

Final Report: SmartRoute

CS6604: Spatial Databases

May 12, 2014

Kyle L. Schutt, Wei Wang, and Parang Saraf

Abstract— In this paper, we design an android application to help the traveller to efficiently and economically explore the city through the public transit system. The application will provide smart travel routes with diverse demands by combining WMATA with the local place information from Google Places such as popular attractions, convenience store, and other points of interest.

Index Terms— Data Mining, Android, Travel, WMATA, Twitter, Route, Transit

I. INTRODUCTION

It's no surprise that smartphones are swiftly taking over the mobile phone market. With Nielsen's mobile insight report, it shows that the smartphone owners in the US make up 62.5% of the market in 2013[1]. There are two mainstream Operation Systems for smartphone platform: Apple iOS and Google Android. Apple iOS takes 40.6 percent of the market share, while Google Android takes 52.2 percent.

Given the rapid adoption of smart phones worldwide, there is clear evidence that mobile phones are becoming the "second screen" to receive and public information. With hi-resolution and larger screen, the smartphone users are more comfortable to with their mobile devices. With the hundreds and thousands applications (apps) in Apple App Store and Google Play, the smart phone can be used just about anything.

Congruent to the booming consumer market for smartphone apps, there are more and more enterprises and government departments provide their data or API to public free. For example, Twitter provides stream API [3] to access the real-time tweets data which can be used to monitor the current trending topics or detecting abnormal events. The association managing the DC Metro system, WMAA, provides a transparent API [2] to access the open metro data such as the schedule of the rail and bus, the routes information and the incidents. With the high accessibility and diversity of open data and the prevalence of smartphones, it provides a great opportunity for developers and researchers to design creative and novel approaches to solve real world problems and meet

explicit or potential demand of consumers.

Imagine the following scenario, you are a traveler and this is the first time you have visited Washington, DC. Everything here is new to you. You want to visit the museums in the Capitol region, and after that you may want to go to a theater to watch a show. Then you may have such questions: Which route shall I take with most efficiency and lowest cost? Where are the nearby bus/rail stops around my hotel or destinations? When is the next bus coming? Furthermore, if I want to get a cup of coffee in the middle of my trip, which stop should I get out at, and how much time and money would it cost me? As a newcomer to a strange city, you may ask a lot of questions like this.

With this scenario in mind, the WMATA set out to challenge developers to extend and create solutions leveraging their open datasets that answer these questions [6]. The WMATA is focused on creating next-generation applications that leverage real-time data for bus and rail routes within the system. The overall goal being to provide a user-centric application that can help mitigate some of the issues and headaches a user may face when riding the metro. Additionally, these challenges focused on creating a system that is more accessible to riders with disabilities by providing important notifications about elevator outages and locations of elevators at certain stops in the system. Finally, the WMATA's other challenge is to create a unified "multi-modal" transportation application that links Metro, Amtrak, MARC, and VRE to provide users with seamless scheduling.

In this paper, we propose a novel approach and design for an Android application to solve the above questions through open data. Through the WMATA API, we can get the real-time information of the entire metro transit system. For instance, we can get the geolocation of each stop as well as the schedule of each line. Also, we can obtain rail and bus incidents as they appear in the public display throughout the transit system. Through Google Places, we can easily get the rich location information about popular attractions, restaurants, cafés, groceries, and convenience stores. By combining these two data sources, our approach could efficiently and effectively provide users multiple real-time routes to satisfy different demands.

The major contributions of our paper are:

- Combine transit routing information with rich location information.
- Dynamically design the route according to the real-time information and user requirements.

We organize the paper as below: in the section 2, we introduce the related work. In section 3, we will provide an overview of our approach. In section 4, we will talk about the detail of our system design. And finally in section 5, we will talk about our conclusion and future work.

I. RELATED WORK

There are a lot of apps have been developed related to the transit system in cities. Some apps are focused on providing the bus and rail schedule information. For example, the app DC Metro and Bus provides the functionality of the next bus, next train, nearby stops, metro maps and metro alerts. Some apps try to utilize the tweets along with bus/rail schedule information. For instance, Metro Twitter will show the latest tweets about each of Washington DC's WMATA metro rail lines in addition to getting real time train arrivals. But it just simply outputs the original tweets without any analysis. In our approach, we will predict the congestion situation of each line according to the tweets and then suggest the alternative route to avoid the bad trip. Most of the apps provide the function to search the bus or rail stops nearby or given location. But in some situation, what around the stops are more important than where is the bus stop. Take our previous example, if you drive to a new city, you may care more about which stops have parking plots around it rather than a specific stop.

Other application, for both Android and iOS, include very detailed metro information based on data ingested from the WMATA Transparent API that is provided to developers. The DC Metro Transit [4] application on Android shows the generic metro map, time predictions for busses and rail, trip planners, and metro news aggregators. It also allows the user to search for nearby stations based on their current location. Again, this type of application falls short of providing points of interest along the route, specifically at rail and bus stops. Many applications are simply a rehash of the same functionality. Embark DC and DC Metro are a few free apps that are available, while other apps are pay for use.

As noted above, many of these application share functionality including the display of the *normalize* DC metro rail map. This map is highly deceiving in how it is organized and does not provide an accurate visual representation of the system layout. Without reference points, or the use of a real basemap, it is hard to discern the location of certain landmarks and POI's with respect to certain locations in the system. Therefore, from a visualization point of view, it would be beneficial to show the user an accurate, to scale, map of the system and be able to geolocate the POI's directly onto the map. Many users are

already familiar and comfortable with using smartphone mapping applications in this way.

During our research, there was one glaring feature that never showed up: calculating the cost of a specific route. Out of all the applications we sample, none included a cost or even an estimated cost for a specific route selected by the user. After researching some more, we found that it should not be all that difficult to analyze the route and properly calculate the cost based on the time, distance, enter and exit points, and the type of transportation. We found that the algorithm for cost on bus routes was significantly easier to obtain than the cost of a rail route because of the reliance on enter/exit points within the system. Additionally, the system does not take into account the time you have been in the system. That is the time factor determines Peak or Off-Peak fares.

