On My Way: Optimizing Driving Routes for Crowdsourcing Applications

Xueyuan Song
Rutgers University
Email: xueyuansong@rutgers.edu

Fengpeng Yuan
Rutgers University
Email:fengpeng@cs.rutgers.edu

Janne Lindqvist
Rutgers University
Email: janne@winlab.rutgers.edu

***Abstract*—Conventionally, the route recommendation given by GPS has been considered as the optimal route search problem only between two locations - origin and destination [1]. However, in reality, restrictions may need to be considered. The restrictions could be several intermediate destinations belonging to user's specified types which must be visited before reaching the final destination. For example, a user wants to visit a bank and a gas station before arriving at his work place, though many choices may be available along his route to company, only one place from each type is to be chosen. This paper tries to bring this issue to light, and provides a solution to the problem. Our work makes three major contributions: first, we proposed an strategy that helps search nearby places. Second, an algorithm is proposed to help get the optimal route through multiple intermediate waypoints. Last but not least, an Android application is designed and implemented which gives the optimal route recommendation by users' specified place types.**

I. Introduction

With the development of the Intelligent Transportation Systems (ITS) technologies, a vast amount of real time information is available for the travelers. Since the ITS technology powers the automotive navigation system, people rely on automotive navigation system more than ever before. One of its main functions is to give the optimal route recommendation to the destination using the real time traffic information. In the system, a GPS) [2] navigation device is used to acquire position data and help to locate the user on a road in the unit's map database. Using the road database, the unit can give directions to other locations along roads also in its database.

The GPS navigation has become one of the most popular applications for people. Almost all the Android devices come with a turn-by-turn navigation system, as well as IPhone. World sales of smartphones to end users totaled 968 million units in 2013, an increase of 42.3% from 2012. Moreover, the navigation apps based on the smartphone can retrieve real-time traffic information from internet to plan a faster route by avoiding roads under construction.

Android Market and Apple Store provide a large number of navigation apps, such as Navigon, Google Maps, Sygic, Waze, etc. However, the route recommendations given by these apps, as well as GPS unit, have only considered the optimal route search problem between two locations - origin and destination [1]. Although Google Maps provide a way to calculate a route with specified waypoints, this can be only used via websites and instead of recommended waypoints, those waypoints need to be specified by users and input one by one manually. On the mobile platform, there is only one start and end point setting each time. In real cases, several constraints may need to be applied when providing the route recommendations. The constraints could be several intermediate destinations which must be visited before reaching the final destination, some place must be visited before another one. The case of multiple waypoints destinations is much more complex than a route search only between two locations. But it is common for users to get to the destination through multiple waypoints.

First of all, to find the optimal route to the final destination, the visit oder of the waypoints needs to be optimized. The problem may look similar to the traveling salesman problem. However, in fact, the final destination must be visited at the end, and the visit order needs to be taken into account. In addition, the waypoints are considered to be selected from a set of possible choices. For example, a user wants to visit places which belong to several general types such as a bank and a gas station before arriving destination, occasionally many choices may be available, where only one place from each type is to be chosen. Moreover, the intermediate waypoints should be along with or near to the route to the destination in order to further optimize the route distance and save time. Necessarily, the driving route which goes through the optimal waypoints needs to be calculated. The shortest path calculation should be based on the actual real time driving route, and the routes information should be transformed to a human readable route in order to navigate.

In this paper, we have made three main contributions: 1) a strategy to optimize the results when searching nearby places is proposed, 2) an algorithm to find the optimal routes is proposed, and 3) we have designed and implemented an Android navigation app, which has the following advantages to find the optimal route through multiple waypoints. First, it recommends the nearby waypoints which belong to user's specified general types. Second, it recommends optimal routes and route information on the map, including time duration, total distance, etc. Third, the app can notify users about the recommended routes when they are leaving home or work place. Furthermore, the strategy and algorithm were extensively evaluated.

II. Related Work

Most smartphones have some types of navigation system [3] or at least have access to a market where they can download one. One of its main functions is to provide the optimal route to destination. The optimal route search problem

has been extensively studied and many algorithms have been proposed.

Given a graph with a set of vertices (or nodes) and edges, where each edge has a weight, single source shortest path algorithms, such as Dijkstra's [4] and Bellman-Ford's [5] [6] algorithms, can compute the shortest path between a single node to every other node in the graph. In order to compute all pairs shortest path, algorithms, such as Johnson's [7], [**?**] and Floyd-Warshall's [8] can be applied. Since these algorithms are all assuming that the weights on the edges are static, they must be re-executed whenever the weight of an edge changes.

