CS219: Project Report - Mobile Search

Jared Lindblom, Pragadheeshwaran Thirumurthi, Shao-Cheng Lu
203-975-285 904-000-582 704-085-247

Abstract— Mobile Internet usage is predicted to overtake desktop Internet usage in 2014. As much as half of the overall search volume in the United States will come from 95 million mobile Internet users by 2013. Existing mobile local search engines perform well when used for a particular narrow range of queries where the relevance of different Points of Interest is clear, but perform poorly when the query is not particularly straightforward. Future search engines must satisfy the demands of their users by taking into account more contextual information to provide better search results and providing this information in an efficient, timely manner. In this report, we propose, design and evaluate a technique to improve location-based searching in situations where the user is highly mobile. By predicting where the user will be in the future, we have proven that it is possible to provide results that are more relevant to the user in this situation, and therefore "mobility-aware."

Keywords- Location-Based Search, GPS, Predicted Search Points, Motion Vector

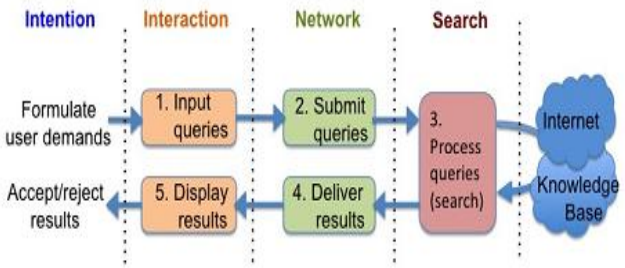
I. PROBLEM STATEMENT

Existing approaches in mobile search are not suitable in situations where the user is highly-mobile. Today, location-based search engines do not take into consideration a user's mobility when responding to requests. The user is assumed to be static for the time period the request is being issue and fulfilled. When the user is mobile, this assumption causes a large portion of the results returned by these search engines to become irrelevant to the user. Most of these results are for places that the user has already passed or is now far away from. For these reasons, we argue that future search engines should be "mobility-aware."

II. INTRODUCTION

Cloud Computing allows users to use resources provided by cloud providers at reduced cost. Mobile Cloud Computing is an integration of cloud computing into the mobile environment. It takes data storage and data processing out of the mobile device and performs operations in the cloud. This overcomes obstacles related to performance (battery life, storage and bandwidth), environment (heterogeneity, scalability and availability) and security (reliability and privacy). Mobile user's requests and information are transmitted to the central processors that are connected to servers provided by mobile network services. Then the user's requests are transmitted to the corresponding cloud provider through the Internet. In the cloud, cloud controllers process the users' requests to provide them with appropriate cloud services. Cloud computing services ensure that the services could be requested on demand and can be customized. So cloud computing helps in overcoming the obstacles involved in mobile computing.

FIGURE I. MOBILE FLOW SEARCH PROCEDURE



Context-aware mobile cloud services monitor the user's preferences and provide appropriate services to each user. Mobile search gets information from the user and integrates it with information provided by the search provider. Using cloud service for mobile search helps users in guiding them during their travel. Though mobile search would perform the same way as desktop search, it should consider several factors to ensure that users can be on the move. Compared with a traditional desktop search, both user demands and platform capabilities vary in the mobile scenarios. Mobile search should offer more flexibility and accurate results by taking user's preferences. For example, more location-dependent and time-critical results are expected even when only a general term is queried in the mobile search. Figure I shows the conventional procedure flow involved in cloud-based mobile search. Mobile phones have smaller memory, less storage and slower processors. These factors bring in technical challenges, but also provide greater scope to improve mobile search technique.

In the rest of the paper, we explain our approach towards improving the mobile search by calculating user's predicted location. We analyze the existing approaches in Section III. We propose our design in Section IV. In Section V we explain in detail how we implemented the system as an android application. We evaluate our application in Section VI and Section VII concludes the paper.

III. EXISTING SOLUTIONS IN MOBILE SEARCH

A. *Shape Search on the Go*

In [1], the author discusses the capabilities and limitations of mobile devices with regard to mobile search. She summarizes key guidelines that can be used to address technical issues that arise, and she gives an example of how a traffic search application may be implemented with respect to these key guidelines. The traffic search application she proposes in her paper focuses on how this traffic information should be accessed, and does not specifically solve the problem of high mobility. However, she recommends adopting

optimized operations specific to traffic search like adjusting the GPS update frequency to save power when the user is moving slow. This energy conservation technique has been adopted in our proposal.

B. *Hapori: Context-based Local Search for Mobile*

- In [2], the author proposes a dynamic search technique by considering user specific information for every search query.
- Hapori also leverages contextual factors, personal preferences along with similarity across mobile users to display highly personalized search results.
- This is done by building a behavioral model for users which exploits the similarity between users to tailor search results.

C. *Improving Mobile Search User Experience with SONGO*

In [3], a search cache system is presented. It proposes an idea to treat partial mobile memory as a cache system in order that frequently used queries can be served locally instead of through the Internet. The cache system will not work for our proposal because it focuses mainly on the performance of mobile search, and not on the search results.

