International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 06 Issue: 06 | June - 2022 Impact Factor: 7.185 ISSN: 2582-3930

A Fast, Efficient Technique for Finding a Path through Multiple Destinations

Snehal Shivaji Gundal

Abstract—In a real world situation, employees of different companies or organizations often need to travel to multiple destinations without passing through the same place twice. However, the problem of identifying an efficient path to travel through multiple destinations can be extremely difficult and timeconsuming. In this work, I propose an algorithm to find an efficient path in real world scenarios, within a short amount of time. This heuristic uses Dijkstra's algorithm to find different paths between pairs of destinations and then determines the efficiency of each path. It runs in a recursive manner until it covers all of the destinations, including returning back to the starting location. With that, the algorithm creates an efficient path through all of the destinations. As a result, this approach allows people and organizations to save time, and money which could be used in other places. In order to test how beneficial this technique is, I have run several sample test situations involving between 5 and 50 destinations. My data shows that the algorithm produces an efficient path within 20 to 150 seconds, depending on the number of destinations. Since it requires very little time and knowledge, this approach will be quite appealing and useful to the companies and organizations.

Index Terms—Algorithms and Theory; Computational Intelligence; Systems and Software Engineering; Computing in Management Technology;

I. Introduction

On a typical day, a busy person in the service industry or sales industry visits multiple places. For example, a courier delivery person probably visits 50 households and drops the boxes in each house. Most likely, those houses are within a few cities, but certainly not next to each other. To address the need of these people, there is certainly a need to develop a methodology to quickly identify an efficient route that cover multiple locations.

For many years, researchers have proposed various methods to identify efficient route between two points. Different definitions of efficiency has been used, along with many different real-life factors. Dijkstra's algorithm [1] is the classic approach for finding the shortest paths between two destinations in a graph. The graph can represent any network like road network across cities, or inside a city. Hart et. al proposed a faster version of the Dijkstra's algorithm, for finding shortest paths in graphs. This approach is called A* algorithm [2].

In [3], a road map production system model using bidirectional search algorithm was proposed for navigating a mobile robot. Fetterer et. al proposed a performance analysis of hierarchical shortest path algorithms in [4]. They analyzed the effect of memorizing individual data structures for the storage overhead and computation time of hierarchical routing algorithms based on A^* technique. In [5], Liu et. al explored the approach of combining a shortest path algorithm with knowledge about the road network. Seo et. al presented an effective alternative path-finding algorithm based on a genetic algorithm, in [6]. They developed efficient genetic operators for path calculation, and used that to efficiently used similarities among the paths.

In [7], Zhang et. al proposed a Bi-Road Method (BRM) that generates the shortest collision-free path with consideration of safety from the given start point to goal point. This was used for mobile robots working in the static and known 2D environment. Noto and Sato proposed a fast method for obtaining a path that is as close as possible to the path obtained by the traditional Dijkstra method (the optimum path) [8]. That method extends the conventional Dijkstra method so as to obtain a solution to a problem given within a specified time, such as path search in a car navigation system. In [9], an improved version of the shortest routing algorithm was proposed. In that approach, a coefficient weighing the evaluated distance was used. This coefficient has an influence on the performance of the shortest path algorithm.

Girish et. al discussed about approximating Shortest Path in Large-Scale Road Networks with Turn Prohibitions (rightturn only, left turn only) and multi-constrained Path Algorithm [10]. The results show that the efficiency of that MCP algorithm is 84.5conventional routing algorithms in terms of execution time. In [11], the authors find realistic or feasible path, and not necessarily only the shortest path. In that work, both lane changes and turn restrictions are considered, which were necessary to achieve a realistic shortest path. In [12], Qi and

Schneider presented a generic two-layered framework for moving objects in road networks environment and demonstrated the important role of traffic factors on path finding and route planning. They developed a parallel algorithm in road networks with the consideration of traffic influence. A general simulated annealing algorithm was proposed in [13] that took into account all the restrictions of real network and found the optimal solution between any two nodes efficiently without changing its structure. In [14], authors presented an effective approach that works well in realistic road networks using multi constraint routing algorithm.

