The Optimal Sequenced Route Query

Mehdi Sharifzadeh, Mohammad Kolahdouzan, Cyrus Shahabi
Computer Science Department
University of Southern California
Los Angeles, CA 90089-0781
[sharifza, kolahdoz, shahabi]@usc.edu

ABSTRACT

Several variations of nearest neighbor (NN) query have been investigated by the database community. However, real-world applications often result in the formulation of new variations of the NN problem demanding new solutions. In this paper, we study an unexploited and novel form of NN queries named *Optimal Sequenced Route (OSR)* query in both vector and metric spaces. OSR strives to find a route of minimum length starting from a given source location and passing through a number of *typed* locations in a specific sequence imposed on the types of the locations. We first transform the OSR problem into a shortest path problem on a large planar graph. We show that a classic shortest path algorithm such as Dijkstra's is impractical for most real-world scenarios. Therefore, we propose LORD, a light threshold-based iterative algorithm, that utilizes various thresholds to filter out the locations that cannot be in the optimal route. Then we propose R-LORD, an extension of LORD which uses R-tree to examine the threshold values more efficiently. Finally, LORD and R-LORD are not applicable in metric spaces, hence we propose another approach that progressively issues NN queries on different point types to construct the optimal route for the OSR query. Our extensive experiments using both real-world and synthetic datasets verify that our algorithms significantly outperform the Dijkstra-based approach in terms of processing time (up to two orders of magnitude) and required workspace (up to 90% reduction on average).

1. INTRODUCTION

A nearest neighbor query is defined as finding the object(s) with the shortest distance(s) to a query point. Although this type of query is useful, but more often, a user intends to make a plan for a trip to *several* (and possibly different types of) locations in some *sequence*, and is interested in finding the optimal route that minimizes her total traveling in distance or time. Besides commercial applications such as navigation devices in vehicles or online map services, where this type of queries has great demand and numerous benefits, this query is of absolute importance in crisis management and defense/intelligense systems, where being able to respond to a series of incidents in fastest time is vital. In this paper, we introduce and address this type of query in spatial databases.

1.1 Motivation

Suppose that we are planning a car trip in town as following: first we intend to leave home toward a gas station to fuel the car, then we plan to stop by a library branch to check in a book, and finally, we need to go to a post office to mail a package. Naturally, we prefer to drive the minimum overall distance to these destinations. In other words, we need to find the locations of the gas station g_i, the library branch l_j, and the post office p_k, which driving toward them considering the sequence of the plan shortens our trip (in terms of distance or time). We call this the *Optimal Sequenced Route* or *OSR*.

Using Figure 1, we can show that this query may not be optimally answered by simply performing a series of independent nearest neighbor queries at different locations. The figure shows a network of equally sized connected squares, three different types of *point sets* shown by white, black and gray circles, which represent gas stations, libraries, and post offices, respectively, and a starting point p (shown by $\triangle$).

A greedy approach to solve our query is to first locate the closest gas station to p, g_2, then find the closest library to g_2, l_2, and finally find the closest post office to l_2, p_2. Assuming the length of each edge of the squares is 1 unit, the length of the route specified by this greedy approach, (p, g_2, l_2, p_2), shown by dotted lines in the figure, is 15 units. However, the route (p, g_1, l_1, p_1) (shown with solid lines in the figure) with the length of 12 units is the optimum answer to our query. Note that g_1 in not the closest gas station to p and l_1 is actually the farthest library to g_1. This shows that the optimum result for our specific query can be substantially different from what a greedy approach would suggest.

1.2 Uniqueness

To the best of our knowledge, although different variations of nearest neighbor queries have been extensively studied by the database research community, no one has explored the problem of *Optimal Sequenced Route (OSR) Query*. This problem is closely related to the *Traveling Salesman Problem (TSP)*. TSP asks for the minimum cost round-trip route from a starting point to a given set of points. As a classic problem in graph theory, TSP is the search for the *Hamiltonian cycle* with the least weight in a weighted graph. With TSP, all the points in the set are participating in the route and the sequence in which the points must be visited is requested. In contrast, OSR enforces a specific sequence to find the appropriate points from a number of point sets.

The most similar TSP-related problem to OSR is *Sequential Ordering Problem (SOP)* in which a Hamiltonian path with a specific node precedence constraint is needed. Similar to all TSP variations, the solution path to SOP must still pass through *all* the given points. Conversely, the main challenge with OSR is to efficiently select a sequence of points,

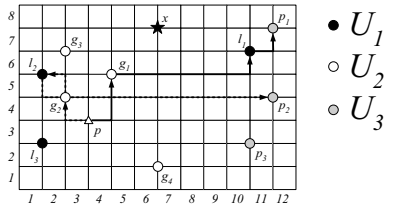


Figure 1: A network with three different types of point sets

where each of which can be any member of a given point set. The commercial online Yellow Pages such as those of Yahoo! and MapQuest can only search for the k-nearest neighbors in *one* specific category (or point set) to a given query location and cannot find the optimal sequenced route from the query to a *group* of point sets.

1.3 Contributions

In this paper, we introduce and formally define the problem of OSR query in spatial databases. We also propose alternative solutions to the OSR queries for both vector and metric spaces.

For vector spaces, we first propose a solution that is based on generating a weighted directed graph from the input road network, and then utilizing Dijkstra's algorithm to find the distances from a starting point to all possible end points on the generated graph. This solution becomes impractical when the generated graph is large, which is the case for most real-world problems. Hence, we propose a second solution, LORD, that utilizes some threshold values to filter out the points that cannot possibly be on the optimal route, and then generates the optimal route in reverse sequence (i.e., from ending to the starting point). We then propose R-LORD, which is an optimization of LORD by transforming the concept of the thresholds in LORD to some range queries and performing the range queries using an R-tree index structure. Finally, we propose PNE to address OSR queries in metric spaces. PNE is based on progressively finding the nearest neighbors to different point sets in order to construct the optimal route from the starting to the ending point.

We also discuss two variations of OSR queries: 1) when the query must end in a specific point, and 2) when more than one optimum route is requested; and show how our proposed solutions can address these variations of OSR as well. Finally, through extensive experiments with both real-world and synthetic datasets, we show that R-LORD can efficiently answer an OSR query, scale to large datasets, and perform independently from the distribution and density of the data.

The remainder of this paper is organized as follows. We first formally define the problem of OSR queries and the terms we use throughout the paper in Section 2. In Section 3, we discuss our alternative solutions for OSR queries in vector and metric spaces. We address two variations of OSR query in Section 4. The performance evaluation of our proposed algorithms is presented in Section 5. The related work to OSR and similar nearest neighbor queries are presented in Section 6. Finally, we conclude the paper and discuss our future work in Section 7.

2. FORMAL PROBLEM DEFINITION

In this section, we describe the terms and notations that we use throughout the paper, formally define the OSR query, and discuss the unique properties of OSR that we utilize in our solutions.

2.1 Problem Definition

Let $U_1, U_2, \ldots, U_n$ be n sets, each containing points in a d-dimensional space $\mathbb{R}^d$, and $D(.,.)$ be a distance metric defined in $\mathbb{R}^d$ where $D(.,.)$ obeys the triangular inequality. To illustrate, in the example of Figure 1, U_1, U_2, and U_3 are the sets of black, white, and grey points, representing libraries, gas stations and post offices, respectively. We first define the following five terms.

Definition 1: Given n, the number of point sets U_i, we say $M = (M_1, M_2, \ldots, M_m)$ is a *sequence* if and only if $1 \leq M_i \leq n$ for $1 \leq i \leq m$. That is, given the point sets U_i, a user's OSR query is valid only if she asks for existing location types. For the example of Figure 1 where $n = 3$, $(2, 1, 2)$ is a sequence (specifying a gas station, a library, and a gas station) while $(3, 4, 1)$ is not because 4 is not an existing point set.

Definition 2: We say $R = (P_1, P_2, \ldots, P_r)$ is a *route* if and only if $P_i \in \mathbb{R}^d$ for each $1 \leq i \leq r$. We use $p \oplus R = (p, P_1, \ldots, P_r)$ to denote a new route that starts from starting point p and goes sequentially through P_1 to P_r. The route $p \oplus R$ is the result of adding p to the head of route R.

Definition 3: We define the *length* of a route $R = (P_1, P_2, \ldots, P_r)$ as

$$L(R) = \sum_{i=1}^{r-1} D(P_i, P_{i+1}) \quad (1)$$

Note that $L(R) = 0$ for $r = 1$. For example, the length of the route (g_2, l_2, g_3) in Figure 1 is 4 units where D is the Manhattan distance.

Definition 4: Let $M = (M_1, M_2, \ldots, M_m)$ be a sequence. We refer to the route $R = (P_1, P_2, \ldots, P_m)$ as a *sequenced route* that follows sequence M if and only if $P_i \in U_{M_i}$ where $1 \leq i \leq m$. In Figure 1, (g_2, l_2, g_3) is a sequenced route that follows $(2, 1, 2)$ which means that the route passes only through a white, then a black and finally a white point.

