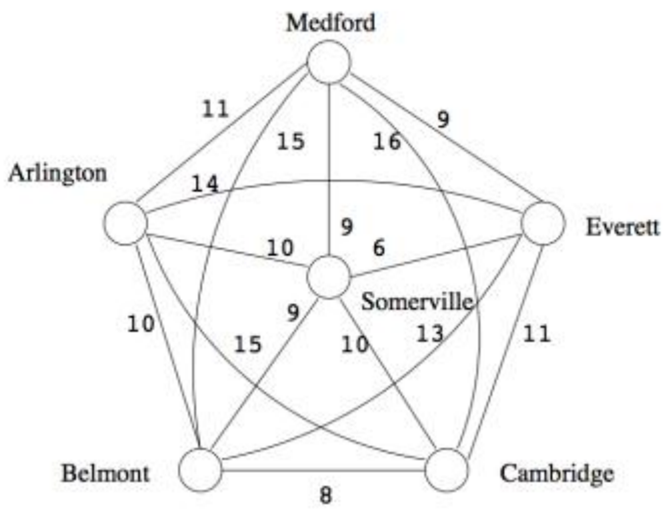


Figure 2.5.1: Completely connected graph[12]

We create a minimum spanning tree as shown in Figure 2.5.2

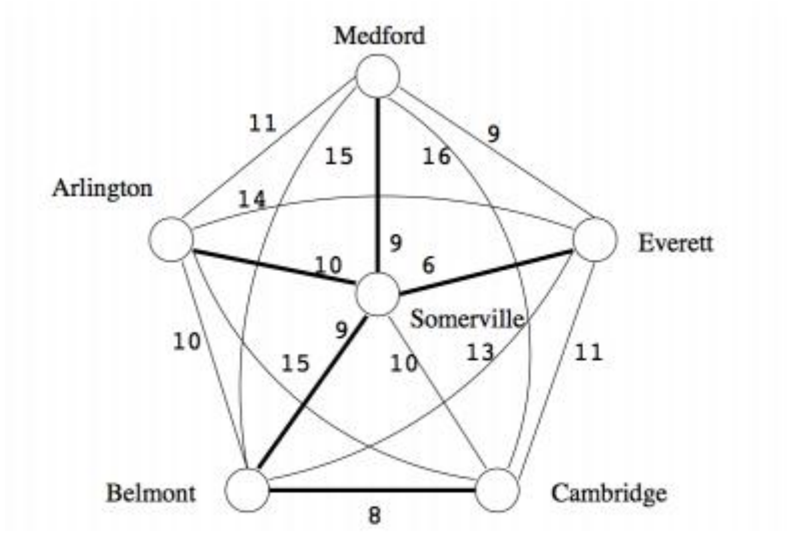


Figure 2.5.2: Minimum Spanning Tree[12]

We then derive a minimal perfect matching on vertices with odd degree in the minimum spanning tree as shown in Figure 2.5.3

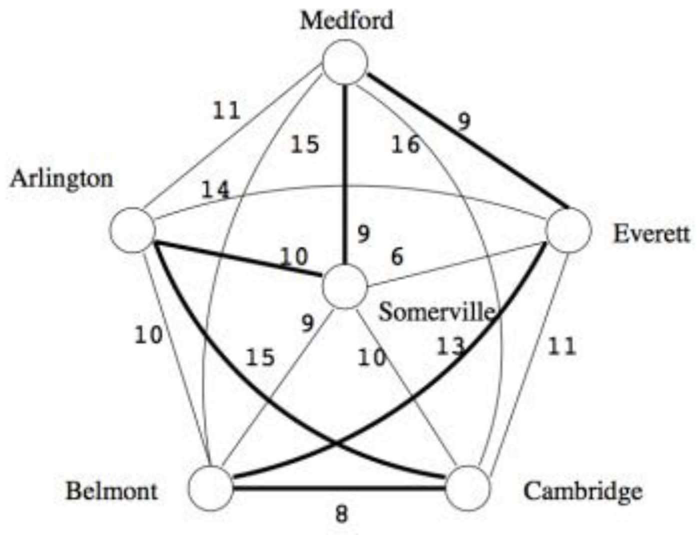


Figure 2.5.3: Minimal Matching on vertices with odd degree in the MST[12]

Then we construct a union of the minimum spanning tree and the minimal matching, as shown in Figure 2.5.4

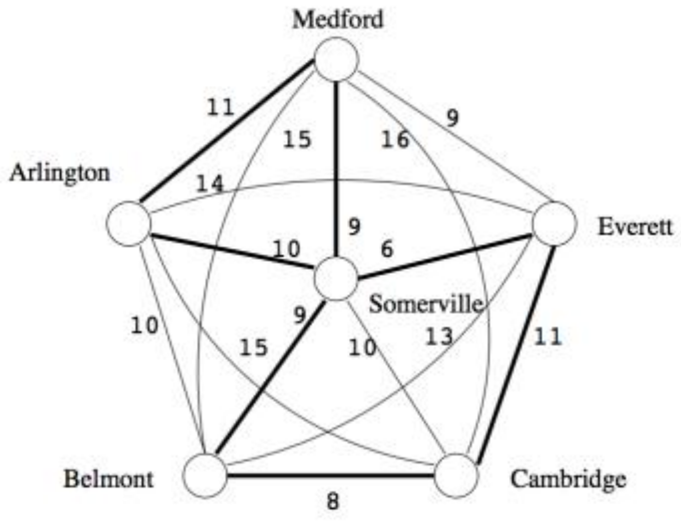


Figure 2.5.4: Union of MST and Minimal Matching[12]

Finally, we produce a solution with weight less than 1.5 times the optimal (OPT) as shown in Figure 2.5.5

