Privacy Preserving Reverse Spatial and Textual Nearest Neighbour Query

Student Name: SIDDHARTH DAWAR

IIIT-D-MTech-CS-IS-12-020
June 09, 2014

Indraprastha Institute of Information Technology
New Delhi

Thesis Committee
Dr. Vikram Goyal
Dr. Debajyoti Bera

Submitted in partial fulfillment of the requirements
for the Degree of M.Tech. in Computer Science,
with specialization in Information Security

Keywords: Reverse Nearest Neighbour Query(RkNN), Spatial-Textual data, IUR Tree, Location Privacy, Location-Based Services

Certificate

This is to certify that the thesis titled **"Privacy Preserving Reverse Spatial and Textual Nearest Neighbour Query"** submitted by **Siddharth Dawar** for the partial fulfilment of the requirements for the degree of *Master of Technology* in *Computer Science & Engineering* is a record of the bonafide work carried out by him under our guidance and supervision in the Security and Privacy group at Indraprastha Institute of Information Technology, Delhi. This work has not been submitted anywhere else for the reward of any other degree.

Dr. Vikram Goyal
Indraprastha Institute of Information Technology, New Delhi

Dr. Debajyoti Bera
Indraprastha Institute of Information Technology, New Delhi

Abstract

The rapid increase in popularity of location-based services have resulted in huge amount of spatial textual data being generated by applications like Foursquare, Facebook Places, Flickr etc. The location-based services offer convenience but threaten the location and query privacy of the user. The data collected by such servers can be to used study user behaviour or for stalking personal locations. A novel query which became popular in the past few years is Reverse k Nearest Neighbour Query (RkNN). Given a set of database objects O and a query point Q, the RkNN query returns those objects $o \in O$, for which Q is one of its k^{th} nearest neighbour, using an appropriately defined similarity function on pairs of database objects. We propose a generalized framework for finding the reverse nearest neighbours of a query point which is independent of the underlying hierarchical indexing structure used as well as the used similarity measure. Our framework is independent of the type of database objects, but the only requirement is to define lower and upper bound similarity between any two object/groups of objects E and E' of the given index structure and calculate the number of objects for every group of objects. We present two different approaches, namely, Lazy and Eager for performing monochromatic Reverse Nearest Neighbour query on spatial textual data. We conduct extensive experiments on real datasets and study the performance of both approaches. We address the problem of performing Reverse Nearest Neighbour (RkNN) search while preserving the location privacy of a user. Location Privacy can be preserved by anonymizing the location of a user using techniques like k-anonymity[1] or l-diversity[2]. The idea is to send a cloaked region to the server instead of the user's exact location so that location privacy is preserved. We formalize the problem of performing Reverse Nearest Neighbour Search on spatial objects when the exact location of database objects is not known to the server. A key challenge in performing such queries is to strike a balance between maintaining the correctness of results versus maintaining the privacy of a user.

Acknowledgments

I would like to thank Dr. Vikram Goyal and Dr. Debajyoti Bera for their guidance, encouragement and immense support during my thesis work. I respect the confidence they showed in me. I would like to thank Dr. Gaurav Gupta and Amit Semwal for their help and support. I would also like to thank Robin Verma and Mridula Singh for their valuable advice and feedback. I am grateful to everyone who has directly or indirectly helped me to achieve this goal. No amount of words written on this page would be sufficient to quantify their advice, prayers, love and support for me.

In addition, I would like to thank my family for supporting me throughout my life.

Contents

List of Figures

Chapter 1

Introduction and Research Aim

Location-based services are the services offered to users based on their locations. Location-based services have many diverse applications like finding the nearest stores in a particular area, location-based advertisement, analysing wildlife and traffic movements, location-based gaming etc. The advancements in database and mobile technology and rapidly increasing popularity of location-based services results in huge amounts of data being collected in databases. Location-based services have attracted significant attention from the industrial and research community. The most common type of query performed on location-based server is Nearest Neighbour query. An example of a Nearest Neighbour query is a person trying to find the nearest pizza store or a soldier finding the nearest enemy troop. Another type of query gaining popularity nowadays is Reverse Nearest Neighbour query. Reverse Nearest Neighbour queries are of two types: monochromatic and bichromatic. In monochromatic RkNN, both query point and database objects are of the same type, while in bichromatic RkNN both are of different types. RkNN query finds applications in decision support systems where the task is to open a new facility like restaurant in an area such that it will be least influenced by its competitors and attract good business. Another application is a profile based marketing [3], where a company maintains profiles of its customers and wants to start a new service such that the service is under the influence of maximum number of customers i.e. maximum number of customers are interested in that service. The RkNN query in decision support system is an example of a monochromatic query as the database objects and the query are of the same type i.e. restaurants. The application of RkNN in profile based marketing is an example of a bichromatic query as the database objects are customers and the query is service to be started by the company.

Location-based services require the users to report their exact location continuously. A user who doesn't want to send his/her exact location has to stop using the location-based services provided by the service provider. The data collected by such servers can be used to study the user behaviour, visiting patterns, stalking personal locations [4]. A compromised location-based server can be used to get historical position information of several users. Krumm et. al [5] presents a literature survey of computational algorithms for compromising and protecting location data. There is a need to find a way such that the user can enjoy the benefits of using location-based services while preserving his/her location privacy. Mokbel et al. [6] proposed a framework for preserving location privacy of a user and defined three types of Nearest Neighbour queries namely, *Public Query over Private Data*, *Private Query over Public Data* and *Private Query over Private Data*. *Public Query over Private Data* signify that the exact location of the querying user is known to the server but the exact location of database objects is unknown. An example of nearest neighbour *Public Query over Private Data* is a location based advertisement

where a restaurant wants to send its advertisement to customers in its vicinity. In this scenario, the customers want to preserve their privacy and the restaurant (query point) wants to find its nearest customers. An example of *Private Query over Public Data* is a customer trying to find the nearest restaurant. An example of *Private Query over Private Data* is a friend finder application, where a user wants to find his/her nearest buddy. Here, both the querying user and his friends (the database objects) want to hide their exact location from the location-based server.