Definition 5: Given a starting point p, a sequence $M = (M_1, \ldots, M_m)$, and point sets $\{U_1, ..., U_n\}$, we refer to $R_g(p, M) = (P_1, \ldots, P_m)$ as the *greedy sequenced route* that follows M from point p if and only if it satisfies the followings:

1. P_1 is the closest point to p in U_{M_1}, and
2. For $1 \leq i < m$, P_{i+1} is the closest point to P_i in $U_{M_{i+1}}$.

It is clear that $R_g(p, M)$ is unique for a given point p, a sequence M, and the sets U_i. Moreover, by definition, the optimal sequenced route R is never longer than the greedy sequenced route for the given sequence M, i.e., $L(p, R) \leq L(p, R_g(p, M))$.

We now formally define the OSR query.

Definition 6: Assume that we are given a sequence $M = (M_1, M_2, \ldots, M_m)$. For a given *starting* point p in $\mathbb{R}^d$ and

Symbol	*Meaning*
U_i	a *point set* in $\mathbb{R}^d$
$\|U_i\|$	cardinality of the set U_i
n	number of point sets U_i
$D(.,.)$	distance function in $\mathbb{R}^d$
M	a sequence, $= (M_1, \ldots, M_m)$
$\|M\|$	m, size of sequence M = number of items in M
M_i	i-th member of M
R	route $(P_1, P_2, \ldots, P_r)$, where P_i is a point
$\|R\|$	r, number of points in R
P_i	i-th point in R
$L(R)$	length of R
$p \oplus R$	route $R_p = (p, P_1, \ldots, P_r)$ where $R = (P_1, \ldots, P_r)$
$L(p, R)$	length of the route $p \oplus R$

Table 1: Summary of notations

the sequence M, the **Optimal Sequenced Route** (OSR) Query, $Q(p, M)$, is defined as finding a sequenced route R that follows M where the value of the following function L is minimum over all the sequenced routes that follow M:

$$L(p, R) = D(p, P_1) + L(R) \quad (2)$$

Note that $L(p, R)$ is in fact the length of route $R_p = p \oplus R$. Throughout the paper, we use $Q(p, M) = (P_1, P_2, \ldots, P_m)$ to denote the *optimal SR*, the answer to the OSR query Q. For the example in Section 1.1 where $(U_1, U_2, U_3) = (black, white, gray)$, $M = (2, 1, 3)$, and D is the shortest path, the answer to the OSR query is $Q(p, M) = (g_1, l_1, p_1)$. We use *candidate* SR to refer to all other sequenced routes that follow sequence M.

Table 1 summarizes the notations we use throughout the paper.

2.2 Properties

Before describing our algorithms for OSR queries, we present the following three properties which are exploited by our algorithms.

Property 1: For a route $R = (P_1, \ldots, P_i, P_{i+1}, \ldots, P_r)$ and a given point p, we have

$$L(p, R) \geq D(p, P_i) + L((P_i, \ldots, P_r)) \quad (3)$$

Proof: The triangular inequality implies that $D(p, P_1) + \sum_{j=1}^{i-1} D(P_j, P_{j+1}) \geq D(p, P_i)$. Adding $\sum_{j=i}^{r-1} D(P_j, P_{j+1}) = L((P_i, \ldots, P_r))$ to both sides of the inequality and considering the definition of the function $L()$ in Equation 2, yields Equation 3.□

As we will show in Section 3.2.1, we utilize property 1 to narrow down the candidate sequenced routes for $Q(p, M)$ by filtering out the points whose distance to p is greater than a threshold, and hence cannot possibly be on the optimal route. Note that this property is applicable to *all* routes in the space.

The answer to the OSR query $Q(p, M)$ demonstrates the following two unique properties. We utilize these properties to improve the exhaustive search among all potential routes of a given sequence.

Property 2: If $Q(p, M) = R = (P_1, \ldots, P_{m-1}, P_m)$, then P_m is the closest point to P_{m-1} in U_{M_m}.

Proof: The proof of this property is by contradiction. Assume that the closest point to P_{m-1} in U_{M_m} is $p_x \neq P_m$. Therefore, we have $D(P_{m-1}, p_x) < D(P_{m-1}, P_m)$ and hence $L(p, (P_1, \ldots, P_{m-1}, p_x)) < L(p, (P_1, \ldots, P_{m-1}, P_m))$. This contradicts our initial assumption that R is the answer to $Q(p, M)$.□

Property 2 states that given that $P_1, \ldots, P_{m-1}$ are subsequently on the optimal route, it is only required to find the first nearest neighbor of P_{m-1} to complete the route and subsequent nearest neighbors cannot possibly be on the optimal route and hence, will not be examined. Note that this property does not prove that the greedy route is always optimal. Instead, it implies that *only* the last point of the optimal sequenced route R (i.e., P_m) is the nearest point of its previous point in the route (i.e., P_{m-1}).

Property 3: If $Q(p, M) = (P_1, \ldots, P_i, P_{i+1}, \ldots, P_m)$ for the sequence $M = (M_1, \ldots, M_i, M_{i+1}, \ldots, M_m)$, then for any point P_i and $M' = (M_{i+1}, \ldots, M_m)$, we have $Q(P_i, M') = (P_{i+1}, \ldots, P_m)$.

Proof: The proof of this property is by contradiction. Assume that $Q(P_i, M') = R' = (P'_1, \ldots, P'_{m-i})$. Obviously $(P_{i+1}, \ldots, P_m)$ follows sequence M', therefore we have $L(P_i, R') < L(P_i, (P_{i+1}, \ldots, P_m))$. We add $L(p, (P_1, \ldots, P_i))$ to the both sides of this inequality to get

$$L(p, (P_1, \ldots, P_i, P'_1, \ldots P'_{m-i})) < L(p, (P_1, \ldots, P_m))$$

The above inequality shows that the answer to $Q(p, M)$ must be $(P_1, \ldots, P_i, P'_1, \ldots, P'_{m-i})$ which clearly follows sequence M. This contradicts our assumption that $Q(p, M) = R$. □

3. OSR SOLUTIONS

In this section, we propose alternative solutions for OSR queries in vector and metric spaces. We start by discussing a naive solution based on the Dijkstra's algorithm. We then propose LORD, an approach that employs some threshold values to efficiently prune non-candidate routes. Next we discuss R-LORD, that is an optimization of LORD by utilizing an R-tree index structure. Finally, we discuss a solution that progressively performs nearest neighbor queries on different point sets to find the optimal route for metric spaces.

3.1 The Dijkstra-based Solution

Suppose we have an OSR query for a network with a starting point p, a sequence M, and point sets $\{U_{M_1}, \ldots, U_{M_n}\}$. We construct a weighted directed graph G for the given network where the set $V = \bigcup_{i=1}^{m} U_{M_i} \cup \{p\}$ are the vertices of G and its edges are generated as follows. The vertex corresponding to p is connected to all the vertices in point set U_{M_1}. Subsequently, each vertex corresponding to a point x in U_{M_i} is connected to all the vertices corresponding to the points in $U_{M_{i+1}}$, where $1 \leq i < m - 1$. Figure 2 illustrates an example of such graph. As shown in the figure, the graph G is a *k-bipartite* graph where $k = m + 1$. The weight assigned to each edge of G is the distance between the two points corresponding to its two end-vertices. This graph is in fact showing all possible candidate sequenced routes (candidate SRs) for the given M and the set of Us. To be precise, it shows all the routes $R_p = p \oplus R$ where R is a candidate SR. By definition, the optimal route for the given OSR query is the candidate SR R for which R_p has the minimum length. Considering graph G, we notice that the OSR problem can be simply considered as finding the

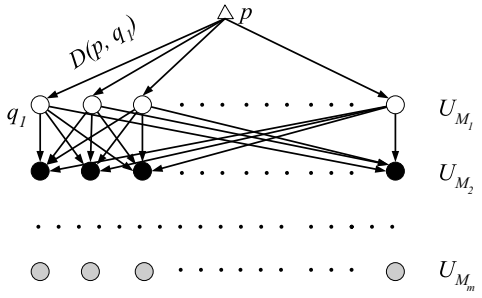


Figure 2: The weighted directed graph G for a sequence M

shortest paths (i.e., with minimum weight) from p to each of the vertices that correspond to the points in U_{M_m} (i.e., the last level of points in Figure 2), and then returning the path with the shortest length as the optimal route. This can be achieved by performing the Dijkstra's algorithm on G.

There are two drawbacks with this solution. First, the graph G has $|E| = |U_{M_1}| + \sum_{i=1}^{m-1} |U_{M_i}|.|U_{M_{i+1}}|$ directed edges which is a large number considering the usually large cardinality of the sets U_i. For instance, for a real-world dataset with $40,000$ points and $|M| = 3$, G has 124 million edges (see Section 5). The time complexity of the Dijkstra's classic algorithm to find the shortest path between 2 nodes in graph G is $O(|E| \log |V|)$. Hence, the complexity of this naive algorithm is $O(|U_{M_m}||E| \log |V|)$. Second, this huge graph must be built and kept in main memory. Although there exist versions of the Dijkstra's algorithm that are adjusted to use external memory [9], but they result in so much of overhead which makes them hard to employ for OSR queries (see Section 6 for the complete discussion). This renders the classic Dijkstra's algorithm to answer OSR queries in real-time impractical.

