**Compare methods for efficient computation of shortest distance queries on road networks**

Chao Fang, Xiangyu Zhang

**Abstract**

The computation of shortest distance or route planning problem on road networks is one of the widely studied topic in recent years. Algorithms were devised aiming for solving various problems and satisfying performance requirements. The goal of this project is to find an efficient way to calculate the shortest distance on road networks. We will compare the different algorithm to find the shortest path. We will also try different indexing methods and compare the benchmark. The final model should be able to give a quick response to distance queries, in a dynamically updated database with traffic.

**1. Introduction and motivation**  
There are many methods to find the shortest path in a graph. The most famous one, dijkstra, can always give a correct answer to a shortest path query. However, the time complexity of running dijkstra is O(V^2). With the help of a heap queue, we can reduce it to O(ELogV).

For a small graph, this approach seems viable. But for a large graph with thousands or millions of nodes and edges, dijkstra might be very inefficient.

Indexes can be used to reduce the query time, but on the other hand, they also add additional computing for making the indexes, and additional space to store the indexes. So when using indexes, we should also consider the indexing time and index size. A good approach should have demanded performance on both query time, index time and index size.

**2. Problem definition**   
We will use the database of road network of california, to test the performance of some selected methods for computing the shortest distance in a graph. The database contains 21047 nodes and 43386 edges. The selected methods are: dijkstra, random-landmark-based-index, pruned landmark labeling, presorted pruned landmark labeling, hop doubling label indexing.

**3. Approach and method description**

**(1) Dijkstra**

This is the most straightforward solution for a query, we implement this method as a comparison of our other method. Dijkstra should have a slowest query time, but do not need a index.

**(2) Random Selected Landmarks**

This is an approximate method, which means it does not always returns a correct answer. For a dataset, to get all the node indexed, suppose we use dijkstra, we need O(V\*E\*log(V)) time to generate the index, which will cost a large amount of time. The index size will be huge as well. So this method reduce the index time and size by using only some selected "landmarks" to create the index. During a query of (u,v), it checks all the label L(i) of landmarks, and compute the shortest distance by d(u,v) = Min(d(u, i) + d(v, i)) for all i in selected landmarks.

The output is the correct answer when at least one of the landmarks is on the shortest path, if none of the landmarks is on the path, the result will be a approximate value which is larger than the real value.

The error of this method is very small when the distance of the two points in the query is very far from each other. However, when they are very close to each other, the error could be huge.

The performance of this method also depends on the number of landmarks selected. The more landmarks, the more accurate, and, on the other hand, need more indexing time and size.

**(3) Pruned Landmark Labeling**

This method is described in [1]:For every node u, we select a set of node C(u) and precompute the distance from u to the node in C(u), then if we make sure each pair of node (u, v) has at least on node m in both C(u) and C(v), and it is on the shortest path on path(u, v), then we can compute the d(u, v) as: min(d(u,m) + d(v,m)), where m is the nodes that C(u) and C(v) shared.

To create the precomputed labels, if we use BFS to visit each node on the graph, it will be very inefficient because some of the work are redundant. If we are visiting a node u started from v, and we have a node w that we have precomputed label, and we have d(u, v) = d(v, w) + d(w, u). Then, we do not need to traverse any edge from u.

This method largely decreased the computation needed for indexing and the size of the index by removing a large amount of unneeded visit. Compare to the random selected landmarks method, it always returns the correct answer. And, compare to index all nodes by BFS or dijkstra, it has much shorter index time and smaller index size.

**(4) Presorted Pruned Landmark Labeling**

To further improve the performance of pruned landmark labeling, we sort the nodes by their degrees before indexing. By doing this, we first index the nodes that has more edges. Those nodes has more probability of on the shortest path of another pair of node. If we precompute them first, we will have more "prunes" in our method.

**(5) Hop Doubling Label Indexing**

Hop doubling label indexing [2] is a indexing method tends to solve distance querying problems in massive scale-free networks. This indexing method produces smaller label size and would give acceptable bounded complexity on either computation time and required memory space to store constructed labels. The main algorithm, Hop-Doubling, is made of three major components:

1. Label index construction via label entry generation and label pruning.

2. I/O efficient algorithm for implementing the iterative process.

3. Hop-Stepping for enhancement

For this project, we were mainly focused on implementing label index construction and label pruning.