However, our thesis is a privacy preserving evaluation of reverse nearest neighbour query. Privacy preserving reverse nearest neighbour queries also find its applications in many scenarios. Examples of Reverse Nearest Neighbour *Public query over Private data* is a shopping mall trying to find customers which have the shopping mall as one of its k nearest neighbours. Another example is a pizza store owner trying to find customers which have the pizza store in its k nearest neighbours in order to send them discount coupons. An example of *Private Query over Public Data* is a customer finding a good location for his home such that he is in the influence of at least k facilities like School, Hospitals etc. Examples of *Private query over Private Data* include peer to peer applications like file sharing and match fixing application where a match fixer wants to avoid an area where there are other match fixers around him. Another application is location-based gaming, where a gamer wants to be in the influence of at least k friends without revealing their exact location.

1.1 Problem Statement

Reverse Spatial and Textual k Nearest Neighbour Query(RSTkNN): Given a set of objects O in database D and query point Q each represented by a pair (loc,vct) where loc is the spatial location and vct is the associated textual description which is of the form (key,w) $\forall$ key $\in$ doc , where doc is the set of words associated with every database object and w is the weight associated with very word (key), returns those objects o $\in O$ in D that have query point Q as one of its k^{th} Nearest Neighbour. The weights can be calculated on the basis of TF-IDF scheme [7]. The spatial-textual similarity (SimST) is defined by Jiaheng et. al [8] as follows:

$$SimST(o_1, o_2) = \alpha * (1 - \frac{dist(o_1.loc, o_2.loc) - \varphi_s}{\psi_s - \varphi_s}) + (1 - \alpha) * (\frac{EJ(o_1.vct, o_2.vct) - \varphi_t}{\psi_t - \varphi_t}) \quad (1.1)$$

The parameter α is used to define the relevance factor for spatial and textual similarity while calculating the total similarity scores and is specified at query time. φ_s and ψ_s denote the minimum and maximum distance between any two objects in the database and are used to normalize the spatial similarity to the range [0,1]. Similarly φ_t and ψ_t denote the minimum and maximum textual similarity between any two objects in the database. dist is the Euclidean Distance between o_1 and o_2 and EJ is the Extended Jaccard Similarity[9] defined as:

$$EJ(o_1.vct, o_2.vct) = \frac{\sum_{j=1}^{n} o_1.w_j * o_2.w'_j}{\sum_{j=1}^{n} o_1.w_j^2 + \sum_{j=1}^{n} o_2.w'^2_j - \sum_{j=1}^{n} o_1.w_j * o_2.w'_j} \quad (1.2)$$

where o_1.vct=$\langle w_1, ..., w_n \rangle$ and o_2.vct=$\langle w'_1, ..., w'_n \rangle$.

The problem is to perform RSTkNN query with and without preserving the location of the

users. We want to perform RSTkNN public query over private data. In the figure below, we have some regions and a query point Q. For finding the Reverse k Nearest Neighbour of Q, every region needs to find its k^{th} nearest neighbour and the regions who have the query point in their k^{th} nearest neighbour, are the RkNN of the query point.

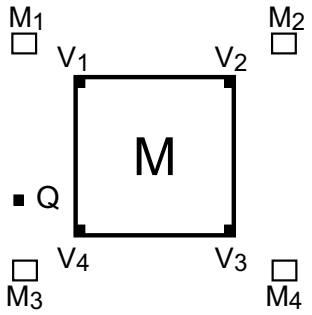


Figure 1.1: Region Objects and Query Point in Space

The notion of minimum and maximum similarity needs to be defined between regions and in between a region and the query point for finding the RkNN of Q. Conditions for accepting or rejecting a region to be the RkNN of Q also needs to be defined.

1.2 Research Contribution

Our contributions can be summarized as follows:

- We propose a generic framework for Reverse Nearest Neighbour Query independent of the underlying indexing structure and the type of data. We also give a proof of correctness of our proposed framework.
- We propose two different approaches, namely, Lazy and Eager for performing monochromatic reverse nearest neighbour query on objects having both spatial and textual information for dynamic values of k and α and algorithm for the same. α is the relevance factor for spatial and textual similarity while calculating total similarity scores.
- We formalize the problem of performing Reverse Nearest Neighbour Public Query over Private Data.
- We conduct extensive experimental studies and compare the performance of our Lazy and Eager approach. We understand the factors responsible for a trade-off in performance of both approaches.

Chapter 2

Related Work

In this section we review the closely related literature. We divide the related work into two sections. The first section covers related work on RkNN Query while the second section covers work done on performing Probabilistic Reverse Nearest Neighbour queries on uncertain data.

2.1 Reverse k Nearest Neighbour Query

Korn et al. [3] introduced the Reverse Nearest Neighbour query, where every database object pre-computes the exact distance of its nearest neighbour (NN) by drawing a sphere around objects of radius equal to the distance of the entry with its nearest neighbour, which is used to decide whether the query point is the nearest neighbour of the object or not. Yang et. al [10] extends the work of Korn et. al [3] by introducing a new indexing structure RdNN tree which can answer both NN and RNN queries. The work done by Korn et. al[3] and Yang et. al[10] compute the exact k^{th} nearest neighbour distance for every object of the underlying indexing structure. However, the above mentioned approaches are limited for fixed value of k and defined only for spatial objects.

Achtert et. al [11] proposed MRkNNCop tree which stores distance approximations of the k^{th} nearest neighbour. They aggregate the maximum and minimum distances to the k^{th} nearest neighbour for objects in every entry to decide true hits or drops. Their approach only works for a fixed range of k i.e. $k \leq k_{max}$. A solution which works for dynamic value of k was proposed by Achtert et. al [12]. The authors proposed a solution to find reverse nearest neighbour for arbitrary values of k in euclidean and metric spaces. Archtert et. al [12] further proposed a framework for Reverse Nearest Neighbour Query which is independent of the hierarchically indexing structure but defined only for spatial objects and the similarity metric is spatial distance.

Tao et. al [13] proposed a RkNN algorithm that works for dynamic value of k without any need for pre-computation and used bisector based pruning techniques. Wu et. al [14] proposed a solution for performing monochromatic and bichromatic reverse nearest neighbour query on two dimensional points by finding and tightening the search space around the query point Q. Cheema et. al [15] used geometric techniques to find the influence zone for a query point Q. Influence Zone is an area around the query point such that all points in that area, are the RkNN of q. However, these techniques can't be applied to find textual similarity between two vectors and are applicable only for finding RkNN in spatial domain.