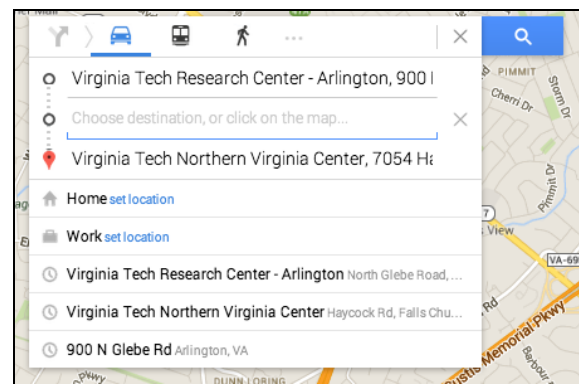


Figure 1: Waypoint directions for driving.

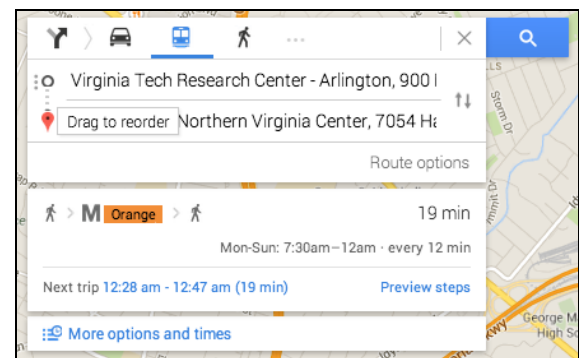


Figure 2: No waypoint directions from transit.

Another very interesting and most used routing application is Google Directions. While it provides a great deal of functionalities and options while determining transit options for cars, it fails to deliver when it comes to using public transit options. One of the biggest disadvantages of Google Maps is the lack of waypoint for public transit. As of now, it can only allow a source and destination for route calculation. Waypoints can be added only when the navigation is through a car. Even with that there are two major problems: 1) It doesn't find the most optimal route. So, say you select two waypoints A and B between a source and destination, Google maps will provide the route connecting waypoints A and B in the order they were entered.

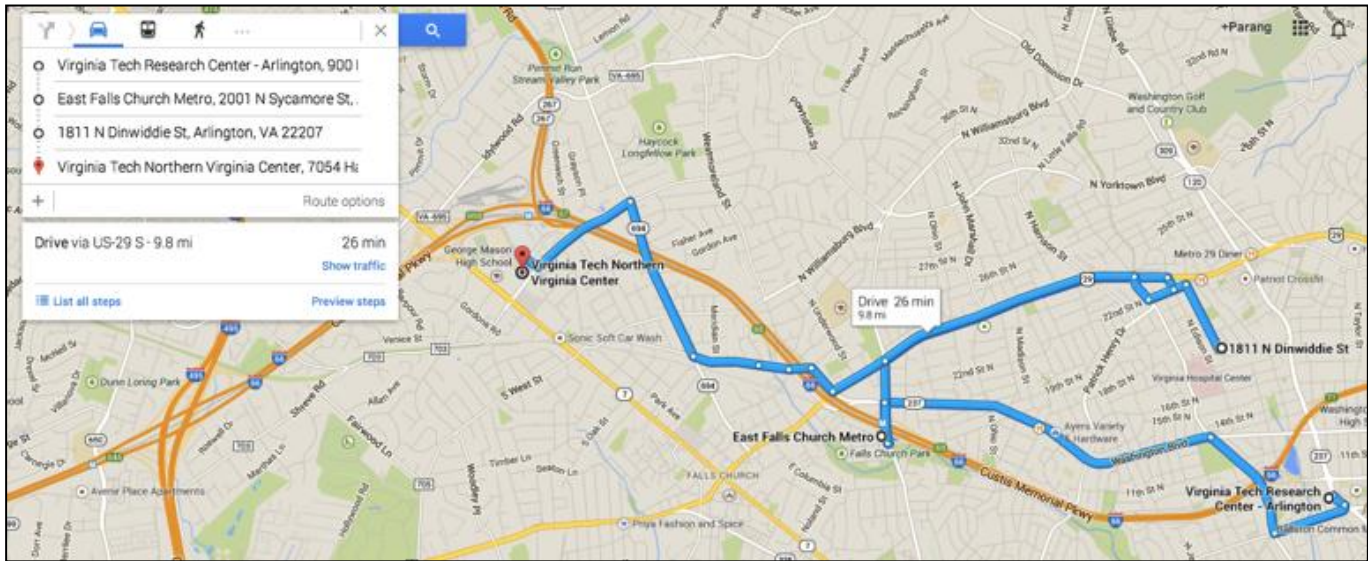


Figure 3: Multiple waypoint routing for driving directions.

If Source – A – B – destination is the order that was entered by the user and the most optimal route would have been Source – B – A – destination, then google will not identify that. 2) Another major problem with google waypoints is the lack of the identification of generic waypoints. So, say if a user enters “supermarket” as a waypoint between a source and destination, without precisely specifying the supermarket then google will not be able to identify the best supermarket that will be enroute the original path.

Our app aims at finding solutions to these two common problems. That is, finding the most optimal route with waypoints using public transit options. And being able to find the waypoints which are enroute and satisfy the user criteria in terms of user ratings, POI timings, POI price range and detour that the user have to take.

The rest of this paper aims to outline the approaches to creating an application that will not only provide the basic functionality that is expected of transit applications, but to provide contextually aware content based on the user’s preferences.

II. PRELIMINARY APPROACH & DESIGN

Since we are focused on WMATA Challenge #1 to provide near real-time information to riders, there are several different approaches we can take to handle the incoming data. Specifically, the data with the WMATA API [5] is a mixture of dynamic and semi-static information about rail and bus lines and associated metadata. For example, rail lines and stations do not change often while arrival information is updated much more frequently. Additionally, this data does not provide exact coordinates of the stations or the corresponding polylines for the rail and bus routes. Therefore, additional GIS layers are required to analyze the correct routes [7].