In order to improve the performance of this naive Dijkstra-based solution, we can issue a range query around the starting point p and only select the points that are closer to p than $L(p, R_g(p, M))$. This is because the length of any route R which includes a point outside this range is greater than that of the greedy route $R_g(p, M)$.Therefore, we build the graph G using only the points within the range instead of all the points. In Section 5, we show that even this enhanced version of the Dijkstra's algorithm is not as efficient as our approaches.

3.2 OSR in Vector Space

In this section, we assume that the distance function $D(.,.)$ is the Euclidian distance between the points in $\mathbb{R}^d$. We provide two solutions for OSR problem in this vector space.

3.2.1 LORD: Light Optimal Route Discoverer

This section describes our *Light Optimal Route Discoverer (LORD)* for addressing OSR queries. LORD has the same flavor as Dijkstra's algorithm but as a threshold-based algorithm it functions in the context of the OSR problem considering its unique properties described in Section 2.2. We name it a *light* algorithm in terms of memory as we show that LORD's workspace is less than the workspace required to apply the Dijkstra-based approach to the OSR problem.

Given an OSR query $Q(p, M)$, LORD iteratively builds and maintains a set of partial sequenced routes (partial SR) in the reverse sequence, i.e., from the end points (points in U_{M_m}) toward p. During each iteration i of LORD, points from the point set $U_{M_{(m-i+1)}}$ are added to the head of each of these partial SRs to make them closer to a candidate SR and finally, to the solution (i.e., optimal SR). To make the solution space smaller, LORD only considers those points in $U_{M_{(m-i+1)}}$ that adding them to the partial SRs will not generate routes which are longer than a *variable* threshold value T_v. LORD further examines the partial SRs by calculating their lengths after adding p, and discards the routes whose corresponding length is more than a *constant* threshold value T_c, where T_c is the length of the greedy route.

We now describe LORD in more details using the example shown in Figure 3. Figure 3a depicts a starting point p and three different sets of points U_1, U_2, and U_3, shown as black (b_i), white (w_i) and grey (g_i) points, respectively. Without loss of generality, we assume that the distance between each two points in the space is their Euclidian distance. Given the starting point p (shown as $\triangle$ in the figure), we want to find the route R with the minimum $L(p, R)$ from a white, to a black and then a grey point. Therefore, the required OSR query is formulated as $Q(p, (2, 1, 3))$.

The first step in LORD is to issue $(m =)3$ consecutive nearest neighbor queries to find the greedy route that follows $(2, 1, 3)$ from p. To be specific, the algorithm first finds the closest w_i to p (i.e., w_2), then the closest b_i to w_2 (i.e., b_2), and finally the closest g_i to b_2 (i.e., g_2). Figure 3b renders the greedy route $R_g(p, (2, 1, 3))$ as (w_2, b_2, g_2). LORD initiates both threshold values T_v and T_c to the length of $p \oplus R_g(p, M)$ (i.e., $L(p, (w_2, b_2, g_2))$. Note that the value of T_c remains the same while the value of T_v reduces after each iteration. Subsequently, it discards all the points whose distances to p are more than T_v, i.e., the points that are outside the circle shown in Figure 3c (i.e., w_1, w_4, and g_1). This is because any route (e.g., R) that contains a point that is outside this circle will lead to $L(p, R) > L(p, R_g(p, M))$ and hence, by definition, cannot be the optimal route. At this point, LORD generates a set, S, for partial candidate routes and inserts the gray nodes (i.e., points in U_M) which are inside the circle in Figure 3c, in to S, i.e., $S = \{(g_2), (g_3), (g_4), (g_5), (g_6)\}$. Note that at this stage, the length of the partial routes in S is zero.

In the first iteration of LORD, each point $x \in U_{M_{m-1}}$ (i.e., b_i's) is added to the head of each partial SR $PSR = (P_1) \in S$ if: a) x is inside the circle T_v, and b) $D(p, x) + D(x, P_1) + L(PSR) \leq T_c$. The rational behind the second condition is property 1; if the inequality does not hold, then $L(p, (x, P_1, \ldots, P_i))$ will be greater than T_c and hence, $(x, P_1, \ldots, P_i)$ cannot be part of the optimal route. For instance, in Figure 3d, point b_4 is added to (g_3) and (g_4) resulting in new partial SRs $\{(b_4, g_3), (b_4, g_4)\}$, but cannot be added to (g_2), (g_5) and (g_6). Moreover, between partial SRs that have the same first point (e.g., (b_4, g_3) and (b_4, g_4)), only the one with the shortest length will be kept in S (i.e., property 2). In addition, any $PSR \in S$ that no x can be added to it will be discarded. For example, in Figure 3d, (g_6) will be discarded because if any b_i is added to it, at least one of the above two conditions will not be met. Hence, at the end of the first step, the set of the partial SRs will become $\{(b_6, g_5), (b_4, g_3), (b_3, g_3), (b_2, g_2), (b_1, g_2)\}$ (Figure 3e).

At the end of each iteration, the value of variable threshold T_v is decreased as follows. Suppose that $Q(p, M) = (q_1, \ldots, q_i, \ldots, q_m)$ and we are examining iteration $(m-i+1)$

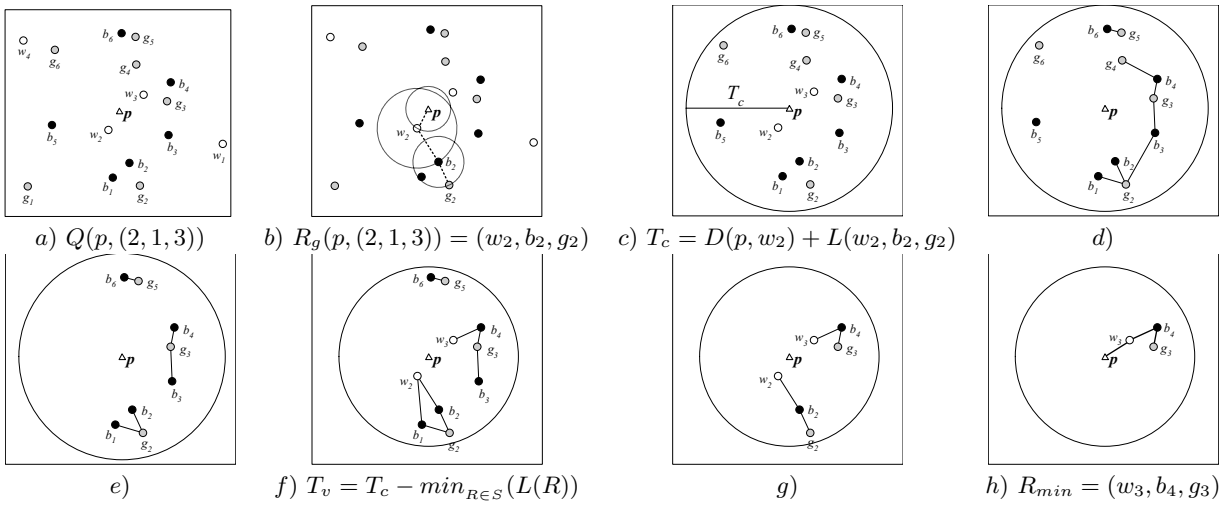


Figure 3: Different iterations of LORD

(i.e., the partial SRs in S are in the form of $(p_{i+1}, \ldots, p_m)$). The definition of the greedy route implies that

$$L(p,(q_1,\ldots,q_m)) \leq \ L(p,R_g(p,M)) = T_c$$

and by considering Property 1, we have:

$$D(p,q_i) + L((q_{i+1},..,q_m)) < D(p,q_i) + L((q_i,..,q_m)) \leq T_c$$

which can be rewritten as:

$$D(p,q_i) \leq T_c - L((q_{i+1},..,q_m)) \quad (4)$$

Note that inequality 4 must hold for all points q_i that are to be examined at iteration $(m-i+1)$. Hence, by replacing $L((q_{i+1},..,q_m))$ with its minimum value, we obtain the maximum value for $D(p,q_i)$ for any q_i. Therefore, for any point q_i that is examined in iteration $(m-i+1)$, we must have

$$D(p,q_i) \leq T_v = T_c - min_{PSR\in S}(L(PSR))$$

Note that at each iteration, the lengths of the partial SRs in S, and hence the value of $min_{PSR\in S}(L(PSR))$ is increasing. This yields to smaller values for T_v after each iteration. This is also shown in Figure 3; the radius of the circle in Figure 3f is smaller than the radius of the circle in Figure 3c.

The subsequent $(m-2)$ iterations of LORD are performed similarly and the partial routes in S will become complete routes (i.e., candidate SRs that follow M) after the last iteration is completed (Figure 3g). Finally, LORD examines the distance from p to the first point in each complete route in S (i.e., $\{(w_2,b_2,g_2),\ (w_3,b_4,g_3)\}$) and selects the one that generates the minimum total distance, i.e., the route with the minimum value for $L()$ function, as the result of $Q(p,(2,1,3))$ (Figure 3h).