In [15], authors presented a use case on scaling up the Time dependent shortest path calculations within an established existing Dynamic Traffic Assignment software framework with distributed computing. Test of this approach with real-world transportation networks show drastic run time performance. In [16], [17], Kaneko and Honda presented a technique to find shortest path involving a specific departure point, destination point and a set of candidate middle transit points. Wang et. al presents an integrated shelter location and route allocation approach for the emergent evacuation problem with multiple sources and multiple shelters in city transportation networks [18]. A mixed integer linear programming model is developed to formulate the problem, in which the overall evacuation time is minimized subject to capacity constraints on both shelters and roads. In [19], Yang et. al build a route selection model in emergency evacuation based on quasiuser optimum dynamic traffic assignment theory, and develop a constrained K-shortest paths algorithm within a dynamic restricted searching area. Jing et. al proposed an efficient road network k-nearest-neighbor query verification technique which utilizes the network Voronoi diagram and neighbors to prove the integrity of query results [20].

In [21], Zhang et. al proposed a parameter-free minimal resource neural network framework for solving a wide range of single-source shortest path problems for various graph types. In [22], Wang et. al proposed a route-planning approach, which introduces factors of turning, and edge removal solve k shortest path problem. Another approach was presented which searches effectively the shortest-time path and avoids collision.

All the above-mentioned research work focus on developing routing algorithm between two locations, but none of them focus on developing a complete route involving multiple destinations. In our proposed approach, we bring in that unique concept where we compute the efficient route covering multiple destination locations.

We have organized the rest of the paper as follows: In the Section II, we present the problem statement and motivation behind working on this solution. In Section III, we discuss the flow and internal details of our proposed approach. The experimental setup and the results are presented in In Section IV. In Section V, we list few possible real-world application of this technique. Conclusions are drawn in Section VI.

978-1-5090-4767-3/17/$31.00 ©2017 IEEE

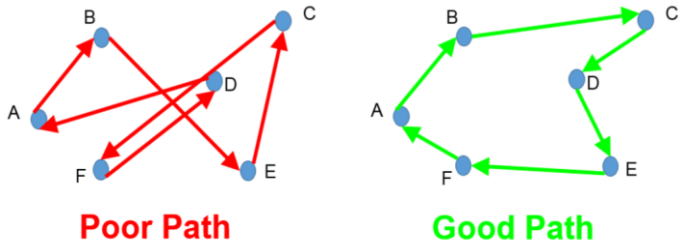


Fig. 1. Example routes showing a good path and a bad path

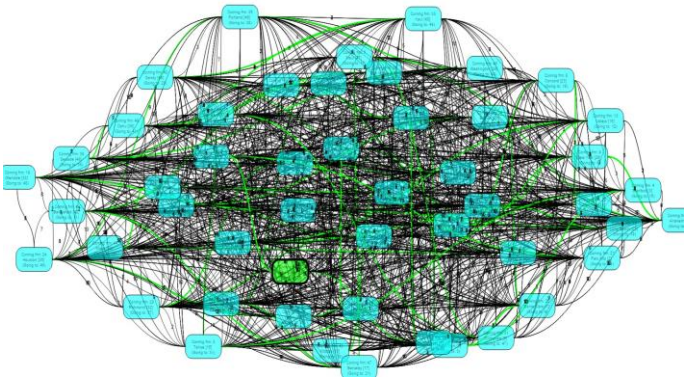


Fig. 2. partial connection graph involving 50 destinations

II. Problem Statement and Motivation

The problem of identifying an efficient path to travel through multiple destinations is extremely difficult and timeconsuming. In this work, I propose an algorithm to find an efficient path in real world scenarios, within a short amount of time. The path will cover all the locations that need to be visited.

Let us first visualize a situation in Fig 1, where a person needs to start from the initial location A, visit five locations (B through F) and then come back to the original starting location. If he uses his intuition and decides the route, it is possible that he would take a route similar to what is shown in the left side of the Figure 1. On the other hand, if there is an intelligent technique to find an efficient path, that would identify the route as shown in the right side of Fig 1. It is evident that the efficient path saves travel time, and resource.

