Categorical Range queries on Spatial Networks

Rahul Saladi, Akash Agarwal and Xiaofei Zhao

October 25, 2011

1 Introduction

Rapid growth of smart phones, GPS technologies and online mapping services (eg, Google Maps, Microsoft MapPoint) has made it possible to answer some fundamental location based queries. A lot of work has been done on $k$-NN (nearest neighbour) queries and shortest-path queries. Surprisingly very little work has been done on range queries or its generalized version called categorical range queries. Categorical range queries are the focus of this work.

A visitor comes over to Minneapolis to spend his vacation. Being new to the city he would want to ask some interesting queries on his smart phone such as: ‘Which are the movie theaters within 10 kms from my current location ?” or ‘Which are the restaurants within 2 kms of Keller Hall ?” Note that when the visitor says report theaters within 10 kms, he does not mean euclidean distance. In a realistic scenario, he can only walk along the road network of Minneapolis and not in arbitrary directions. Therefore, by 10 kms he means that the theaters of interest should be less than 10 kms away from the current location while moving along the network. Refer to Figure 1 for a graphical description.

2 Problem Statement

Categorical range search query on a spatial network We have a spatial network $G = (V, E)$, where $V$ represents the set of nodes/vertices (i.e. intersection points of roads) and $E$ is the set of edges connecting any two vertices. With each edge $e \in E$, we shall associate a weight $w(e)$ which is the length of the edge. A set $F$ of facilities lie along the edges of the network. The set of facilities $F$ are categorized into $l$ disjoint groups s.t. $F = F_1 \cup F_2 \ldots \cup F_l$ and $F_i \cap F_j = \Phi$, $i \neq j$ and $\forall i, j \in [1, l]$. The user gives a two-dimensional query point $q$, a query value $d_q$ and facility type/category $F_i$. The objective is to report all the facilities in $F_i$ which are within a network distance of $d_q$ from $q$. The network is not restricted to be planar.
Figure 1: An example of a road network. The black dots denote the vertices on the network (intersection of roads). All the edges have unit length. The red and the blue circles denote the theaters on the network. Suppose our visitor is at vertex $q$ and wants to know all the theaters within a distance of $3\sqrt{2}$ units (the diameter of the network is $3\sqrt{2}$). If the distance metric was ‘euclidean distance’ then all the theaters will be reported. However, in a realistic scenario we can only move along the edges of the network. If we consider ‘network distance’ then all the theaters colored blue can be reached. All the red colored theaters have a network distance greater than $3\sqrt{2}$ from $q$. 
3 Limitation of existing techniques

The work in [24] presents the first known index structure and algorithms for spatial queries such as range queries, nearest neighbours, spatial joins and closest pairs using the network distance instead of the Euclidean distance. For the range query they propose two algorithms: a) Range Euclidean Restriction (RER), and b) Range Network Expansion (RNE).

In RER technique, first a circular range query is performed on \( S \) to retrieve a subset \( S' \) of facilities which lie within a Euclidean distance of \( d_q \) from the query point \( q \). This helps in taking care of false misses. From the query point \( q \), we shall be expanding along the network to find out all the edges in the network which are within a ‘network distance’ of \( d_q \) from \( q \). All the edges are visited in increasing order of their ‘network distance’ (similar to Djikstra’s algorithm). For each edge visited, all the facilities in \( S' \) which have an overlap with the edge are reported. Clearly we can construct inputs for which RER technique will fail miserably. In an extreme case it might happen that all the facilities in set \( S \) fall within Euclidean distance of \( d_q \) from \( q \), but none of them fall with network distance of \( d_q \) from \( q \). In such a case no facility will get reported but it might happen that we explore \( O(|E||S|) \) edges in the network in the process. The overall query time would be \( O(|E||S|) \) (ignoring the cost of maintaining a queue) and we end up reporting none of the facilities ! Refer to Figure 3.

In RNE technique they first compute a set \( QS \) of all the edges in the network which are within a network distance of \( d_q \) of from \( q \). The facilities are indexed in an R-tree. An intersection join is performed between \( QS \) and the facilities R-tree which is shown to be optimal. Alternate techniques for performing the join are also mentioned. Though the join operation is efficient, the first step of finding the set \( QS \) is the major bottleneck of this process. In an extreme scenario, it might happen that \( O(|E|) \) edges in the network might be within a network distance of \( d_q \) from \( q \) but none of the facilities lie on them ! Refer to Figure 3.

