On My Way: Optimizing Driving Routes for Navigation Applications

On My Way: Optimizing Driving Routes for Navigation Applications

Fengpeng Yuan, Xueyuan Song and Janne Lindqvist
Rutgers University

***Abstract*—Conventionally, the route recommendations given by GPS navigation applications have been considered as the optimal route search problem only between two locations - origin and destination [1]. Sometimes people want to visit several intermediate locations prior to reaching their final destination. For example, travelers may want to visit a diner and a gas station before arriving at their vacation destination. Although there is likely to be many choices that are available along the route to the destination, only one place from each type should be chosen. Furthermore, in new emerging application domains, such as "physical-world crowdsourcing" [2], people may want to opportunistically visit some places in order to complete personal or work related tasks. Our work explores a design space where we try to reduce the amount of requests made to third-party map and route data providers. We explore the simple idea of using the Euclidean distance as a rough estimate for the optimal route between destinations with multiple waypoints. Our preliminary results indicate that with over 80% of test cases, this simple Euclidean distance estimator approach gives at least one optimal routing alternative.**

I. Introduction

GPS-based navigation is a popular application. One of its main functions is to give the optimal route recommendation to the destination with or without using real-time traffic information. A traditional GPS navigation device is used to acquire position data and help locate the user on a road in the unit's map database [3]. Using the road database, the unit can give directions to alternative locations along additional roads in its database.

There are also several smartphone apps available for GPS-based navigation. However, typically the route recommendations given by these apps, as well as with stand-alone GPS navigators, have only considered the optimal route search problem between two locations - origin and destination [1]. Alternatively, users must manually specify and input waypoints.

In this paper, we present our preliminary design and analysis of an app that could help users to navigate on roads via multiple desired waypoints. Toward this end, to find the optimal route to the final destination, the visit order of the waypoints needs to be optimized. Our work explores a design space where we try to reduce the amount of requests made to third-party map and route data providers. We explore the simple idea of using the Euclidean distance as a rough estimate for the optimal route between destinations with multiple waypoints. Our preliminary results indicate that with over 80% of test cases, this simple Euclidean distance estimator approach gives at least one optimal routing alternative.

II. On My Way: App Design and Implementation

A. Application Scenario

Alice and Bob are leaving home towards their vacation in a different state. They forgot to refuel, so their car is low on gas, and therefore they need to find a gas station. Their usual gas station would in a different direction than planned route. The couple also would like to dine in a nice diner somewhere on the trip. With a usual GPS navigator they would have to search for a diner and gas station separately, and choose the waypoints manually. This would be time-consuming and inconvenient. With our app, Alice and Bob need to only input their destination address and the general type of places they would like to go. The app can recommend an optimal route that satisfies all of their requirements.

B. Design Alternatives

In this paper, we decided to explore a design space where we aim to optimize the need to place API calls to the network-based map and routing distance provider. Towards this end, we are investigating if the Euclidean distance is a good estimator for most of the time for the actual routes. If this would be the case, the needed API calls for route data can be significantly reduced.

Other obvious design alternatives would be to have all relevant data locally on the navigator or smartphone app. Such an approach could be built on top of, for example, Caché [4], a system for caching location-enhanced content for privacy. Another obvious design approach is not to limit API calls to the server at all, but do all the computation in the server side. In such case, the Euclidean distance estimator is unnecessary.

C. Finding Places of Interest Near the Route

Towards finding desired places of interest near the route, we used several APIs available from Google. We used the Google Places API [5] to find interesting places and the Google Directions API [5] to obtain the directions from the origin to the destination.

D. Finding Possible Shortest Paths

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. 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. For each of the waypoints permutations, we calculate the shortest path through the waypoints and then select the best three results. Furthermore, all of the waypoints

in each permutation will be retrieved and sent to Google Directions API to get the directions for comparison. Finally, the recommended optimal routes of those combinations are shown on the map.

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

The first strategy is calling Google Direction API for all the waypoints combinations. We can leverage Google Directions API to optimize the order of waypoints visits. It can optimize the provided route by rearranging the waypoints in a more efficient order. This cannot only offload the optimization work from us, but also take traffic conditions into consideration. This is the easiest way to determine the optimal routes. However, there are a lot of shortcomings when using this strategy. First, calculating directions is a time and resource intensive task, and the number of waypoint combinations can easily explode. For example, if we suppose that a 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 that the number of API requests would be 125.