In all situations, the best possible path can be determined by using brute-force approach of trying every possible combination. If number of locations are relatively small, then a person can manually do trial and error to decide on the best possible route. Unfortunately, if the number of locations is large, then that approach is infeasible and will require very long time to identify even for a super-computer.

In the Fig 2, I am showing the network of paths if there are 50 destinations in the problem statement. This image is captured by using the Dracula Graph Library [23] available in the public domain. In the Fig 2, each green box denotes a destination, and each black line denotes a path between the two destination points. From that figure, it is clear that analyzing a network like that is impossible for a human brain. In addition, for a fast super-computer, it will take very long runtime.

If there are 10 locations to visit, then the computer needs to analyze 362,880 paths to determine the best possible path. Similarly, for 15 locations, computer needs to analyze 87,178,291,200 paths. Finally, for 50 locations, computer needs to analyze 60,828,186 * 10^{55} paths.

With that, we realized that it is extremely beneficial to different organizations and companies, if there is an efficient algorithms that can quickly determine an efficient path covering multiple locations.

III. Details of Our Approach

Our proposed approach is a web-based application that can be run from any Internet-enabled modern browser. In this section, we first explain the use-model of this proposed application. After that, we describe the details of the core engine behind the application.

A. Flow of Our Approach

Fig 3 shows the input form for the online interface of this application. In this application, the user needs to enter the following information:

- The Starting Address: This is the place from where the person will start the trip, and visit all the required destinations, before coming back to this starting location. For example, in the case of a courier driver, the warehouse address will be entered in this field, as the starting address.
- All the Different Destination Addresses: In this field, user will enter the addresses of all the locations that need to be visited. For example, in the case of a courier driver, the addresses of all the customers will be entered in this field.
- Mode of Travel: As of now, the supported modes are driving, walking, public transport (bus) and bicycling. This is optional, and driving by car is used as the default value. This field makes the application usable by the people who do not have a car.
- Most Important Criteria: This field allows people to specify whether they want to travel less distance or travel faster, even if that means longer distance. This option can be useful during a busy commute hours or in a highly congested area.
- Optional Settings: User can specify optional parameters like avoid highways or avoid toll-road etc.

After all the information is submitted through this webbased form, our back-end engine computes an efficient route to cover each of the destination locations. Since our flow uses Google Maps [24], it is very intuitive and easy to use. The final output of the application contains the following information:

- The Efficient Path and Complete Direction: All the destination locations are arranged in an efficient order, which constitutes the final path. Finding the order of the locations is the key offering in this approach. Once the order is determined, the application displays clear turnby-turn direction for driving, walking or bicycling (by using Google Maps [24]). As a result, user gets a specific travel order that he can follow from the starting point. For the transit users, the application shows a combination of bus direction and walking.
- Visual Display of Complete Direction: The actual direction is displayed by using Google Maps [24]. Depending on the mode of travel, this can be either driving or walking or bicycling or hybrid direction.
- Total Distance of Travel: The actual total traversed distance is displayed in Miles. This distance is computed

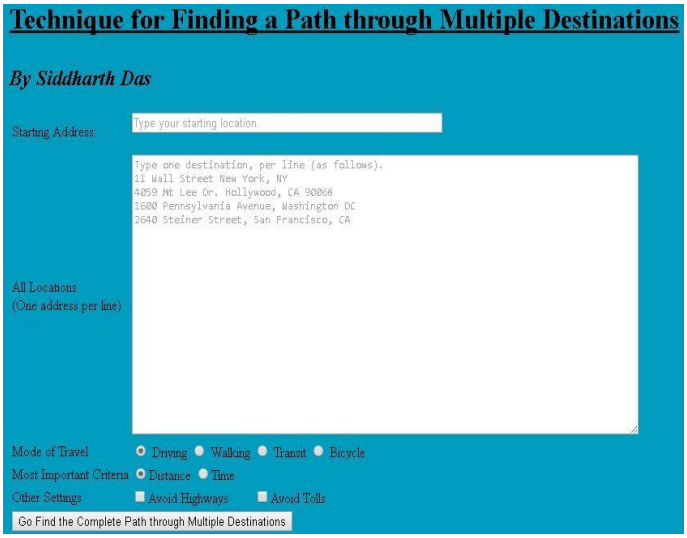


Fig. 3. Web interface of our proposed flow

by using the route information from Google Maps [24]. Hence it is almost always different from the geographical distance between the locations in order.