The Dijkstra shortest path [9] and A* [10] algorithms are usually used to plan the route in the automotive navigation system. As compared to Dijkstra, A* achieves better computation time by using heuristic functions. However, these traditional route planning algorithms [11] [12] use static information such as the speed limit, instead of the real-time speed, of each road segment to calculate the route to the destination, which usually underestimates the actual traveling time and fuel consumption in turn.

To calculate the optimal route through multiple waypoints, Maruyama et. al. [13] proposed a method to find the optimal tour plan using genetic algorithm. Their work focused on tour planning and users' preferences for the destinations. In their work, they worked on determining the optimal tour plan considering restrictions, so some destinations may be left out as a result. Kanoh et. al [14] proposed a GA-based route search algorithm to find the route considering the unspecified waypoints destination. In their method, however, it is not guaranteed that the intermediate destinations are always included in the route. Manoj et. al [1] proposed a method to optimize traveling times and the order of visiting multiple waypoints in order to determine the optimal route from origin to destination via multiple waypoints. An efficient Ran-Discrete Version (RasID-D) has been proposed by Hirasawa et. al [15] to solve the discrete optimization problems. In their paper, they mainly applied RasID-D to optimize the visiting order of the intermediate destinations. However, the paper did not mention how to select waypoints from a set of possible choices and how to get the real time routes between those waypoints. Lin et. al realized a travel route intelligent navigation system based on WEBGIS [16]. In their paper, they did research on the tourism route planning based on the traveler's types, preferences, etc. The basic purpose was to produce a satisfied tourism route planning for travelers based on some restrictions. Jeffrey and Muhammad [17] presented a fastest path algorithm that contains multiple unique destinations, which is a specialization of the Traveling Salesman Problem (TSP) with intermediate cities. In their paper, they proposed the algorithm which can be used with Intelligent Transportation System (ITS) applications for determining the fastest route to travel to a set of destinations, such as required by delivery companies.

To design an application, Chang et. al [18] designed and implemented an Android-based navigation app, which can help reduce the traveling time, fuel consumption and emitted carbon dioxide. In their paper, they were focusing on using vehicular ad-hoc network (VANET)-based A*(VBA*) [19] route planning algorithm which is proposed to calculate the route which takes shortest traveling time or has least oil consumption, depending on two real-time traffic information sources. A GPS navigation app was implemented on the Android platform to realize the VBA* route planning algorithm. However, the application was only designed for proving VBA* achieves significant reductions on both the average traveling time and oil consumption of the planned route, as compared to traditional route planning algorithms. In Mark's [20] work, they presented the algorithm for selecting paths for traveling by road. The algorithm is based on the principle that 'ease of description' is an important consideration in route selection. The route description is represented by a frame, and the route description cost is approximated by the number of slots in that frame. Then total route cost is the sum of the route length and weighted route description cost.

III. Application Design

To design an application on mobile platform, the proposed strategy should consist of three steps: first, waypoints should be recommended which belong to user's specified types, then a real time driving route through specified multiple waypoints need to be optimized. Third, the route information is transformed to user readable routes and shown on the map.

In order to find the waypoints belong to users' specified types, the local businesses and services need to be collected and used for further calculation. The Google Places API provides place information on a variety of categories, such as: establishments, prominent points of interest, geographic locations, and etc. Search for places can be either by proximity or a text string. We chose to use the Google Places API, for the reason that it can reach millions of business quickly and for free.

In real situations, the route needs to be calculated efficiently in real time. It seems impossible to collect all the actual routes information, fortunately, Google Direction API provides a service that calculates directions between locations. The direction search could be several modes of transportation, include transit, driving, walking or cycling. Directions may specify origins, destinations and waypoints. Moreover, the Direction API was designed to return an optimal route through multiple waypoints. So in order to get the real time driving route, Google Direction API will be chosen in our work.

Traditional route recommendations only consider about two locations - origin and destination. However, in reality, this is not the case. People would like to visit some places before arriving the destination. Basically, to implement an application with this purpose, a strategy is needed to help search nearby places and an algorithm is required to help get the real time optimal routes through multiple waypoints. In this section, we firstly describe a scenario where an application which can recommend both intermediate waypoint and optimal route going through them is needed. Then we propose a method that can search nearby interested places of specific types and an algorithm that can deal with the ordering of the visit and find the optimal path.