IV. DESIGN

A. *Overview*

The goal of our design was to build a mobile application that provides better search results in highly mobile situations by improving location-based search results obtained through a cloud-based search engine. To do this, we proposed a technique to provide information to the cloud so that the results returned are relevant to the user when he/she receives them in the future.

If one was to look at the delay that could happen after querying for a search result, we have two cases: static search and mobile (dynamic) search. For these two cases, the user would realize that each case has different requirements for providing relevant search results. In the case of a static search, the results the user obtains may become irrelevant if a long period of time has gone by since he/she first performed the search. For example, if Bob was to search for restaurants nearby his location using his desktop computer, the results would probably be irrelevant to him if he obtains the results after 12 hours. The same case applies to a mobile search. However, in mobile search, the location of the user also comes into play. If, for example, Bob has performed the same query using his mobile phone as he is driving to downtown Los Angeles from Westwood, the results he obtains would probably be irrelevant to him if he obtained the results only when he reaches downtown Los Angeles. This is because the results returned to him were based on his location when he was in Westwood.

We argue that a large portion of results become irrelevant when the user is highly mobile using existing location-based search engines. Therefore we designed our mobile application to predict the future location of the user based on the user's vector of travel. This predicted location substitutes the user's current location whenever the user performs a search query, adding the context of mobility to the query.

B. *Future Location Calculation*

We have designed our mobile application to query the GPS sensor of the mobile device periodically. We store a specific number of past updates in an array to represent the user's vector of travel history. Between each two location updates, we calculate the user's vector of travel. The user's vector of travel is calculated by calculating the difference in latitude, the difference in longitude, as well as the difference in time between the previous location and current location of the user. The difference in latitude and the difference in longitude are divided by the difference in time to normalize the vectors with respect to time.

To predict the user's future vector of travel, we sum up all the vectors in our history to produce a "*true vector*" with a direction that reflects the user's changes in direction in our history. We argue that this gives us a more accurate estimation of the user's future direction of travel. Once the user's future vector of travel is calculated, we normalize the result to omit its magnitude. Since the magnitude of the true vector is the sum of the magnitudes of all the vectors in the history, this magnitude no longer reflects the user's speed. Instead the user's speed is called by summing up the difference in the coordinate degrees between the user's current location and the user's previous location and dividing that by the number of vectors in our history. Since these vectors have been normalized with respect to time, the results obtained represent the user's average speed in coordinate degrees per second.

Using the normalized true vector, we scale it based on how far in the future we want to predict the user's location. We have defaulted this value to 120 seconds. From this result, we are able to obtain the future geographical coordinates of the user i.e. the user's predicted future location.

C. *Performing Mobile Search*

Once the process of predicting the user's future location is implemented, a search can be performed at user's will that contains the context of mobility. This mobile search task is performed asynchronously from the future location calculation III-B. When the user performs a search, the user's search query is passed along with his/her future location to a location-based search engine. We have selected Google Places API [7] as our search engine of choice. The reason for choosing Google Places API is explain in the next paragraph. Once the search results are obtained from Google Places, they are plotted on the map along with the user's current and future location. The user is able to click on any of these results to find out more information about it.

We selected *Google Places API* as our search engine of choice because it provides the ability to perform a location based search using an arbitrary location. For us, this arbitrary location is the user's predicted future location. Also, Google Places API is a cloud-based search engine that is still in its experimental stage. It was released to the public early last year (May 2011). Utilizing this search service in our

application allows us to become a contributor to the improvement of this service itself. Google requires all the users of this service to be registered with the company. This registration is free and after you have been successfully registered to use the service, you are given an API key. This key needs to be embedded in each request you make to the service in order to obtain results. Google limits the request to Google Places API to 1000 requests per day. So using our application we could perform search for a maximum of 1000 queries/keywords each day.

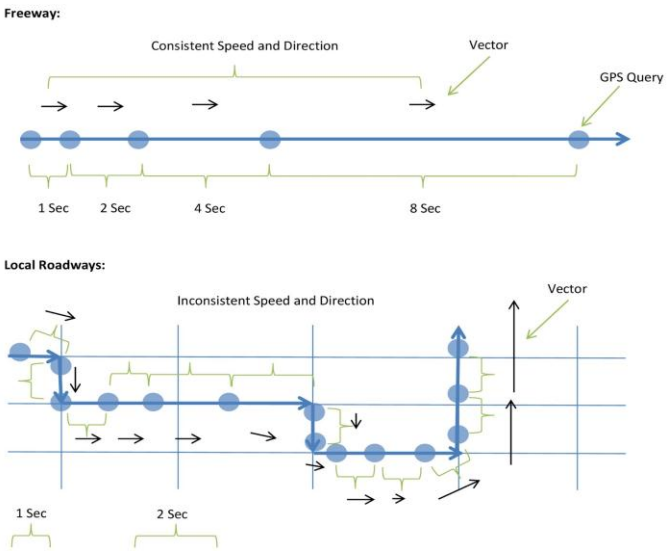
The advantage of using Google Places API ranges from its extremely important database of information and intention for mobile application developers to its simple and transparent concept of transmitting information between mobiles and Google. What Google Places API offers is an interface with a HTTP request as input and a list of JSON (JavaScript Object Notation) as output; both responses are fortunately lightweight for mobile use.