- Total Time of Travel: The actual total travel time is displayed in hours and minutes. This value greatly changes based on the mode of travel (car or walk or bicycle or transit).

B. Our Core Engine

Once the user fills out the form and submits the input data, our core algorithm gets executed in the back-end. In this section, we discuss the internal details of that algorithm.

1) Identification of the distance or time between pair of locations: Once user submits the starting location and the list of destination locations, we first need to collect the data about distance or travel time between each pair of locations. This is obtained by calling the Google Maps developer API [24]. Google Maps allows 50 such requests per second and 100,000 such requests per every 24-hours, which is more than sufficient for our usage. If the user chose the distance to be the most important factor, we call Google Maps in the distance mode. On the other hand, if user entered time as the most important factor, we call Google Maps in the time mode.

2) Building of the graph structure: We used graph theory for the core of our algorithm. In this step, we formulate the graph, where each location is a vertex. Every pair of vertex is connected by an edge, where the edge weight is the number (distance or travel time) returned by he corresponding Google Maps API call. We modeled the graph by using the Dracula Graph Library [23]. We store the complete path in a variable called Final Path. In the beginning, that variable is empty. As we decide different segments of our whole route, we keep on appending the segments to this Final Path, until all the destinations are visited.

Fig. 4. Flowchart of the proposed algorithm

3) Updation (Increase) of the cost: Now, we start an iterative process, until a complete Final Path is computed. In our graph, if any destination was marked as Visited, we increase the cost of visiting that destination again. This is a standard technique in the domain of graph algorithms, so that subsequent steps (in the iterative loop) avoids visiting these destinations again.

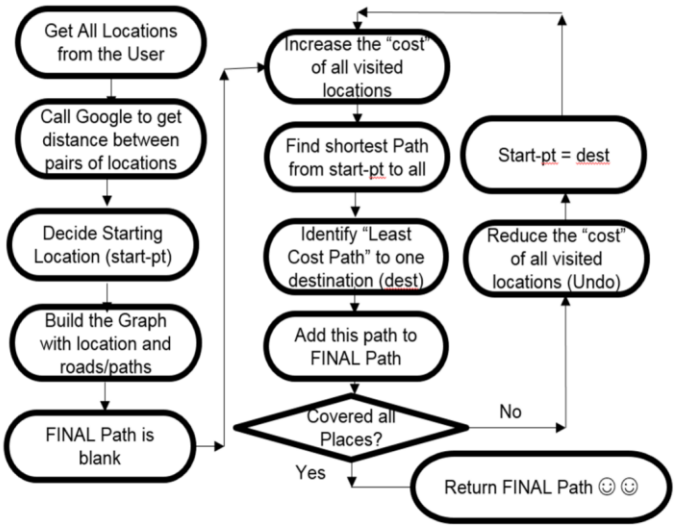
4) Computation of different Shortest Paths: Next, we start the frontier propagation from the starting destination. In the first pass of the iteration, starting destination is the original starting destination. In the subsequent passes, we change the starting destination, so that the frontier propagation starts from the new starting destination. In this step, we find the shortest path from the starting destination to all the other destinations. At the end of this computation, we have many alternate paths that take the visitor from starting destination to all the other destinations.

5) Identification of Least-Cost Path: once all the paths to different destinations are identified, we have cost values for each route. The cost value will be in miles, if the user is running the application in distance mode. On the other hand, if user chose time as the important factor, then the cost will be in minutes. Once the cost for each destination is computed, we find out how many new destinations are being visited in that path. Then, the "effective cost" for that path is computed by diving the total cost by the number of newly visited destinations in that route.