A. Application Scenario

Bob is leaving home for work in the morning. His car is running out of gas, so he needs to find a gas station, and he needs to get some medicine because of his allergy. As usual, he wants to grab a cup of coffee before going to work. With a

normal GPS system he would have to search for coffee shop, gasoline station, and drug store and input their addresses into the GPS. This would be time-consuming and inconvenient. With our app, Bob only need input his work address and the general type of places he would like to go. The app can recommend a optimal route that satisfies all his requirements.

B. Find Nearby Interested Places

We could have hundreds of options to visit all the interested places before arriving the destination. To save the time and cost, it's better that we find all the interested places along the route from origin to the destination, and visit them sequentially. However, we cannot always find all the interested points along the routes. Therefore, we need to find the nearby interested places which belong to users? specified place type when driving on some route. To realize this, we mainly leverage the google Places API [21], which is designed for this purpose. When using the Place API, we need to provide the location where you want to search for the specified place type. Hence, we firstly need to find the locations where we can do the place searching. Due to the API calls limit (1000 calls/day for free), we need to select the locations properly. Certainly, we can use trial-and-error method to find out these locations, but this is exhaustive. Since we know our origin and destination, we could utilize the Google Map Directions API [22] to get the directions from origin to destination. It can return us the directions which are needed to get strictly from point A to point B. We can use intermediate points from the returned directions as the search points.

Google directions responses [22] contain two root elements: status which contains metadata on the request and routes which contains an array of routes from the origin to the destination. Routes consist of nested Legs and Steps. Each element in the legs array specifies a single leg of the journey from the origin to the destination in the calculated route. Each element in the steps array defines a single step of the calculated directions. A step is the most atomic unit of a direction's route, containing a single step describing a specific, single instruction on the journey. The "end_location" field of "step" contains the location of the last point of this step, which is usually a turn point. The strategy to find the closest waypoints is to search at these turn points for the specified place type. The search would only occur if the distance between the previous and current turn was farther than a threshold, so there would not be redundant places to route to.

The places returned by Place API could be organized by rating or distance. We chose distance because we're looking for the most efficient route rather than most pleasant place. Three nearest results of each place search will be saved for further use. This would give a total of 3*(# of types)*(# of turns - # of turns within threshold of last turn) different locations, it is plenty and proved to be more than enough in our experiments.

C. Find Possible Shortest Path

In order to find the optimal route to the final destination, the optimal waypoints need to be selected from the places list returned from the previous step. Because for each place type, there could be many places available after nearby searching. The goal is to select one place from each type, and make them a set of waypoints, which should be a possible waypoints combination. For each waypoints combination, we calculate the shortest path through the waypoints and then select the best three results. Furthermore, all the waypoints in each combination will be retrieved and sent for requesting directions from Google Directions API. Finally, the recommended optimal routes of those combinations are shown on the map.

When calculating the shortest path for all the waypoints combinations, basically there are two strategies can be used to select the best three optimal routes.

The first strategy is calling Google Direction API for all the waypoints combinations, since Direction API automatically optimizes the order of visiting waypoints, it will offload the optimization work from us. Beside, Google Direction API takes traffic condition into consideration. This is a simple way to get the optimal routes. However, there are lots of shortcomings when using this strategy. First, calculating directions is a time and resource intensive task, it takes too much time to get response when calling Google Direction API and if the number of waypoint combinations is too large (more than 200), it can cause the app to crash, because the execution takes too much time. Second, when calling the Google Direction API at such a high frequency, it may lose data and get no response from the remote server. Third, The Directions API has limited 2,500 requests per day, each time calling API counts as a single request. Each request for driving may contain up to 8 intermediate waypoints. If every possible combination needs to call the API, as a result, it will be quite expensive. For example, we suppose that user selects 3 general types of waypoints and each type can return 5 places, the number of possible combinations will be 5^3, which means number of API requests would be 125, we can only call 20 times per day for free.