Spatial keyword queries and Reverse Top k Nearest Neighbour queries are also getting much attention nowadays. A spatial keyword query returns top k objects ordered by spatial and textual relevance. Spatial relevance is ranked by distance and textual relevance is ranked by the similarity with the keywords of the query. Felipe et. al [16] proposed a method to answer top-k spatial keyword queries using IR^2 tree as the underlying indexing structure. IR^2 is an R-tree superimposed with textual signatures. Cong et. al [17] introduced location aware top k text retrieval query which used TF-IDF scheme [7] and IR tree [17] as the indexing structure. IR Tree is an R-Tree with its nodes combined with inverted lists. Cao et al. [18] proposed prestige based spatial keyword query that also takes the number of relevant objects around a point into account along with its textual relevance with the query point. A variant of spatial keyword query is proposed by Li et. al [19] called keyword-based k nearest neighbour query, which returns the k nearest points to the query which contain all the query keywords. The Reverse Spatial and Textual Nearest Neighbour Query is the opposite of spatial keyword queries.

Vlachau et. al[20] introduced Reverse Top-k queries, which finds objects which have query point as their top-k object based on user preferences. Vlachau et. al [21] proposed a branch and bound algorithm for efficiently processing Reverse Top-k queries without accessing individual user preferences or running top k query. However, Reverse Top-k queries are different from RSTkNN, as they consider only user preferences and textual similarity while RSTkNN consider both spatial and textual similarity.

Jiaheng et. al[8] introduced the Reverse Spatial and Textual k Nearest Neighbour Search (RSTkNN) query to find RNN using both spatial and textual information for dynamic values of k and α. The authors proposed an indexing structure named as Intersection Union Tree (IUR-tree), where every node of R-tree was embedded with intersection and union textual vectors. The intersection vector contains the minimum weights of the vectors of objects present in a node of the tree and union vectors contain the maximum weights. The IR^2 Tree [16] and IR Tree [17] are different from IUR Tree as the IUR Tree integrates textual vectors in an R Tree so that textual similarity can be defined between two entries of the R Tree while IR Tree integrates inverted lists and IR^2 Tree integrates signatures to answer top-k and Boolean queries instead of similarity queries.

2.2 Probabilistic Reverse k Nearest Neighbour Search

The work on Probabilistic Nearest Neighbour Query focuses on performing nearest neighbour search when data is uncertain either due to error in measuring equipment or due to privacy concerns. The authors assume a probabilistic database which consists of many uncertain objects. Every uncertain object has a set of possible instances with some assigned probability. Lian et. al [22] proposed an algorithm for finding probabilistic Reverse Nearest Neighbour query where the appearance probability of uncertain objects is represented as continuous PDF. They proposed a spatial and probabilistic pruning scheme for spherical objects but their technique can't be applied for higher dimensional data and for k >1. Cheema et. al [23] also proposed a probabilistic pruning approach along complex spatial pruning approach based on extensive geometric computations. However, their technique is inapplicable for $k > 1$. Emrich et. al [24] proposed a new pruning mechanism and obtained tighter bounds by decomposing target objects. Their solution is not efficient with increment of k and it is non-trivial to find optimal depth to decompose target objects. Li et al. [25] designed a new spatial and probabilistic pruning approach based on conceptual partitioning using angle intervals which works for dynamic k.

Chapter 3

RSTkNN Query

3.1 Generalized Framework for Reverse k Nearest Neighbour Search

We will now propose a generalized framework for answering RkNN queries using a hierarchical tree-like index. Our proposed framework is independent of the explicit indexing structure and is applicable to any type of data as long as similarity can be defined between nodes of the index. For notational consistency, we assume that the leaf nodes of the given index are the data points themselves (to be represented by small letters) and all other nodes contain children nodes. Query point will be denoted by Q, and the dataset by D of size denoted by N. If e' is the k^{th} nearest neighbour of e, then we say that rank$(e', e)= k$ and also write e' as kNN(e). We will use the convention that a point is the 0^{th} nearest neighbour of itself.

Given two sets of data points E and E' and given a similarity measure Sim(.) between them, we define MinSim(E, E') and MaxSim(E, E') as follows:

Definition 1. *MinSim(E, E') gives a lower bound for the minimum similarity between pairs of points from E and E' i.e. $\forall\ e \in E$, $\forall\ e' \in E'$, Sim$(e, e') \geq$ MinSim(E, E').*

Definition 2. *MaxSim(A, B) gives a upper bound for the maximum similarity between pairs of points from E and E' i.e. $\forall\ e \in E$, $\forall\ e' \in E'$, Sim$(e, e') \leq$ MaxSim(E, E').*

Definition 3. *Sim(e, e') is the similarity between two points e and e'. Formally,*
$\forall\ e \in E$, $\forall\ e' \in E'$, minSim$(E, E') \leq$ Sim$(e, e') \leq$ MaxSim(E, E').

We would like to answer RkNN queries for any value of k, and one way to do this is by computing the exact NN(e) list for every data point e: NN$(e)= \langle\ e_1, e_2, e_3, ... \rangle$; here, e_is are other data points such that e_1 is 1NN(e), e_2 is 2NN(e) and so on. Computing this list explicitly for every data point could be very inefficient, hence, our algorithm traverses the index while maintaining two NN-lists with each node - one contains an over-estimate of its nearest neighbours, and another an under-estimate of the same. There are two different approach for traversing the index in a top-down fashion, Depth First Search (DFS) and Breadth-First Search (BFS). We choose to use BFS for traversing the index.

Definition 4. *A NN-list of a node E is a list of tuples: $L = \langle\ (E_1, m_1), (E_2, m_2) ...\ \rangle$, where each E_i is a node and m_i is an integer. The size of such a list, denoted by $|L|$ is $\sum_i m_i$. The NN-lists we will maintain are $NN_U(E)$ and $NN_L(E)$ whose tuples will provide correct estimates*

to the actual r-nearest neighbour (actually we mean, correct estimates to the similarity with r^{th} nearest neighbour) for some r.