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 $shortest\_path$;

The second strategy is to reduce the needed API calls. In addition to have the origin and destination visited first and last, a user has to visit each waypoint of specified types

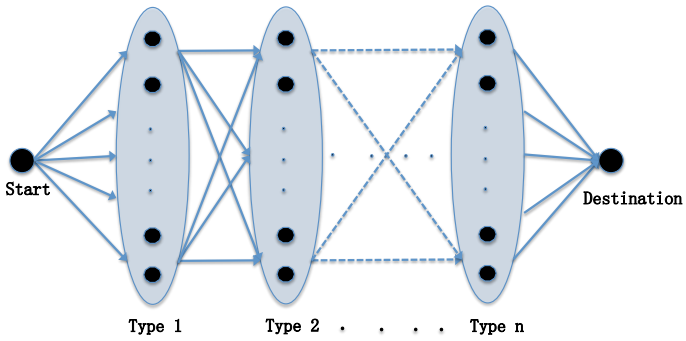

Fig. 1: The graph with layers. The number of layers is the number of places types. Each layer contains all of the nearby places of each specified place type. From start to destination, we must select one node from each layer, which means we must visit one place of each type. If we find a path from start to destination, this path must satisfy the user's need: visit all of the places of specified types before arriving the destination.

once. The problem then 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 this more efficiently, we propose a heuristic algorithm. First, we construct a layered graph with origin and destination as the source and destination on the graph, as shown in Fig. 1. Each layer of the graph contains all the places of each specified type; hence, the number of layers is the number of specified place types. When constructing the graph, for all the intermediate waypoints in two adjacent layers, we connect them (one from each layer) using the Euclidean distance as the weight. To find the path from the origin to destination in the graph, we must select one node from each layer, these nodes are the "must pass" nodes. Therefore, the path from the origin to destination can meet the users' requirements. The top three shortest ones from these routes will be recommended to the user. Since the visit order affects the distance and duration of the routes dramatically, we need to permute the visit order of the specified place types. In real life, people usually do not visit too many places along the route, so the permutation of the visit order (types) would not increase the complexity.

When calculating the path with a fixed visit order, we utilize the well-known shortest path algorithm, for example, Floyd-Warshall algorithm, to obtain all pairs shortest path. Since we must have the origin visited first and destination visited last, we can apply pruning during the permutation calculation, which can save a lot of time. The pseudocode is shown in Algorithm 1. There are several details that are omitted from the pseudocode to increase clarity. For instance, the pruning should be done before calculating the shortest path.

After we receive the response from Google Directions API, we display the best three optimal routes on the map with the waypoints marked, as shown in Fig. 2.

III. Preliminary Experiments and Results

There are two key questions we aimed to answer with our preliminary experiments. First, can Euclidean distance be used as an estimator when selecting actual routes? Second, in which situations is Euclidean distance effective and when does it not work?

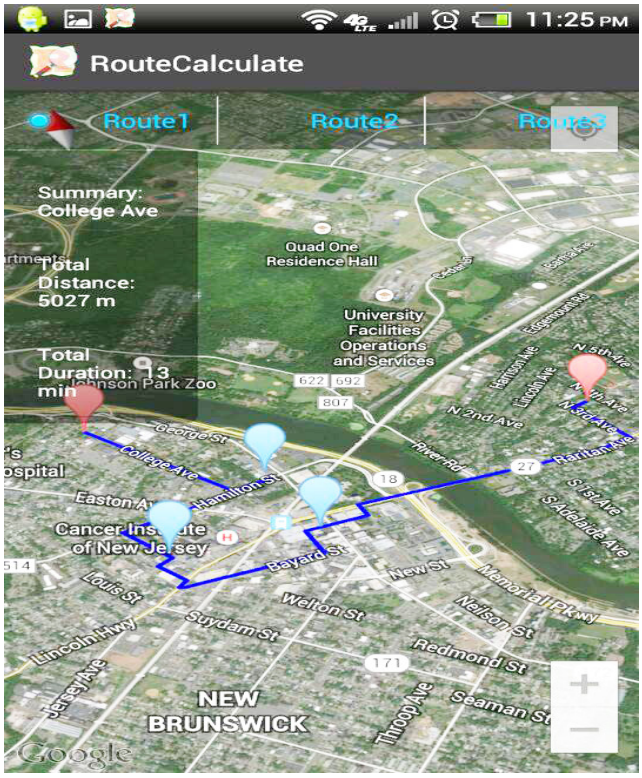

Fig. 2: On the top of the screenshot, a group of three radio buttons can be used for switching three recommended routes, below them there is a information board which is showing the corresponding routes information include main routes name, time duration, and distance. Map data is provide by Google (Imagery ©2013 Google, Map data ©2013 Google).