The second strategy is to reduce the Google API calls. Besides the origin and destination need to be visited firstly and lastly, we have to visit all the waypoints once. The problem becomes finding the shortest path from origin to destination via some "must pass" waypoints. This problem is a variant of well-known traveling salesman problem, which is NP hard. To solve it, we propose a heuristic algorithm. First of all, we connect all intermediate waypoints(including origin and destination)with each other as a graph, the weight of each edge is the Bird-View(Euclidian) distance between two endpoints. Next, we leverage the well-known shortest path algorithm, e.g. Floyd-Warshall algorithm, to get the all pairs shortest paths. Finally, we construct a new graph with the origin, destination and all the shortest path we calculated from the previous step, then permute all the path from origin to destination via specified waypoints to get the shortest path. In the process, since we must have origin and destination visited firstly and lastly, respectively, we can apply pruning during the permutation calculation, which can save lots of time, since complexity of the algorithm is exponential. If we limit the number of waypoints of each type, the running time of the algorithm can be improved. The pseudocode is shown in algorithm 1. The second strategy may not be intuitive, because there is no evidence to show the Bird-View distance is related to both actual driving distance and duration. Therefore, we select three waypoints combinations that yield the best three shortest paths to call Google Directions API. Nonetheless, as implemented

in the app, a great deal of experiments and tests showed that this strategy reduces the time and can return the optimal routes as first strategy. The results will be showed and discussed in following section.

Algorithm 1: Find possible shortest path

Input: origin, destination, all the waypoints that belong to user's specified type
Output: shortest path passes waypoints of user's specified type

n= number of all the waypoints including origin and destination ;
N=number of specified types;
// Find all pairs shortest paths, e.g. using Floyd-Warshall algorithm
for $i \leftarrow 1, n$ **do**
  for $j \leftarrow 1, n$ **do**
    $d[i][j] = \infty$
  end
end
for $k \leftarrow 1, n$ **do**
  for $i \leftarrow 1, n$ **do**
    for $j \leftarrow 1, n$ **do**
      $d[i][j] = \min(d[i][j], d[i][k] + d[k][j])$
    end
  end
end
// try all the permutations to find the shortest one
$shortest\_path = \infty$
foreach *permutation waypt_1, waypt_2, ..., waypt_N of the 'mustpass' waypoints* **do**
  $shortest\_path = \min(shortes\_path, d[origin][waypt\_1] + d[waypt\_1][waypt\_2] + \cdots + [waypt\_N][destination])$
end
return $shortest\_path$;

In addition, to calculate the route, we can specify the transportation mode which includes Driving, Walking, Bicycling and Transit, which will return different waypoints those are convenient to the user's current state.

After, we get the response from Google Direction API, we display the best three optimal routes on the map with the waypoint marked, as shown in Fig. 1.

IV. Experiments and Results

In this paper, Bird-View distance is used as the reference to find the optimal routes. The strategy will be evaluated by several experiments. In real situations, many factors have dramatic effects on the final results. The experiments are much more complex in real situations than theoretical analysis. There are two key questions to be answered in the experiments. The first question is why Bird-View distance can be used as a reference when selecting actual routes. Second is in which situations it is effective and when it becomes infeasible.

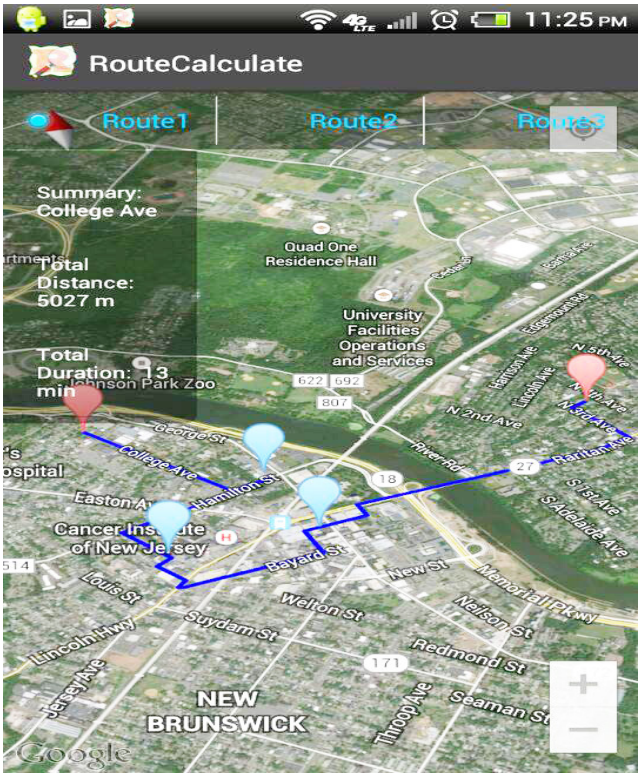

Fig. 1: On the top of the screenshot, a group of three radio buttons can be used for switching three recommended routes, below them there is a information board which is showing the corresponding routes information include main routes name, time duration, and distance.