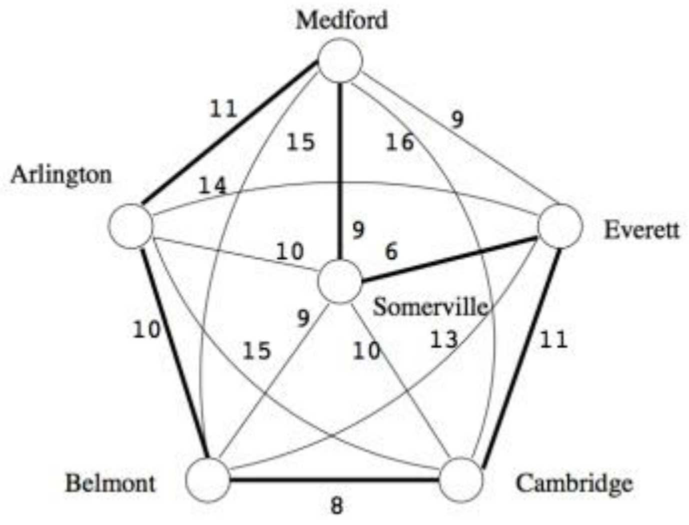


Figure 2.5.5: Solution with weight less than 1.5*OPT [12]

Chapter 3 - Errand Planner Application

The Errand Planner application was developed with the idea that at each stage there should be an application, however limited in capability able to run on the Android device.

3.1 Features to be built

1. The main purpose of the application is to generate a near optimal sequence of errands for the user.
2. The application should allow the user to enter in the addresses/coordinates of the locations to be visited.
3. The user should be able to specify priorities for each of the locations entered on a scale of 1 to 3 with 1 being top priority, 3 being the lowest priority and 2 being medium priority.
4. The user should be able to specify the time he/she will need to spend at each of the locations in order to find the most time consuming errands.
5. The application should show the user an estimate of the total time including the time spent at each of the locations.
6. If the time estimated is too long for the user, the application may suggest some low priority/time consuming tasks to be dropped based on the priority and time provided by the user.
7. The total distance traveled should also be displayed to the user.
8. One way streets should be considered.
9. The initial system is to be built assuming driving as the mode of transport but this can be extended to walking, bicycling and public transport too.

10. The user should be able to specify restrictions such as avoiding highways or avoiding tolls, and the application should attempt to accommodate this restriction if possible.

11. The resulting path and the nodes as geocoded by the API should be displayed on a Google Map.

12. The application should be free.

3.2 Planning

3.2.1 Choice of platform

Android is one of the most popular platforms among mobile users[4]. It is also easier to start coding with Android as most building blocks of an Android application are open source and free. In order to become an ios developer, a membership fee of $99 per year[5] has to be paid upfront which makes it difficult to present ios applications that are truly free to use.

3.2.2 Choice of IDE

Android Studio is the official IDE for Android and receives all the latest updates. It is specifically built for Android development and hence it was the automatic choice for this application.

Android studio also has the following features which makes it a good choice for developing Android applications[6].

- <u>Instant Run</u>: Android Studio has an instant run feature that quickly pushes the code and resource changes to the running app when Run or Debug is clicked, intelligently understanding the changes that have been made to the app and delivers them without need to restart the app or rebuild the APK.

- Intelligent code editor: The code editor has advanced code completion, code analysis and refactoring capabilities. Suggestions are offered in a dropdown list as the developer types. This helps the developer work faster, write better code and be more productive.
- Fast and feature rich emulator: The Android Emulator runs and installs the applications at a speed comparable with a real device. It allows the developer to prototype the application and to test it on various Android device configurations: phones, tablets, Android TV and Android Wear. Many hardware features can also be simulated such as network latency, GPS location, motion sensors and multi-touch input.
- C++ and NDK support: Android Studio offers the capabilities to edit C/C++ project files, so JNI components can be built into the application. An LLDB-based debugger allows debugging of Java and C++ code at the same time and the IDE also offers syntax highlighting and refactoring for C/C++. CMake and ndk-build scripts can also be executed without any modifications using the build and then the shared objects can be added to your APK.
- Firebase and Cloud Integration: The Firebase Assistant offers the capability to connect your application to Firebase and include Authentication, Notifications, Analytics and other services with stepwise procedures inside Android Studio. Due to the Built-in tools for Google Cloud Platform it easier to create and then to deploy a backend for the Android app, with the use of services such as Google