Definition 5. *$NN_U(E)$=⟨ (E_1^U, m_1^U),(E_2^U, m_2^U)...(E_l^U, m_l^U) ⟩ is a valid upper contribution list if, ∀j = 1...l, Sim(e, $\sum_{i=1}^{j} m_i^U$)NN(e)) ≤ MaxSim(e,E_j^U) (the j^{th} entry correctly upper bounds some nearest neighbour(s)).*

Definition 6. *$NN_L(E)$=⟨ (E_1^L, m_1^L),(E_2^L, m_2^L)...(E_l^L, m_l^L) ⟩ is a valid lower contribution list if, ∀j = 1...l, Sim(e, $\sum_{i=1}^{j} m_i^L$)NN(e)) ≥ MinSim(e,E_j^L) (the j^{th} entry correctly lower bounds some nearest neighbour(s)).*

Since, we know that the similarity of an entry e with its k^{th} nearest neighbour is more than its similarity, with its k' nearest neighbour as shown in the equation below,

$$Sim(e, kNN(e)) \geq Sim(e, k'NN(e)) \quad \text{for any} \quad k' \geq k \tag{3.1}$$

The lower contribution list provides us correct estimates for r^{th} nearest neighbour for any r ≤ $\sum_i m_i$ and the upper contribution list provides correct estimates for a specific range of r as we will see in the following example. Let $NN_L(e)$= ⟨ $(E_1, 3)$,$(E_2, 5)$,$(E_3, 9)$ ⟩ be a valid lower contribution list. Then, by the property of lower contribution list, we know,

$$\begin{aligned} Sim(e, 3NN(e)) &\geq MinSim(e, E_1) \quad (\text{for} \quad j = 1) \\ Sim(e, 8NN(e)) &\geq MinSim(e, E_2) \quad (\text{for} \quad j = 2) \\ Sim(e, 17NN(e)) &\geq MinSim(e, E_3) \quad (\text{for} \quad j = 3) \end{aligned} \tag{3.2}$$

Also, by equation 3.1, Sim(e,1NN(e)) ≥ Sim(e,3NN(e))
Therefore, by equation 3.2,
Sim(e,1NN(e)) ≥ MinSim(e,E_1)
Similarly, Sim(e,2NN(e)) ≥ MinSim(e,E_1)
Sim(e,iNN(e)) ≥ MinSim(e,E_2) ∀ i=4...7
Sim(e,iNN(e)) ≥ MinSim(e,E_3) ∀ i=9...16
Therefore, Sim(e,iNN(e)) ≥ MinSim(e,E_1) ∀ i=1...m_1^L

Let $NN_U(e)$= ⟨ $(E_4, 2)$,$(E_2, 4)$,$(E_6, 6)$ ⟩ be a valid upper contribution list. Then, by the property of upper contribution list, we know,

$$\begin{aligned} Sim(e, 2NN(e)) &\leq MaxSim(e, E_4) \\ Sim(e, 6NN(e)) &\leq MaxSim(e, E_2) \\ Sim(e, 12NN(e)) &\leq MaxSim(e, E_6) \end{aligned} \tag{3.3}$$

Therefore, by equation 3.1,
Sim(e,3NN(e)) ≤ Sim(e,2NN(e)) ≤ MaxSim(e,E_4)
Sim(e,4NN(e)) ≤ Sim(e,3NN(e)) ≤ MaxSim(e,E_4)
Sim(e,5NN(e)) ≤ Sim(e,4NN(e)) ≤ MaxSim(e,E_4)
Therefore, Sim(e,iNN(e)) ≤ MaxSim(e,E_4) ∀ i=m_1^U...L

A point to be observed is we can't say whether Sim(e,iNN(e))≤ MaxSim(e,E_4) ∀i =1...m_1^U-1, will hold true or not.

We derive additional efficiency by computing and storing estimates for an entire node, instead of individual data points. The above definitions are extended in a straight forward way to nodes. The list L = ⟨ (E_1^L, m_1^L),(E_2^L, m_2^L)...(E_l^L, m_l^L) ⟩ is a valid $NN_L(E)$ list for a node E, if L is a valid $NN_L(e)$ for all $e \in E$. $NN_U(E)$ is similarly defined.

Claim 1. *$NN_L(E)$ is a valid lower contribution list for the region E if and only if $\forall e \in E$, $NN_L(E)$ is also a valid list for e.*

Proof. In order to prove our claim, we need to prove that if $NN_L(E)$ is a valid contribution list for E, it implies that $NN_L(E)$ is a valid contribution list for every children of E and vice versa. Let us assume that $NN_L(E)$ is a valid contribution list for E and $NN_L(E)$ is:
$NN_L(E)$=$\langle$ (E_1,m_1), (E_2,m_2) $\rangle$
A lower contribution list has the following property:
$\forall$ e $\in E$ and $\forall$ $e' \in E_1$
Sim(e,e') $\geq$ MinSim(E,m_1NN(E)).
Similarly, the property holds for E and E_2. Suppose $\exists$ $e'' \in E$, such that $NN_L(E)$ is not a valid contribution list for e'' i.e. there exists a similarity score, say S_1' such that $S_1' <$ MinSim(E,m_1NN(E)) and $\forall$ $e' \in E_1$, Sim(e'', e') $\geq S_1'$, which contradicts our assumption that $NN_L(E)$ is a valid contribution list for E. So, a valid lower contribution list $NN_L(E)$ implies that it is also a valid contribution list for every children of E.
If $NN_L(E)$ is a valid contribution list $\forall$ $e \in E$, it implies that Sim(e,iNN(e)) $\geq$ MinSim(E,m_1NN(E)), for i $\in$ [1,2....,m_1] and, Sim(e,jNN(e)) $\geq$ MinSim(E,m_2NN(E)), for i $\in$ [m_1+1,...,m_2] which implies that $NN_L(E)$ is a valid lower contribution list for E since it is valid for every child of E. Hence Proved. □

We will prove a similar claim for $NN_U(E)$ in the next section.

3.1.1 Computing NN-List

Now we discuss some properties of these NN-lists which can be used to compute them efficiently.

NN_L **lists:** The central idea behind the NN_L list comes from the following observation. Suppose for a set of m points $\{e_1', e_2', ..., e_m'\}$ and another point e, we have that Sim(e, e_i') $\geq$ s. Then, it is obvious that if e does not belong to this set, Sim($e, mNN(e)$) $\geq$ s; and if e belongs to this set, then Sim($e, (m-1)NN(e)$) $\geq$ s. Extending this concept to nodes, consider any node E with m data points; now, if MinSim(E, e) $\geq$ s then, Sim($e, mNN(e)$) $\geq$ s if $e \notin E$ and Sim($e, (m-1)NN(e)$) $\geq$ s if $e \in E$. Notice that these bounds are tight.