First of all, the Hop-Doubling algorithm ranks all vertices according to non-increasing degree. Then it generates label entries to cover shortest paths. A label entry is a pair (v, d) where v is a vertex and d is a distance value. Each vertex is assigned with two labels, label\_in and label\_out, and each label contains a set of label entries, representing the distance between this label to target label. The label entry generation is based on 4 specific rules given by Hop-Doubling algorithm. The ordering of vertices plays an important role in applying rules to label generation. For example, a label entry (v, d) is added to label\_out(u) if there exists a path from u to v with distance d, where the degree of u is less than degree of v. Similarly, a label entry (u1, d2) is added to label\_in(u) if there exists a path from u1 to u, where the degree of u is less than degree of u1. In this case, another label entry (u1 -> u) is added to both label\_in(u) and label\_out(u1) based on the Rule 1.

Besides label entry generation, label pruning is implemented to reduce the label size. It basically checks all the new generated labels that if the new generated path is not the current shortest path. In this case, the checked label entry is pruned.

Although Hop-Doubling algorithm has powerful and efficient label index construction and pruning functionalities, it is not effective dealing with road networks. Because that label entry generation is heavily depending on the ordering of vertices, while road networks typically do not have high degree vertices. The similarity in degree of each vertex leads to lower frequency of label entry generation.

We implement Hop-Doubling algorithm and attempt to run distance query with Hop-Doubling on California road network. The result shows that Hop-Doubling ends up with the lowest performance, which significantly lower than the performance of other methods in both indexing time and searching time. Therefore, in section 4 we only compare the performance between Dijkstra and three landmark labeling algorithms.

**4. Results**

**(1) Environment and code**

To test the those methods, we used the following test environment:

OS: Mac OS

CPU: Intel(R) Core(TM) i7-4870HQ CPU @ 2.50GHz

Memory: 16GB

Graphs are using the same design for each methods. The graph class and dijkstra method is in dijkstra.py, the random selected landmarks method is in random\_landmark.py. The pruned landmark labeling method is in pruned\_landmark.py, the presorted method is in sort\_by\_degree.py.

**(2) Query time**

For query (0, 10000):

|  |  |
| --- | --- |
| Methods | Query Time |
| Dijkstra | 104.05898094177246ms |
| Random Landmark | 1.1691383361816406ms |
| Pruned Landmark Labeling | 0.10895729064941406ms |
| Pruned Landmark Labeling(pre-sorted) | 0.119052352905ms |

The query time for dijkstra is significantly longer than the other methods. The Random landmark is slightly longer than the two purned methods.

**(3) Index time and size**

|  |  |  |
| --- | --- | --- |
| Methods | Index Time | Index Size |
| Dijkstra | N/A | N/A |
| Random Landmark (500 landmarks) | 66.39699816703796s | 456MB |
| Pruned Landmark Labeling | 325.66018295288086s | 166MB |
| Pruned Landmark Labeling(pre-sorted) | 263.763673067s | 134MB |

The random landmark shows a very bad performance on index, it only uses 500 landmarks but occupies a very large space. Its index time and size depends largely on the numbers of landmarks it chooses. Also, this is the only method that does not guarantee a exact result. It performs badly when the queried two point are close. As shown below.

The pruned landmark labeling performs relative good. For this dataset, the about 300s index time means we can update the index every 5 min, according to the traffic, which is acceptable.

**(4) Accuracy**

|  |  |
| --- | --- |
| Methods | Accuracy |
| Dijkstra | 100% Accurate |
| Random Landmark | 2200% larger than real value for query(0,10) by average, 0.1% larger than real value for query(0,10000) by average. |
| Pruned Landmark Labeling | 100% Accurate |
| Pruned Landmark Labeling(pre-sorted) | 100% Accurate |

Compare to the other three exact methods, Random Landmark performs very bad for queries on short distances. As a conclusion, it can only works in cases of most of the queries are long distance query.

**5. Conclusion**

For the Road Network of California dataset, Pruned Landmark Labeling with pre-sorting performs the best, It has a much shorter query time than dijkstra, and also have a comparatively small size of index since the "prune" operation largely reduced the index size and index time. The index time is sufficient to update the index according to the traffic. Compare to the statistics in [1], the pre-sorting operation has limited improvement on this dataset, we think it is because the this dataset has too much nodes with same degree(most of nodes have degree of 2).

**References**

[1] Takuya Akiba, Yoichi Iwata and Yuichi Yoshida. Fast Exact Shortest-Path Distance Queries on Large Networks by Pruned Landmark Labeling *arXiv:1304.4661 [cs.DS]*

[2] Minhao Jiang, Ada Wai-Chee Fu, Raymond Chi-Wing Wong, Yanyan Xu. Hop Doubling Label Indexing for Point-to-Point Distance Querying on Scale-Free Networks *arXiv:1403.0779 [cs.DB]*