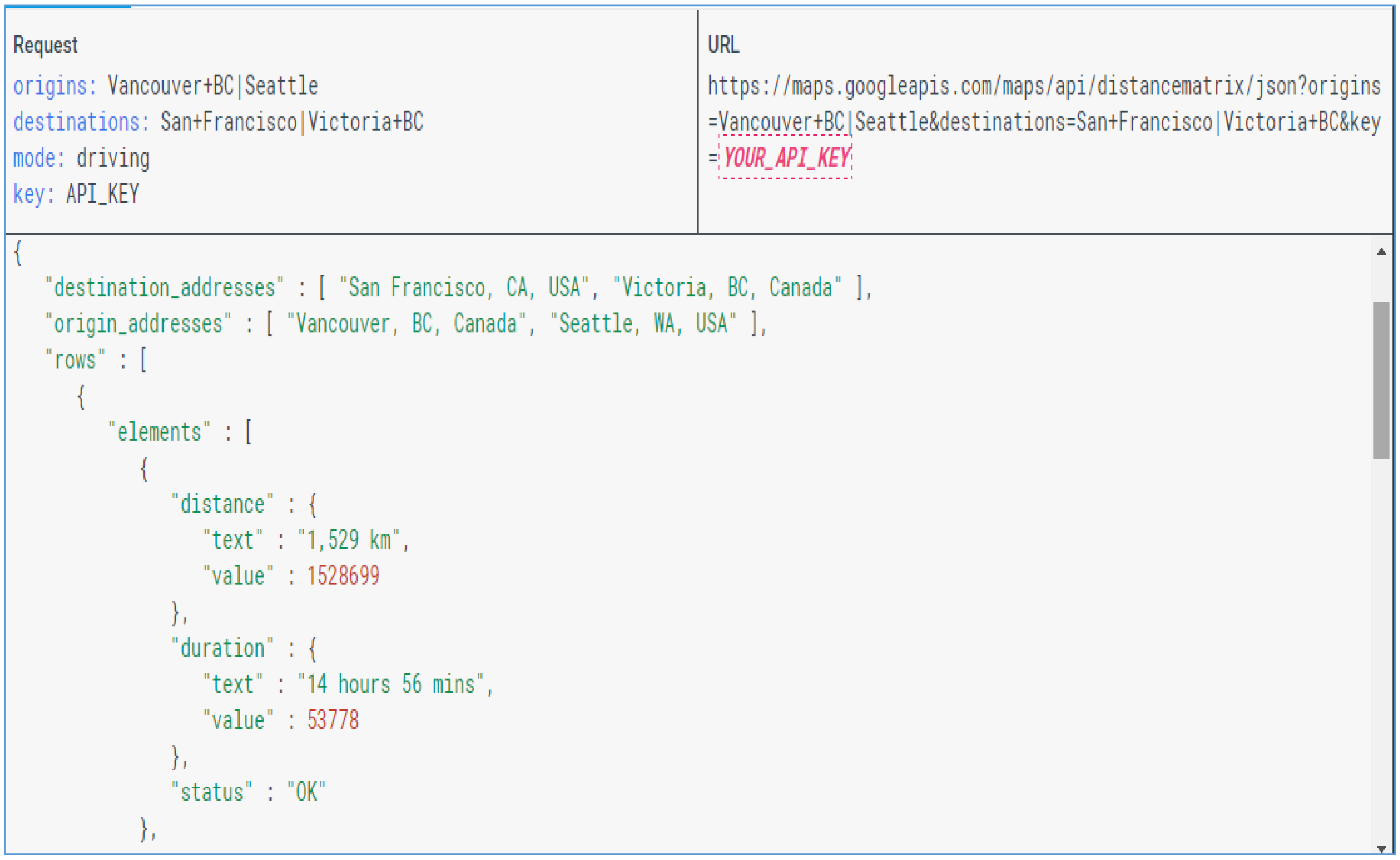
Cloud Endpoints and project modules that are purpose designed for Google App Engine.

- Layout Editor: Android Studio provides a drag-and-drop visual editor, so it becomes easy to work with XML layout files, creating a new layout quickly. The Constraint Layout API is built in conjunction with Layout Editor so a layout can be built by dragging in views and then adding layout constraints. The layout adapts to different screen sizes.
- APK analyzer: The contents of the APK can be easily inspected using the APK analyzer. A breakdown of the APK size is generated by giving the size of each component so ways to reduce the total APK size can be identified. It also offers the capability to inspect the DEX files, to troubleshoot multidex issues, preview packaged assets and compare two APKs.
- Translations editor: A single view of all translated resources can be displayed using the translations editor. This makes it easy to change or add translations, and to figure out any missing translations. This can be done without the need to open each version of the strings.xml file. It also provides a link to translation services.

3.2.3 Choice of API to calculate distances

There was only one choice for the API to use to calculate the distance between the nodes i.e. the weights of the edges in the graph which was Google's Distance Matrix.

This API allows us to retrieve the distance and duration for multiple destinations and transport modes. On providing the list of origins and destinations over https, the API generates a JSON/XML output with the values. An API key is needed to use this API which can be obtained for free from Google API console[3].



```
Request
origins: Vancouver+BC|Seattle
destinations: San+Francisco|Victoria+BC
mode: driving
key: API_KEY
```

```
URL
https://maps.googleapis.com/maps/api/distancematrix/json?origins
=Vancouver+BC|Seattle&destinations=San+Francisco|Victoria+BC&key
=YOUR_API_KEY
```

```
{
   "destination_addresses" : [ "San Francisco, CA, USA", "Victoria, BC, Canada" ],
   "origin_addresses" : [ "Vancouver, BC, Canada", "Seattle, WA, USA" ],
   "rows" : [
      {
         "elements" : [
            {
               "distance" : {
                  "text" : "1,529 km",
                  "value" : 1528699
               },
               "duration" : {
                  "text" : "14 hours 56 mins",
                  "value" : 53778
               },
               "status" : "OK"
            },
```

Figure 3.2.3.1 Sample JSON response of Google Distance Matrix

3.2.3.1 Building a Google Distance Matrix API request URL

A Google Distance Matrix Request URL is of the form:

https://maps.googleapis.com/maps/api/distancematrix/outputFormat?parameters

- <u>outputFormat</u> may be either of the following values:
  - JSON: This is the recommended outputFormat and will generate the output in JavaScript Object Notation format.
  - XML: This will generate the output as XML

- <u>HTTPS or HTTP:</u>
    - HTTPS is the recommended wherever possible, more so for applications that include private user data, such as location, in requests. Use of the HTTPS encryption makes for a more secure application that is more resistant to tampering and snooping.
    - If HTTPS is not possible, one can access the Google Maps Distance Matrix API over HTTP, using:

      http://maps.googleapis.com/maps/api/distancematrix/outputFormat?parameters
- <u>Required Parameters:</u>
    - <u>Origins</u>: The starting point provided to the API in order to calculate travel distance and time. Multiple locations can be listed out separated by the pipe character (|), and can be addresses, latitude/longitude coordinates, or a place IDs.

      If an address is provided, the API automatically geocodes the string into to a latitude/longitude coordinate and uses it to calculate distance. It is possible for this coordinate to be different from the coordinate returned by the Google Maps Geocoding API, for example a building entrance rather than its center.