D. Increasing Energy Efficiency

GPS sensors are one of the most power hungry components in mobile devices. For this reason, we researched ways we can improved the energy efficiency of our design. Although this is not implemented in our mobile application, we argue it is a viable way using which one can improve the energy efficiency. We designed our system to make accurate predictions of the user's future location to provide more relevant results. It queries the location of the user at specific intervals and compute the user's vector (speed and direction) of travel. We propose a technique to change the rate at which our program queries the GPS sensor, thus improving its energy efficiency.

Our system can be designed to query the location of the user at a fast rate until it is able to determine consistency in the user's vector of travel. Once this has been determined, the rate at which the location of the user is queried can be exponentially backed-off to save energy. A drawing of how this might occur is shown in Figure II.

FIGURE II. MOVEMENT OF TRAFFICE IN A FREEWAY COMARAED WITH LOCAL ROADWAYS



There are two types of mobile situations we will cover in our system design:

a) *Freeway driving*

b) *Local driving*

Our system should behave differently in each situation. When a user drives on the freeway, we would expect to see a vector that stays mostly consistent, without drastic changes in direction or speed. When a user drives on local roadways, we would expect to see frequent changes in direction and speed. Local driving will cause our system to recalculate the user's predicted future location, so it is important for our system to be able to detect the difference between the two situations. There are a few ways we can detect this difference: speed, location service and vector changes.

Usually when a person travels on the freeway, they travel at speeds in excess of sixty miles an hour. When our system detects a consistency in the user's high rate of speed and direction of travel, it will assume the user is on a freeway and it will query for the user's location less often. When our system detects a fluctuating rate of speed and direction of travel, it will assume the user is driving on local roadways and it will query for the user's location more often to provide up-to-date results.

E. Past Design Choices - The Haversine Formula

Originally we calculated the user's vector of motion using Haversine Formula. But during implementation, we found that it causes too much overhead. Even though calculations using this formula produced more accurate results, we found this difference in accuracy to be negligible.

The Haversine Formula is defined as follows: Given any two points on a sphere, the great-circle distance between the two points can be found. In other words, if we treat two points on a sphere as two coordinates on earth, we can calculate the distance between the two points, taking into consideration the curvature of the earth. This distance (in miles), divided by the time it took to travel from the first point to the second point (in seconds), becomes the magnitude of the vector between the two points (miles/second).

Once the true vector (the summation of the history of vectors) was calculated, we predicted where the user would be in the future using a derivative of the Haversine Formula. Given the user's current bearing (calculated as the radians, clockwise from North of the "true vector") and the predicted distance (calculated as the magnitude of the "true vector" multiplied by the number of seconds we are looking into the future) we were able to find the future predicted location of the user.

The formulas for calculating latitude and longitude using the Haversine derivative are the following:

$$lat2 = \arcsin(\sin(lat1)*\cos(d/R) + \cos(lat1)*\sin(d/R)*\cos(\theta))$$
$$lon2 = lon1 + \operatorname{atan2}(\sin(\theta)*\sin(d/R)*\cos(lat1), \cos(d/R) - \sin(lat1)*\sin(lat2))$$

where,

- lat1, lon1 are the latitude and longitude positions of user's current position

- lat2, lon2 are the latitude and longitude positions of user's future/predicted position

- θ is the bearing (in radians, clockwise from north);

- d/R is the angular distance (in radians), where d is the distance travelled and R is the earth's radius

V. IMPLEMENTATION:

We implemented an Android application to put our design to practice. This application is developed for android operating systems from version 2.3.3 and later. We utilize the Google Maps API [6] to provide our map interface for plotting locations and search results. We use the Google Places API [7] to act as our location-based search engine in the cloud.

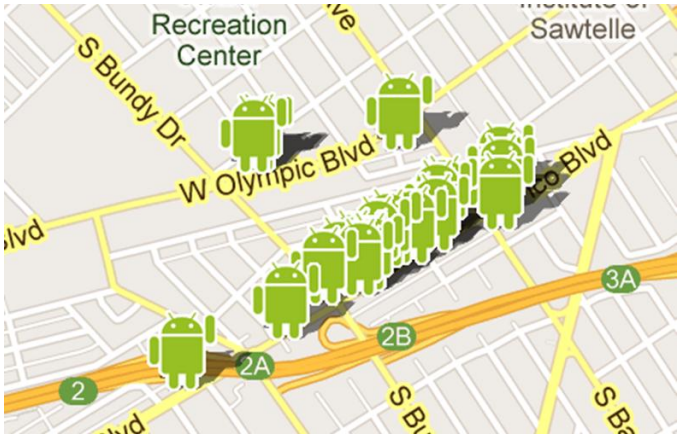
A. Google Maps API

The Google Maps API is native to Android. It is available in the Android SDK. To plot objects onto the map interface, we implemented three overlays.

- Current Location,
- Future Location and
- Search Results.

The current location overlay receives location updates from the GPS sensor and plots the location on the map interface in real time. The future location overlay update overhears these updates, calculates the future location and plots that location on the map interface in real-time as well. The search results overlay plots search result locations and information about each location on the same map interface whenever the user performs a search. Figure III shows the how search results are displayed for predicted future location as a result of a search query.

