Shopping Planner

Lattu, Devendra
dev12@umbc.edu

Ahirkar, Vineet
vineeta1@umbc.edu

Pisupati, Kanni Sailakshmi
pika1@umbc.edu

Department of Computer Science and Electrical Engineering
University of Maryland, Baltimore County
December 2016

***Abstract*— With the growing number of services and service providers over the world, the problem has always been to pick the best of the service providers that can match to the requirements of the customers. Most of the requirements generally involve cost, the quality and the time taken. In this paper, we are going to discuss the shopping planner problem with respect to time and distance. Considering multiple services across a given location, we try to create the path that an individual can take to cover the given set of services with the optimal path cost. Here we will be using the common term service providers for different kinds of shops, agencies, offices, schools etc.**

Keywords- Optimal path, Traveling Salesman Problem, Services, Shopping Planner

I. INTRODUCTION

The motivation of this project was when we were looking for the services around the vicinity of Baltimore and had to pick the best of the service and also get the best of the path to cover these services. This project is more like the shopping aisle algorithm wherein the customer has to pick all the items in the given shop, peek at various category locations, choosing the best (utility maximizing) option, and then continuing in a similar manner until the path is complete. In our project, we have a data set of services across the globe, of which shops are the major of the entire set. User of the services always had the issue of picking the best service within their vicinity and make a route that they can cover with the parameters to have the considered all the required categories, get the best route, cost and time efficiency. In this project, we are going to cover the time and the distance factor to get the best route. Routing problems have always been in the picture in most of the use cases, like the package delivery problem, the vehicle routing problem and the shopping aisle problem. Research has proved that the loss incurred on account of lack of a proper route planning is huge. The routing problems generally found are for the applications of the logistic domain; the common customer is also in need of a solution to this domain. This motivated us to come up with the project to solve the routing issue for common man.

II. RELATED WORK

The Traveling Salesman Problem, is being NP-hard has been attempted by many. The Department of Combinatorics and Optimization at the University of Waterloo [12] works mainly on the optimization of the Traveling Salesman Problem, where they have considered multiple use cases such as the route across the major cities of US, World city tour using TSP and many other.

III. DATASET

The data set for the project was retrieved from "Archive.org"- SimpleGeo Public Spaces CC0 Collection Services Data set [4]. The data set is in a geojson format and contains the following attributes -service name, category, location co-ordinates, address. The dataset size is 20GB and contains the details of services across the globe. For the purpose of our project, we considered the data of only US locations. The US dataset is 7.4 GB and contains 12,924,903 entries.

A. Dataset Cleansing and Loading

The raw data set which was in the geojson format [13] was in a form of a big text and had lots of extra attributes which we didn't require for the project. Also, some of the attributes were nested inside hashed attributes. We wrote a script in nodeJS which asynchronously spawned threads and read a line from the JSON file, parsed it and inserted it into mongoDB. We added 2dsphere index [14] to the data which was needed for the geospatial queries.

IV. PROBLEM STATEMENT

Customers are given the option to select a maximum of five service categories. Some of the service categories that we are available with us for our project are Groceries, Furniture, Electronic, School, Bank, Health care, Doctor etc. which totals to 80 categories. The user is given the option to search the categories from the current location or can enter a location of his/her own. The user is also given an option to set a radius limit in miles of up to how much radius should the services search be restricted to from the set location. Our application will search for only services categories given by the user within the given radius i.e in the vicinity of the given input location. If services are not found, the user will be prompted that the service category is not available in the vicinity. If all the services are found, the application will pick the services within the vicinity, pick the best shop from every category and compute the best route for these categories. The user will then be provided with a map having detailed navigation information of the entire route.