      Origins=Bobcaygeon+ON|24+Sussex+Drive+Ottawa+ON

      If latitude and longitude coordinates are passed to the service, they are used as-is to calculate the distances and durations. There should be no space between latitude and longitude coordinates.

      origins=41.43206,-81.38992|-33.83748,151.20699

  - Destinations — Multiple locations can be specified to use as the ending point for calculating travel distance and time. The destinations parameters have the same options as the origins parameter.
  - Key: This is an API key which is used by Google services to identify each application for usage quota management.
- Optional Parameters
  - Mode: The transportation made can be specified in order to calculate the distance and time. Driving is considered as the default mode but the following travel modes are also offered:
    - driving (default) distance is calculated assuming for driving along the road network
    - walking distance is calculated considering pedestrian path and walkways where available.
    - bicycling distance is calculated considering bicycle paths and preferred streets where available
  - Restrictions: Certain restrictions may be specified in order to calculate distances. Restrictions are specified with the avoid parameter, and the argument to that parameter is the restriction to be avoided. The following restrictions are supported:

      avoid=tolls

      avoid=highways

      avoid=ferries

      avoid=indoor

The addition of restrictions does not exclude routes that fail the restriction condition, but the result will be biased towards routes adhering to those restrictions.

3.2.3.2 Distance Matrix Response Elements

Distance Matrix Responses contain the following root elements:

- <u>status</u>: Status is the metadata about the response. This field contains valuable debugging information. There are two levels of status codes returned by the API. The Distance Matrix API response has a top level status, with the status of the complete request, as well as status fields for each of the element fields, for that particular origin-destination pairing.

  Top-level Status Codes:

  - OK indicates the response has a valid result.
  - INVALID_REQUEST indicates that the request was invalid.
  - MAX_ELEMENTS_EXCEEDED indicates that the product of origins and destinations has exceeded the per-query limit that is limited by the usage limits.
  - OVER_QUERY_LIMIT indicates the application has exceeded the number of permitted requests within the allowed time period.
  - REQUEST_DENIED indicates a service denial
  - UNKNOWN_ERROR indicates a server error that caused the request to fail and that the request may work upon retry

Element- level Status Codes:

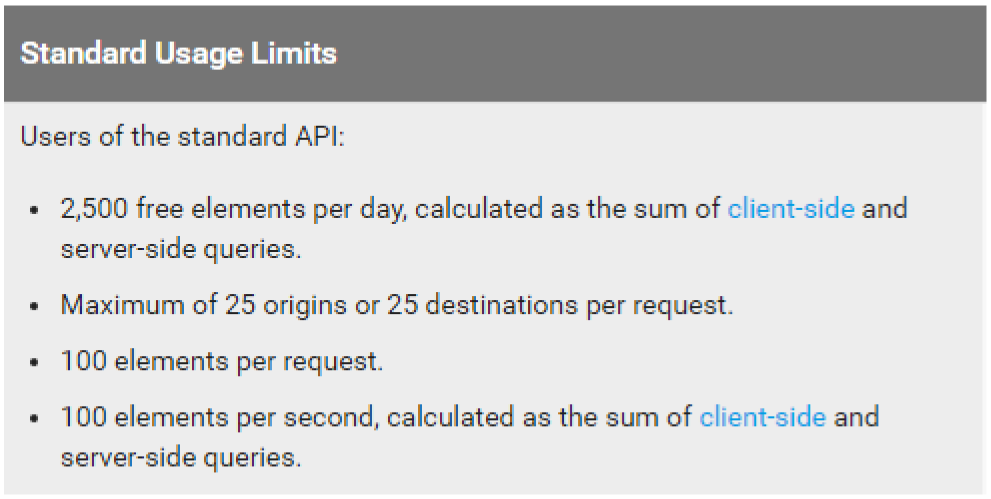
- OK indicates a valid result in the response.
- NOT_FOUND indicates that the service was unable to geocode the origin and/or destination for this pairing.
- ZERO_RESULTS indicates the service could find no route between the origin and destination.

- origin_addresses contains the array of addresses geocoded and localized by API according to the language parameter passed with the request.
- destination_addresses contains the array of addresses constructed from the original request returned by the API. As with origin_addresses, these are localized if appropriate.
- rows: The results returned by the Google Maps Distance Matrix API are placed within a JSON rows array. Even if no results are returned (for instance if the origins and/or destinations don't exist), an empty array is returned. XML responses have of zero or more <row> elements. One or more element entries are contained in the Rows array, which in turn contain the information about one origin-destination pairing each. The details for each origin-destination pairing is returned in the form of an element entry. An element contains the following fields:
    - status: The status will be from the list of status codes listed before and the interpretation of each status is the same as described before.

- duration: The estimated time to travel this route, expressed in seconds (the value field) and as text. The text field is localized according to the query's language parameter.
- distance: The total distance of this route, expressed in meters (value) and as text. The text field uses the unit system specified with the unit parameter of the original request, or the origin's region.

3.2.3.3 Usage limits

Google Distance Matrix API is free to use, subject to some usage limits:


Standard Usage Limits

Users of the standard API:

- 2,500 free elements per day, calculated as the sum of client-side and server-side queries.
- Maximum of 25 origins or 25 destinations per request.
- 100 elements per request.
- 100 elements per second, calculated as the sum of client-side and server-side queries.

Table 3.2.3.3.1 Standard Usage Limits

It allows 2500 free elements per day. This means we can get a total of 2500 edges for our graphs. We address the question of the maximum number of nodes we can support given this limit. To execute the Christofides algorithm, we need a complete graph.

Since n(n-1)/2 edges = 2500, solving for n we get n= 71. Hence, we can support a maximum of 71 nodes.

3.2.4 Choice of library to send/receive HTTP requests

I chose OkHttp as the HTTP client. There are many options available for this, such as Volley, NuGet and Retrofit, but I found OkHttp simplest to use. On the online forum such as stackoverflow, I saw that OkHttp consistently had good reviews.

Code snippet to download a URL and print its contents as a string

```
OkHttpClient client = new OkHttpClient();


String run(String url) throws IOException {
  Request request = new Request.Builder()
      .url(url)
      .build();


  Response response = client.newCall(request).execute();
  return response.body().string();
}
```

3.2.5 Choice of JSON processing library

Once the JSON string is received from the API with the response, we need to parse the response to retrieve data of interest and pass it along to the Christofides algorithm.

To choose the JSON processing library, I considered Simple JSON, org.JSON and Google’s GSON library. The JSON response to be parsed for this application is nested. The distance and duration data which we are interested in is nested 4 levels deep. So I needed a library that would work best in this case.