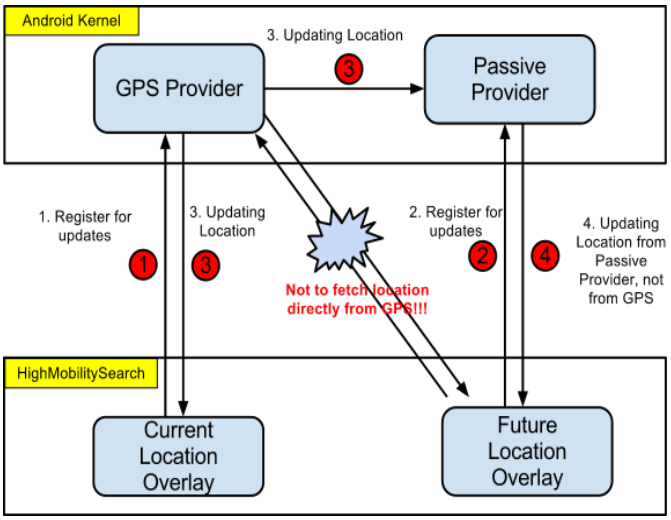
FIGURE III. MOVEMENT OF TRAFFICE IN A FREEWAY COMARAED WITH LOCAL ROADWAYS



B. Providers

Early on in our implementation phase, we realized that multiple overlays do not need to query the GPS provider. Querying the GPS provider using a single overlay is sufficient. The other overlays just need to register updates with a passive location provider. This passive provider overhears updates being passed by the GPS provider to the requesting overlay. In our case, the current location overlay registers for updates from the GPS provider. The future location overlay, on the other hand, registers for updates from a passive provider. In our case, we were able to cut down the number of requests for location updates to the GPS provider by half. Figure IV illustrates this optimization:

FIGURE IV. USE OF PASSIVE PROVIDER FOR ENERGY EFFICIENCY



C. Google Places API

Google Places API works by taking a HTTP request as input and sending a list of JSON or XML objects as output to the mobile device. In our implementation, we created a string to hold the HTTP request and send it to the Google Places API. In return our application will receive the list of JSON objects. We chose JSON because it has an advantage over XML in that the response is lightweight; parsing the result would be trivial in Java since the format is native in our environment. Once we received the JSON list of objects, we simply iterate over the list and over each individual values including latitude, longitude, and description, etc. By recording the geographical coordinates of each JSON object we then plot them on Google Maps and provide information of each interest points with the description value stored in JSON.

D. Challenges

1) Implementing within the search engine

One of the challenges we faced in designing such a "mobility-aware" search engine was the inability to implement this awareness from within the search engine itself. Instead, we were only left with designing a solution from within areas of the search process that we could control. Therefore, we focused on designing such a system by improving the context of the search we provide to the search engine. To provide this context of mobility, we designed and developed a system that would predict the user's future location based on a history of his/her direction and speed of travel.

2) Signal Loss and GPS fluctuations

Another challenge we faced was in the implementation of our design. We experienced frequent signal loss and fluctuations from the GPS sensor as we polled it constantly for the user's current location. These periods of loss and fluctuation wreaked havoc on our calculations as we tried to predict the future location of the user. To gracefully account for these periods, we developed a technique to weed them out of our calculations that is comprised of a thresholding scheme based on a moving average of the user's speed. Every time a new location update through, we calculate the speed the user would need to travel to get from the previous location to this current location. If the speed it takes to get from the previous location to the current location is in excess of 100 miles per hour and/or is drastically different from the moving average of the user's speed, the current location is deemed impossible and weeded out of our calculations.

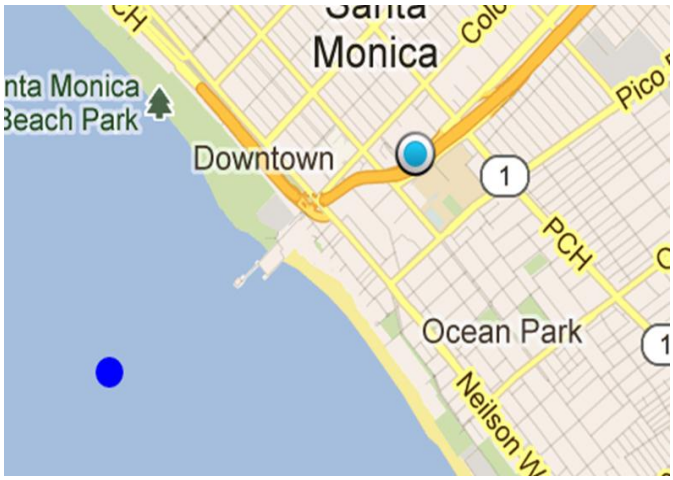
3) Drastic change of user's travel direction

Yet another challenge we faced in our implementation was how to gracefully handle drastic changes in a user's vector of travel. Since our design revolves around the use of the user's vector history to make a more accurate prediction of his/her future location, these drastic changes initially caused our predicted point to lag in the reflection of this change. To overcome this challenge, we were able to develop a technique involving vector summation to improve the accuracy of the predicted point while lowering the window of history needed to calculate it. We have explained this technique in our implementation section as well.