By creating a native Android application, we can take advantage of the processor more effectively than a simple web-based application in that some of the analysis can be offloaded to the client-side to improve the overall responsiveness. Initially, we thought to create a web-based application that could work cross-platform and was browser agnostic. However, because this application relies on real-time data ingestion and updates, we thought the majority of our development time would be spent creating a dynamic, single-page website instead of focusing on the GIS analytics as part of the system. Creating a native Android application provides many benefits over a web-based version of the system because of increased usability, performance, efficiency, and overall user experience. We could have easily created an iOS application instead of Android; however, Android is the cheaper and easier option for this project.

In order to meet our three contributions described previously, we have to use several different types of tools to properly analyze the data and display it to the user. In order to expedite the usage of the basic GIS data layers, we will be using OpenGeo Suite as our database and middle layer management application. This will provide us a RESTful web-service for access to our data as needed and in real-time.

On the server side, the database will contain a variety of support datasets that are static. These datasets include Metro Bus Lines and Stops, Entrances, Rail Lines, Park-n-Ride Lots, Station Entrances, Stations, and Station-to-Line cross references [4]. These are all GIS-enabled data layers that we can use for further analysis and visualization. For example, the Rails Lines polyline layer will provide an accurate representation of the Metro Rail routes (as opposed to the skewed metro map).

Furthermore, these datasets only provide a partial picture of the entire system since it is also required to include points-of-interest, arrival and schedule times, weighted routing, fare pricing, and congestion analysis. For this, we must incorporate data from the WMATA API [5]; specifically, Station Predictions for Rail, and Schedule by Stop or Route for

Bus. Additionally, we must incorporate fare pricing into the analysis as a weighted function in order to determine the shortest and cheapest route. This data is available from WMATA and the Washington Post as an API [6]. Additionally, this data can be accessed via a custom RESTful API built with Ruby on Rails (RoR) in such a way that minimizes the hit on the API limits by using simple caching of the static data. The RoR application will also act as an aggregator for social media data from Twitter and Yelp!. The above mentioned datasets shall be accessible from the Android application to improve performance and reliability of the system. The Android application shall be able to visualize this data in a useful manner.

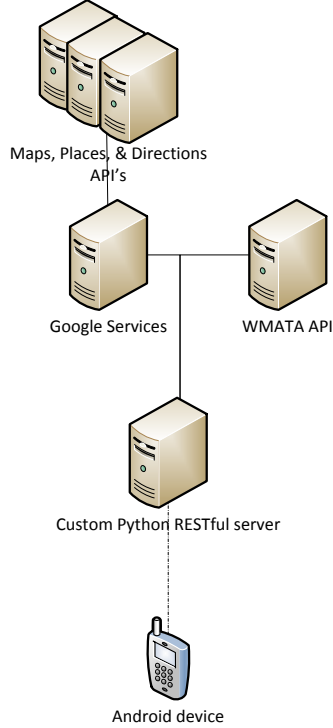


Figure 4: Final Architecture Diagram

As mentioned above, some of the analysis is performed on the client-side and some is performed on the server-side. One of the more intensive calculations will be the routing of a user from one location to another. For this, we can utilize two different approaches: a web-based routing API (Google or MapQuest), or utilize PostGIS pgRouting [8] which provides database level routing functions. In fact, we might utilize both options to compare and visualize the differences for the user. The latter of the two approaches provides more flexibility and for us to analyze the route against the fare pricing and traffic congestion data already stored in the database. Additionally, it keeps the entire system more contained than it otherwise would be. It also allows us to perform analysis on a stop-by-stop basis to pull points-of-interest in the surrounding area (similar to walkscore.com).

On the client-side, it has already been mentioned that some analysis will occur with respect to the real-time data to provide a seamless user experience. The user shall be presented a list of the closest rail and bus stations based on the current location or the user can select from an alphabetical list. The user shall select an origin and destination in the application. The application shall display multiple routing options based on price, congestion, length, duration, and other factors. These other factors may include a simple selection based on the user's mood. For example, they may select "Hungry" to get a list of restaurants or food places at each stop along their journey. They may also select "Get me home!" which will give them the fastest route without regard to price. There shall be similar options to these such as "Scenic route" for tourists or "Cheapest" for the budget conscious rider.

III. PROPOSED APPROACHES

The app aims at providing solution to a very simple day-to-day problem: If a user is interested in multiple points of interest, what is the most efficient way for him to visit all of them given some user specified constraints.

Even though the problem is very straight forward, the solution becomes increasingly complex as we add multiple real life constraints. Hence, in order to deal with this incremental complexity we develop our app in multiple stages. These stages are also summarized in a tabular format in table 1.

1) STAGE I

Problem Statement: Given multiple waypoints between a source and destination, find all possible routes that use public transport.

Key Assumption(s): We know the exact geo-location for the waypoints. There are no user-specified constraints in terms of route preference.

Sample Use-Case: Say there are three waypoints A, B, C between source S and destination D. This means there are six possible ways to visit these destinations: S-A-B-C-D, S-A-C-B-D, S-B-A-C-D, S-B-C-A-D, S-C-A-B-D and S-C-B-A-D. Each of these routes will have multiple options for travel based on mode of transport. For each possible route, the app will display all travel options.

Proposed Solution: Finding all possible permutations of the string S-A-B-C-D where the first and last letters are always S and D can generate the different route combinations. For each of route combination, we query Google Directions API to find all travel options.

2) STAGE II

Problem Statement: Given multiple waypoints between a source and destination, find all possible routes that use public transport and also satisfy some user-defined constraints. The constraints include time for departure, time for arrival, time of travel, cost of travel, number of transfers and walking distance.

Key Assumption(s): We know the exact geo-location for the waypoints.