We can even extend this idea to multiple nodes to get the following claim. Let e be a data point and $E_1, E_2, ..., E_k$ be a collection of non-overlapping nodes which doesn't contain e, where the list is sorted in decreasing order of MinSim(E_i,e). Let m_i denote the number of data points in E_i, and let s_i be a lower bound on MinSim(E_i, e). Then, for all j= 1...k, Sim(e, ($\sum_{i=1}^{j}$)m_i)NN(e)) $\geq s_j$. If $e \in E_i$, then m_i must be replaced with m_i-1 in all tuples containing m_i.

We will use this concept to compute NN_L lists; in fact, for further optimization we will compute $NN_L(E)$ for nodes E instead of data points. Any list $\langle (E_1, m_1), ... \rangle$ of non-overlapping nodes is a valid $NN_L(E)$ when the following holds:

- the list is sorted in decreasing order of MinSim(E_i, E)
- if E does not overlap any E_i, then $m_i \leq |E_i|$ for all i
- if E overlaps with some E_i, then $m_i \leq |E_i|$ - 1 for that specific i and $m_j \leq |E_j|$ for $j \neq i$
- E doesn't have ancestor-descendant relationship with any $E_i's$ present in its contribution list.

As explained above, the crucial property of this list is that, for any t, if $m_{i-1} < \text{t} \leq m_i$, then for all $e \in E$, $\text{Sim}(e, tNN(e)) \geq \text{MinSim}(e, E_j)$.

NN_U **lists:** Correct computation of these lists requires an additional concept of completeness. We say that an NN-list is complete if it provides estimates for r^{th} ranked nearest neighbour for all possible r; another way, if every data point is present in some node in the NN-list. For example, if e belongs to the index, then L is a complete NN-list for e if $|L| = N$-1. It must be noted that the NN_L list computed above need not be complete. However, similar arguments will not work for NN_U. Take for example, a claim similar to the first claim for NN_L: we have a set of points $\{e'_1, e'_2, ...e'_m\}$ and another point e (all disjoint). But even if we know that $\text{Sim}(e, e'_i) \leq$ s for some s and for all i, it is nevertheless not true that $\text{Sim}(e, mNN(e)) \leq$ s, unless, all points other than e are in the set.

Therefore, similar concepts and results as above can be defined for NN_U as well, as long as we also require that the lists are complete (and of course, using MaxSim instead of MinSim). We are stating the final claim and avoiding other definitions to avoid repetition.

Claim 2. *For a node E, $L = \{(E_1, m_1), ..., (E_l, m_l)\}$ is a valid $NN_U(E)$ if $E'_i s$ are sorted in decreasing order of MaxSim(E_i, E) and L is complete.*

For a node E, L=$\{(E_1, m_1), ...\}$ of non-overlapping nodes is a valid $NN_U(E)$ when the following holds:

- the list is sorted in decreasing order of MaxSim(E_i,E)
- the list of nodes is complete
- if E does not overlap any E_i, then $m_i \leq |E_i|$ for all i
- if E overlaps with some E_i, then $m_i \leq |E_i|$ - 1 for that specific i and $m_j \leq |E_j|$ for $j \neq i$
- E doesn't have ancestor-descendant relationship with any $E'_i s$ present in its contribution list.

We will now prove a claim for $NN_U(E)$ similar to the Claim 1 for $NN_L(E)$.

Claim 3. *$NN_U(E)$ is a valid upper contribution list for the region E if and only if $\forall e \in E$, $NN_U(E)$ is also a valid list for e.*

Proof. In order to prove our claim, we need to prove that if $NN_U(E)$ is a valid contribution list for E, it implies that $NN_U(E)$ is a valid contribution list for every children of E and vice versa. Let us assume that $NN_U(E)$ is a valid contribution list for E and $NN_U E$) is:
$NN_U(E)$=$\langle$ (E_1,m_1), (E_2,m_2) $\rangle$
A upper contribution list has the following property:
$\forall\ e \in E$ and $\forall\ e' \in E_1$
Sim(e,e') $\leq$ MaxSim(E,m_1NN(E)).
Similarly, the property holds for E and E_2. Suppose $\exists\ e'' \in E$, such that $NN_U(E)$ is not a valid contribution list for e'' i.e. there exists a similarity score, say S'_1 such that $S'_1 >$ MaxSim(E,m_1NN(E)) and $\forall\ e' \in E_1$, $\text{Sim}(e'', e') \leq S'_1$, which contradicts our assumption that $NN_U(E)$ is a valid contribution list for E. So, a valid upper contribution list $NN_U(E)$ implies that it is also a valid contribution list for every children of E. If $NN_U(E)$ is a valid contribution list $\forall\ e \in E$, it implies that Sim(e,iNN(e)) $\leq$ MaxSim(E,m_1NN(E)), for i $\in$ [1,2....,m_1] and,

Sim(e,jNN(e)) $\leq$ MaxSim(E,m_2NN(E)), for i $\in [m_1+1,...,m_2]$ which implies that $NN_U(E)$ is a valid lower contribution list for E since it is valid for every child of E. Hence Proved. □

In the previous section we gave an example to demonstrate that lower contribution list provides us correct estimates for r^{th} nearest neighbour for any r $\leq \sum_i m_i$ but upper contribution list provides us correct estimates for a specific range of values. We will extend our previous example below to demonstrate that upper contribution list can also provide us correct estimates for any r $\leq \sum_i m_i$ if the upper contribution list is complete. For example,
Let $NN_U(e)$= $\langle$ $(E_4, 2)$,$(E_2, 4)$,$(E_6, 6)$ $\rangle$ be a valid upper contribution list. In the previous example, we observed that, Sim(e,iNN(e)) $\leq$ MaxSim(e,E_4) $\forall$ i=$m_1^U ...L$
We were not able to claim that Sim(e,iNN(e))$\leq$ MaxSim(e,E_4) $\forall$i =1...m_1^U-1, will hold true.
However, if we assume that $NN_U(e)$ is sorted in decreasing order of similarity score and is complete i.e. every other point is in the upper contribution list of e, we can say that,
Sim(e,iNN(e))$\leq$ MaxSim(e,E_4) $\forall$i =1...m_1^U-1