Simple JSON and org.JSON , in my opinion did not do too well with nested JSON. It was quite hard to code and the code was not easily understandable.

GSON, I found was a good choice as it allowed me to mirror the structure of the JSON response by creating classes for each of the JSON arrays and objects.

Important features of GSON[7]:

- Provides easy to use mechanisms like toString() and constructor (factory method) to convert Java to JSON and vice-versa
- Allows pre-existing unmodifiable objects to be converted to and from JSON
- Allows custom representations for objects
- Supports arbitrarily complex object
- Generates compact and readability JSON output

Although it does require more coding upfront for creating classes for all the arrays and objects it was easier to use overall.

3.2.5.1 Code snippets

Class created for the JSON response

```
class Response {

   private String[] destination_addresses;
   private String[] origin_addresses;
   private Item[] rows;
   private String status;
```

```
public String getStatus() {
    return status;
}

public void setStatus(String status) {
    this.status = status;
}

public String[] getDestination_addresses() {
    return destination_addresses;
}

public void setDestination_addresses(String[] destination_addresses) {
    this.destination_addresses = destination_addresses;
}

public String[] getOrigin_addresses() {
    return origin_addresses;
}

public void setOrigin_addresses(String[] origin_addresses) {
    this.origin_addresses = origin_addresses;
    }

    public Item[] getRows() {
        return rows;
    }

    public void setRows(Item[] rows) {
        this.rows = rows;
    }
}
```

Class created to process the rows array:

```
class Item {
    private Elements[] elements;
    public Elements[] getElements() {
        return elements;
    }

    public void setRows(Elements[] elements) {
        this.elements = elements;
    }
}
```

Class created to process the Elements array

```
class Elements {

    Duration duration;

    Distance distance;

    String status;

    public Duration getDuration() {

        return duration;

    }


    public void setRows(Duration duration) {

        this.duration = duration ;

    }

    public Distance getDistance() {

        return distance;

    }


    public void setRows(Distance distance) {

        this.distance = distance ;

    }

}
```

Class created to process the Distance Object

```
class Distance {

  private String text;

  private String value;


  public String getText() {

    return text;

  }


  public void setText(String text) {

    this.text = text;

  }


  public String getValue() {

    return value;

  }


  public void setValue(String value) {

    this.value = value;

  }

}
```

Class created to process the Duration object

```
class Duration {
    private String text;
    private String value;

    public String getText() {
        return text;
    }

    public void setText(String text) {
        this.text = text;
    }

    public String getValue() {
        return value;
    }

    public void setValue(String value) {
        this.value = value;
    }
```

3.2.6 Choice of API to display the results to the user

For this API, there was only one choice, namely the Google Maps Directions API. This API allows us to resolve the addresses entered by the user into proper addresses and GPS coordinates. We can show these addresses on a Google Map and highlight the path to be followed on the Google Map. This will help the user see the nodes to be visited in order and also navigate from one node to the next.

The parameters for sending a request to this API are the similar to the parameters for Google Distance Matrix API and the response is to be interpreted in a similar way.

Sample Request URL:

https://maps.googleapis.com/maps/api/directions/json?origin=Brooklyn&destination=Queens&departure_time=1343641500&mode=transit&key=YOUR_API_KEY

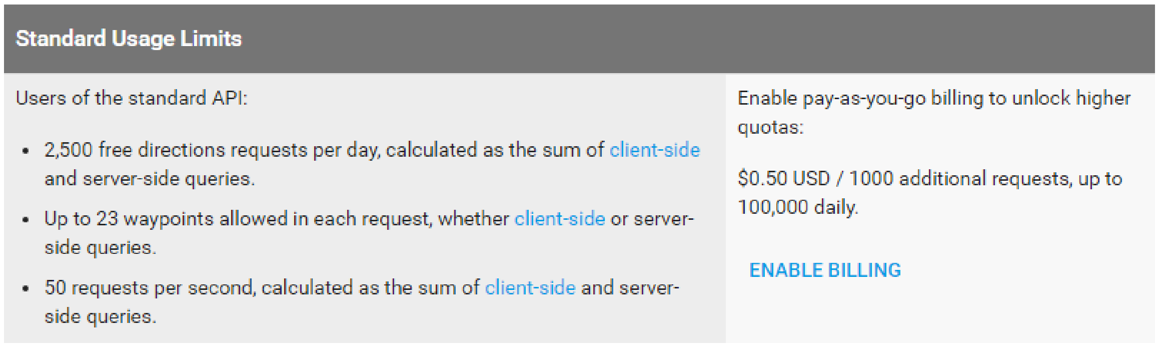
Sample JSON response is shown below, the response is self-explanatory

```
{
  "status": "OK",
  "geocoded_waypoints" : [
    {
      "geocoder_status" : "OK",
      "place_id" : "ChIJ7cv00DwsDogRAMDACa2m4K8",
      "types" : [ "locality", "political" ]
    },
    {
      "geocoder_status" : "OK",
      "place_id" : "ChIJ69Pk6jdlyIcRDqM1KDY3Fpg",
      "types" : [ "locality", "political" ]
    },
    {
      "geocoder_status" : "OK",
      "place_id" : "ChIJgdL4flSKrYcRnTpP0XQSojM",
      "types" : [ "locality", "political" ]
    },
    {
      "geocoder_status" : "OK",
      "place_id" : "ChIJE9on3F3HwoAR9AhGJW_fL-I",
      "types" : [ "locality", "political" ]
    }
  ],
  "routes": [ {
    "summary": "I-40 W",
    "legs": [ {
      "steps": [ {
        "travel_mode": "DRIVING",
        "start_location": {
          "lat": 41.8507300,
          "lng": -87.6512600
          },
          "end_location": {
            "lat": 41.8525800,
            "lng": -87.6514100
          },
          "polyline": {
            "points": "a~l~Fjk~uOwHJy@P"
          },
          "duration": {
            "value": 19,
            "text": "1 min"
          },
          "html_instructions": "Head \u003cb\u003enorth\u003c/b\u003e on
\u003cb\u003eS Morgan St\u003c/b\u003e toward \u003cb\u003eW Cermak
Rd\u003c/b\u003e",
          "distance": {
            "value": 207,
            "text": "0.1 mi"
          }
        },
        ...
        ... additional steps of this leg
      ...
```

```
... additional legs of this route
  "duration": {
    "value": 74384,
    "text": "20 hours 40 mins"
  },
  "distance": {
    "value": 2137146,
    "text": "1,328 mi"
  },
  "start_location": {
    "lat": 35.4675602,
    "lng": -97.5164276
  },
  "end_location": {
    "lat": 34.0522342,
    "lng": -118.2436849
  },
  "start_address": "Oklahoma City, OK, USA",
  "end_address": "Los Angeles, CA, USA"
} ],
"copyrights": "Map data ©2010 Google, Sanborn",
"overview_polyline": {
```

```
          "points":
"a~l~Fjk~uOnzh@vlbBtc~@tsE`vnApw{A`dw@~w\\|tNtqf@l{Yd_Fblh@rxo@b}@xx
SfytAblk@xxaBeJxlcBb~t@zbh@jc|Bx}C`rv@rw|@rlhA~dVzeo@vrSnc}Axf]fjz@xfFb
w~@dz{A~d{A|zOxbrBbdUvpo@`cFp~xBc`Hk@nurDznmFfwMbwz@bbl@lq~@loPp
xq@bw_@v|{CbtY~jGqeMb{iF|n\\~mbDzeVh_Wr|Efc\\x`Ij{kE}mAb~uF{cNd}xBjp]fu
lBiwJpgg@|kHntyArpb@bijCk_Kv~eGyqTj_|@`uV`k|DcsNdwxAott@r}q@_gc@nu`Cn
vHx`k@dse@j|p@zpiAp|gEicy@`omFvaErfo@igQxnlApqGze~AsyRzrjAb__@ftyB}pIl
o_BflmA~yQftNboWzoAlzp@mz`@|}_@fda@jakEitAn{fB_a]lexClshBtmqAdmY_hLx
iZd~XtaBndgC"
        },
        "warnings": [ ],
        "waypoint_order": [ 0, 1 ],
        "bounds": {
          "southwest": {
            "lat": 34.0523600,
            "lng": -118.2435600
          },
          "northeast": {
            "lat": 41.8781100,
            "lng": -87.6297900
          } }
      } ] }
```