Sample Use-Case: The use specifies the waypoints, along with his route preferences. The system generates the best possible

3) STAGE III

Problem Statement: Given multiple waypoints between a source and destination, find all possible routes that use public transport and satisfy user-defined and point of interest specific constraints. In addition to user-defined constraints from stage II,

	Waypoints	Waypoints Restrictions	User Specified Restrictions
Stage I	Fixed Geo-located Single POI per waypoint Known Source and Destination Ex: Union Station	None	None
Stage II	Fixed Geo-located Single POI per waypoint Known Source and Destination Ex: Union Station	None	Time of Departure Time of Arrival Minimize Time of Travel Minimize Cost of Travel Minimize Number of Transfers Minimize Walking Distance
Stage III	Fixed Geo-located Single POI per waypoint Known Source and Destination Ex: Union Station	Waypoint Visiting Hours Waypoint Visiting Duration	Time of Departure Time of Arrival Minimize Time of Travel Minimize Cost of Travel Minimize Number of Transfers Minimize Walking Distance
Stage IV	Generic Single POI per waypoint Known Source and Destination Ex: coffee shop	Waypoint Visiting Hours Waypoint Visiting Duration	Time of Departure Time of Arrival Minimize Time of Travel Minimize Cost of Travel Minimize Number of Transfers Minimize Walking Distance
Stage V	Generic Single POI per waypoint Known Source and Destination Ex: coffee shop	Waypoint Visiting Hours Waypoint Visiting Duration Waypoint User Ratings Waypoint Cost Ratings	Time of Departure Time of Arrival Minimize Time of Travel Minimize Cost of Travel Minimize Number of Transfers Minimize Walking Distance
Stage VI	Generic Clustering of POIs Known Source and Destination Ex: coffee shop	Waypoint Visiting Hours Waypoint Visiting Duration Waypoint User Ratings Waypoint Cost Ratings	Time of Departure Time of Arrival Minimize Time of Travel Minimize Cost of Travel Minimize Number of Transfers Minimize Walking Distance
Stage VII	Generic Clustering of POIs Source and Destination are Same Ex: coffee shop	Waypoint Visiting Hours Waypoint Visiting Duration Waypoint User Ratings Waypoint Cost Ratings	Time of Departure Time of Arrival Minimize Time of Travel Minimize Cost of Travel Minimize Number of Transfers Minimize Walking Distance

Table 1: Table of stages with varying restrictions

route by considering all user-specified constraints and displays them in some sorted order.

Proposed Solution: From Stage I, once we generate multiple travel options from all possible route combinations, we filter them based on user-specified constraints and sort the resultant set in some particular order for display.

the points of interest can have their own constraints like they can be visited only during certain time of the day. The use can further specify the duration of visit.

Key Assumption(s): We know the exact geo-location for the waypoints. The user specifies time and duration constraints for Points of Interest.

Sample Use-Case: Say waypoint A can be visited only between 10 am and 4 pm, and the user wants to visit A for approximately

1 hr. Similarly waypoint B can be visited only between 3 pm and 7 pm and there is no associated duration with it. Based on this information and other user specified constraints, the app will find the preferred travel suggestions in a way such that the user reaches waypoint A on or before 3 pm and waypoint B is included in the route only after waypoint A.

Proposed Solution: Time constrained waypoints introduce an ordering for visiting waypoints. For the waypoints that specify some duration, the latest possible visit time will be adjusted based on the closing time and duration of visit. The waypoints that don't specify any duration, default 'x' min duration is assumed, where x can be adjusted. From all possible travel combinations the app will determine the ones that satisfy the time constraint first. The qualifying routes will be further imposed to other user-specified constraints and the resultant set will be displayed to the user.

4) STAGE IV

Problem Statement: Given multiple generic waypoints between a source and destination, find all possible routes that use public transport and satisfy some user defined and point of interest specific constraints. The waypoints are generic in nature for example coffee shop and thus don't have a fixed geo-location. The visiting time for waypoints can either be automatically determined from Google Places or can be user specified.

Key Assumption(s): Multiple points of interests can't be clustered together in one waypoint. There are no restrictions in terms of user ratings for points of interest. The system will select the best point of interest from several contenders, based on user specified constraints. In case, if there is a tie between multiple best points of interest, the system will pick one at random.

Sample Use-Case: The user wants to visit any coffee shop, any grocery store and one particular library during his commute from work to home. There can be several contending coffee shops and grocery stores. The system will identify the ones that are on the way of travel, are open for visit and also satisfy all other user specified criterions.

Proposed Solution: In order to determine contending routes, only waypoints with known geo-locations are considered. If there are no waypoints with known geo-locations, only source and destination will be considered. The generic waypoints are searched near train/bus stops that are enroute the contending routes. In order to identify a particular generic waypoint, the visiting time for the waypoint along with the distance from the train/bus stop is used. Google places API is used to gather information about the points of interest.

5) STAGE V

Problem Statement: Given multiple generic waypoints between a source and destination, find all possible routes that use public transport and satisfy some user defined and point of interest specific constraints. The waypoints are generic in nature, have associated user ratings, visiting time and cost ratings.

Key Assumption(s): Multiple points of interests can't be clustered together in one waypoint.

Sample Use-Case: The user wants to visit any coffee shop, any grocery store and one particular library during his commute from work to home. Between contending coffee shops and grocery stores, the ones those satisfy rating, cost, visiting time criteria and closest location are filtered out. Based on some formula that takes into account the importance of all these four factors, the best point of interest is identified and used as waypoint.

Proposed Solution: Among the contending points of interest from stage IV, the best point of interest is identified based on a formula that give different weightages to user ratings, cost, visiting time criteria and distance from stop. For example, a coffee shop with rating of 3.8 from 100 reviewers might be a better place than another coffee shop with a rating of 4 given by 10 reviewers. Similarly, between two contending coffee shops with close by ratings of 3.8 and 4, the one that is less costly might be more preferred than the other one. Hence, there is a need to identify a formula that takes into account all these different combinations and then gives score to points of interest. User ratings and place information specified on Google Places API are used.