V. ALGORITHM

The Traveling Salesman Problem(TSP) [1] is a classic algorithm which solves the issue of route optimization. It is an NP-hard problem and has a brute force solution in $O(n!)$ time complexity. TSP tries to find a Hamiltonian cycle [3] with the smallest possible overall routing weights. In other words, TSP answers to the following question - "Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the origin city?". However, in our version of TSP, we need not visit all the service providers to find the shortest route. The shortest route needs to be planned between home and only those service provides that can fulfill the shopping item list of the customer. From the notation perspective, we will consider n as the number of categories and m as the maximum number of service providers per category. We will be listing the algorithms that we implemented as well as the ones which we studied for our usecase.

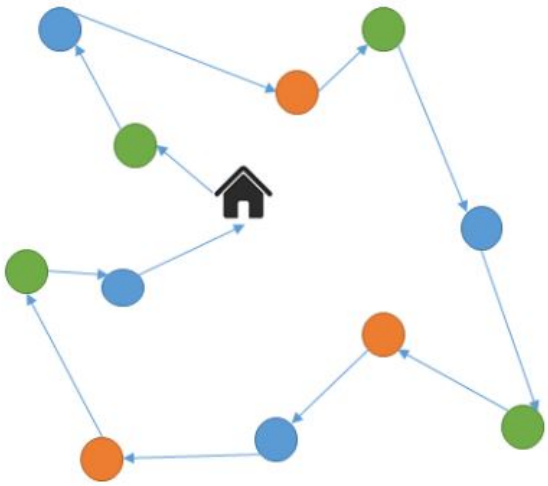

Fig. 1. Classic TSP: Traverse all the data points and find the best route.

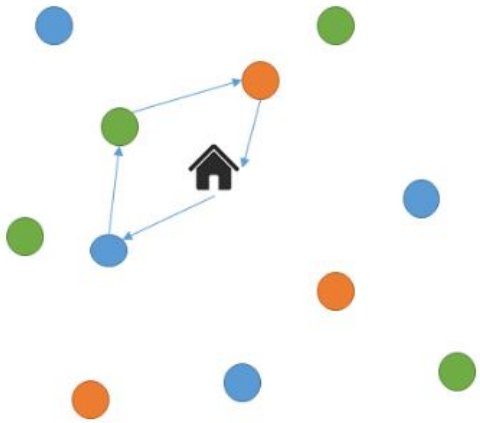

Fig. 2. Requirement in our case: Plan the shortest route by selecting one data point for every category such that total route cost is the smallest

A. *Nearest Neighbor (NN)*

The nearest neighbor algorithm [5] was one of the first algorithms used to determine a solution to the traveling salesman problem. In this algorithm, the salesman starts at a random city and repeatedly visits the nearest city until all have been visited. It quickly yields a short tour, but usually not the optimal one.
We experimented with the following implementation for this algorithm:

1) Start with home location and set the current location pointer to home.
2) Select a service provider that is closest to the

home. Move current location pointer to this service provider.
3) Mark the item corresponding to the visited service provider as completed.
4) Select the service provider that is closest to the current location and has it's category unmarked.
5) Repeat step 3 and 4 till all items are marked as completed.

The Nearest Neighbor is a greedy approach and many not lead to the best path and at times may give out the worst path. Here, we need to calculate distance of every point from home location and thus the time complexity will be $O(n*m)$. The time complexity to form distance matrix will be $O(n^2)$. Therefore, the running time complexity for this algorithm is $O(n^2)+O(n*m)$.

B. Dijkstra's algorithm

Dijkstra's algorithm [6] finds the shortest path between two points in a graph. Input to the algorithm is a set of vertices 'V' and a set of edges 'E'. Also, every edge between vertices u and v should have a weight w(u, v). Thus, we generate a path from source vertex 'S' to the destination vertex 'T'. The algorithm proceeds in an incremental fashion by considering the adjacent vertices and selecting the vertex with the minimum weight. The running time of this algorithm is O((E+V) log V). This algorithm is not a good fit for us, as the source and destination is the same in our case and we have to make sure that the selected path covers shops from each product type. This criteria cannot be maintained in the Dijkstra's algorithm and will thus give insufficient results.