Figure 4 shows the pseudocode of LORD. Lines $3-5$ perform the first range query using the threshold T_v and initialize the set of partial SRs S. The $(m-1)$ iterations are performed in lines $6-16$ where lines 9 and 12 check if a point can be added to the partial SRs in S and line 16 updates the value of T_v. Finally, line 17 selects the route in S that generates the minimum $L(p,R)$ as the result of $Q(p,M)$.

```
    Algorithm LORD(point p, sequence M)
1.    S = {};
2.    T_v = T_c = L(p, R_g(p, M));
3.    for q in U_{M_m}
4.        if (D(p,q) ≤ T_v)
5.            S = S ∪ {(q)};
6.    for i = m − 1 downto 1
7.        S' = {};
8.        for q in U_{M_i}
9.            if (D(p,q) ≤ T_v)
10.               S'' = {};
11.               for R = (P_1, ..., P_{m−i}) in S
12.                   if (D(p,q) + D(q,P_1) + L(R) ≤ T_c)
13.                       S'' = S'' ∪ {(q, P_1, ..., P_{m−i})};
14.               S' = S' ∪ {argmin_{R''∈S''}(L(R''))};
15.       S = S';
16.       T_v = T_c − min_{R∈S}(L(R));
17.   R_min = argmin_{R∈S}(L(p,R));
18.   return R_min;
```

Figure 4: Pseudocode of the LORD Algorithm

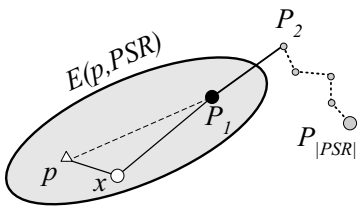


Figure 5: The locus of the points x for LORD

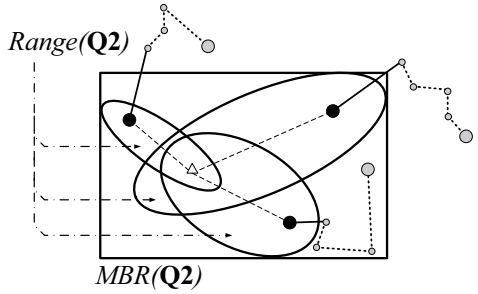


Figure 6: Range query Q2 and its MBR for partial routes in LORD

3.2.2 R-LORD: R-tree based LORD

We described LORD in Section 3.2.1 without any assumption on the structure of the points in each U_i. We now discuss the situation that the points in U_i's are stored in an R-tree index structure. We utilize the features of the index structure to develop an R-tree-friendly version of LORD. The core idea behind this solution is to use the points' neighborhood information implicitly stored in R-tree MBR's to more efficiently prune the candidate points at each iteration of LORD. Towards this goal, we transform the LORD's point selection criterium to the range queries applicable on an R-tree. Then, we show that the point selection can be performed using a single range query. Finally, we describe our algorithm which uses this range to find the solution for an OSR query utilizing an R-tree.

3.2.2.1 Point Selection Criterium in LORD

As we discussed in Section 3.2.1, at each iteration i, LORD prunes the points in U_{M_i} in two steps. First, it ignores any point of the set U_{M_i} that is farther than the value of the variable threshold T_v from the starting point p. This is a simple range query **Q1** given the range $Range(\mathbf{Q1})$ as a circle with a known radius T_v centered at p. Second, any point x resulting from query $Range(\mathbf{Q1})$ is checked against all partial SRs $PSR \in S$. If for each $PSR = (P_1, ..., P_{|PSR|}) \in S$, the value of $D(p,x) + D(x,P_1) + L(PSR)$ is greater than the constant threshold T_c (i.e., the length of the greedy route), then point x is not added to the beginning of that PSR. Otherwise, a new partial SR, $(q_i, P_1, ..., P_{|PSR|})$, is generated. This clearly shows that the second query **Q2** uses a more complicated range to prune the results of **Q1**.

To identify $Range(\mathbf{Q2})$, we first find the locus of the points x which can possibly be added to a $PSR = (P_1, ..., P_{|PSR|}) \in S$. For such a point x, we must have $D(x,p) + D(x,P_1) \leq T_c - L(PSR)$ (Line 12 in Figure 4). As $L(PSR)$ and T_c are constant values for a given PSR and query $Q(p,M)$, the sum of x's distances from two fixed points p and P_1 cannot be larger than a constant. Hence, x must be on or inside an ellipse defined by the foci p and P_1 and the constant $T_c - L(PSR)$. Figure 5 shows the locus of the points x for a given route PSR as inside and on an ellipse $E(p, PSR)$.

Query **Q2** is defined in terms of the set of partial SRs stored in S in the current iteration. For each PSR, we showed that LORD appends points inside ellipse $E(p, PSR)$ to the head of the PSR in order to build a new partial candidate route. All such ellipses, each corresponding to a partial SR in S, are intersecting as they all share the common focus point p. The union of these ellipses contains all the points x (of the appropriate set), where for each, there is exactly one route starting with x built at the end of the current iteration. In other words, this union should be the range used in query **Q2**. Figure 6 illustrates an example for the current set S during an iteration of LORD. The set includes three partial SRs of the same length each starting with a black point. The sequence M of the query $Q(P,M)$ dictates the type of the point which must be added to the head of each partial SR. Any point outside the union of these three ellipses is ignored by LORD.

Up to this point, we have identified the range of the two main queries **Q1** and **Q2** used in LORD. In the following, we show that any ellipse for the range **Q2** is entirely inside the circle for range **Q1** and hence, the range of **Q2** is completely inside that of **Q1**.

LEMMA 1. *During each iteration of LORD for $Q(p,M)$, given a partial SR $PSR \in S$, any point x inside or on the ellipse $E(p,PSR)$ has a distance less than current value of the variable threshold T_v from point p (i.e., $D(x,p) < T_v$).*

PROOF. As point x is inside or on ellipse $E(p,PSR)$ corresponding to the route PSR, we have

$$\begin{aligned} D(x,p) + D(x,P_1) &\leq T_c - L(PSR) \\ &\leq T_c - min_{PSR \in A}(L(PSR)) \end{aligned} \quad (5)$$

The right side of the above inequality has the same value as that of the current value of T_v. It directly yields that $D(x,p) \leq T_v - D(x,P_1)$ and subsequently, we have $D(x,p) < T_v$. □

Lemma 1 shows that any ellipse $E(p,PSR)$ is completely inside the circular range of **Q1**. Now, as $Range(\mathbf{Q2})$ is the union of all ellipses $E(p,PSR)$ corresponding to all the partial SRs in S, it can be concluded that it is entirely inside $Range(\mathbf{Q1})$.

Note that at each iteration, LORD builds a new route using only the points in the intersection of $Range(\mathbf{Q1})$ and $Range(\mathbf{Q2})$. Given Lemma 1, this intersection is the same as $Range(\mathbf{Q2})$. Hence, the algorithm must only consider the points which are within the range of **Q2** from p, to be added to the partial SRs in S.

3.2.2.2 R-tree Friendly LORD

Recall that our goal is to transform the threshold values utilized by LORD to the range queries that can be performed on R-tree index structures. In Section 3.2.2, we showed that the two range queries **Q1** and **Q2** employed by LORD can be reduced to only one as **Q2** is entirely inside **Q1**. However, as Figure 6 illustrates, the range specified by **Q2** (union of the ellipses) is a complex parameterized curved shape which cannot be efficiently handled by an R-tree range

```
    Function RQ1(point p, number dist, number index)
1.    L = empty; R = {};
2.    insert R-tree root into list L;
3.    while L is not empty
4.       N = first node in the list L;
5.       if N is a data point q and q ∈ U_index then
6.          if (D(p,q) ≤ dist) then R = R ∪ {q};
7.       else // N is an intermediate node
8.          remove N from L;
9.          for each child node N' of N
10.            if (mindist(N',p) ≤ dist) then add N' to L;
11.   return R;
```

Figure 7: Range query Q1 using R-tree

query algorithm. To make this range simpler, we employ its minimum bounding box ($MBR(\mathbf{Q2})$) as shown in Figure 6. However, $MBR(\mathbf{Q2})$ is no longer inside the range of **Q1**. Therefore, our R-tree version of LORD must use the intersection of $MBR(\mathbf{Q2})$ and $Range(\mathbf{Q1})$ to examine the points in U_{M_i}'s.

To retrieve the points in a specific range, we need to traverse the R-tree from its root down to the leaves and report those points that are within the given range. To make the search efficient, existing search algorithms on R-tree prune subtrees of the main tree utilizing some metrics. The most common metric, $mindist(N, q)$, gives a lower bound on the smallest distance between the point q and any point in the subtree of node N. We utilize *mindist* for **Q1** as its range is relative to a fixed point p. Any R-tree node N with $mindist(N, p)$ greater than threshold T_v cannot contain a point q with the distance $D(p, q)$ less than or equal to T_v. Such node can be easily pruned when traversing the R-tree during our first range query (i.e., **Q1**). Moreover, query **Q1** is used to initialize the PSRs of LORD (Line 3-5 in Figure 4). Figure 7 shows how we use *mindist* metric in **Q1** to initialize the set of routes S. It also demonstrates the way a circular range query can be answered on an R-tree.

The second rectangular range query (i.e., $MBR(\mathbf{Q2})$) can be performed as follows. We first check whether a node N of the R-tree intersects with the rectangle. If their intersection is empty, the node N must be pruned; otherwise, the child nodes of N must be checked for their intersection with $MBR(\mathbf{Q2})$.