6) STAGE VI

Problem Statement: Given multiple generic points of interest between a source and destination, find all possible routes that use public transport and satisfy user defined and point of interest specific constraints. Multiple points of interest can be clustered together in one waypoint and have associated user ratings, visiting time and cost ratings.

Key Assumption: The points of interest will be determined along some contending routes. Hence, there must be at least two points with fixed geo-locations (source and destination) to determine contending routes.

Sample Use-Case: The user wants to visit any coffee shop, any grocery store and one particular library during his commute from work to home. The system identifies that there is a coffee shop right next to the library and also satisfies all user specified criteria. Hence, it clubs coffee shop and library in one waypoint. While clustering points of interest together the system automatically takes into account the walking distance between them and clusters them such that the walking distance/ time taken is minimized.

Proposed Solution: In order to cluster points of interest, we always start with points of interest with the least number of options. For example, between coffee shop and grocery store, the number of qualifying grocery stores will be less as compared to the coffee shop. Among the qualifying grocery stores, our first preference would be to identify one that is in the vicinity of stops with fixed geo-location (source, destination, particular library). There will be a tradeoff between the walking distance at a fixed geo-location and getting down at a stop on the way and then walking to the point of interest. For example, a user might want to walk to a grocery store that is 0.3 miles from a fixed waypoint than to walk to another grocery store which is 0.2 miles but requires user to get down at an unscheduled stop. Once we identify a list of qualifying grocery stores, we will look

for a similar list of coffee shops. While doing so, we will use a formula to determine the tradeoff between walking distance and getting down at unscheduled stop. This will result in clustering of multiple points of interest together while still qualifying all the constraints.

7) STAGE VII

Problem Statement: Given a source and multiple generic waypoints, find the most efficient route such that it satisfies all the user defined and point of interest specific constraints. The user starts from the source, goes to all the points of interest and comes back to the source.

Key Assumption: None

Sample Use-Case: The user wants to visit any coffee shop, any grocery store and any library. He starts from his home, goes through all these points of interest and comes back to his home. The system displays multiple travel routes that satisfy user's entire criterion.

Proposed Solution: Because there are no two fixed points, the initial number of contending routes is not limited now. Hence, the app starts with the particular points of interest that results in the least number of options. It arranges them in the decreasing order of distance and identifies the one with the least time of

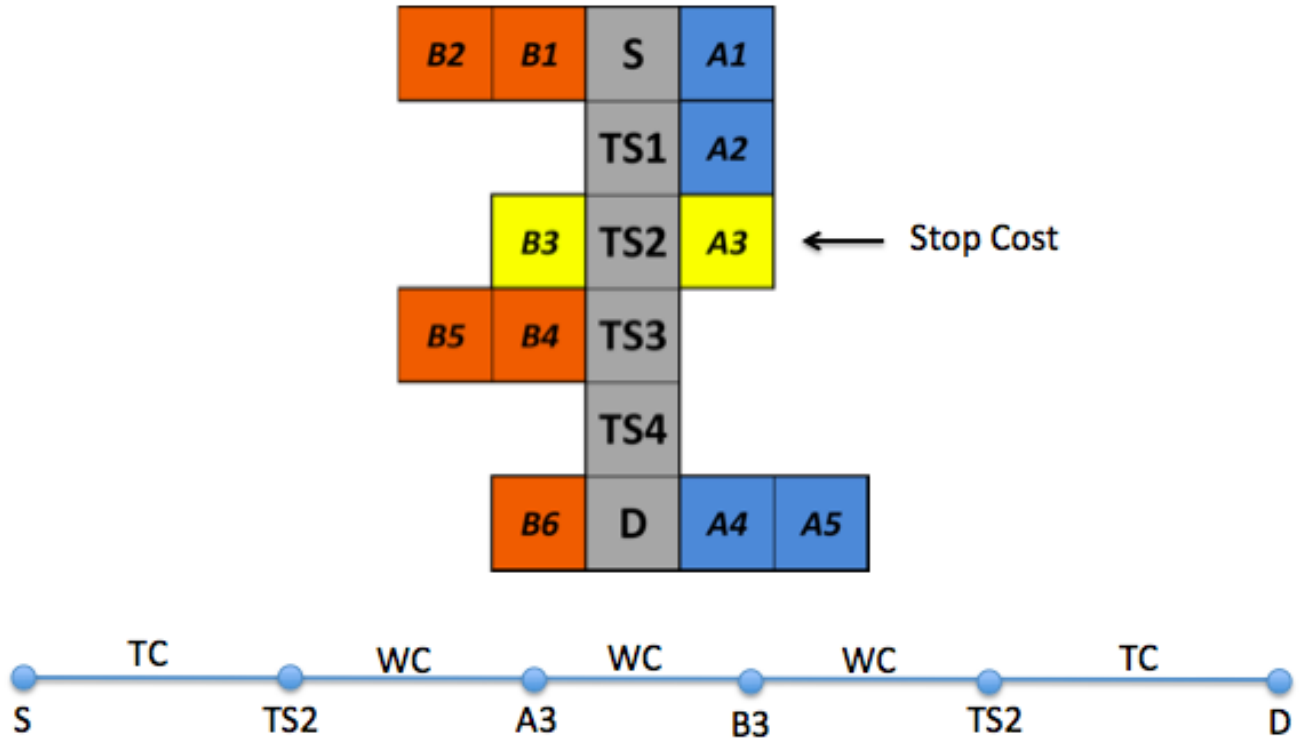
travel and considers that as fixed point. Once one of the waypoints is fixed, the problem becomes same as the one in stage VI.

Several stages with increasing complexity are mentioned in this section; however, not all of them qualify as being in scope for the class project. There is a blurred boundary between what constitutes the scope for a class project and a yearlong research project. We will try to implement as many of these stages as possible and plan to work on it even after the semester is over. We believe that a complete implementation till the last stage will result in a very useful app and will also yield a good research publication.