C. 2-OPT

The 2-OPT [7] is a optimizing algorithm for solving the traveling salesman problem. The approach is to compute a route at the initial stage which may not be the best. The next step is to select nodes from the route randomly and swap it with every valid possible combination of nodes and check for the best of these swapping. This approach is generally used in the vehicle routing problem. The number of swaps make this approach time consuming.

D. Simulated Annealing

Simulated Annealing [8] is another approach for optimizing the traveling salesman algorithm. The main idea of this algorithm is unlike the 2-opt [7] and hill climbing algorithm [9], the solution is not stuck to the local minima [10], where it believes to have obtained the optimal solution but there exists a global maxima [11] that can provide the best solution.

E. Clustering and Centroid Analysis

We brainstormed about the possible ways to solve our version of TSP using a new approach which we named as clustering and centroid analysis.
Our implementation of the algorithm was as follows:

1) Create multiple clusters, each having randomly allocated single service providers from each of the individual categories such that no service provider can be present in more than one cluster.
2) Calculate the centroid of every cluster.
3) Measure the distance of the every centroid from home as well as the distance between every centroid to all the services providers contained within the cluster with those particular centroids.
4) Sum up all the measures and select the cluster who has the shortest measure.
5) Use our formulation from Nearest Neighbor to calculate the distance matrix and decide upon the ordering of the service providers from home location.

Even though this seemed to be a version of fast approximation approach for calculation of the closest possible route, the output is based upon the randomness in formation of clusters. The runtime complexity for creating random clusters, getting centroid, and calculating distances from home and centroid is $O(n*m)$ individually. The runtime complexity for calculating the distance matrix is and $O(n^2)$ as before. Therefore, the overall runtime even for this algorithm is in $O(n*m)+O(n^2)$.

Another problem that we faced when we had to find the route was unlike the naive node and edge approach where we had the distances to be of a single unit. This being a real world scenario, wherein the distances of two different points is not the flat distance between the two points. The distance depends on the spherical parameter of the Earth and also two different points can have one or more route, so theses routes can have multiple distances. We had to pick one distance that can not be the shortest but also look into the fact if the other nodes lie within or at the vicinity of this route. In order to solve this problem of multiple distances and spherical distance calculation, we made use of MongoDB and Google Maps API [15]. MongoDB's distance calculation provided us with

the distance between locations co-ordinates. Google Maps API provided us the optimal route between two locations co-ordinates, if there are multiple route to reach a particular place. Also, Google Maps API makes use of 2-OPT and it majorly solves the TSP problem when provided with multiple points.

Keeping in mind the above issues, we also had to pick an algorithm for our project which would not only solve the problem of routing but also pick the best service point from the service categories that we have. The problem of picking one from every cluster was a constraint that we had to address. After discussing with Dr. Halem on different algorithms we can use, we found that the naive approach had to be used since we had to traverse and pick the best of the node from every cluster. After the best pick from every cluster we decided to feed these nodes for solving the routing problem.

F. Nearest Neighbor(NN) from Home

We designed a modified version of Nearest Neighbor algorithm to get all the unique item list category wise nearest neighboring service providers.
Our implementation of the algorithm was as follows:

1) Start with home location and set the current location pointer to home.
2) Create a list of arrays where each array contains service providers for a particular item.
3) From each of these arrays we shortlist a single service provider who has a distance closest to home.
4) Create a distance matrix for home and for each of the shortlisted service providers by calculating the distances of every service provider with each other and from home.
5) Calculate the sum for each row of this distance matrix and select the row with the max sum.
6) Finally do ordering over the service providers by selecting the columns corresponding to the ascending order of selected matrix row.

There are certain advantages of our modified NN algorithm which includes keeping the user as close to home as possible. Our modified algorithm also performs in $O(n^2) + O(n * m)$ time. The output from this algorithm might not be the optimal one as we are not considering all the possible routes like the one done by brute force approach. However, given the usecase for our problem, this algorithm gives a fairly descent result.