kNN_L and kNN_U lists: If we are interested in RkNN for a given k, then we may not require to store complete NN-lists for all nodes and points. This fact is obvious for the NN_L list: for any b < N, and any valid $NN_L(E)$ list L, if we choose the first t tuples of L such that $\sum_{i=1}^{t} m_i \geq$ b, to create a list L', then L' is also a valid $NN_L(E)$ list which provides correct estimates for the first b nearest neighbours of data points in E.
For a node E, consider its lists $NN_U(E)$ and $NN_L(E)$.
$NN_L(E)$= $\langle$ (E_1,5), (E_2,2),(E_3,4),(E_4,2) $\rangle$
$NN_U(E)$= $\langle$ (E_1,5), (E_2,2),(E_3,4),(E_4,2) $\rangle$
Let k=7 and $NN_L(E)$,$NN_U(E)$ be valid contribution lists. We will consider $NN_L(E)$ first. We are interested in finding RkNN for k=7 and need upper and lower bound estimates for the7^{th} nearest neighbour. Since, $NN_L(E)$ is sorted is decreasing order of MinSim(E,E_i), we need only the first two tuples as Sim(E,7NN(E)) $\geq$ MinSim(E,E_2). This concept is not applicable to $NN_U(E)$ as although, Sim(E,7NN(E)) $\leq$ MaxSim(E,E_2), but there is no guarantee that the exact 7^{th} nearest neighbour of E will be in the first two tuples, since MaxSim(.) is an upper bound on the similarity with 7^{th} nearest neighbour. So, simply prefixing $NN_U(E)$ like $NN_L(E)$ is incorrect. The $NN_U(E)$ will only have those $E_i's$ which have MaxSim(E,$E_i's$) $\geq$ MinSim(E,$kNN^L(E)$).

Before we derive the conditions for pruning or accepting a point or node to be the RkNN of Q, we would like to state some conditions which must be maintained for ensuring correctness of results. The conditions are as follows:

Locality Condition: A node trying to find its k nearest neighbours must look at the nodes/points it contains along with other nodes/points in the neighbourhood. Let us consider two nodes E_1 and E_2 as shown in the figure 3.1. Assume k=2. If the node E_1 while finding its k nearest neighbours, sees only the points in E_2 without looking at the points contained in itself, it will see itself as the $RkNN$ of Q which is incorrect.

Completeness Condition: Now we will motivate the need for ensuring that contribution lists are complete as maintaining incomplete contribution lists might result in incorrect results. Let us consider an example as shown in the figure 3.2.

Lets assume k=3. Node E_1 sees objects in itself as well as its neighbouring entry E_2. However, since the query point is also located inside E_2, there is a need to access the points inside E_2.

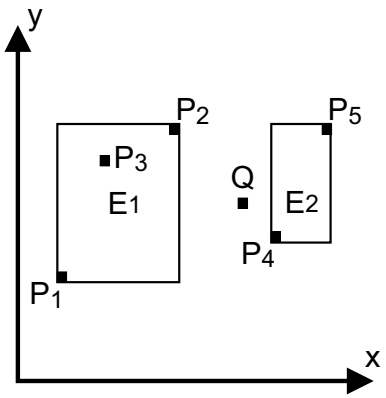


Figure 3.1: Locality Condition

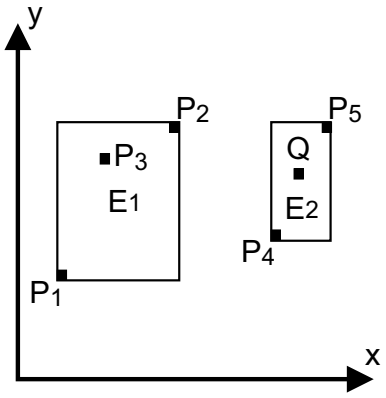


Figure 3.2: Completeness Condition

Now, if E_1 sees point P_5 as its k^{th} nearest neighbour, it will be accepted as the RkNN of q. However, E_1 should have seen both points P_5,P_4 and then decide about its k^{th} nearest neighbour.

3.1.2 Pruning using NN_L lists

1. We will first consider the case when the elements in the lower contribution list of point e are points only. We observe the following:
(a) MinSim(e,e_i^L) $\leq$ Sim(e, iNN(e))
The similarity of e with its i^{th} nearest neighbour(iNN(e)) is greater than or equal to the minimum similarity between e and i^{th} entry in its lower contribution list(e_i^L). The i^{th} entry correctly lower bounds the i^{th} nearest neighbour.
(b) MinSim(e,e_i^L) $\geq$ MinSim(e,e_{i+1}^L)
The minimum similarity of e with the i^{th} entry in its lower CL i.e. its i^{th} nearest neighbour, is greater than or equal to the minimum similarity of e with the $(i+1)^{th}$ entry in its lower contribution list i.e. its $(i+1)^{th}$ nearest neighbour.

2. We will now consider a region R such that $|R|$=m. Assume MinSim$(e, R) = S_1$ which

implies $\forall\ e' \in R$, Sim(e,e') $\geq S_1$.
We can also say that MinSim(e,iNN(e)) $\geq S_1\ \forall$ i = 1 to m,so
$NN_L(e) = \langle\ (R, m)\ \rangle$

3. We will now consider two regions $|R_1|$=m_1, $|R_2|$=m_2 such that MinSim(e,R_1)=S_1 and MinSim(e,R_2)=S_2. Let us assume S_1>=S_2. The lower contribution list of e will be
$NN_L(e) = \langle\ (R_1,m_1),\ (R_2,m_2)\ \rangle$

4. Let us now consider a region R such that, the lower bound list of R is:
$NN_L(R)= \langle\ (R_1,m_1),\ (R_2,m_2)\ \rangle$. We will now derive the condition using which we can prune a point/region to be the RkNN of Q.