For example, to visit from San Francisco to New York, our algorithm might be going through Las Vegas, Dallas and Orlando. If all those destinations were not visited earlier, then this specific path will have effective cost value equal to (total distance or time)/4. Similarly, if our route reached New York only via Las Vegas and Dallas, then the specific path will have effective cost value equal to (total distance or time)/3. Another

situation would be, our route reached New York via path will have effective cost value equal to (total distance or time)/2.

By using this approach, we identify the Least Cost Path among all the paths possible. This Least Cost Path is appended to our Final Path. In addition, we mark all the visited locations appropriately, so that subsequent steps try to avoid visiting those locations again.



6) More Iterations Needed?: After covering any additional location, this algorithm performs a quick check to see if there are still any more locations that are not visited. This is a trivial step. If all the locations are visited, then the algorithm successfully terminates and returns the Final Path as the answer. In addition to the Final Path, the algorithm will then interact with Google Maps [24] and display the actual traveling directions along with travel time and travel distance. On the other hand, if the algorithm identified some unvisited locations, then following steps are performed

7) Updation (Reduction) of the cost: In this step, we reduce the cost of visiting any Already Visited location. This step is important, so that we do not increase the cost repeatedly, in the step 3.

8) Reformulating the problem for Remaining Path: In this step, we prepare the data structure for starting a new search for the next segment. To achieve that programmatically, we mark the last visited location as the new starting point. With that, the algorithm goes back to the step 3 and performs the iterations, until the step 6 successfully terminates the algorithm.

IV. Experimental Setup and Results

We have implemented our proposed algorithm using Javascript programming language on Windows-10 laptop, with 4GB memory. HTML and CSS were used for the forms to accept user's input values. For the output display, HTML was used, along with Google Maps [24].

To run this software, users need only a modern web browser and internet connection. For all our runs, we used dual-core

Fig. 5. Sample Path when destinations are in different states in USA

Las Vegas and Dallas, but Dallas was already visited in our overall algorithm (through some other path), then this specific Windows-10 computer containing 1.7 GHz i3 processor and 8GB memory. In addition, the application was also installed

Fig. 6. Sample Path when destinations are in different cities in Silicon Valley

in a web-hosting area, so that other people can use it from their computer.

To collect all the different data-points regarding the capability and usefulness of our proposed approach, we used different combinations of input factors, as described in the Section III. Following variations were used:

A. Different Types of Destinations

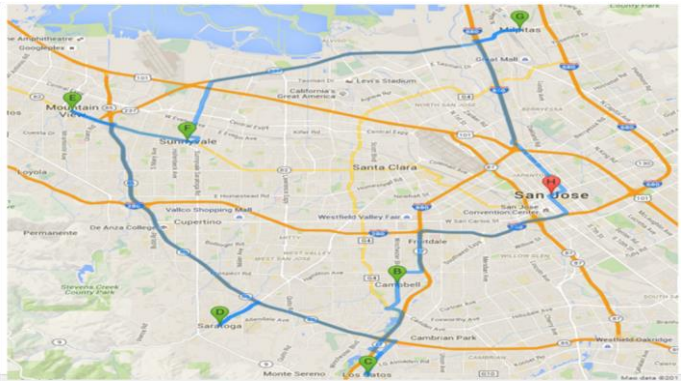
- Short distance travel: We chose different addresses in a specific city like the New York City. Generally, we used the addresses of local fire stations and police stations in the city.
- Medium distance travel: We chose different addresses in different cities in a specific state. For example, in some runs, we chose following 7 cities in the state of California: San Jose, Mountain View, Saratoga, Los Gatos, Campbell, Milpitas, and Sunnyvale
- Long distance travel: We chose different destinations across America, that spans through multiple states. For

example, in some runs, we chose the city centers of following cities: San Francisco, Denver, Orlando, New York, Las Vegas, Chicago, Seattle and Dallas

B. Varying Number of Locations

We ran the proposed application on different number of destination locations. The number varied from 5 to 50.

C. Different Modes of Travel



We have used Walking, Driving, Cycling and Public Transit mode, for different address combinations. Walking and bicycling were mostly used for short-distance mode.