VI. FUNCTIONAL FLOW DESCRIPTION

As per the use case we have, we have designed the following sequence of steps that the user will be following to the get the best route.
The user has to provide the following inputs:
1. Location - Current/ Enter a location (within USA)
2. Radius (in miles)
3. Categories (max of 5)

The data is then computed as follows:
1. Based on the input, the data will be filtered to get all the shops within the given radius.
2. Out of this data, the shops/service providers that fall into the given category of the input are filtered.
3. One shop/service provider out of the entire set is picked for every category. These are then fed to our algorithm i.e. Nearest Neighbor from Home.
4. The route is then finally computed using the nodes.

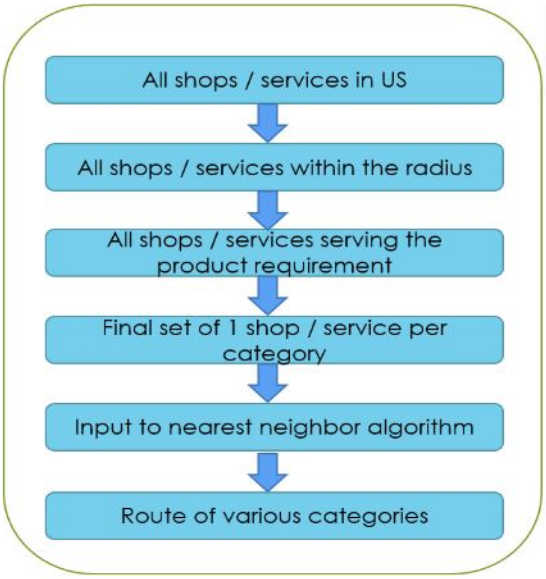

Fig. 3. Data Flow Diagram

VII. SERVICE ARCHITECTURE

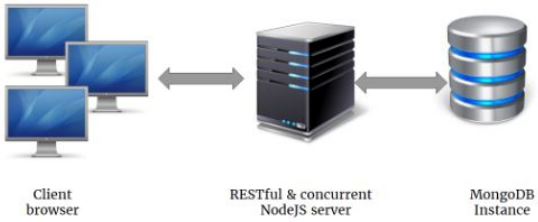

Fig. 4. Service Architecture

For the purpose of this project, we planned to design a scalable REST API [16] which can be used by other web or mobile clients. We also created a website which will consume this API and provide a service to the clients. Keeping in mind the huge dataset and the constraint of calculating geo-distances, we needed a database which was good in handling location related information. Previously, we did work on PostGreSQL [17] and ArcGIS [18]. But PostGreSQL has issues in reading the geojson files and ArcGIS was tough to integrate it to our rest of the application. MongoDb [19] solved both the problems of integration and also provides a high support geospatial queries [14].

A. Technologies used

1) NodeJS: NodeJS [20] is an runtime environment for JavaScript on the server side created in 2009. It is based on Google's V8 JavaScript engine. It makes use of the Reactor pattern and runs a single threaded event loop which delegates requests to multiple worker threads. Due to such event driven architecture it sports an asynchronous I/O providing high levels of concurrency and availability.

2) Rest API: Rest API [16][21] is nothing but a standardized way for communication between devices on the internet. It involves data transfer in the HTML, JSON, JS and XML protocols. HTTP is the protocol used on the internet to send and receive data. According to the RESTful conventions, there are HTTP verbs such as GET, POST, PUT and DELETE depending on the task to be performed. When everyone follows these conventions, it makes things easier and simpler to understand for other developers. In this project, we created a REST API using NodeJS JavaScript environment and Express application framework. This API was used by the website which fetched HTML pages and made Ajax calls to the NodeJS server.

3) MongoDB: MongoDB [19] is a NoSQL document based database which stores data in the BSON format (Binary JSON). It is famous for its fast writes, totally flexible schema design, simple queries, etc.
It has a high support for geospatial queries. There many operators which query on the basis of the location coordinates. It has the '$geoNear' operator which finds out the points near to a certain point within the specified radius. Such geo-spatial queries are extremely powerful. They are executed only if there is a 2dsphere index (geospatial index) on the document. A geospatial index creates a tree based data-structure from the co-ordinates making retrieval very easy.
MongoDB has a full proof aggregation framework called the Aggregation Pipeline [22]. It is used for processing data on multiple levels to get aggregated results. Some of the operators it supports are - $match, $group, $project, $sort, $unwind, $geonear. Internally an aggregate query is divided into chunks and processed in parallel giving a tremendous boost to the performance. It can be called as an alternate to MongoDB's map-reduce and sometimes may even give better results than the former.