1) Future position pointing to impossible co-ordiante

Another interesting challenge we came across during our test run was the predicted future point being calculated in the ocean. As we were driving at the a good speed in the same direction on a freeway that goes all the way till Pacific ocean, our future point gets calculated in the ocean. This is show in Figure V. One approach that we could use to overcome this problem is to use Reverse Geo-Coding. We can make use of Latitude and Longitude co-ordinates of the future point and pass it to Google API for reverse geo-coding which would give us the address of the location.

FIGURE V. IMPOSSIBLE FUTURE POSITION



The attributes that could be used check for a proper/valid co-ordinate are Accuracy and Status returned. A threshold could be set on accuracy parameter to ensure that the co-ordinate is valid. If the accuracy falls below the threshold or if the 'Status returned' gives an 'Unknown location' or a failed result, the distance between current position and the future position shall be reduced by employing a recursive back-off algorithm. If the future predicted point falls in the ocean again, the distance between the current and the future position is recursively reduced till the predicted point becomes a valid co-ordinate. The base case assumption for this approach to work is that the current location of the user should point to a valid co-ordinate. Suppose the user is on a ship due to which the user's current location maps to an invalid co-ordinate then this approach would not be applicable.

VI. EVALUATION

To evaluate the design and implementation of our Android application, we decided to get out on the road and test its performance, reliability and usability in real driving scenarios. We evaluated our application for two hours as we drove from Los Angeles to Santa Monica and back. Throughout the testing duration, we drove on freeways and surface streets, performing maneuvers that may occur in normal day-to-day driving scenarios. Some of these maneuvers include U-turns, sharp forty-five degree and ninety degree turns, abrupt stops to yield to traffic and stoplights.

A. Performance

High Mobility computes fast yet it's able to do so without much battery consumption. It is shown in the evaluation log that, in most cases, High Mobility finishes calculating the future location in zero milliseconds. With such computation being calculated every second, our application was still able to consume only three percent of battery on Droid Razr after the two-hour drive. Not only does High Mobility computes fast, but it also recovers fast. The time it takes to recover from a large change in direction is about five seconds only.

B. Reliability

Our evaluation proved our application to be reliable in predicting accurate prediction points with most scenarios. On the freeway without traffic, our application was able to pick up a prediction point that is closed to the freeway. While in traffic, our application accommodates its new situation and thereby output a prediction point relevantly close to the current location.

On local, our application loses not its ability to predict the right location. At complete stop, the distance between the future location and the current location will decrease gradually until the prediction point is on top of the current location or when the mobile user starts moving again. At regular or sharp turns, the prediction point will move in an arc in relation to the direction of the turn and be fixed in the direction where the user is moving toward. Same result will be obtained when the user experiences a U-turn, but with more time would our application need to calibrate its new prediction point.

C. *Usability*

High Mobility is simple and user-friendly. No more than inputting their preferences would the users need to do. Upon each search query, the users can simply click on any desired interest point to view its description.

D. *Log Analysis*

We found some interesting observation from the evaluation log regarding the following parameters: the time between the current location and the previous location, the time to calculate the prediction point, and the impossible speed ratio, which ratio describes the percentage of disregarded locations.

1) Time between locations -> fluctuates inconsistent not always available

The time between each pair of consecutive queried location points are periodically recorded in our log. Originally, we intended that High Mobility would pull the GPS sensor location every second. This expectation failed once we spot that the time between each location retrieval is not the same. To resolve the fluctuation of our parameter, we investigated and observed that the GPS sensor location has no ready coordinates to give periodically, or every one second precisely. Given its nature in yielding locations in arbitrary time, we conclude the fluctuation to be normal due to the GPS.

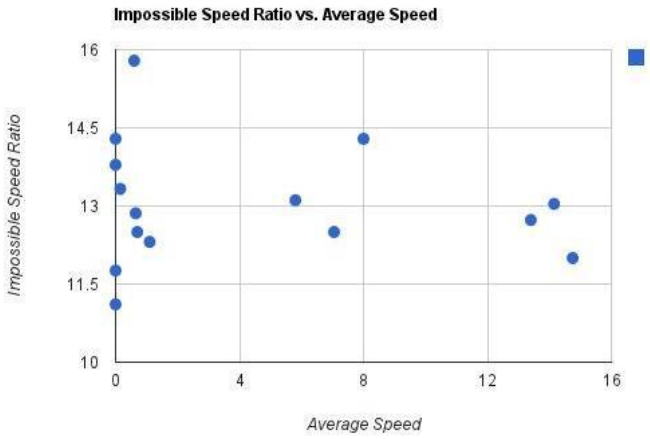
2) Time to calculate future location -> shows 1 milliseconds sometimes

Another interesting observation lies in the time to compute the future location. Although High Mobility computes the future location in sub milliseconds, it is shown, though rarely, in the log that at some circumstances it's computed in one millisecond. Since the impact by a delay of one millisecond is negligible, further analysis is apparently meaningless.

3) Impossible speed ratio -> percent of discarded points

The impossible speed ratio, ranging from ten percent to fifteen percent, is the percentage of location disregarded. Since the impossible speed ratio weed out location with a threshold set on average speed of the current user, we expected the two of the parameters would have a correlation. However, with the plot of impossible speed ratio vs. average speed, it showed that there exists no correlation.