Lemma 1. *if $\forall\ e \in R$, sim(e,kNN(e))>MaxSim(R, Q) then R can be pruned.*

Proof. We know that an entry e is not reverse nearest neighbour of Q if, Sim(e, Q)<Sim(e,kNN(e)) i.e. the similarity of e with its k^{th} nearest neighbour is more than its similarity with the query point. We can also say if MaxSim(e, Q)<MinSim(e,kNN(e)), then also e can be pruned.
The above argument implies that if MaxSim(R,Q)<Sim(e,kNN(e)) then e can be pruned, since MaxSim(R,Q) is the upper bound of the actual similarity of every children of R with Q. Therefore, if $\forall\ e \in R$, sim(e,kNN(e))>MaxSim(R, Q) then R can be pruned □

5. Till now, we need to compute the actual similarity of every child entry of R with its k^{th} nearest neighbour to decide whether R can be pruned or not. We now state a pruning theorem using which we can prune an entry without accessing its children. The lower contribution list of R is given by:
$NN_L(R)= \langle\ (R_1, m_1), (R_2, m_2)..... \rangle$
Let us assume that $m_1 \geq k$. The pruning theorem is as follows:

Theorem 1. *If the lower bound similarity of a point/region with its k^{th} nearest neighbour is greater than or equal to its maximum similarity with the query point, then the point/region can be pruned i.e. if MinSim(R, R_1) $\geq$ MaxSimR, Q), then R can be pruned.*

Proof. MinSim(R,kNN(R)) gives a lower bound of its similarity with its k^{th} nearest neighbour. A point/region can be safely pruned if MaxSim(R, Q) $\leq$ MinSim(R,$kNN(R)$), as it guarantees that there are at least k objects whose similarity is at least equal to the maximum similarity of the point/region with the query point. □

3.1.3 Accepting using NN_L and NN_U lists

1. We will first consider the case when the elements in the upper contribution list of point e are points only. We observe the following:
(a) MaxSim(e,e_i^U) $\geq$ Sim(e, iNN(e))
The similarity of e with its i^{th} nearest neighbour(iNN(e)) is less than or equal to the maximum similarity between e and i^{th} entry in its upper contribution list(e_i^U). The i^{th} entry correctly upper bounds the i^{th} nearest neighbour.
(b) MaxSim(e,e_i^U) $\geq$ MaxSim(e,e_{i+1}^U)
The maximum similarity of e with the i^{th} entry in its upper contribution list i.e. its i^{th} nearest neighbour, is greater than or equal to the maximum similarity of e with the $(i+1)^{th}$ entry in its upper contribution list i.e. its $(i+1)^{th}$ nearest neighbour.

2. We will now consider a region R such that $|R|$=m. We assume that NN_U(R) is complete and MaxSim$(e, R) = S_1$ which implies $\forall\ e' \in R$, Sim(e,e') $\leq S_1$.
We can also say that MaxSim(e,iNN(e)) $\leq S_1\ \forall$ i = 1 to m,so
$NN_U(e) = \langle\ (R, m)\ \rangle$

3. We will now consider two regions $|R_1|$=m_1, $|R_2|$=m_2 such that MaxSim(e,R_1)=S_1 and MaxSim(e,R_2)=S_2. Let us assume S_1>=S_2. The upper contribution list of e will be
$NN_U(e) = \langle\ (R_1,m_1),\ (R_2,m_2)\ \rangle$

4. Let us now consider a region R such that the upper contribution list of R is:
$NN_U(R)= \langle\ (R_1,m_1),\ (R_2,m_2)\ \rangle$. We will now derive the condition using which we can accept an entry to be the RkNN of Q.

Lemma 2. *if $\forall\ e \in R$, sim(e,kNN(e))<MinSim(R, Q) then R is the RkNN of Q.*

Proof. We know that an entry e is the reverse nearest neighbour of Q if, Sim(e, Q)>Sim(e,kNN(e)) i.e. the similarity of e with its k^{th} nearest neighbour is less than its similarity with the query point. We can also say if MaxSim(e,kNN(e))<MinSim(e,Q)), then also e belongs to the result. The above argument implies that if Sim(e,kNN(e))< MinSim(R,Q) then e is the RkNN of Q, since MinSim(R,Q) is the lower bound of the actual similarity of every children of R with Q. Therefore, if $\forall\ e \in R$, sim(e,kNN(e))<MinSim(R, Q) then R is the RkNN of Q. □

5. The lower contribution list can also be used to decide whether a region R is the RkNN of Q or not. A region R is the RkNN of Q if $\forall\ e \in R$, Sim($e, kNN(e)$) < MinSim(e, Q). However, for deciding about the region R to be the RkNN using lower contribution list requires to access the children of R. We now state a result theorem using which we can add an entry to the results without accessing its children. The upper contribution list of R is given by:
$NN_U(R)= \langle\ (R_1, m_1), (R_2, m_2)..... \rangle$
Let us assume that $m_1 \geq k$. The result theorem is as follows:

Theorem 2. *If the upper bound similarity of an entry with its k^{th} nearest neighbour is less than its minimum similarity with the query point, then the entry can be added to the results i.e. if S_1 $\leq$ MinSimR, Q) such that $m_1 \geq k$, then R can be added to the results.*

Proof. MaxSim(R,kNN(R)) gives a upper bound of its similarity with its k^{th} nearest neighbour. A point/region can be added to the results if MinSim(R, Q) > MaxSim(R,$kNN(R)$), as it guarantees that there are at most k objects whose similarity is smaller than the minimum similarity of the point/region with the query point. Completeness condition must be satisfied to guarantee the correctness of results.

□

3.2 Public Query over Public Data

We will now discuss how to perform Reverse k Nearest Neighbour Query on spatio-textual data when both database objects and query point are public i.e. the location-based server knows the exact location of the database objects and query point, for dynamic values of α and k. For the sake of clarity, we refer to the nodes/points of the IUR tree as entries. Jiaheng et. al [8] proposed similarity metrics and algorithms to perform RSTkNN query when both query object and database objects are public. We will discuss their proposed solutions and give

counter examples to prove the in-correctness of their proposed similarity metrics and algorithm. After discussing the conditions necessary for correctness of results, we will discuss our proposed algorithm and will give a proof of correctness.