Fig. 5. Given the fact of different clusters and select on point from every cluster and then compute the route for these data points.

```
Shop.aggregate([{
    $geoNear: {
        near: {type: "Point", coordinates: [lng, lat]},
        $maxDistance: radius,
        spherical: true,
        distanceField: "distance",
        includeLocs: "coordinates",
        distanceMultiplier: 0.000621371
    }
},{
    $match: {
        subcategory: {$in: product_types}}
},{
    $group: {
            _id: "$subcategory",
            coordinates: {$push: '$coordinates'},
            distances: { $push: '$distance' },
            ids: {"$push": '$_id'},
            names: {$push: '$name'},
            addresses: {$push: '$address'}
        }
    }
], function (err, data) {
  // callback
});
```

Fig. 6. Mongo Pre-Aggregation Code Snippet

B. Geohash

MongoDB requires a 2dsphere index to be created before any geo-spatial operation can be performed on the collection. Internally a 2dsphere index works on the concept of geohashing [26]. Geohashing is the technique of dividing the map into multiple the entire map into small quadrants and assigning a unique string of bits to each quadrant such that neighboring quadrants always have some part of their geohash common. This helps in geo-spatial queries, and thus can be used to reduce the search space. Using geohashes a tree based data structure called the R-tree is created which brings down geo-spatial searching to O(log(n)) complexity. Thus, geohashing is a very powerful technique which mongoDB makes full use of.

VIII. RESULT

Following are the screenshots of the images from our application.

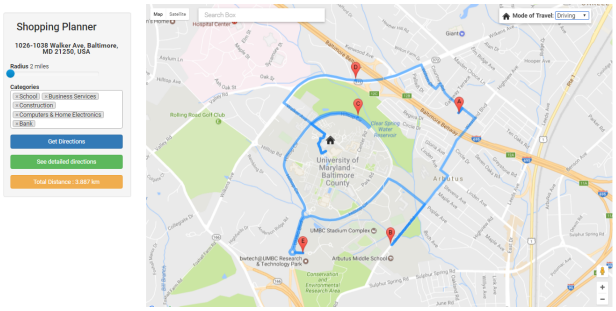

Fig. 7. User Interface - User Input and Map (driving)

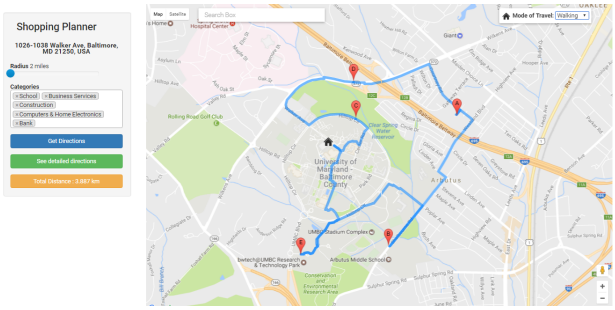

Fig. 8. User Interface - User Input and Map (walking)

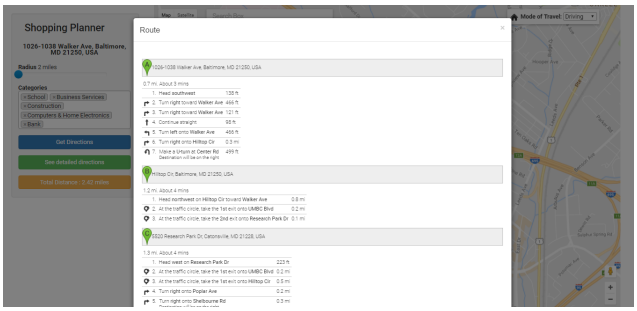

Fig. 9. User Interface - Detailed Navigation

IX. CONCLUSION

Thus, we were able to provide a service for users to get a descent shortest route according to their shopping lists ,i.e., selecting the shops offering the required services in the locality and then finding the shortest path to traverse these shops. The output route with detailed directions along with total distance was also displayed on map using Google API.