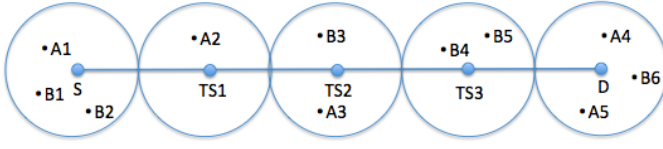
IV. WAYPOINT CLUSTERING MECHANISM

We have implemented till stage VI, as described in the last section. While stage VI implies a good number of restrictions in terms of what a waypoint implies and what a user can specify, our solution currently handles a limited number of restrictions. The current solution is able to cluster generic waypoints together such that the most optimal route is identified. In terms of waypoint restrictions, the solution takes into account waypoint visiting hours, waypoint user ratings and waypoint cost ratings. In terms of user defined restrictions, the system currently only identifies the route that is most efficient in terms of time.

Figure 5: Cost algorithm details.



$$\text{Total Cost} = [\text{Dist}(S, TS2) + \text{Dist}(TS2, D)] * \text{transit_factor} + [\text{Dist}(TS2, A3) + \text{Dist}(A3, B3) + \text{Dist}(B3, TS2)] * \text{walking_factor} + \text{Stop_cost} - [\text{Rating}(A3) + \text{Rating}(B3)] * \text{Rating_factor}$$



Potential POIs are filtered by time, price range, and ratings

User Inputs	
• Source	• Waypoint A (Starbucks)
• Destination	• Waypoint B (Supermarket)

Figure 6: Potential POI selection mechanism

The routing mechanism starts with identifying the most time efficient route using public transit between the given source and destination. Figure 6 explains the potential POI identification step. Once the initial route has been identified, all the transit stops are identified next. These stops can be identified based on the stored routes and bus stops from WMATA API. For each of these stops we have associated geo-coordinates. We use these geo-coordinates to query Google Places API to identify the potential POIs for each of the generic waypoints that has been entered by the user. An important point to notice here is that each of the generic waypoints will result in several potential POIs. For example, a user's search for Starbucks will result in several Starbucks location that can be enroute. For each of these potential POIs there will be associated Google Places information like User ratings, visiting timings, cost ratings etc.

Once each of the transit stops is identified, the system queries the Google Places API to identify potential POIs for both waypoint types in a 1-mile radius. Please note that we have taken the 1-mile as the maximum one way walking distance that a user might be willing to walk. For this radius-based search we get potential POIs that are then filtered based on cost and user ratings. As can be seen in the diagram there can be some transit stops that don't have one qualifying waypoint type. For example, TS1 has only waypoint type of A present. Also there can be transit stops with none qualifying waypoint type. Once all the potential POIs have been identified, waypoint clustering mechanism is invoked which identifies the most suitable POIs to act as waypoints.

In order to understand the waypoint clustering mechanism, it is helpful to visualize the potential POIs in a different way as shown in Figure 7. Here the nodes highlighted in gray correspond to the source, destination and the enroute transit stops. For each of these nodes, the potential POIs that can act as waypoints are shown in blue and red. The POIs to type A waypoint are shown in blue and the POIs of type B waypoints are shown in red. The goal is to identify a combination of A_i and B_j (where $i < j$) such that the route $S - A_i - B_j - D$ leads to the most time efficient route. For cases where $i > j$, the route will become $S - B_j - A_i - D$.

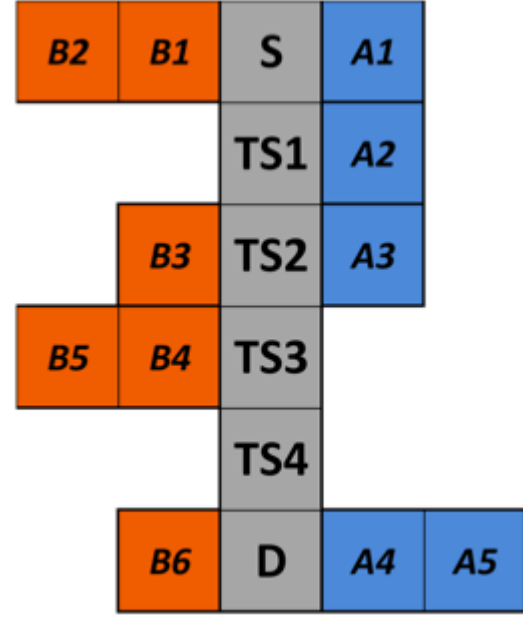


Figure 7: Potential POIs for each stop

For each of such possible combination of A_i s and B_j s the system calculates cost of travel. The cost of travel are calculated based on the distance of transit stops from source, distance of potential POIs from transit stop, number of transit stops and the user ratings of each of the POIs. Different transit parameters are associated with each of the transit modes.

There can be only two possible scenarios for the distribution of waypoints: 1) both the waypoints correspond to the same transit stop and 2) each of the waypoints corresponds to different transit stops. The cost calculation for each of these mechanisms is explained further.

The case where there are two transit stops for each of the waypoints, the user has to break his journey twice to reach each of these POIs. Corresponding to each of the break journey there will be cost associated.

V. SYSTEM DESIGN

The app uses two different APIs: Google Maps Directions API for finding travel routes, and Google Places for finding information about points of interest. The system design will involve both the client side implementation and server side implementation. However, for the initial few stages, till stage V are not computationally intensive and can be implemented at client side only.