D. Different Key Goals

In some cases, we have tried to minimize travel-time. In some other cases, we have tried to minimize travel distance.

E. Different Miscellaneous Settings

For the long distance travel, we have used some combinations where we tried to avoid highways or toll roads.

Fig. 7. Sample Path when destinations are in the city of San Jose, CA

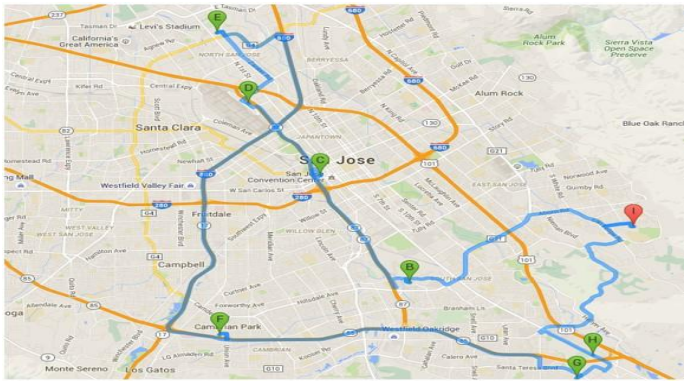
We tried our application with over 100 different variations of these input combinations. When the number of locations is less than 8, we could implement an exhaustive brute-force search approach and compare our algorithms' result with the best-possible outcome from the brute-force approach. In each of the cases, our approach produced the best possible route.

Then, we tested our algorithm on wider network (with number of destinations greater than 8) and visually observed the results in Google Maps [24]. Since brute-force approach cannot compute such a large number of destinations, there was no way for us to verify that. visually, the final complete paths seemed quite efficient.

In this section, for the sake of brevity, we are displaying three sample outputs from our application. These are out of 100 different combinations that we tried with.

Fig 5 shows the complete path when the destinations are in different states in USA. Chosen locations are: San Jose, Denver, Orlando, New York, Las Vegas, Boston, Seattle and Dallas

Fig 6 shows the complete path when the destinations are in different cities in Silicon Valley (San Francisco Bay Area). Chosen locations are city centers of San Jose, Mountain View,



Saratoga, Los Gatos, Campbell, Milpitas, and Sunnyvale.

Fig 7 shows the complete path when the destinations are quite close to each other, in the city of San Jose. I chose the addresses of few fire stations and police stations.

TABLE I
COMPUTATION TIME OF MY ENGINE

Number of Destinations	Computation Time in Seconds
5	17
10	28
15	41
20	54
25	70
30	89
35	102
40	107
45	124
50	146

In Table I, we present the time consumed by our approach, to compute the complete path. It is observed that our approach consumes less than 30 seconds to compute the complete path covering upto 10 locations. In the other spectrum, when the number of locations is around 50, then our approach takes less than only 3 minutes. This observation confirms that our proposed approach is extremely fast and easy to use in realworld applications.

V. READ-WORLD APPLICATION

This application can potentially be used by different organizations across the world. Currently, this is being presented to different departments in the state of California. This application has been successfully demonstrated to one of the largest public transportation companies in the USA. We have proposed them to use it in their on-demand services offered to the senior citizens in the city. Another potential usage will be in the department of Waste management, where they deliver the replacement garbage bins to different houses. Few other potential departments are servicing/inspection divisions in different departments, like Food and Drugs, Gas and Electric Services, Cable and Telephone Services companies etc. All these departments' applications require a vehicle (driver) to start from a specific location, visit different destinations locations in an on-demand manner, and finally come back to the starting location.

VI. CONCLUSION

In this work, we propose an algorithm to find an efficient path in real world scenarios, within a short amount of time. Our primary goal is to provide this tool to different companies or organizations that have people who need to visit multiple destinations. Using this tool, they can determine the efficient path quickly, and that eliminates the need for them to manually identify a path. As a result, this approach allows people and organizations to save time, and money which could be used in other places. This proposed methodology is practical and extremely easy to use. Our experimental data-set indicates that our proposed approach determines a path within a very short amount of time, and the path is almost always the best possible path that a human could have generated by using trial and error approach over several hours and days of time.