X. FUTURE WORK

The future scope of our project will be to consider the cost factor of the goods and services. The user will be able to provide the cost range for the services and then the results will be calculated considering the cost factor and also optimizing it on this factor. In addition to the route traversal, and displaying the route on map, the user will also be able to search for the previous route i.e. history of the routes will be maintained. Also, user's search history and travel history can be used to pick the service categories. This project can be used to solve the major vehicle routing problem and package delivery problem wherein parameters would be time, cost and order of delivery (prioritization).

References

[1] Traveling Salesman Problem, Wikipedia: *https://en.wikipedia.org/wiki/Travelling_salesman_problem*

[2] Traveling Salesman Problem, Google Developers: https://developers.google.com/optimization/routing/tsp

[3] Hamiltonian Path, Wikipedia: *https://en.wikipedia.org/wiki/Hamiltonian_path*

[4] Dataset, Archive.org: *https://archive.org/details/2011-08-SimpleGeo-CC0-Public-Spaces*

[5] Nearest Neighbor Algorithm, Wikipedia: *https://en.wikipedia.org/wiki/Nearest_neighbour_algorithm*

[6] Dijkstra's Algorithm, Wikipedia: *https://en.wikipedia.org/wiki/Dijkstra's_algorithm*

[7] 2-OPT Algorithm, Wikipedia: *https://en.wikipedia.org/wiki/2-opt*

[8] Simulated Annealing: *http://www.bookstaber.com/david/opinion/SATSP.pdf*

[9] Hill Climbing Algorithm: *http://www.cs.cornell.edu/gomes/selman-gomes-encycl-hillclimbing.pdf*

[10] Local minima, Quora: *https://www.quora.com/What-is-the-difference-between-local-minima-maxima-and-absolute-minima-maxima*

[11] Global minima, Quora: *https://www.quora.com/On-a-graph-whats-the-difference-between-global-and-local-maximum-minimum-values*

[12] University of Waterloo, TSP: *http://www.math.uwaterloo.ca/tsp/*

[13] GeoJSON: *http://geojson.org/geojson-spec.html*

[14] MongoDB GeoNear Reference Manual: *https://docs.mongodb.com/manual/reference/command/geoNear*

[15] Google Maps API: *https://developers.google.com/maps/*

[16] REST API: *http://searchcloudstorage.techtarget.com/definition/RESTful-API*

[17] PostGreSQL: *https://www.postgresql.org/*

[18] ArcGIS: *https://en.wikipedia.org/wiki/ArcGIS*

[19] MongoDB: *https://www.mongodb.com/*

[20] NodeJS: *https://nodejs.org/en/*

[21] REST API, Wikipedia:
https://en.wikipedia.org/wiki/Representational_state_transfer
[22] MongoDB Aggregation Pipeline:
https://docs.mongodb.com/manual/core/aggregation-pipeline/
[23] Siddhartha Jain, M. M. (2010, Fall). Parallel Heuristics for TSP on Map Reduce. Retrieved from Brown University: http://cs.brown.edu/courses/csci2950-u/s14/papers/tsp.pdf
[24] Stuart J. Russell, P. N. (n.d.). Artificial Intelligence, A modern approach (3rd Edition). Pearson.
[25] NPM. (n.d.). Retrieved from Library to perform geo specific tasks: https://www.npmjs.com/package/geolib
[26] Solution Methods for VRP. (n.d.). Retrieved from Networking and Emerging Optimization: http://neo.lcc.uma.es/vrp/solution-methods
[27] MongoDB GeoHashing:
https://docs.mongodb.com/v3.2/core/geospatial-indexes/