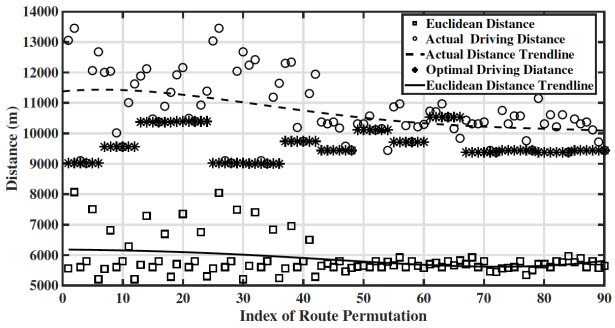

Fig. 3: This figure shows the Euclidean distances, actual driving distances and the optimal driving distance. Euclidian distance simply connects all the waypoints from origin to destination in order, e.g. Origin, waypoint 1, waypoint 2, ..., waypoint n, destination. Actual driving distance means the driving distance if the user visits the waypoints in that specific order corresponding to the Euclidean distance order. Optimal driving distance is the distance if the user visits these waypoints in a more efficient order (optimized by Google). The lines are the trend lines of Euclidean distance and actual driving distances. The x-axis represents the number of waypoints permutations, each number represents a possible permutation - index of route permutation.

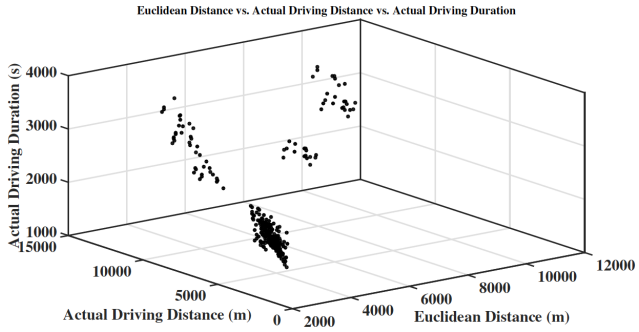

Fig. 4: The points located at the top left corner mean that there is no intuitive route between two places even though they are close to each other. The user has to make a U-turn or bypass some obstacle in order to get to the other place, which results in a greater distance traveled.

A. Experiment Setup and Overview

We selected bank, cafe and gas station to represent typical waypoints. Both origin and destination are chosen from ten different states, which cover both cities and rural areas. When running the application, for each pair of origin and destination, the Euclidean distance of all the possible intermediate waypoints permutations were calculated. The actual driving distance and time duration of visiting all the waypoints were calculated as well. The search radius and the minimum distance between two consecutive searches were set to 300 meters and 1000 meters, respectively. To make the recommendations more useful, we set the place rating threshold to 4.0 (5.0 is maximum). The average route distance from origin to destination was 4.54 miles, and the average number of possible routes was 122.57.

B. Is Euclidean distance a Valid Estimator?

Fig. 3 shows the experiment results with places in Seattle. The figure shows the Euclidean distance, actual driving distance and optimal driving distance of a driving path by visiting different places in Seattle. The actual driving distance is always longer than the Euclidean distance, although the change trend of driving distance is similar to that of Euclidean distance. This indicate the potential of using the Euclidean as an estimator.

In Fig. 4, the area with dense and darker points located are what we would hope to see. These points have a short Euclidean distance, short actual routes distance and the shortest time of travel. In these cases, we can use Euclidean distance as a criteria to select actual routes. In the figure, most points are distributed at the front corner. The points at the top left corner are invalid routes, as discussed below. If the points are uniformly distributed along the space diagonal (face to back), it could be more accurate using Euclidean distance as the criterion, because the diagonal line means that actual route distance and time duration are proportional to Euclidean distance.

From Fig. 4, we can see that points are distributed in three parts. The points scattered along the diagonal are regarded as valid routes, without regarding the efficiency. The points located at the top left corner represent a situation when there is no viable or intuitive route between these two places. For instance, A, B are two places belonging to a gas Station and a cafe, respectively. They are close to each other but located at two sides of highway, which means the user would have to drive around on the highway to find an exit and make a U-turn to get to the other location.