Figure 3.2.6.1 Sample JSON response from Google Maps Directions API



Standard Usage Limits	
Users of the standard API: • 2,500 free directions requests per day, calculated as the sum of client-side and server-side queries. • Up to 23 waypoints allowed in each request, whether client-side or server-side queries. • 50 requests per second, calculated as the sum of client-side and server-side queries.	Enable pay-as-you-go billing to unlock higher quotas: $0.50 USD / 1000 additional requests, up to 100,000 daily. ENABLE BILLING

Table 3.2.6.1 Usage Limits for Google Maps Directions API

3.3 Implementation

3.3.1 Features implemented

- The system should generate a sequence of errands for the user.
- The user can enter the addresses/coordinates separated by a # character in the nodes field.
- The user can enter the priorities separated by a # character for each of the locations specified in the nodes field. The priorities are on a scale of 1 to 3, 1 being the highest priority. This will help identify the low priority locations that the user may consider dropping if the estimate time is too long for him/her.
- The user can enter the durations of the errands separated by a # character at each of the locations. This will help identify the most time consuming errands in case the user needs to drop some errands to save time.

- The application calculates the estimated time in transit according to the path generated and adds the time spent at the locations to arrive at the total time for the entire trip.
- The system suggests some low priority/time consuming errands that the user may consider dropping if the estimated time is too much. The user can choose to drop these errands and generate an updated path.
- Display the total distance travelled.
- Google API already factors in one way streets where this information is available.
- Choice of mode of transport i.e. Driving, Cycling and Walking offered to the user.
- The user is able to specify restrictions such as avoid highways or avoid tolls or avoid indoors or avoid ferries and the application should attempt to accommodate this restriction if possible.
- Display the addresses as resolved by the Google Maps Directions API on a Google Map and highlight the path between them.
- Label each of the Google Map markers with the resolved address for the location.
- Show the resolved list of locations in the proper order in the EditText field which the user used to enter the addresses
- The application should be free.

3.3.2 Battery use

I used the GSAM Battery monitor application to measure battery usage. I ran the Errand Planner application for the following input criteria:

Addresses: San Francisco#Santa Barbara#Sacremento#las vegas#Los Angeles #Vancouver Canada#Orlando Florida#New York#New Jersey#Washington DC