A. Preliminary Experiment

In the first stage of the experiment, we selected three most common place types as testing types: gas station, restaurant and bank. Both origin and destination are chosen from ten different states all over US which cover both cities and rural areas. When running the application, for each pair of origin and destination, the bird view distance of all the possible intermediate waypoints combinations were calculated. The actual driving distance and time duration of all the waypoints combinations were also calculated. The average number of possible combinations for all the pairs is 104.7. One of the experiment results is shown in Fig.2, which is experimented with places in Florida. From the figure, we can see that the actual driving distance and driving time are increasing as the bird view distance increases.

B. Experiment on Effectiveness Validation

In the second stage, the experiment was designed to answer the question why the bird view distance could be a valid criterion for selecting actual route. There are lots of factors that should be considered when answering this question as mentioned above. To reduce the influence of different factors, the experiments was done in big cities where roads are criss-crossed.

Selecting the start and end point is very important when doing the experiments. The strategy we were using was searching the city on Google map, and randomly choosing the points as a start point and an end point. For rural area, the addresses

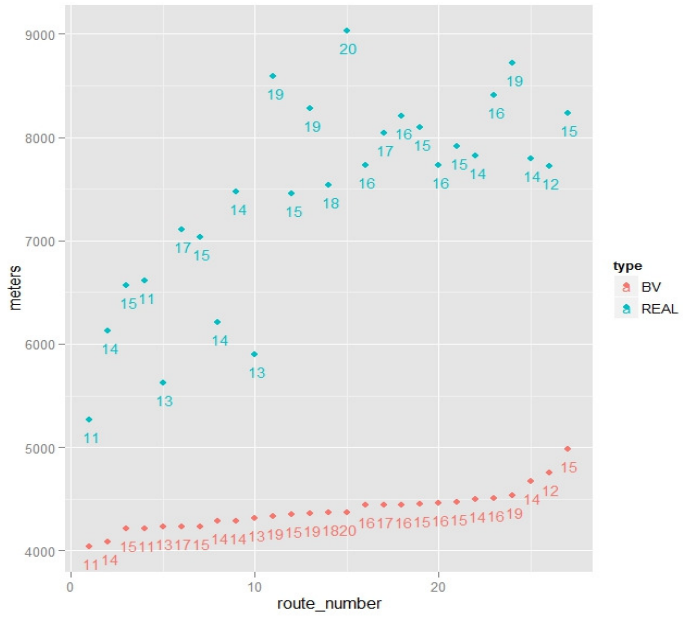

Fig. 2: x-axis represents the number of waypoints combinations, each number represents a possible waypoints combination. The combinations are sorted in an ascending order based on their bird view distance. Y-axis represents the distance. Both origin and destination are in Florida.

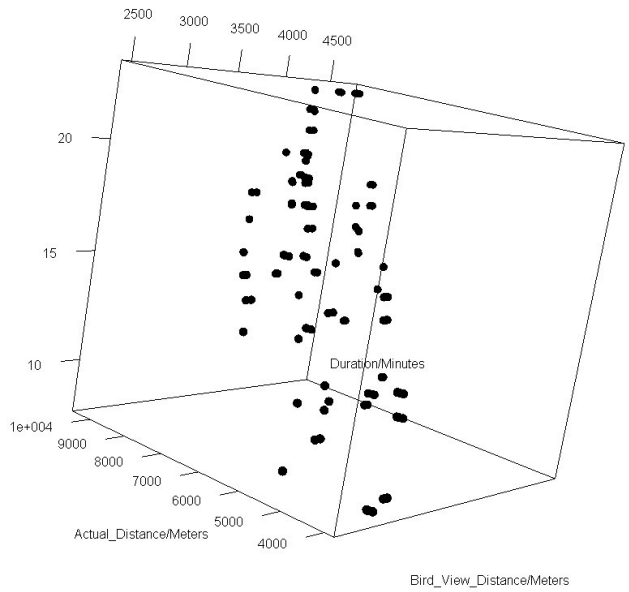

Fig. 3: Each point in the figure represents a possible waypoints combination for one pair of origin and destination. X-axis represents the bird-view distance, y-axis represents the actual route distance and z axis represents the driving time.

of state universities are searched as the start points and the nearby hotel as the end points.

In the 3D figures, points which are located in lower right corner are actually what we needed, because those points have less bird view distance, less actual routes distance and less time cost, which means when using bird view distance as a criteria to select actual routes, these points will be selected and should be effective for calling Google Direction API. In Fig.3, all points are distributed along the diagonal. If the points are uniformly distributed along the diagonal, it could be more accurate using bird view distance as the criteria, because the diagonal line means bird-view distance, actual route distance and time duration are proportional with each other.