FIGURE VI. IMPOSSIBLE SPEED RATIO VS AVERAGE SPEED



VII. CONCLUSION

Existing approaches in mobile search are not suitable in situations where the user is highly-mobile. Today, location-based search engines do not take into consideration a user's mobility when responding to requests. The user is assumed to be static for the time period the request is being issue and fulfilled. When the user is mobile, this assumption causes a large portion of the results returned by these search engines to become irrelevant to the user. Most of these results are for places that the user has already passed or is now far away from. For these reasons, we have argued that future search engines should be "mobility-aware."

The goal of our design was to build a mobile application that provides better search results to the user in highly mobile situations. We have proposed, designed and implemented a mobile application that improves the search results obtained by current location-based search engines by providing specific information to the cloud during the search query phase. With today's location-based search engines, results are usually returned within a certain radius of the user's current location. By returning search results tailored to the user's predicted future location, we have produced results that are more relevant to the user in highly mobile situations.

We have observed through our evaluation that our future location prediction of the user is overall accurate, the future location point can be calculated in sub-millisecond time and our application can recover from drastic changes in direction in about five seconds. The predicted future location is updated at the same rate as the updates we receive from the GPS sensor for the user's current location, allowing us to provide the most up to date information as possible. In terms of energy efficiency, our application consumes about 3% of the battery on a Droid Razr running Android OS 2.3.6. On the same phone, Google Maps consumes about 5.5%. In addition, we have researched and proposed methods of improving the energy efficiency of our current implementation. Overall, we argue that our method provides results that are more relevant to the user in highly mobile situations in an efficient, user-friendly manner.

VIII. REFERENCES

[1] Chunyi Peng, "Shape Search on the Go," University of California, Los Angeles, Computer Science Department.

[2] Nicholas D. Lane, Dimitrios Lymberopoulos, Feng Zhao and Andrew T. Campbell, "Hapori: Context-based Local Search for Mobile Phones using Community Behavioral Modeling and Similarity," in Ubicomp 2010.

[3] Emmanouil Koukoumidis, Dimitrios Lymberopoulos, Jie Liu, and Doug Burger, "Improving Mobile Search User Experience with SONGO", Microsoft Research, February 2010.

[4] Eemil Lagerspetz and Sasu Tarkoma, "Mobile Search and the Cloud: The Benefits of Offloading" in PERCOM 2011.

[5] Hoang T. Dinh, Chonho Lee, Dusit Niyato and Ping Wang, "A Survey of Mobile Cloud Computing: Architecture, Applications, and Approaches" in ICIS 2011.

[6] http://developer.android.com/guide/topics/location/index.html

[7] https://developers.google.com/maps/documentation/places/

Appendix

I. Response to Comments And Concerns

Review Comment #1: Project Oriented more towards predicting the location than improving the search.

Response: We understand that in the course of designing and developing our mobile application to improve search results obtained from a cloud-based search engine, we have ended up with a result that can be considered more of a location prediction problem than a search problem.

In our report, we have argued that our technique is a viable option to improve existing cloud-based search engines by making them sensitive to a user's mobility. We believe future location-based search engines will acquire additional context about the mobility of the user as part of its normal search routine through techniques similar to ones proposed by HAPORI [2]. This techniques followed by Hapori is also explained in Section III-B and in Appendix III-B. The cloud would take this context into consideration before returning the results to the user and thus improving the search results. We would also like to indicate that the prediction of the user's future location will be a pivotal part of the cloud's calculations.

Review Comment #2: Handling Signal loss and insufficient optimal bandwidth

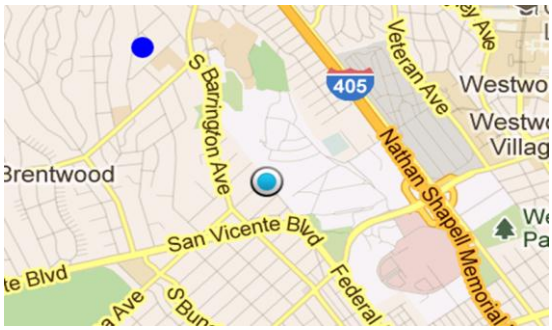
Response: We have addressed this issue of Signal loss and GPS fluctuations in Section V-D-2 with our approach in resolving the issue.

II. Evaluation

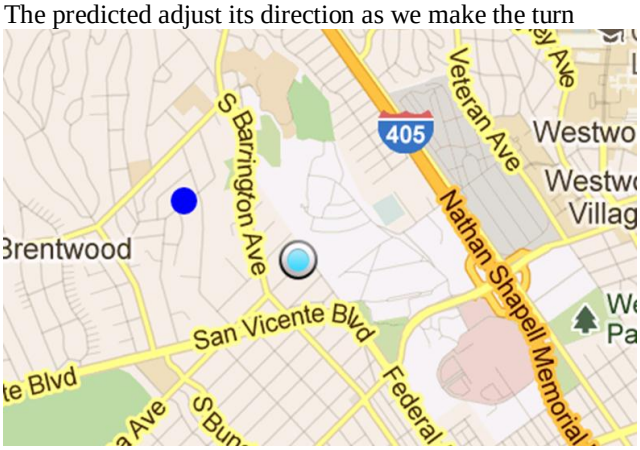
We performed our evaluation by driving from Westwood to Santa Monica. We ran the following set of experiments.