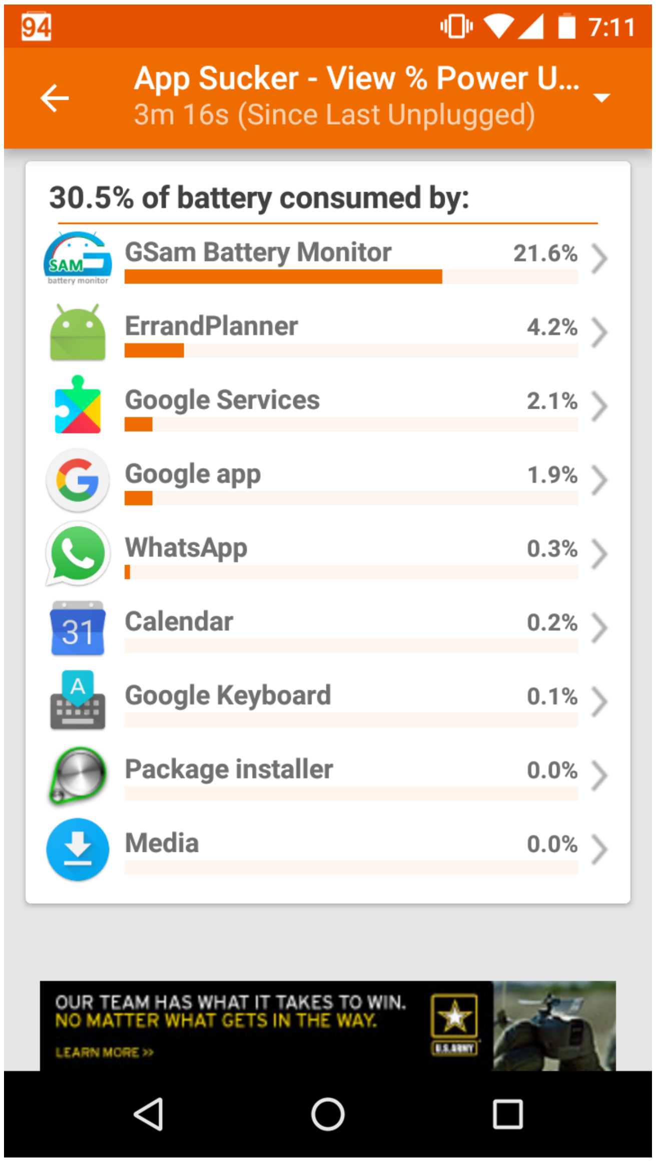
Time: 10#20#30#40#50#60#70#80#90#100

Priorities: 3#1#1#2#3#2#1#3#1#2

Mode of transport: Driving

Avoid: Tolls

I obtained the following statistics from the monitor application:


94 7:11

App Sucker - View % Power U...

3m 16s (Since Last Unplugged)

30.5% of battery consumed by:

GSam Battery Monitor	21.6%
ErrandPlanner	4.2%
Google Services	2.1%
Google app	1.9%
WhatsApp	0.3%
Calendar	0.2%
Google Keyboard	0.1%
Package installer	0.0%
Media	0.0%

OUR TEAM HAS WHAT IT TAKES TO WIN. NO MATTER WHAT GETS IN THE WAY.

LEARN MORE »

Figure 3.3.2.1 Battery usage

3.3.3 The application in use

Data entered:

Addresses: San Francisco#las vegas#Los Angeles #Vancouver Canada#New York

Time: 10#20#30#40#50

Priorities: 3#1#1#2#1

Mode of transport: Driving

Avoid: Tolls

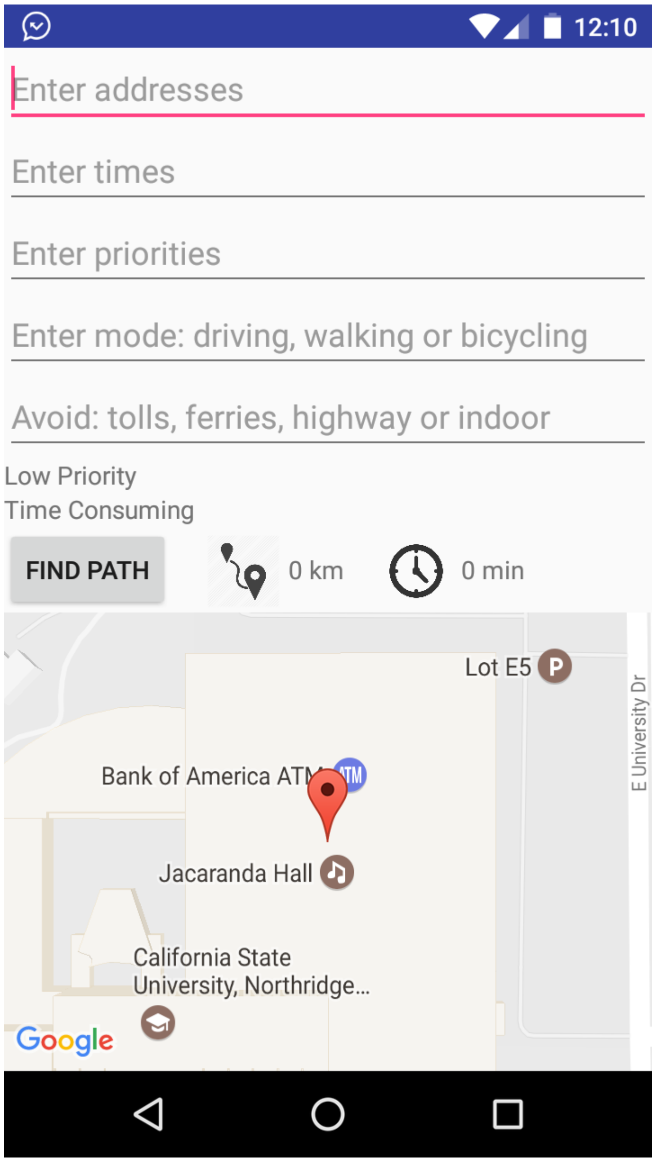
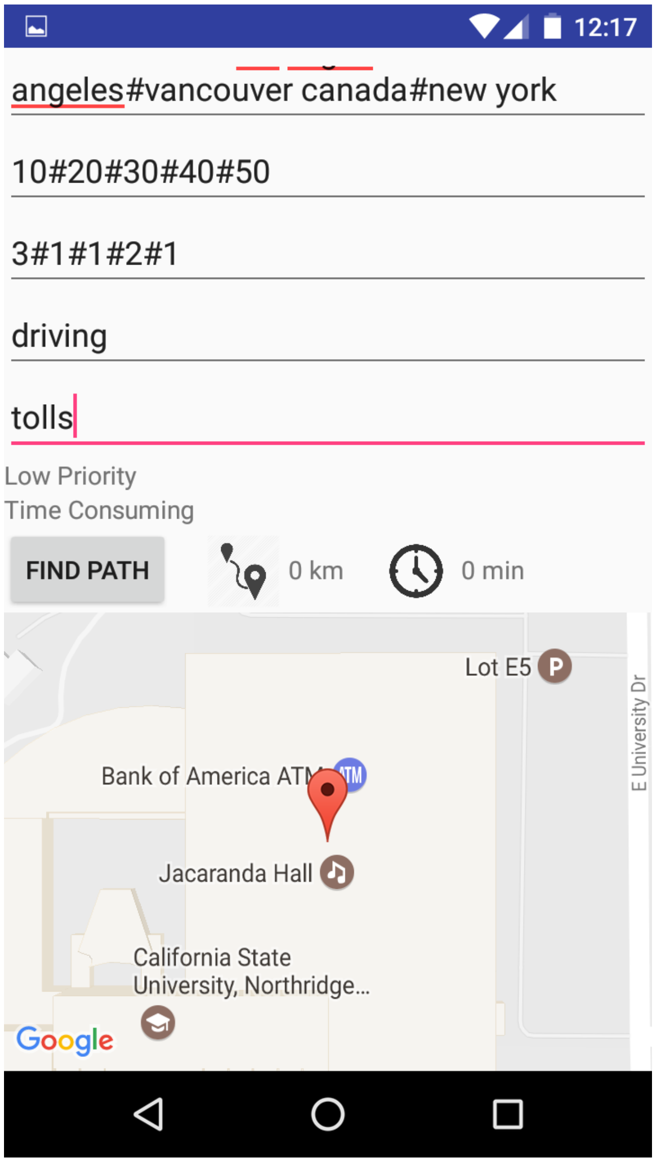