From Fig.4, we can see that points are distributed in two corners. This situation usually happens when there is no intuitive route between two places which belong to same type.

With visualizing the relationship among bird view distance, actual driving distance and driving time, the points may have different distributions and patterns. But no matter what patterns they exhibit, there always exist points which have less bird view distance, less actual distance and less time cost. Such points can be used for calling Google API. To make the result more accurate, the first N (N<<total number of possible waypoints combination) of bird view distance can be also used for calling Google API, however, deciding the value of N becomes another question, because the numbers of waypoints combinations vary depending on the pair of start and end points. If the number of the waypoints combinations is too small, the value of N should be small, and vice versa.

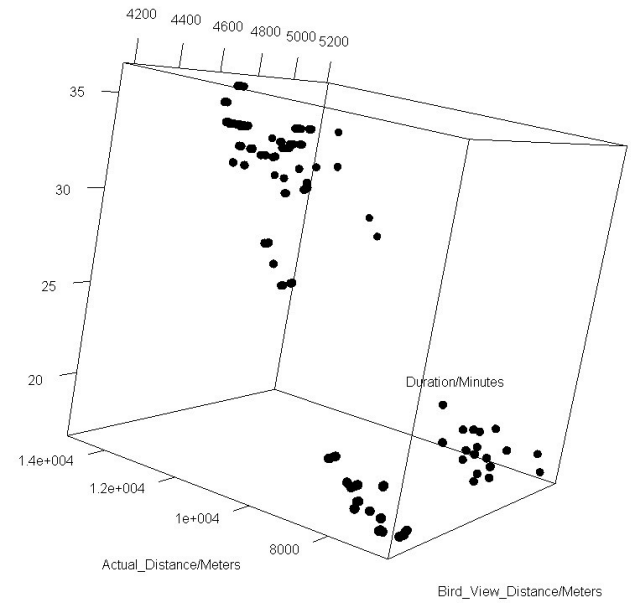

Fig. 4: When there is no intuitive route between two places which belong to same general type, the points are separated into two parts: the top-left corner and bottom right corner

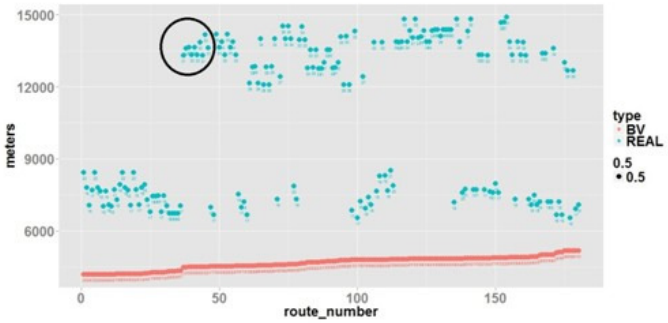

Fig. 5: The invalid points is circled in the graph, which means it may be invalid by using the strategy

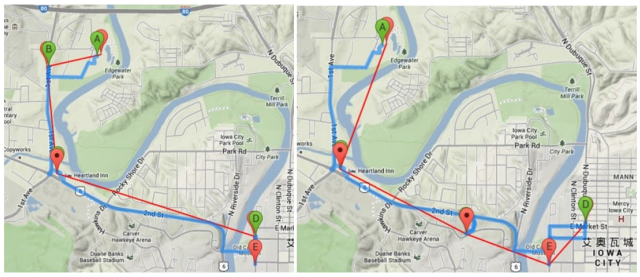

Fig. 6: The invalid points is circled in the graph, which means it may be invalid by using the strategy

C. When is Effective, When Is Not

Finding the invalid points may help to improve the strategy. In the third stage, another experiment was implemented to answer question in what situation this strategy is effective and when it is infeasible.

Ideally, the actual route distance should be increased with the increasing of bird view distance. However, due to the complexity of the topography, many factors will have an influence on the actual route distance. The experiment began with finding the points which are invalid using the proposed strategy and figuring out why it is invalid by comparing with valid point. The actual driving distances of some points are rapidly changed and then go back to normal level soon after several points. Such points can be regarded as the invalid points and the points which go back to the normal level can be regarded as valid points. For example, the point within the circle in Fig. 5 can be regarded as invalid points, because those points compared with the previous points, the distance has been rapidly changed and then after several points, the distance goes back to normal level. After selecting those invalid points, those points will be plotted and as it is mentioned before, each point represents for a possible waypoints combination. Hence, the optimal routes through the waypoints of this point will be drawn on the map which can show a more intuitive result.