Now that we have identified both of the range queries used to select the points in LORD and studied how they can be evaluated using an R-tree, we propose R-LORD, the R-tree version of LORD. The only difference between R-LORD and LORD is that R-LORD incorporates the R-tree implementation of two range queries of LORD in its iterations. First, it initializes the set S, with the partial SRs of length zero, each including a single point of the set of points returned from the function $RQ1(p, T_c, M_m)$ (Figure 7). Then, in each iteration, R-LORD traverses the entire R-tree starting from the root to prune the nodes that are outside $MBR(\mathbf{Q2})$ and $Range(\mathbf{Q1})$ and then selects the points that must be added to the PSRs. At the end of each iteration, R-LORD updates $MBR(\mathbf{Q2})$ by examining the recently built PSRs in S.

3.3 OSR in Metric Space

The previous proposed solutions for OSR queries (discussed in Sections 3.2.1 and 3.2.2), although efficient in vector spaces, are impractical or inefficient for an arbitrary sequence M in a *metric* space. Even though LORD can be applied to both vector and metric spaces, its extensive usage of the $D(.,.)$ function renders it inefficient for metric spaces where the distance metric is usually a computationally complex function. Moreover, R-LORD can only be applied to vector spaces since it is based on utilizing R-tree index structure.

In this section, we describe our proposed algorithm, Progressive Neighbor Exploration (PNE), to address OSR queries in metric spaces for arbitrary values of M. Unlike LORD, the idea behind PNE is to incrementally create the set of candidate routes for $Q(p, M)$ in the same sequence as M, i.e., from p toward U_{M_m}. This is achieved through an iterative process in which we start by examining the nearest neighbor to p in U_{M_1}, generating partial SR from p to this neighbor, and storing the candidate route in a heap based on its length. At each subsequent iteration of PNE, a partial SR ($e.g., PSR = (r_1, r_2, ..., r_{|PSR|})$) from top of the heap is fetched and examined as follows.

1. If $|PSR| = m$, meaning that the number of nodes in the partial SR is equal to the number of items in M and hence PSR is a candidate SR that follows M, the PSR is selected as the optimal route for $Q(p, M)$ since it also has the shortest length.
2. If $|PSR| \neq m$:
   (a) First the last point in PSR, $r_{|PSR|}$, (which belongs to $U_{M_{|PSR|}}$) is extracted and its next nearest neighbor in $U_{M_{|PSR|+1}}$, $r_{|PSR|+1}$, is found. This will guarantee that a) the sequence of the points in PSR always follows sequence specified in M, and b) the points that are closer to $r_{|PSR|}$ and hence may potentially generate smaller routes are examined first. The fetched PSR is then updated to include $r_{|PSR|+1}$ and is put back in to the heap.
   (b) We then find the next nearest neighbor in $U_{M_{|PSR|}}$ to $r_{|PSR|-1}$, $r'_{|PSR|}$, generate a new partial SR $PSR' = (r_1, r_2, ..., r_{|PSR|-1}, r'_{|PSR|})$, and place the new route in to the heap. This is because once the point $r_{|PSR|}$, which we can assume is the k-th nearest point in $U_{M_{|PSR|}}$ to $r_{|PSR|-1}$, is chosen in step (a) above, the $(k+1)$-st nearest point in $U_{M_{|PSR|}}$ to $r_{|PSR|-1}$ (e.g., $r'_{|PSR|}$) is the only next point that may generate a shorter route and hence, must be examined. If $|PSR| = 1$, we find the next nearest point in U_{M_1} to p.

We describe PNE in more details using the example of Section 1.1. Recall that our OSR query was to drive toward a gas station, a library, and then a post office (i.e., $M = (g, l, p)$ and $|M| = m = 3$). Figure 2 depicts the values stored in the heap in each step of the algorithm. In step 1, the first nearest g_i to p, g_2, is found and the first partial SR along with its distance, $(g_2 : 2)$, is generated and placed in to the heap. In step 2, first $(g_2 : 2)$ is fetched from the heap. Since for this route $|PSR| \neq 3$, the steps $2(a)$ and $2(b)$ are performed. More specifically, first the next nearest l_i to g_2, l_2, is found; the partial SR is updated by adding l_2 to it; and is placed back into the heap. Second, the next nearest g_i to p, g_1, is found and is placed in to the heap. Similarly, this

step	heap contents (candidate route R : $L(p, R)$)
1	$(g_2 : 2)$
2	$(g_1 : 3), (g_2, l_2 : 4)$
3	$(g_2, l_2 : 4), (g_3 : 4), (g_1, l_2 : 6)$
4	$(g_3 : 4), (g_2, l_3 : 5), (g_1, l_2 : 6), (g_2, l_2, p_2 : 15)$
5	$(g_2, l_3 : 5), (g_4 : 5), (g_1, l_2 : 6), (g_3, l_2 : 6)$ $(g_2, l_2, p_2 : 15)$
6	$(g_4 : 5), (g_1, l_2 : 6), (g_3, l_2 : 6), (g_2, l_1 : 12)$ $(g_2, l_3, p_3 : 14)$, ~~$(g_2, l_2, p_2 : 15)$~~
7	$(g_1, l_2 : 6), (g_3, l_2 : 6), (g_4, l_3 : 11), (g_2, l_1 : 12)$ $(g_2, l_3, p_3 : 14)$
8	$(g_3, l_2 : 6), (g_1, l_3 : 9), (g_4, l_3 : 11), (g_2, l_1 : 12)$ $(g_2, l_3, p_3 : 14)$, ~~$(g_1, l_2, p_2 : 17)$~~
9	$(g_1, l_3 : 9), (g_3, l_3 : 9), (g_4, l_3 : 11), (g_2, l_1 : 12)$ $(g_2, l_3, p_3 : 14)$, ~~$(g_3, l_2, p_2 : 17)$~~
10	$(g_3, l_3 : 9), (g_1, l_1 : 10), (g_4, l_3 : 11), (g_2, l_1 : 12)$ $(g_2, l_3, p_3 : 14)$, ~~$(g_1, l_3, p_3 : 18)$~~
11	$(g_1, l_1 : 10), (g_4, l_3 : 11), (g_2, l_1 : 12), (g_3, l_1 : 12)$ $(g_2, l_3, p_3 : 14)$, ~~$(g_3, l_3, p_3 : 18)$~~
12	$(g_4, l_3 : 11), (g_2, l_1 : 12), (g_3, l_1 : 12), (g_1, l_1, p_1 : 12)$ ~~$(g_2, l_3, p_3 : 14)$~~
13	$(g_2, l_1 : 12), (g_3, l_1 : 12), (g_1, l_1, p_1 : 12)$ ~~$(g_4, l_3, p_3 : 20)$~~

Table 2: PNE for the example of Figure 1

process is repeated until the route on top of the heap follows the sequence M (i.e., (g_1, l_1, p_1) in step 13). Note that we only keep one candidate SR (i.e., route with m points) in the heap. That is, if during step $2(a)$ a route with m points is generated, it is only added to the heap if there is no other candidate SR with a shorter length in the heap. Moreover, after a candidate SR is added to the heap, any other SR with longer length will be discarded. For example, in step 6, adding the route (g_2, l_3, p_3) with the length of 14 to the heap will result in discarding the route (g_2, l_2, p_2) with the length of 15 from the heap (crossed out in the figure).

The only requirement for PNE is a nearest neighbor approach that can progressively generate the neighbors. Hence, by employing an approach similar to INE [16] or VN^3 [12], which are explicitly designed for metric spaces, PNE can address OSR queries in metric spaces. In theory PNE can work for vector spaces in a similar way; however, it is inefficient for these spaces where distance computation is not expensive. The reason is that PNE explores the candidate routes from the starting point which might result in an exhaustive search. Instead, R-LORD optimizes this search by building the routes in the reverse sequence utilizing the R-tree index structure.

4. VARIATIONS OF OSR QUERIES

In this section, we address two variations of OSR queries. The first variation is when a destination point also exists, and the second variation is when k optimal routes are requested.

4.1 OSR-I

Assume that the user asks for an optimal sequenced route that follows the given sequence which starts from a given source and ends in a given destination. A special case of this query is where the source and destination points are the same, i.e., the user intends to return to her starting location. We start by formally defining this type of query as:

Definition 8: Given source point p, destination point q and a sequence M, the *OSR-I* query is defined as finding $R = (P_1, ..., P_m)$, a sequenced route that follows M, where the following function G is minimum over all sequence routes that follow M:

$$G(p, R, q) = D(p, P_1) + L(R) + D(P_m, q) \quad (6)$$

The above equation is similar to $L(p, R) + D(P_m, q)$. We show that this new form of OSR can easily be reduced to the general form of OSR.

We define a new set $U_{n+1} = \{q\}$. Including this new set in the set of U_i's makes $M' = (M_1, \ldots, M_m, n+1)$ a valid sequence in the new setting of the problem. Now if we assume that $Q(p, M') = R' = (P'_1, ..., P'_{m+1})$, we know that P'_{m+1} will be q as q is the only member of U_{n+1}. Moreover, $L(p, R')$ is minimum over all candidate routes that follow M'. Recall that the length of the route $R'_p = p \oplus R'$ (i.e., $L(p, R')$) is equal to $D(p, P'_1) + L(R')$. We define the route R as $(P'_1, \ldots, P'_m)$ by excluding q from R'. It is clear that $L(p, R')$ is the same as $D(p, P_1) + L(R) + D(P_m, q)$. By comparing the latter expression with $G(p, R, q)$ of Equation 6, we conclude that R is the answer to the OSR-I query given the source p, destination q, and sequence M.