Figure 3.3.3.1 Application


12:17

angeles#vancouver canada#new york

10#20#30#40#50

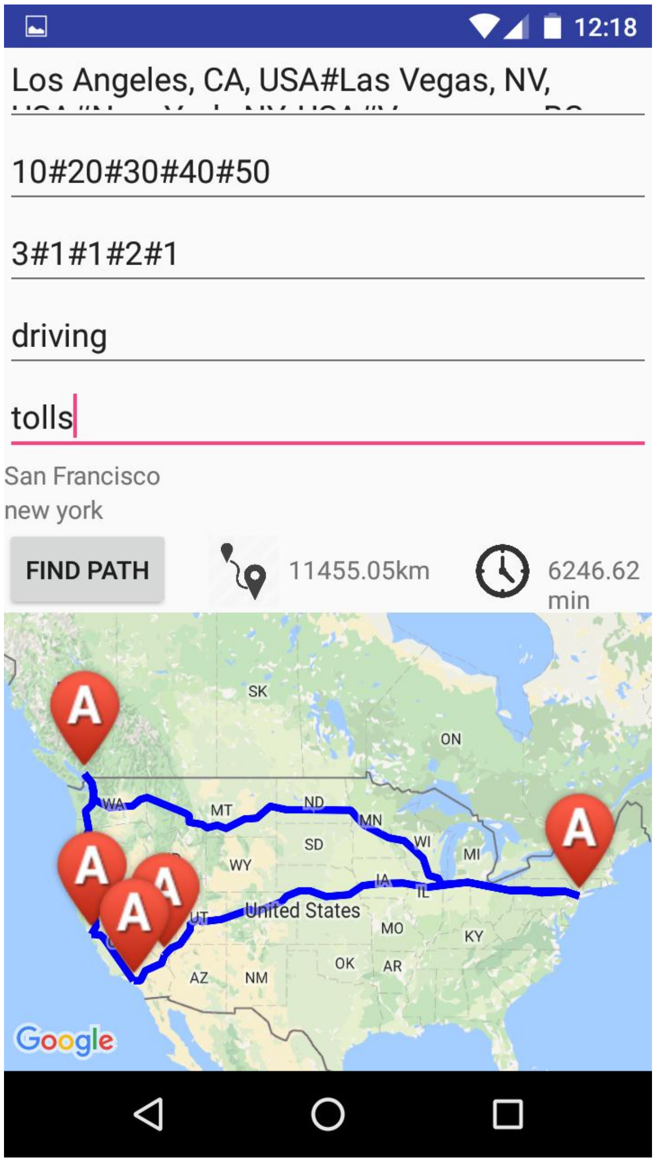
3#1#1#2#1

driving

tolls

Low Priority
Time Consuming

Figure 3.3.3.2 The user enters the parameters


12:18

Los Angeles, CA, USA#Las Vegas, NV,

10#20#30#40#50

3#1#1#2#1

driving

tolls

San Francisco
new york

FIND PATH

11455.05km

6246.62 min

Figure 3.3.3.3 Hamiltonian path displayed on the map

The resolved list of addresses is:

Los Angeles, CA, USA

Las Vegas, NV, USA

New York, NY, USA

Vancouver, BC, Canada

San Francisco, CA, USA

Chapter 4 - Conclusion

4.1 Conclusion

The application is not meant to compete with the commercial routing applications available in the market as they are targeted towards businesses who need to perform route planning on a large scale and are all paid applications. [8][9][10] The application is subject to Google’s free usage limits but the limits are enough to support personal use.

4.2 Future Enhancements

The following Future Enhancements could be considered:

- The application to consider which side of the road the location that the user wants to visit is on, so as to factor any U-turns.

  Google distance matrix does not currently provide this functionality.
- Closing time for various businesses to also be considered when mapping out the path. Google Places API has an opennow parameter that returns a Boolean value[11]. However, it does not offer a way to check for a time in the future.
- Public Transport can also be included in the modes of transport supported.
- A comparison with the paid services can be made.

References

[1] Nicos Christofides, Worst-case analysis of a new heuristic for the travelling salesman problem, Report 388, Graduate School of Industrial Administration, CMU, 1976.

[2] https://en.wikipedia.org/wiki/Travelling_salesman_problem

[3] https://developers.google.com/maps/documentation/distance-matrix/intro

[4] http://techland.time.com/2013/04/16/ios-vs-android/

[5] https://developer.apple.com/programs/how-it-works/

[6] https://developer.android.com/studio/features.html

[7] https://sites.google.com/site/gson/gson-user-guide

[8] https://www.route4me.com/

[9] https://www.roadwarriorllc.com/

[10] http://carobapps.com/products/inroute/

[11] https://developers.google.com/places/web-service/

[12] http://www.cs.tufts.edu/comp/260/Old/tspscribe-final.pdf