Basically, the experiment was set with the origin and destination located in cities and rural areas. Both bird view distance and actual distance were calculated. Based on the visualized data and our data validation rules, we can infer the valid data and invalid data.

To make it more intuitive and specify each situation, an HTML and JavaScript application is designed to show the bird view and actual routes on maps. As shown in Fig 6, the actual distance in right figure is longer than the one in left, since the waypoints of two pictures are different but closed to each other. To go through all the waypoints to arrive B, in right figure, one needs to make a turn which produces more distance, because when calling Google Direction API, it is very likely to return a route which has light traffic and high speed limit but produce more distance.

V. Discussion

In the preliminary experiment, we selected three most common place types as testing types: gas station, restaurant and bank. After the evaluations, we found that some points are grouped by three. In order to better understand the waypoints in the combination, we have retrieved the API calling URLs. We found that the parameter “optimize” in the URL is already set to true, which means the routes returned by calling API are optimized.

An interesting finding is that with two waypoints fixed while changing the third one, the actual driving route distances are pretty close, usually these routes seem to be grouped together. That’s because some types such as restaurants will return the waypoints which are closed to each other. This usually happens because they may be located in the same shopping-mall or plaza.

Time duration has relationship with both bird-view distance and actual route distance, but the relationship between them is not intuitive. The results showed that some routes have longer actual distances but may cost less time. Different periods during the day, different speed limits and other factors may have a great influence on the driving time duration. Because the time duration is based on current traffic conditions, if the traffic is heavy on the route or the speed limit is slow on this route, it could take more time than the one with longer distance.

The best three points of bird-view distances could not always be the actual three shortest routes. Most results show only one or two best bird-view distances that exist in actual three shortest routes and sometimes even no one exists at all. What we want to propose here is that this method can be used as a reference to find the best three routes. The results showed bird-view distance is related with actual route distance, and the actual distance is related with driving duration. In Fig 2, with the increasing number of bird-view distance, the trend of actual distance is also rising. If drawing a trend line to show both of them, we can find that the slopes of these two trend lines are almost same. This usually happens in the city area. Since in the city area, roads are always highly structured and organized. Streets and avenues are crossed at most corners, which means, when you are driving in the city, you do not need to make detour to get to the destination. However, the route could not be always completely straight, so the actual driving route distance is still longer than bird-view route distance. To quantify the relationship between the bird-view distance, actual distance and driving time is complicated enough that beyond the scope of this paper.

In Fig. 3, the bird-view distances of these points are divided into two parts. This usually happens where there is

roadblock, e.g. river, between origin and destination. To arrive the destination, the user have to detour the roadblock, e.g. drive over the bridge.

Our application is currently taking into account all of the places on the route to the destination. The application works good while producing choices to go to different places. An application like this would help ease the normal hassle of figuring out where to go in order to not be late to work in the morning, find the optimal route or it could help the user discover their new favorite cafes and stores. The application can display the information of different routes which include a route summary, driving time duration and distance. This information can be used for navigation, route selection and further optimization. In current strategy, the best three optimal bird view routes will be sending to calling Google Directions API, it may get better result if more bird view optimal results is used to call Directions API. Moreover, calculating directions is a time and resource intensive task. So the optimization of calculation process is pending to implement in the future work.

VI. Conclusions

In this paper, we proposed an strategy for the application which helps find nearby places, and an algorithm to recommend efficient routes. Also, an application on mobile platform to find the optimal route through multiple intermediate waypoints was implemented. It can recommend three optimal routes while passing through multiple waypoints. At the moment, the application only returns recommended three optimal routes, but it can be easily extended to display more routes. We doubt that more than three are needed for users to choose from. Studies have shown that when users have too many choices they become overwhelmed and unhappy [23].

Waypoints and efficient routes are used every day by companies such as FedEx, UPS, DHL, and many more. Those companies use high end equipment to calculate their efficient routes. Getting similar functionality into a common user's phone is a great start for them.

A great deal of experiments have been done to evaluate the strategy and algorithm. The result shows that the best three recommended routes may not be always the top three shortest paths, however, the strategy still can be used to find good results. 80% of the experiment results showed at least one of the best three shortest driving routes will be selected by using Google Map. 90% of results showed at least one of the best three shortest time durations will be selected.