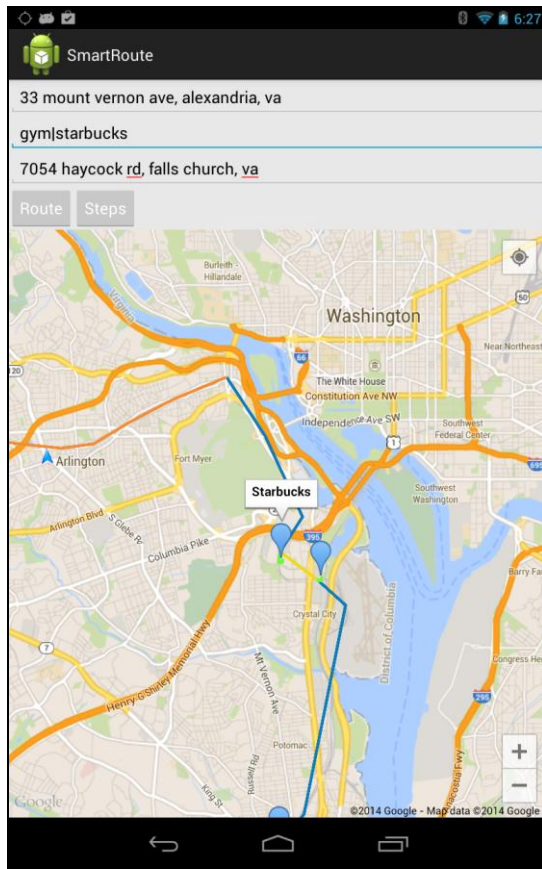


Figure 8: Screenshot of the map view with two points of interest.

In order to implement the initial three stages, we only require Google Maps Directions API. The API returns a list of multiple travel options between any two points. The introduction of multiple waypoints and the resulting different contending routes will require the system to query the API as many times as the number of contending routes. Once the App has received information about all possible travel options for each of the contending routes, it will identify most suited routes by applying user specified filtering. The list of contending routes can be narrowed down a bit further before the initial querying by applying the time restriction imposed by waypoints visiting hours. Because this is not computationally intensive, this can be easily handled at the client side, ex an android phone, that has limited computation power.

The advanced stages will require querying Google Places API, in order to get information about Points of Interest. Because these later stages involve generic waypoints, Google Places API will be used to get a list of possible points of interest that satisfy that criterion. This list will be subjected to user specified filtering criterions like ratings, cost, visiting hours etc. Only the qualifying points of interests will be further subjected to the distance requirements and will be used for clustering purposes. In order to work around Google Place API limits, several optimization mechanisms will be required. For example, searching nearby places of a particular location returns a list of qualifying POIs but doesn't return their distance from the particular location. In order to identify the distance between

POI and the particular location, the system needs to query the Google Maps API, which can result in lots of queries. We can rather use the geo-coordinates of the POI and the particular place to get a rough estimate of the distance and thus can save on multiple queries. Only the most qualified POIs will be subjected to actual Google Maps Distance API.

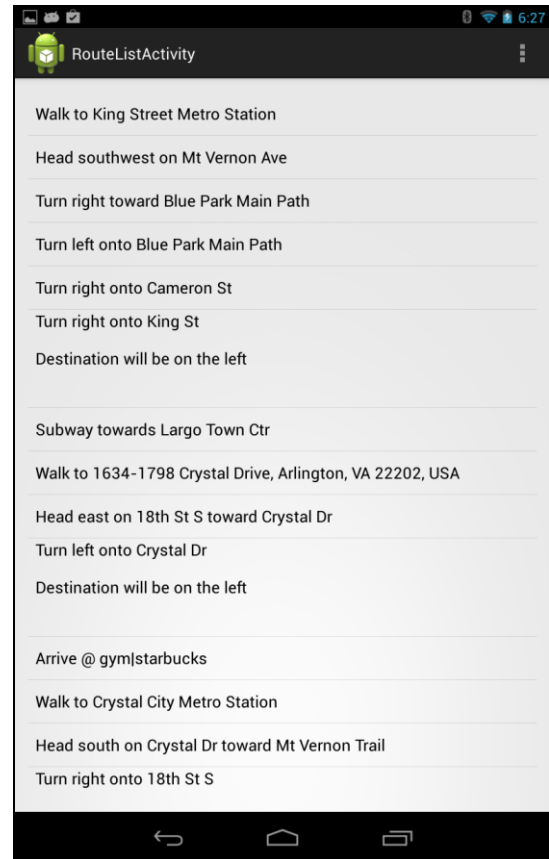


Figure 9: Step-by-step directions for a route with bus, rail, and walking directions.

Now that we have discussed our proposed approaches, system design decisions, and phase descriptions, we elected to create a native Android application that leverages the Google Maps API v2 for Android as the basis for displaying and communicating the results of our custom algorithm for determining multi-modal transit routes between destinations and specific points of interest. The algorithm itself was developed in python to be run on a server. We decided to offload this computation from the client-side because of the complexity and resource-intensive calculation that must occur to determine the least costly route from an origin to a destination via some points of interest. The details of this algorithm will be discussed later in this report.

The client side application is a simple single-page app that provides the user with auto-completion on the addresses or points of interest that they are searching. The points of interest can be as simple as a single category for a business type (e.g., a gym) or as specific as a restaurant (e.g., The Greene Turtle). The client side interface allows the user to enter multiple points of interest by using the "or" or pipe operator (i.e., "|") to separate

the search terms. Once the user has submitted their request and the server has returned results, the application displays the appropriate polylines and markers on the map interface. These polylines represent metro rail, metro bus, and walking routes and can be determined based on the color of the route. The metro rail lines are colored based on their principal colors used within the system. That is to say that the polyline for the Orange Line is in fact an orange polyline on the map. Likewise, bus lines are colored blue and walking directions are colored green. Additionally, markers along the route show the points of interest, the source, and the destination. For the points of interest, we display the name of the business we are to visit based on the results from the algorithm. This allows users to quickly view what business or point of interest they should visit.

Once the client has submitted their request, the server ingests the incoming RESTful request and begins processing the data. The python web server utilizes the Google Places API and the WMATA API to obtain points of interest information and rail/bus stop details, respectively. The server will then respond with a properly formatted JSON body that includes the routing details and costs. Currently, the server responds with two arrays. The first array contains the different combinations of directions between the source, points of interest, and the destination. The second array contains detailed information about the points of interest that the algorithm selected, and the final cost of the route. For example, a search for “coffee shops” might return a “Starbucks” from the Google Places API.