Since we showed that OSR-I can be reduced to a general OSR problem, we are able to use our LORD (or R-LORD) algorithm to answer this query. Specifically, the answer to OSR-I given the source p, destination q, and sequence M is the same as the answer to $LORD(p, M')$ excluding the point q, where $U_{n+1} = \{q\}$ and $M' = (M_1, \ldots, M_m, n+1)$. Although R-LORD can similarly solve OSR-I, we can further optimize it for OSR-I. This is achieved by neglecting the range query **Q1** (i.e., $RQ1(p, T_c, n+1)$). This is because we know that the only point in this range is q. Therefore, the set S can be directly initialized to $\{(q)\}$.

4.2 k-OSR

The second variation of OSR is when the user asks for the k routes with the minimum total distances to its location. We define this as $k-OSR$ query. We can easily address this type of query using our PNE approach discussed in Section 3.3.

Recall that in PNE, we maintain a heap of the partially completed sequenced routes and only keep one candidate sequenced route (or in other words, a route that follows M), that is the one that has the minimum total length. By modifying this policy to maintain k candidate SRs in the heap and continuing the iterations until k candidate SRs are fetched from the heap, PNE can also address $k - OSR$ queries.

5. PERFORMANCE EVALUATION

We conducted several experiments to evaluate the performance of our R-LORD approach with respect to: 1) disk I/O accesses incurred by its underlying R-tree index structure, 2) effectiveness of its range queries, and 3) its overall query response time. Moreover, we compared the query response time of R-LORD with that of the Dijkstra-based solution. In our experiments, we evaluated R-LORD by investigating the effect of the following parameters on its performance: 1) size of sequence M in $Q(p, M)$ (i.e., number of points in the optimal route), 2) cardinality of the datasets (i.e., $\sum_{i=1}^{n} |U_i|$), and 3) density and distribution of the datasets. We also investigated the performance of the PNE approach with respect to the density of the datasets. We used one real and

two synthetic datasets for our experiments. The real data is obtained from the U.S. Geological Survey (USGS) and consists of the location of different businesses (e.g., schools) in the entire country. The synthetic datasets consist of randomly generated set of points with uniform and Zipf distributions. Table 5 shows the characteristics of the datasets. The real dataset has a total of 950,000 points. However, in our experiments, we randomly selected sets of $40K$, $70K$, $250K$ and $500K$ points from this dataset. The cardinality of each synthetic dataset is 480,000. Each dataset is indexed by an R*-tree [3] index with the page size of 1K bytes and the maximum of 50 entries in each node (capacity of the node). The experiments were performed on a DELL Precision 470 with Xeon 3.2 GHz processor and 3GB of RAM. We ran 1000 OSR queries initiated from randomly selected starting points and report the average of the results.

In the first set of experiments, we compared the performance of R-LORD with that of the Dijkstra-based solution. Note that the weighted directed graph G (see Section 3.1) for even a small dataset is a substantially large graph. For example, for a real dataset with 40,000 points and $|M| = 3$, G has 22,400 nodes and 124 million edges. This will result in substantially large query response times for the naive Dijkstra-based solution (e.g., 40 seconds for the 40K example). Therefore, we do not report the query and workspace costs of this expensive approach. Instead, we compare R-LORD's costs with those of enhanced Dijsktra-based approach in which the length of the greedy route is used to reduce the number of candidate points (see Section 3.1).

Figure 8 shows the query response time for R-LORD and the enhanced Dijkstra-based approach (EDJ) when the number of points in optimal route (i.e., $|M|$) varies from 3 to 12. While the figure depicts the results from an experiment on 250K USGS dataset, the trend is the same to those of all of our datasets with different cardinalities and distributions. As shown in the figure, both approaches answer an OSR query very quickly for small values of $|M|$ (less than 100msec for $|M| = 3$). The figure also shows that as the value of $|M|$ increases, the response time of the EDJ increases with a rate that is substantially more than that of R-LORD, confirming the impracticality of the Dijkstra-based solution for OSR on large graphs.

In the second set of experiments, we varied the size of M and measured the performance of R-LORD. Figures 9(a,b,c) depict the performance of R-LORD on a randomly selected real dataset with $250K$ points when the size of M varies from 3 to 12. For this dataset, 7291 nodes are generated in R*-tree. Figure 9a illustrates the percentage of R*-tree nodes that were accessed by R-LORD. As shown in the figure, between 1% (for small values of $|M|$) to 11% (for large values of $|M|$) were accessed by R-LORD. The figure also shows that the rate in which the number of accessed nodes increases is slightly more than linear. That is, while the percentage of accessed nodes increases from 1% to 2% (i.e., 2 times) when $|M|$ increases from 3 to 6, it increases from 2% to 11% (i.e., 5.5 times) when M increases from 6 to 12. This is because for larger values of $|M|$, more nodes are examined against **Q2** and $mindist()$ function. Figure 9b shows the total query response time of R-LORD for the same dataset. As shown in the figure, even for a large value of 12 for $|M|$, R-LORD can answer the query in less than 0.8 seconds. Moreover, it shows that the rate of increase in the processing time closely follows the rate of increase in accessed nodes, indicating that traversing R*-tree is the major factor in R-LORD. Figure 9c shows the performance of the range queries of R-LORD. The bars in the figure indicate the required workspace of R-LORD (WS) as the maximum number of points that were stored in the partial SRs of S (see Section 3.2.2). As shown in the figure, the number of points filtered in by the range queries are substantially less than the cardinality of the points (e.g., for $|M| = 6$, only 110 points out of 250,000 are selected). This shows that the range queries of R-LORD are extremely effective. The figure also compares the effectiveness of the two range queries of R-LORD. It shows the percentage of reduction in the number of selected points as compared to the Dijkstra-based approach. In the later approach the only filter is one simple range query with a range based on the length of the greedy sequenced route ($L(p, R_g(p, M))$). This is shown as vertical lines in the figure, where each line indicates the maximum, minimum, and average value of this reduction for a given M. The figure confirms that our range queries provide a filter with better selectivity as compared to the simple range query. For example, for $|M| = 6$, the decrease in the size of the candidate points is between 48% to 97.7% with an average of 77.4%. Figures 9(d,e,f) show the result of the same set of experiments for our first set of synthetic data (i.e., with uniform distribution). This dataset has 250,000 points and generate 7,291 nodes in the R*-tree. The figures show identical behavior for the synthetic date as compared to the real dataset. It also shows that the range queries can filter out up to 99% of the points as compared to the simple range query with the range equal to $L(p, R_g(p, M))$.

Figures 10(a,b,c) show the results of our third set of experiments, where we investigated the impact of the cardinality of the points on the efficiency of R-LORD. We varied the cardinality of our real dataset from $40K$ to $500K$ and ran OSR queries of sequence size $|M| = 6$. Figure 10a shows the per-

USGS		**Synthetic**	
Points	*Size*	*Points*	*Size*
Hospital	5,314	P1 (uniform)	32,000
Building	15,127	P2 (uniform)	64,000
Summit	69,498	P3 (uniform)	128,000
Cemetery	109,557	P4 (uniform)	256,000
Church	127,949	P5 (Zipf)	32,000
School	139,523	P6 (Zipf)	64,000
Populated place	167,203	P7 (Zipf)	128,000
Institution	319,751	P8 (Zipf)	256,000

Table 3: Datasets used in our experiments

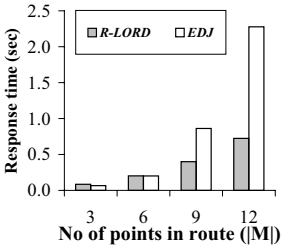
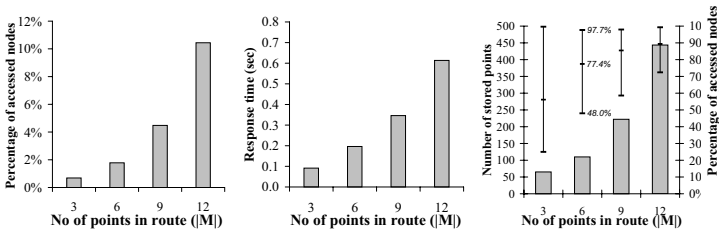
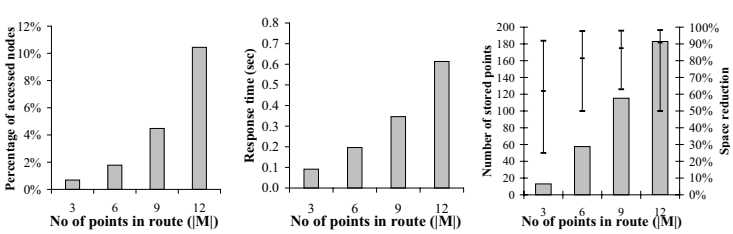


Figure 8: Query response time vs. sequence size $|M|$ (i.e., number of points in the optimal route $Q(p, M)$)