By visualizing the relationship among Euclidean 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 routes which have shorter

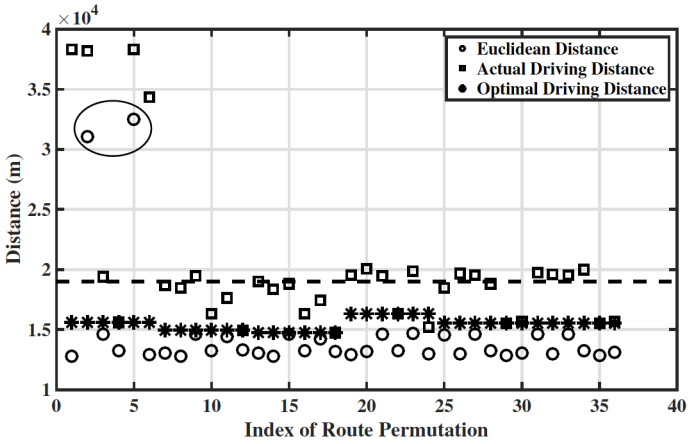

Fig. 5: The invalid points are circled in the graph, which means these routes are invalid by using the strategy. Because the Euclidean distance is longer than the optimal driving distance, it is impractical in reality due to the complexity of the topography, traffic conditions, etc. The dotted line is the average distance of the valid routes, which is used to further select the valid routes.

Euclidean distance, shorter actual distance and shorter time to travel.

C. When is the Euclidean distance Effective?

It is not surprising, due to the complexity of the topography, that sometimes the Euclidean distance does estimate well the actual route distance. The experiment began with finding the routes which are invalid using the proposed strategy and figuring out why it is invalid by comparison with valid routes. If the Euclidean distance is longer than the optimal driving distance, which is optimized by an efficient algorithm, this means that the visit order is much more inefficient. Such routes are invalid, and should be discarded. For example, the points circled in Fig. 5 are invalid routes, because the Euclidean distance of those routes are much longer than the optimal routes, which is impractical. After removing the invalid routes, we choose the routes whose distances are shorter than the average of all the "valid" routes to call the route data API, for example, actual driving routes below the dotted line.

Fig. 6 shows the relationships between Euclidean distance and actual driving distance, optimal driving distance, actual driving duration and optimal driving duration. The circled points (above) are the routes candidates we want to recommend to the user. These routes have shorter distances and less time cost. Unless the user has to visit the specified types of places in a specific order, the routes on the right would not be employed to call the route data API.

To validate the effectiveness of the proposed strategy, we investigated the relationship between Euclidean distance and the optimal driving distance. The results are shown in Fig. 7. From the figure, we can see that most of the optimal routes are covered by those routes that have the shortest Euclidean distance, 249 out of 288 optimal routes are covered as shown in the figure. In addition, we can see that the optimal driving duration is mostly proportional to the optimal driving distance.

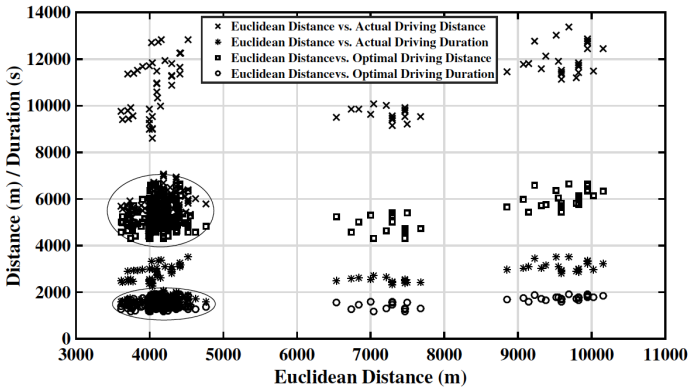

Fig. 6: This figure shows all the relationships between Euclidean distance and actual driving distance, optimal driving distance, actual driving time, and optimal driving duration. The points circled are the routes which require less distance and less time. Other routes would not be selected unless the user has a specific visit order requirement.