The Android application is then tasked with deserialization this information into the appropriate data structures. We use a JSON parser to reconstruct the strings into JSON objects for use within the application. While the JSON string returned from the server is full of data, we only parse out the details that are necessary for displaying and informing the user. For example, the JSON parser will pull out step-by-step instructions, transit details (i.e. rail line or rail color), distance, duration, and an encoded polyline. The encoded polyline is also decoded during this step to form a set of lat-lon pairs which can be displayed in the Android application.

Now that the application has deserialized the JSON string, we reconstruct the polyline, drop info markers, and fill out any details with the step-by-by instructions. Any subsequent requests by the user will clear out the details and the system will regenerate the view appropriately. This makes the application very lightweight and easy to use on the client side by offloading all of the computation to the server. However, this does leave the application at a disadvantage when unfavorable network conditions arrive for the user, or for the server.

VI. FUTURE WORK

The solution we created only addresses Stage 1 in our outline of work which leaves many additional features and functionalities to still be developed. Additionally, due to the complexity of the algorithm at hand, our solution only allows for two points of interest to be entered at a time. However, the system should be

scalable enough to handle any number of points of interest by creating a multidimensional matrix of potential routes based on the selected points of interest. We could then leverage parallel computing methods to analyze the multidimensional matrix to determine the least costly route. The main bottleneck for this kind of computation will be the reliance on external API's to generate the routing information. In this case, the future solution may be deployed for a smaller area to allow us direct access to the features and polylines on our local machine to increase the performance of the algorithm even further. The problem at hand lends itself quite well to parallelization because each of the routes is disjoint and do not rely on information from similar or nearby routes. Therefore, the cost of each route can be calculated independently on its own thread. A single master thread could therefore be employed to handle the cost analysis based on the unique thread ID and the route cost. Thus, the master thread only needs to select the least costly routes.

Other work includes providing useful turn-by-turn navigation instructions that many users expect from a mapping application. As mentioned in the stages above, the algorithm and cost analysis may be expanded to include additional factors. One important factor includes traffic data to ensure that these routes are weighted properly. While the routing details obtains from the Google Maps API does contain a duration factor, it is unclear where the traffic is occurring along the route in question. Again, it might be less costly to furnish a route that minimizes the users overall time in a congested area, and, therefore, does not suggest points of interest within that congested area.

Additionally, work on properly determining the weight factors for walking, bus, or rail must be determined. Walking is by far the most costly when compared to rail. We have implemented baseline factors attributed to anecdotal and hypothetical factors, but a more rigorous study is required to determine the appropriate levels. If necessary, different factors may vary from location to location, or times of day. There may be times (i.e., during rush hour) that will be easier to walk between two locations then to take a bus.

For increased usability and personalization for the user, a profiling system could be setup to help store and retrieve common routes used by the user on a daily basis. Additional parameters could also be set on a per-user basis such as review and price tolerances to help narrow the search results and tailor them directly for the user. Additionally, some design thought needs to be put in place for offline or unfavorable network conditions that keep the application usable even in times of limited network connectivity. A mechanism to allow users to download and save commonly used routes will be critical to solving this particular problem.

There are many more features that users would want in the application to make it more user friendly. Focusing on the user experience will be crucial for determining the long term usability and importance of such an application because of the overall market saturation of similar solutions.

VII. CONCLUSION

To summarize, our application fills a massive void in the current navigation application market by providing multi-modal transit and walking directions by computing the least costly route between a source/destination pair and points of interest. While the details and computational complexity of the problem can yield to unsuitable calculation times, the algorithm itself is not impossible to solve in a parallel computing environment. Furthermore, the application provides a useful solution for one or two points of interest in such a way that creates a step by step solution to a complex problem.

In today's application markets, and specifically in the mobile application market, there is an issue with market saturation and the ability for an application to gain enough market capitalization to be a viable product in the face of these concerns. In order for an application to be successful, a very specific need should be addressed and solved in the most effective and efficient way possible while providing an overall positive user experience. The sheer volume of applications requires user to sift through a large number of applications before discovering the correct application for the problem they wished to be solved.

There is a growing school of thought that applications developed by startups and small project groups should be highly local and highly specific. Both of these factors should be addressed and executed to the highest levels of speed, scalability, and dependability. If an application can achieve these metrics for a specific problem, then it has a high likelihood of being successful in the marketplace. As an anecdotal example, Instagram and snapchat both started with a very simple, specific, and local goal in mind: providing instant access to photos to your friends and be able to perform some type of manipulation on those photos. Snapchat had the unique idea that photos were not stored and immediately deleted after a certain amount of time. Both of these applications have grown tremendously since their inception. Instagram now includes videos and a web-based portal for users, freeing them from their mobile devices.

With this in mind, SmartRoute and the server-side infrastructure we designed and discussed throughout this paper solves a highly specific and local problem for users. A problem that is not easily solved, but not impossible either. By providing the ability to generate dynamic multi-modal walking, rail, or bus directions between two locations and points of interest allows users to interact and access mapping data in a new, more fluid manner. It frees the user from the rigid restrictions of single-point directions by allowing them to search on more open terms. SmartRoute allows the user to get back to enjoying their vacation, trip, or Sunday drive without having to worry about the hassles of multiple waypoint routes.

REFERENCES

- [1] http://www.comscore.com/Insights/Press_Releases/2013/12/comScore_Reports_October_2013_US_Smartphone_Subscriber_Market_Share
- [2] http://developer.wmata.com/Application_Gallery
- [3] <https://dev.twitter.com/docs>
- [4] <https://play.google.com/store/apps/details?id=com.jazzmonstewdio.android.dcmetro.activity>
- [5] http://developer.wmata.com/Application_Gallery
- [6] http://developer.wmata.com/page/read/Wmata_Developer_Challenge
- [7] [http://dcatlascgdcgisdcgovcatalog/results.asp?pretype=All&pretype_info=All&alpha=M](http://dcatlascgdcgisdcgovcatalogresults.asp?pretype=All&pretype_info=All&alpha=M)
- [8] <http://pgrouting.org/>