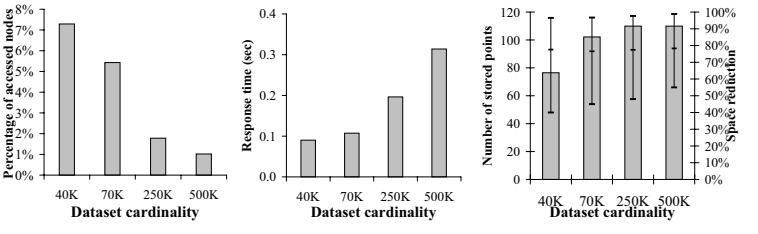


a) I/O (250K USGS) b) Time (250K USGS) c) WS (250K USGS)

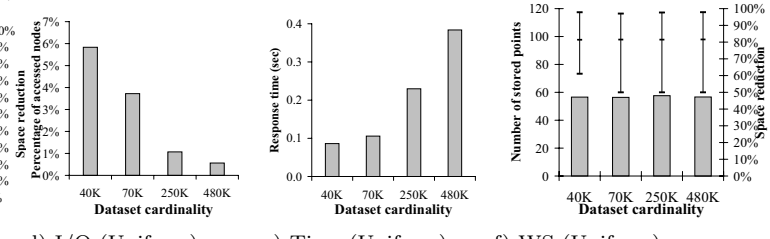


d) I/O (250K Uniform) e) Time (250K Uniform) f) WS (250K Uniform)

Figure 9: Query cost vs. sequence size $|M|$



a) I/O (USGS) b) Time (USGS) c) WS (USGS)



d) I/O (Uniform) e) Time (Uniform) f) WS (Uniform)

Figure 10: Query cost vs. cardinality ($|M| = 6$)

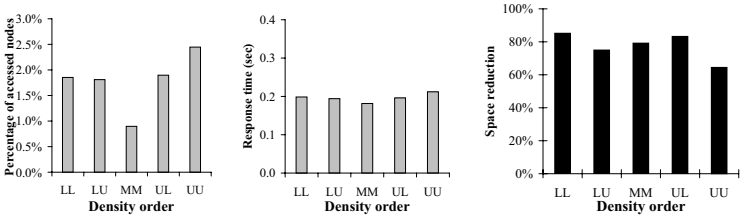
centage of accessed nodes of R*-tree for different cardinalities of the dataset. As shown in the figure, the percentage of accessed nodes in R*-tree decreases as the cardinality of the data increases, indicating that R-LORD can efficiently scale to large datasets. Moreover, Figure 10b shows that the processing time of R-LORD slightly increases as the cardinality of the data increases. For example, where the query response time is 0.09 seconds for $40,000$ points, it only increases to 0.32 seconds (i.e., factor of 3.5) where the number of points increases to $500,000$ (i.e., factor of 12). This also verifies the scalability of R-LORD. Figure 10c shows the performance of the range queries for different cardinalities of the dataset. The figure shows that for a dataset with $70,000$ points, only 100 (0.142%) of them are selected as the result of the range queries. The figure also indicates that this percentage decreases for larger cardinalities of data. For example, in the dataset with $500,000$ points, only 110 (0.022%) are selected. Figures 10(d,e,f) show the results of the same set of experiments on the first set of synthetic data. Once again, the figures indicate similar behavior of R-LORD for synthetic data as compared to the real datasets.

Our next set of experiments were aimed to evaluate the performance of R-LORD when the densities of the datasets U_{M_i}'s specified by the query sequence M are different. We used R-LORD to answer five different categories of queries $Q(p, M)$, each with a different *pattern of change* in the density of the datasets. The categories are:

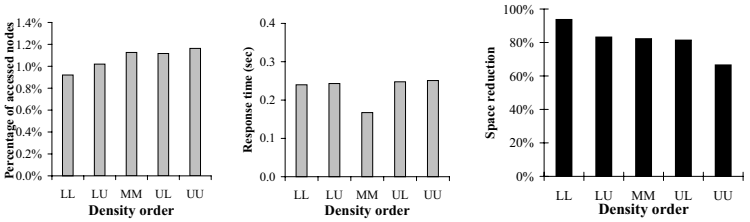
1. **LL:** The density of points is significantly decreasing from U_{M_1} to U_{M_m}. For example, a query for an optimal route to an institution (i.e., 319,751 points), then to a church (i.e., 127,949 points) and finally to a hospital (i.e., 5,314 points) in USGS dataset falls in this category.

2. **LU:** There is an $1 < i < |M|$ where the density is decreasing from U_{M_1} to U_{M_i} and increasing from U_{M_i} to U_{M_m} (e.g., (church, hospital, school)).

3. **MM:** The density of all U_{M_i}'s is almost the same (e.g., (school, church, school)).



a) I/O (250K USGS) b) Time (250K USGS) c) WS (250K USGS)



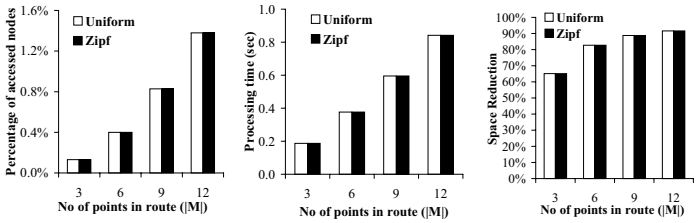
d) I/O (250K Uniform) e) Time (250K Uniform) f) WS (250K Uniform)

Figure 11: Query cost vs. density ($|M| = 6$)

4. **UL:** There is an $1 < i < |M|$ where the density is increasing from U_{M_1} to U_{M_i} and decreasing from U_{M_i} to U_{M_m} (e.g., (church, school, hospital)).

5. **UU:** The density is significantly increasing from U_{M_1} to U_{M_m} (e.g., (hospital, church, institution)).

Figures 11(a,b,c) illustrate the results of our experiments where M follows the above density distribution categories. In these experiments, $|M| = 6$ and the data is $250,000$ points selected from USGS dataset. Figure 11a shows that although the percentage of the accessed nodes varies for different density categories, they are still in the range of 1% to 2%. Moreover, the query response times shown in Figure 11b indicate that regardless of the density of the points, R-LORD answers OSR queries with almost identical response times. Figure 11c depicts that although the range queries perform similarly for different density categories, the selectivity of the range queries for LU and UU is slightly less than that of LL, MM and UL. The reason for this is that the last



a) I/O b) Time c) WS

Figure 12: Query cost vs. distribution for 32K synthetic data ($|M| = 6$)

set of points in M (i.e., U_{M_m}), which are selected first by **Q1** (recall that R-LORD constructs the partial roads from U_{M_m} toward U_{M_1}), are denser and hence, **Q1** selects more number of points to be included in the PRCs in S. Figures 11(d,e,f) show the results of the same experiment for the synthetic datasets. These figures also show similar trend in behavior of R-LORD for synthetic as compared to the real datasets.

Figure 12 shows the results of our last set of experiments, where we studied the effect of the distributions of the datasets on R-LORD. In this set of experiments, we used the synthetic datasets with uniform and Zipf distributions (Table 5). As shown in the figure, R-LORD shows similar I/O cost (Figure 12a) and query response time (Figure 12b), and its range queries perform similarly for the given datasets (Figure 12c). This indicates that the performance of R-LORD is also independent from the distribution of the datasets.

Due to lack of space, we only itemize the major observations of our experiments performed on the similar datasets for PNE. The complete discussion of the results for PNE will be presented in an extended version of this paper.

- Contrary to R-LORD, PNE's performance is sensitive to the distribution of the densities in the data set. That is, it performs efficiently for UU and MM categories, while its performance suffers for LL, LU and UL. The intuition here is that when the last group of points in the given sequence (e.g., U_{M_m}, $U_{M_{m-1}}$,...) are sparse and hence, their distances to each other are much more than the distances of the first group of the points in the sequence (e.g., U_{M_1}, U_{M_2},...) to each other, PNE will perform exhaustive search on $\{U_{M_1}, U_{M_2}, ...\}$ before examining $\{U_{M_m}, U_{M_{m-1}}, ...\}$. This leads to execution of numerous NN queries.

- The query response time of PNE is largely incurred by the underlying nearest neighbor technique and the overhead of PNE to maintain the heap is negligible as compared to the time required by the NN approach.

6. RELATED WORK

In this section, we first review the related work in the area of graph theory. We then provide an overview of the related studies on variations of the nearest neighbor queries in spatial databases.

The only similarity between Traveling Salesman Problem (TSP) and OSR is that both search for a route of minimum cost in a graph. The general form of TSP first was studied in the 1930s by Karl Menger in Harvard [4]. The most similar instance of TSP to OSR is Sequential Ordering Problem (SOP) which sets some precedence constraints on the route. Each constraint requires that a node of the graph be visited before some other node. The polyhedral structure of the TSP with precedence constraints was investigated by Balas et al. in [2]. Ascheuer et al. [1] use the results of [2] to provide a branch and cut solution for the problem and solve it efficiently for real instances of 200 nodes. Hernádvölgyi [7] derives lower bounds for the solution from the state space and provides an optimal search method. Although the number of nodes is small in SOP and general TSP, the unknown traveling sequence makes them NP-hard. However, OSR dictates a given strict sequence order of point *types* where each point must be selected from a large set per type.