References

[1] M. K. Mainali, S. Mabu, X. Li, and K. Hirasawa, "Optimal route planning with restrictions for car navigation systems," in *Systems Man and Cybernetics (SMC), 2010 IEEE International Conference on*. IEEE, 2010, pp. 393–397.

[2] B. Hofmann-Wellenhof, H. Lichtenegger, and J. Collins, *Global positioning system: theory and practice*. Springer Science & Business Media, 2013.

[3] R. K. Ganti, N. Pham, H. Ahmadi, S. Nangia, and T. F. Abdelzaher, "Greengps: a participatory sensing fuel-efficient maps application," in *Proceedings of the 8th international conference on Mobile systems, applications, and services*. ACM, 2010, pp. 151–164.

[4] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[5] R. Bellman, "On a routing problem," DTIC Document, Tech. Rep., 1956.

[6] L. Ford and D. R. Fulkerson, *Flows in networks*. Princeton Princeton University Press, 1962, vol. 1962.

[7] D. B. Johnson, "Efficient algorithms for shortest paths in sparse networks," *Journal of the ACM (JACM)*, vol. 24, no. 1, pp. 1–13, 1977.

[8] R. W. Floyd, "Algorithm 97: shortest path," *Communications of the ACM*, vol. 5, no. 6, p. 345, 1962.

[9] D. B. Johnson, "A note on dijkstra's shortest path algorithm," *Journal of the ACM (JACM)*, vol. 20, no. 3, pp. 385–388, 1973.

[10] R. Dechter and J. Pearl, "Generalized best-first search strategies and the optimality of a*," *Journal of the ACM (JACM)*, vol. 32, no. 3, pp. 505–536, 1985.

[11] B. M. Sathyaraj, L. C. Jain, A. Finn, and S. Drake, "Multiple uavs path planning algorithms: a comparative study," *Fuzzy Optimization and Decision Making*, vol. 7, no. 3, pp. 257–267, 2008.

[12] M. G. Bell, "Hyperstar: A multi-path astar algorithm for risk averse vehicle navigation," *Transportation Research Part B: Methodological*, vol. 43, no. 1, pp. 97–107, 2009.

[13] A. Maruyama, N. Shibata, Y. Murata, K. Yasumoto, and M. Ito, "A personal tourism navigation system to support traveling multiple destinations with time restrictions," in *Advanced Information Networking and Applications, 2004. AINA 2004. 18th International Conference on*, vol. 2. IEEE, 2004, pp. 18–21.

[14] H. Kanoh and N. Nakamura, "Route guidance with unspecified staging posts using genetic algorithm for car navigation systems," in *Intelligent Transportation Systems, 2000. Proceedings. 2000 IEEE*. IEEE, 2000, pp. 119–124.

[15] K. Hirasawa, H. Miyazaki, J. Hu, and K. Goto, "Discrete random search method "rasid-d" for optimization problems," *Journal of Signal Processing*, vol. 8, no. 4, pp. 351–358, 2004.

[16] J. Lin, J. Du, and S. Wang, "Study on travel route intelligent navigation system based on webgis," in *2009 International Conference on Artificial Intelligence and Computational Intelligence*. IEEE, 2009, pp. 560–564.

[17] J. Miller and M. Ali, "Dynamic fastest paths with multiple unique destinations (dynfast-mud)-a specialized traveling salesman problem with intermediate cities," in *Intelligent Transportation Systems, 2009. ITSC'09. 12th International IEEE Conference on*. IEEE, 2009, pp. 1–6.

[18] C. Chang, H.-T. Tai, D.-L. Hsieh, F.-H. Yeh, and S.-H. Chang, "Design and implementation of the travelling time-and energy-efficient android gps navigation app with the vanet-based a* route planning algorithm," in *Biometrics and Security Technologies (ISBAST), 2013 International Symposium on*. IEEE, 2013, pp. 85–92.

[19] M. Rudack, M. Meincke, and M. Lott, "On the dynamics of ad hoc networks for inter vehicle communications (ivc)," 2002.

[20] D. M. Mark, "Automated route selection for navigation," *Aerospace and Electronic Systems Magazine, IEEE*, vol. 1, no. 9, pp. 2–5, 1986.

[21] "Google Place API," 2015, https://developers.google.com/places/.

[22] "Google Maps Directions API," 2015, https://developers.google.com/maps/documentation/directions/.

[23] J. Chen, "Flow in games (and everything else)," *Communications of the ACM*, vol. 50, no. 4, pp. 31–34, 2007.