There a couple of other works [29] which claim to handle range queries based on network distance but they do not provide any concrete theoretical bounds on the space and the query time requirements.

4 Other Related Work

Euclidean space For static objects there has been extensive work done on range searching queries in the database and computational geometry community. Orthogonal range searching query is perhaps one of the most widely studied problem in the computational geometry and database community due to its wide range of applications. Here the set of objects are points lying in a \( d \)-dimensional space and the query is an axis-parallel orthogonal query box.

Traditionally, the researchers in computational geometry have been interested in building solutions for this problem which aim at coming up worst-case efficient query time solutions, i.e., ensuring that the query time is low
Figure 2: An example to show the limitations of RER and RNE techniques proposed in [24] for answering range queries on spatial networks. Assume that all the edges are unit length. The facilities are shown in blue and the vertices in black. The query point is $q$ and the query distance is 2 units. In RER, the circular range query around $q$ with radius two units would report all the facilities as candidate points. From $q$ all the edges within network distance of 2 units are explored but unfortunately none of these edges will have facilities on them. Similarly, in RNE the first step is to find out all the edges in the network which are within a network of 2 units from $q$. As stated before none of the edges explored will have a facility lying on them!
for all possible values of the query. The most popular among them are the range-trees [10] and the K-d tree [6]. The original range-tree when built on \( n \) points in \( d \)-dimensional space took \( O(n \log^d n) \) space and answered queries in \( O(\log^{d-1} n + k) \) query time, where \( k \) are the number of points lying inside the query region \( q \). By using the technique of fractional cascading we can reduce the query time by a log factor [8]. In a dynamic setting, the range-tree uses \( O(n \log^{d-1} n) \) space, answers queries in \( O(\log^{d-1} n \log \log n) \) time and handles updates in \( O(\log^{d-1} n \log \log n) \) time [8]. A K-d tree when built on \( n \) points in \( d \)-dimensional space takes up linear space and answers a range-query in \( O(n^{1-1/d} + k) \) time. Updates can be handled efficiently in it.

In the field of databases there have been a significant number of index structures proposed to answer an orthogonal range-query. The researchers in database community try to come up with practical solutions which are optimized to work well for the average case queries (with the assumption that the worst-case query will occur rarely). Perhaps no other structure has been more popular than the R-tree proposed in 1984 by Guttman [17]. According to Google scholar, as on 16th June 2011 this paper has been cited 5260 times! R-trees is extremely popular for indexing multi-dimensional information. It’s main advantage arises from the fact that it can handle multiple queries efficiently (range-queries being one of them). Contrast this with a range-tree which is tailor made for handling only range-queries. Variants of R-tree such as R* tree [5], R+ tree [30], Hilbert tree and Priority R-tree are also quite popular index structures for storing multi-dimensional data.

Circular range searching [7] and Halfspace range searching [7] are some other important range queries where the point set are queried with a disk and a halfspace, respectively.

**Range queries on moving objects** In the field of moving object databases, there has been work on indexing moving objects which can answer range queries or window queries. However, most of these assume the objects to have an unconstrained two-dimensional movement. A couple of index structures such as FNR-tree [11], MON-tree [9] and [25] assume the objects to be constrained to move along the spatial network and appropriately model the index structures. These structures store the past history of movement of all the objects and range queries are asked on this past data. However, the range queries which they handle are different than what we are attempting in this paper. The range query which they want to answer is the following: Given a spatio-temporal window, return all the objects whose movements overlap the query window. So in some sense they still consider euclidean metric distance.

**Highway dimension** Recently there was a paper [1] which introduced the notion of highway dimension. They observed that there were a lot of heuristics for finding shortest-path among vertices which are based on a wide variety of ideas, such as arc flags [21, 18, 4], A* search with landmarks [15], highway hierarchies [27, 28], reach [14, 13, 16], transit nodes [2, 3], and contraction hierarchies [12]. In experiments on real-world data these heuristics performed a lot better than a plan Dijkstra. Also the preprocessing involved in them is practical and space occupied was only slightly more than than the road network
alone. However, these were all purely experimental with no good theoretical bounds. It is possible to construct inputs for which these heuristics will perform miserably.