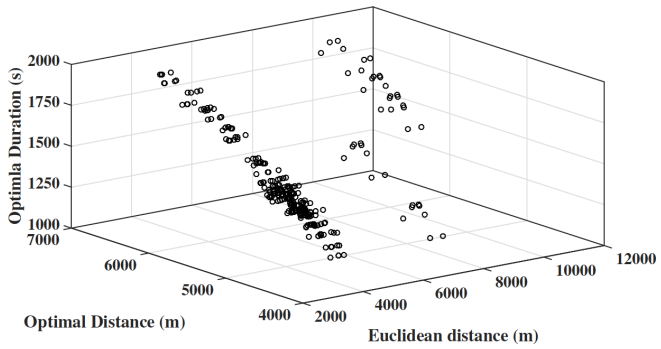

Fig. 7: The grouped points on the left are the routes which have both less Euclidean distance and less optimal driving distance. These are the routes the we would like to recommend to users. With the proposed strategy, 86.46% of the optimal routes are covered by the shortest Euclidean distance routes.

IV. Related Work

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 [6] and Bellman-Ford's [7] [8] algorithms, can compute the shortest path between a single node to every other node in the graph. In order to compute all the pairs shortest path, algorithms, such as Johnson's [9] and Floyd-Warshall's [10] 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 [6] and A* [11] algorithms are usually used to plan the route in automotive navigation systems. Compared to Dijkstra, A* achieves better computation time by using heuristic functions. However, these traditional route planning algorithms [12] [13] 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.

To calculate the optimal route through multiple waypoints, Maruyama et al. [14] proposed a method to find the optimal tour plan using a genetic algorithm. Their work focused on

tour planning and users' preference for the destinations. Kanoh et al. [15] proposed another genetic algorithm based route search algorithm to find the route considering the unspecified waypoints destination. 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 time and the order of visiting multiple waypoints in order to determine the optimal route from origin to destination via multiple waypoints. Hirasawa et. al [16] proposed RasID-D for solving the discrete optimization problems. RasID-D can optimize the visiting order of the intermediate destinations. Lin et al. [17] realized a travel route intelligent navigation system based on WEBGIS. The basic purpose of their work was to produce a satisfying tourism route planning for travelers based on predefined restrictions. Jeffrey and Muhammad [18] presented a fast path algorithm that contains multiple unique destinations. In their paper, they proposed an algorithm which can be used for determining the fastest route to travel to a set of destinations, such as those required by delivery companies. Recently, Chang et al. [19] designed and implemented an Android-based navigation app, which can help reduce the traveling time, fuel consumption and emitted carbon dioxide. Gao et al. [20] have developed an "elastic pathing" algorithm, which can be used to derive driving routes merely with the knowledge of the starting location and time-stamped speed data.

V. Discussion and Conclusions

In this paper, we presented preliminary experiments on finding optimal routes between multiple possible waypoints with a resource-constrained mobile device. In contrast to having unlimited access to server-side map and route data, we implemented an app that can do most of the computation locally with few API calls for requesting data. Towards this end, we tested whether Euclidean distance between places would work as an effective estimator for actual driving routes. With our preliminary experiments, using bank, cafe and gas stations as waypoints, more than 80% of the experiments showed at least one of the best three shortest driving routes were found using Euclidean distance estimates.

Acknowledgments

This material is based upon work supported by the National Science Foundation under Grant Number 1211079. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

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] R. Teodoro, P. Ozturk, M. Naaman, W. Mason, and J. Lindqvist, "The motivations and experiences of the on-demand mobile workforce," in *Proceedings of the 17th ACM Conference on Computer Supported Cooperative Work & Social Computing*, ser. CSCW '14. New York, NY, USA: ACM, 2014, pp. 236–247. [Online]. Available: http://doi.acm.org/10.1145/2531602.2531680

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

[4] S. Amini, J. Lindqvist, J. Hong, J. Lin, E. Toch, and N. Sadeh, "Caché: Caching location-enhanced content to improve user privacy," in *Proceedings of the 9th International Conference on Mobile Systems, Applications, and Services*, ser. MobiSys '11. New York, NY, USA: ACM, 2011, pp. 197–210. [Online]. Available: http://doi.acm.org/10.1145/1999995.2000015

[5] "Google maps api," 2015, https://developers.google.com/maps/.

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

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

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

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

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

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] X. Gao, B. Firner, S. Sugrim, V. Kaiser-Pendergrast, Y. Yang, and J. Lindqvist, "Elastic pathing: Your speed is enough to track you," in *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, ser. UbiComp '14. New York, NY, USA: ACM, 2014, pp. 975–986. [Online]. Available: http://doi.acm.org/10.1145/2632048.2632077