- Freeway Driving
  - Smooth driving
  - Congestion due to Traffic
- Local Driving
  - Test on traffic lights
  - Test on left/right/U turns

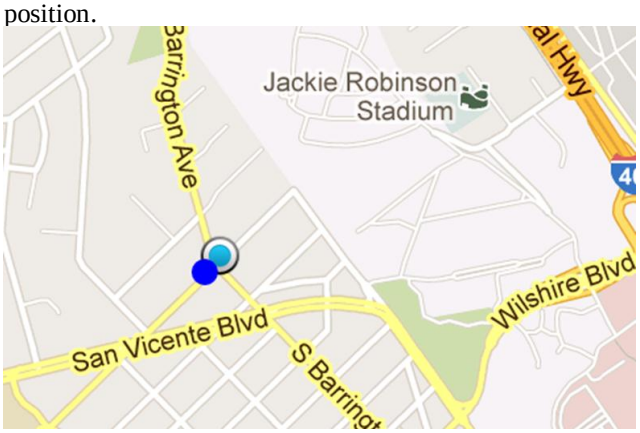
Before making the left turn as we were travelling straight, the future position is in direction of travel.



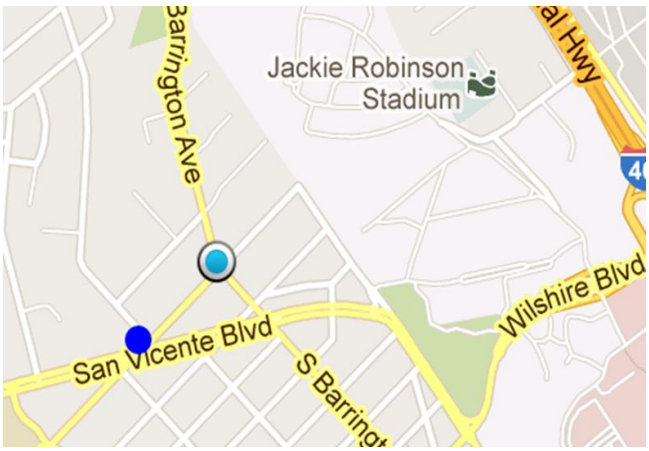
The predicted adjust its direction as we make the turn



Future position gets adjusted correctly as the user starts moving straight. In the below scenario due to traffic congestion the predicted position is near the current position.

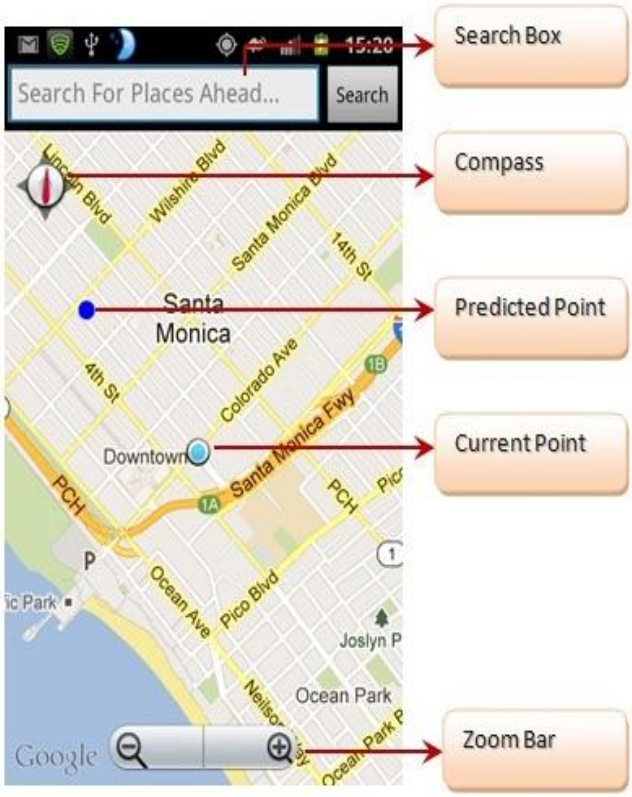


The predicted point corrects itself once the user picks up speed and it correctly points in the user's direction of travel.



III. Application Screenshot

The following figure shows a complete screenshot of our android application with current as well as future predicted point along with the search box where user can enter their search query.



IV. Surveys of Related Work

A. *"Shape Search on the Go" Chunyi Peng*

In this paper, Chunyi discusses the capabilities and limitations of mobile devices with regard to mobile search. She summarizes key guidelines that can be used to address technical issues that arise, and she gives an example of how a traffic search application may be implemented with respect to these key guidelines. Chunyi points out that mobile internet usage is predicted to overtake desktop internet usage in 2014, which will make mobile search a critical component of the overall search volume in the future. She goes on to add that the purpose of mobile search is different from conventional desktop search. Mobile search results are expected to be near the user's current location and arrive as soon as possible to the phone. Personal and social information on mobile phones can provide to personalized search, while the phone's physical features like the camera, GPS and accelerometers can provide user behavior. This information can be combined to form a specific mobile search.