In [1] they tried to bring all these techniques within a theoretical framework and tried to identify factors which lead to good performance of these heuristic techniques. In this process they define the notion of highway dimension and showed that having ‘small’ highway dimension gives a theoretical guarantee of good query performance for these heuristic algorithms. Interestingly, the line of approach of this work seems promising for coming up with good theoretical guarantees for algorithms which perform range queries on spatial networks.

**Other problems** Apart from the shortest path query, the nearest neighbour or $K$-nearest neighbour query in terms of network distance happens to be an active field of research [31, 19, 26, 23]. It looks possible that we can borrow ideas from shortest path and K-nearest neighbour algorithms for answering the range query problem.

In [32] they provide various techniques to estimate the number of nodes and edges which lie within $d_q$ distance of the query point $q$ which might help us in predicting the space and query time requirements for range queries. In some scenarios users of location based services would not want to reveal their local information. In [20] they come up with nearest neighbour and range search queries on spatial network with network distance without revealing the local information of the user. In this work we are looking at static range queries but there has been work done on dynamic range queries in spatial networks [22].

## 5 Our Technique

In this section we shall present an initial solution we could come up with for this problem. We start off with a couple of definitions.

**Definition 1 Neighbour of a vertex** Suppose we are interested in facility type $F_i \in F$. A vertex $v \in V$ and a facility $f \in F_i$ are neighbours iff there is a path $P_{vf}$ from $v$ to $f$ which does not have any other facility on $P_{vf}$. Then $f$ becomes a neighbour of $v$.

**Definition 2 Neighbourhood of a vertex w.r.t $F_i$** For a given facility type $F_i$, the neighbourhood of a vertex $v$ (denoted by $n_i(v)$) are the set of facilities in $F_i$ which are its neighbours.

**Definition 3 Neighbourhood of a facility** For a given facility type $F_i$, the neighbourhood of a facility $f \in F_i$ (denoted by $n_i(f)$) are the set of facilities in $F_i \setminus f$ which are its neighbours.

### 5.1 Preprocessing steps

For each facility type $F_i \in F$ the following steps are carried out:
1. For each vertex \( v \in V \), identify its neighbours i.e. \( n_i(v) \). Maintain this list in increasing order of their network distance from \( v \).

2. We shall build a Facility-Neighbourhood graph for \( F_i \) (say \( FNG_i \)) as follows: The vertices of \( FNG_i \) are the facilities in set \( F_i \). An edge is made between vertices \( u \) and \( v \) if they are neighbours. The weight of the edge is the path \( P_{uv} \in G \) which does not have any other facility on \( P_{uv} \) and has the minimum possible weight (weight of a path is the sum of the weights of the edges on it). The edges incident on a vertex are stored in increasing order of their weights.

**Space requirements** The input graph \( G \) occupies \( O(|V| + |E| + |F|) \) space. Let \( N_i = \sum_{v=1}^{|V|} |n_i(v)| \). Let \( N = \sum_{i=1}^{|F|} N_i \). The space occupied by \( FNG_i \) is \( F_i = \sum_{i=1}^{|F|} n_i(f) \). Let \( F = \sum_{i=1}^{|F|} F_i \). Therefore, the total space occupied by the data structure is \( O(|V| + |E| + |F| + N + F) \). If we assume the graph to be a road network then the space complexity would be \( O(N + F) \).

5.2 Query Algorithm

Suppose the query is to find out all the facilities of type \( F_i \) which are within a network distance of \( d_q \) from the given query point \( q \). For simplicity assume that \( q \in V \). This assumption can be easily removed later. The following steps shall be followed:

Figure 3: This networks represents an example where the algorithm proposed would work really well. The blue circles represent the facilities. The number of neighbours for each vertex and each facility is exactly two which makes our algorithm work well.
Figure 4: The blue circles represent facilities. The black circles represent vertices. Here each vertex has each facility as its neighbour. Also each facility has every other facility as its neighbour. This makes our algorithm perform poorly.

1. The list $n_i(q)$ is scanned to identify the facilities lying within network distance $d_q$. These facilities are pushed into a queue $Q$.

2. Now we shall run a Dijkstra’s style algorithm on $FNG_i$. However, unlike in the traditional Dijkstra where we start with an empty queue, we shall here start with the entries in $Q$. We shall keep exploring the graph till we cross the network distance $d_q$.

Query time The second step dominates the running time. The time taken to run a Dijkstra’s like algorithm would be $O(\sum_{j=1}^{k} n_i(f_j))$ where $f_1, f_2, \ldots, f_k$ are the facilities reported.

References