The OSR problem is also related to the problem of finding Shortest Path (SP) in directed weighted graphs. Two classic algorithms for solving SP in main memory are Dijkstra's and Bellman-Ford algorithms. However, for addressing SP on the huge graph G of Section 3.1, an external memory algorithm is required. Hutchinson et al. [9] propose a tree data structure for answering SP queries on a planar graph stored in external memory. Chan at al. [5] describe a disk-based algorithm to find SP on large network systems. They partition the original large graph and search for the shortest path by locally searching in its smaller pieces. While these approaches eliminate the overheads of loading the huge graph in main memory, they are not applicable for OSR queries. The reason is that OSR graph's topology is dependent on the user's query $Q(p, M)$. Since point p and sequence M are not known in advance, this graph must be built on demand as described in Section 3.1. Therefore, if we intend to use an external memory SP approach, we need to store the graph on disk blocks before processing it. This makes the approach expensive and therefore impractical.

Numerous algorithms for k-nearest neighbor queries in spatial databases have been proposed. A majority of these algorithms are based on utilizing spatial index structures such as R-tree and usually perform in two filter and refinement steps. Roussopoulos et al. in [17] present a branch-and-bound R-tree traversal algorithm that uses two *mindist minmaxdist* metrics. Korn et al. in [13] present a multi-step k-nearest neighbor search and Seidl et al. in [18] propose an optimal version of this method. Hjaltason et al. [8] propose an incremental nearest neighbor algorithm that is based on utilizing an index structure and a priority queue. Jung et al. in [11] propose an algorithm to find the shortest distance between any two points by partitioning a large graph into layers of smaller subgraphs and pushing up the pre-computed shortest paths between the borders of the subgraphs in a hierarchical manner. Jensen et al. in [10] discuss data models and graph representations for NN queries in road networks and provide alternative solutions for it. Papadias et al. in [16] propose a solution for nearest neighbor queries in network databases by generating and expanding a search region around a query point. Kolahdouzan et al. in [12] propose a solution that is based on utilizing network Voronoi diagrams. Other variations of k nearest neighbor queries have also been studied and their solutions are usually motivated by the solutions of their regular k nearest neighbor queries. Sistla et al. in [19] first identified the importance of the continuous nearest neighbors (CNN) and described modeling methods and query languages for the expression of these

queries. Song et al. in [20] propose the first algorithms for CNN queries based on performing several point-NN queries at predefined sample points. Tao et al. in [21] propose a solution for CNN queries based on performing one single query for the entire path. Ferhatosmanoglu et al. in [6] introduce the problem of constrained NN queries, where the nearest neighbors in specific range or direction are requested. Koudas et al. in [14] discuss approximate NN queries with guaranteed error for streams where access to the entire data is not feasible. Finally, the class of group nearest neighbor queries has been recently introduced by Papadias et al. in [15].

To the best of our knowledge, no other work by the database community has studied the problem of optimal sequenced route query.

7. CONCLUSIONS AND FUTURE WORK

We studied the novel problem of optimal sequenced route query in both vector and metric spaces. To tackle the problem, we first proposed a Dijkstra-based approach and showed that it is not efficient for large point sets and routes of large number of points. We described our novel threshold-based algorithm, LORD, which is applicable on vector spaces. Furthermore, we proposed R-LORD which utilizes an R-tree index structure to address OSR queries in vector spaces. Our extensive experiments showed the followings:

- R-LORD is *light* in terms of required workspace because as compared to the Dijkstra-based approach it always reduces the required workspace by a factor of (on average) 55%-90%. The maximum of this space reduction reaches 99.6% for some instances of our experiments.

- R-LORD is *efficient* in terms of query response time as it answers an OSR query in a time which increases with almost a linear rate as the sequence size $|M|$ increases. R-LORD's response time for large sequence size $|M|$ of 12 is less than a second as compared to 40 seconds response time of the Dijkstra's classic algorithm for $|M| = 3$ on a small dataset.

- R-LORD is *efficient* in terms of I/O as it accesses at most 10.5% of the R-tree nodes while iterating to complete its set of partial routes to answer an OSR of sequence size $|M| \leq 12$.

To overcome LORD's extensive usage of distance function, we proposed PNE, a progressive OSR algorithm for metric spaces that generates the optimal route from the starting to the ending point. We showed that the overhead of PNE is negligible as compared to the nearest neighbor approach that it employs.

We plan to extend our definition of OSR query to include more general precedence constraints on the points of the optimal route. Moreover, we have observed that a caching scheme can be combined with a pre-computation approach to scale our algorithms with respect to the number of queries. We intend to investigate the impact of using these approaches for the users' frequently used sequence constraints.

8. REFERENCES

[1] N. Ascheuer, M. Jünger, and G. Reinelt. A branch & cut algorithm for the asymmetric traveling salesman problem with precedence constraints. *Comput. Optim. Appl.*, 17(1):61–84, 2000.

[2] E. Balas, M. Fischetti, and W. R. Pulleyblank. The precedence-constrained asymmetric traveling salesman polytope. *Math. Program.*, 68(3):241–265, 1995.

[3] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The r*-tree: an efficient and robust access method for points and rectangles. In *Proceedings of the 1990 ACM SIGMOD international conference on Management of data*, pages 322–331. ACM Press, 1990.

[4] N. Biggs, E. K. Lloyd, and R. J. Wilson. *Graph Theory, 1736-1936*. Clarendon Press, 1986.

[5] E. P. F. Chan and N. Zhang. Finding shortest paths in large network systems. In *Proceedings of the 9th ACM international symposium on Advances in geographic information systems*, pages 160–166. ACM Press, 2001.

[6] H. Ferhatosmanoglu, I. Stanoi, D. Agrawal, and A. E. Abbadi. Constrained nearest neighbor queries. In *SSTD*, pages 257–278, 2001.

[7] I. T. Hernádvölgyi. Solving the sequential ordering problem with automatically generated lower bounds. In *Operations Research Proceedings 2003*, pages 355–362. Springer Verlag, September 3-5, 2003.

[8] G. R. Hjaltason and H. Samet. Distance Browsing in Spatial Databases. *TODS, ACM Transactions on Database Systems*, 24(2):265–318, 1999.

[9] D. Hutchinson, A. Maheshwari, and N. Zeh. An external memory data structure for shortest path queries. *Discrete Appl. Math.*, 126(1):55–82, 2003.

[10] C. S. Jensen, J. Kolářvr, T. B. Pedersen, and I. Timko. Nearest neighbor queries in road networks. In *Proceedings of the 11th ACM international symposium on Advances in geographic information systems*, pages 1–8. ACM Press, 2003.

[11] S. Jung and S. Pramanik. An Efficient Path Computation Model for Hierarchically Structured Topological Road Maps. In *IEEE Transaction on Knowledge and Data Engineering*, 2002.

[12] M. Kolahdouzan and C. Shahabi. Voronoi-Based K Nearest Neighbor Search for Spatial Network Databases. In *VLDB 2004, Toronto, Canada.*

[13] F. Korn, N. Sidiropoulos, C. Faloutsos, E. Siegel, and Z. Protopapas. Fast Nearest Neighbor Search in Medical Image Databases. In *VLDB'96, Proceedings of 22th International Conference on Very Large Data Bases, September 3-6, 1996, Mumbai (Bombay), India*, pages 215–226. Morgan Kaufmann, 1996.

[14] N. Koudas, B. C. Ooi, K.-L. Tan, and R. Z. 0003. Approximate nn queries on streams with guaranteed error/performance bounds. In *VLDB*, pages 804–815, 2004.

[15] D. Papadias, Q. Shen, Y. Tao, and K. Mouratidis. Group Nearest Neighbor Queries. In *ICDE 2004, 20th International Conference on Data Engineering March 30 - April 2, 2004, Boston, USA.*

[16] D. Papadias, J. Zhang, N. Mamoulis, and Y. Tao. Query Processing in Spatial Network Databases. In *VLDB 2003, Berlin, Germany.*

[17] N. Roussopoulos, S. Kelley, and F. Vincent. Nearest Neighbor Queries. In *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data, San Jose, California, May 22-25, 1995*, pages 71–79. ACM Press, 1995.

[18] T. Seidl and H.-P. Kriegel. Optimal Multi-Step K-Nearest Neighbor Search. In *SIGMOD 1998, Proceedings ACM SIGMOD International Conference on Management of Data, June 2-4, 1998, Seattle, Washington, USA*, pages 154–165. ACM Press, 1998.

[19] P. Sistla, O. Wolfson, S. Chamberlain, and D. S. Modeling and Querying Moving Objects. In *IEEE ICDE 1997,*

Proceedings of the Thirteenth International Conference on Data Engineering, April 7-11, 1997 Birmingham U.K.

[20] Z. Song and N. Roussopoulos. K-Nearest Neighbor Search for Moving Query Point. In *The Seventh International Symposium on Spatial and Temporal Databases, SSTD'2001, Redondo Beach, CA, USA.*

[21] Y. Tao, D. Papadias, and Q. Shen. Continuous Nearest Neighbor Search. In *VLDB 2002, Proceedings of 28th International Conference on Very Large Data Bases, August 20-23, 2002 Hong Kong, China.*