Chunyi explains that one complete search operation is relevant to four technical aspects: intention, interaction, network and search. A user's intention can be gathered from his/her location, time, activity and social interactions. A user's interaction gives way to the user's feeling, and implies that a good mobile search should be easy to use. The third technical aspect, network, is based on how the search adapts to available network conditions. Chunyi outlines four-subgoals which are considered unique requirements in mobile search. The subgoals are as follows: be Applicable, be precise, be prompt, be efficient and be easy. She explains that a good user experience corresponds to accurate results, rapid response, easiness of use and low cost. Also, the collaborative behaviors of mobile users can enhance multi-user search experience by social means.

Chunyi lists three kinds of approaches to enhance mobile network connections. The first approach is to optimize the existing network operations. The second approach is to utilize secondary network channels like short message service (SMS). The third approach is to re-build the current network architecture. Optimizations like caching popular queries to reduce the number of connections to remote servers could decrease latency and traffic. Chunyi believes more needs to be done to utilize mobile opportunities to improve search performance. She discusses three approaches. The first approach is to develop a context-aware ranking of results. She points to a paper that adds the concept of context by developing a 4W (Who, what, where and when) model to represent the context structure needed for a mobile search. The second approach is to employ a vertical search. Chunyi explains that about 40% of all mobile search queries are to find places based on the user's current location. The third approach is to utilize and exploit the "wisdom of phones and crowds."

To conclude her survey of mobile search, Chunyi proposes a traffic search application that could be implemented in the future. In this application, a specific search is chosen instead of a general search, since traffic search has a clear and explicit purpose. She recommends that traffic information be provided by authenticated sources like Sigalert. Chunyi proposes the deployment of proxies to further enhance search and network performance, while using various network communication schemes to deliver the information. Last, she recommends adopting optimized operations specific to traffic search like adjusting the GPS update frequency to save power when the user is moving slow. Overall, Chunyi believes that making full use of specific user demands, rich sensing capability, pervasive context, and community collaboration on mobile platforms can contribute to a better mobile search experience.

B. *"Hapori: Context-based Local Search for Mobile Phones using Community Behavioral Modeling and Similarity." Lane et al.*

In this paper, the authors propose a novel way to display personalized search results by analyzing environmental information combined with user specific information. This paper talks about data mining, information preserving and distance metric learning in-order to create multi-dimensional models from local search logs of a mobile phone. It considers more contextual information that could be obtained from mobile phones for refining the search results.

The authors model Point of Interest (POI) preferences for every mobile search user. These preferences are combined based on the similarity between different users. They also analyze the search logs of users obtained from popular search engine like Bing over a period of 6 months to identify and quantify the search queries. The mobile sensors could be used to obtain details such as, if the user is inside or outside a building, with a group of friends or alone, walking or driving etc., in-order to customize search results.

Just the context such as weather conditions, time of the day is insufficient, as different people sharing same context could select different results. So a POI preference model is built to reflect (i) contextual information, (ii) personal preferences and behavior of a user and (iii) group of people with whom they have similarities. The first phase of the project involves and off-line training model which builds the POI preference model over a set of stages. This preference model is used by the online search to respond to search queries. The search query is augmented with contextual information taken from mobile sensors as well as historical information about the user which is built over time by Hapori.

Thus the proposed Hapori search technique meets the diverse needs of different people by taking into account their context, behavioral profile and behavioral similarities with other local search users.

C. "Improving Mobile Search User Experience with SONGO." Microsoft Reserach.

In this paper, the authors propose a cache search system for mobile, which cache is generated by mining the most popular queries and link from the mobile search logs. Since popular queries and links change daily, the cache will be updated in a day-to-day basis in order to keep up with the most up-to-date popular information for the mobile device.

The design takes advantage of the increasing memory resources on mobile devices. From this, a cache is built on two observation from data analysis. First, there is a set of queries that is frequently submitted by all the mobiles. Second, each mobile users send repeated queries over time. The cache will store popular queries and repeated queries to enable mobile users perform local searches.

In the web search logs, the term volume represents the number of times in the search logs that the specific link was selected after entering the query. The higher the volume, the more popular it's query is. The cache decides which entry to store from the search logs, which is straightforward because the cache will go through the list from highest volume to the lowest volume. The amount of popular link is a rather more complicated process for the cache to be constructed or updated. Because of this, the cache will specify a memory threshold to refrain itself from allocating too much memory.

In order to access the cache, a hash table is needed. SONGO develops a hash table within that stives for minimal usage of memory in searching and fast performance under ranking look-ups, etc. The hash table have three main functions: to quickly identify if query exists in cache, to point at the hit query, and to provide ranking score. Over time, SONGO updates both the hash table and the cache to make sure that the user will get the right result with the correct ranking scores.

Personalization also plays an important part in cache searching. The phone can learn from the user's past activities (number of clicks and time of clicks of queries) to constantly update the value of ranking scores. This then results in a better chance of presenting contents that fits the preference of each particular user. All in all, the cache search system provides a faster performance with offline query service and a method to save more energy with the trade-off of using a slight percentage of memory in the mobile phones.