3.2.1 Similarity Approximations

Definition 7. *(MinST) The authors[8] defined minimum similarity between two entries E and E′ as follows:*

$$MinST(E,E') = \alpha * (1 - \frac{MaxS(E,E') - \varphi_s}{\psi_s - \varphi_s}) + (1-\alpha) * (\frac{MinT(E,E') - \varphi_t}{\psi_t - \varphi_t}) \tag{3.4}$$

MaxS is the maximum Euclidean Distance between E and E′ and MinT is the minimum textual similarity defined as:

$$MinT(E,E') = \frac{\sum_{j=1}^{n} E.w_j * E'.w_j}{\sum_{j=1}^{n} E.w_j^2 + \sum_{j=1}^{n} E'.w_j^2 - \sum_{j=1}^{n} E.w_j * E'w_j} \tag{3.5}$$

where the weights are assigned according to the equation below:

$$\begin{aligned} &E.w_j = E.u_j, E'.w_j = E'.i_j \quad if \quad E.i_j * E.u_j \geq E'.i_j * E'.u_j \\ &E.w_j = E.i_j, E'.w_j = E'.u_j \quad otherwise \end{aligned} \tag{3.6}$$

Definition 8. *(MaxST) The maximum similarity between two entries E and E′ is defined as follows:*

$$MaxST(E,E') = \alpha * (1 - \frac{MinS(E,E') - \varphi_s}{\psi_s - \varphi_s}) + (1-\alpha) * (\frac{MaxT(E,E') - \varphi_t}{\psi_t - \varphi_t}) \tag{3.7}$$

MinS is the minimum Euclidean Distance between E and E′ and MaxT is the maximum textual similarity defined as:

$$MaxT(E,E') = \frac{\sum_{j=1}^{n} E.w_j * E'.w_j}{\sum_{j=1}^{n} E.w_j^2 + \sum_{j=1}^{n} E'.w_j^2 - \sum_{j=1}^{n} E.w_j * E'w_j} \tag{3.8}$$

where the weights are assigned according to the equation below:

$$\begin{aligned} &E.w_j = E.i_j, E'.w_j = E'.u_j \quad if \quad E.i_j > E'.u_j \\ &E.w_j = E.u_j, E'.w_j = E'.i_j \quad if \quad E.u_j < E'.i_j \\ &E.w_j = E'.w_j = E.u_j \quad if \quad E'.i_j \leq E.u_j \leq E'.u_j \\ &E.w_j = E'.w_j = E'.u_j \quad otherwise \end{aligned} \tag{3.9}$$

Definition 9. *(Similarity Preserving Function): Given two functions fsim: $V \times V \rightarrow \mathbb{R}$ and fdim: $\mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$, where V denotes the domain of n-element vectors and $\mathbb{R}$, the real numbers. fsim is a similarity function w.r.t fdim, such that for any three vectors $\vec{p}$= $\langle\ x_1 ..., x_n\ \rangle$, $\vec{p'}$ =$\langle\ x_1{}' ..., x_n{}'\ \rangle$,$\vec{p''}$ = $\langle\ x_1{}'', ... x_n{}''\ \rangle$, if $\forall\ i \in$ [1,n], fdim(x_i,$x_i{}'$) $\geq$ dim(x_i,$x_i{}''$), then we have fsim($\vec{p}$,$\vec{p'}$)$\geq$ fsim($\vec{p}$,$\vec{p''}$).*

Claim 4. *Euclidean Distance function is similarity preserving function, wrt function fdim(x,x′) = |x-x′|.*
Lets consider the scenario where we have points in two dimensions X and Y, and we consider three points $p_1(x_1, y_1)$, $p_2(x_2, y_2)$ *and* $p_3(x_3, y_3)$. *dist*(p_1, p_2)=$\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$ *and dist*(p_1, p_3)=$\sqrt{(x_3 - x_1)^2 + (y_3 - y_1)^2}$. *Now if* $|x_2 - x_1| \geq |x_3 - x_1|$ *and* $|y_2 - y_1| \geq |y_2 - y_1|$, *then dist*$(p_1, p_2) \geq$ *dist*(p_1, p_3). *So Euclidean distance is similarity function wrt |x-x′|.*

Claim 5. *Extended Jaccard is similarity preserving function, wrt function fdim(x,x′) =* $\frac{min(x,x')}{max(x,x')}$, *x,x′ >0.*

Lemma 3. *MinST(E,E′) satisfies the property that* $\forall\ o \in E$, $\forall o' \in E'$, *SimST(o,o′) ≥ MinST(E,E′) and MaxST(E,E′) satisfies the property that* $\forall o \in E$, $\forall o' \in E'$, *SimST(o,o′) ≤ MaxST(E,E′).*

Proof. Based on the claim 4 and 5, the authors tried to prove Lemma 3. The detailed proof can be referred from [8]. We will consider one case of MinST, to get an idea of the derivation. Consider the case when $\sqrt{E.i_j * E.u_j} \geq \sqrt{E'.i_j * E'.u_j}$ i.e. $\frac{E'.i_j}{E.u_j} \leq \frac{E.i_j}{E'.u_j}$, then for $\forall E.w \in [E.i_j, E.u_j] and \forall E'.w \in [E'.i_j, E'.u_j]$, $\frac{E'.i_j}{E.u_j} \leq \frac{min(E.w_j, E'.w_j)}{max(E.w_j, E'.w_j)}$. The authors claimed that only the assignments $E.w_j = E.u_j and E'.w_j = E'.i_j$ can guarantee that $\forall o \in subtree(E)$ and $\forall o' \in subtree(E')$, $MinT(E, E') \leq SimT(o, o')$. Similarly, assignments of values are done for other cases of MinT and MaxT. □

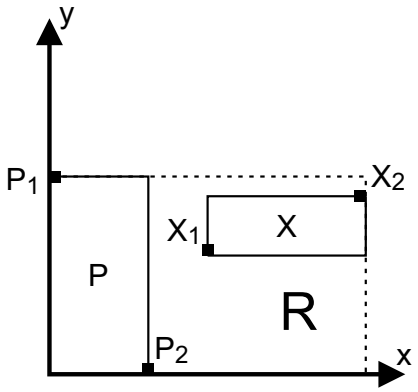


Figure 3.3: Counter Example (MinT and MaxT)

Counter Example 1: Now, we will prove that claim 5 is false by providing a counter example. Let us consider two regions X and P as shown in the figure 3.3, both containing two points each. Region X contains points X_1 with textual vector [100,30] and X_2 with vector [100,35]. Region P contains points P_1 with vector [1,50] and P_2 with vector [1,40]. We will first consider the textual vectors X_1, P_1 and P_2. For the first dimension, we have $\frac{1}{100} \geq \frac{1}{100}$ and for second dimension $\frac{30}{40} \geq \frac{30}{50}$, which according to the claim above about the similarity preserving function of Extended Jaccard implies that EJ(X_1,P_2) ≥ EJ(X_1,P_1). But, the exact textual similarity between X_1,P_2 EJ(X_1, P_2) is 0.116 and EJ(X_1, P_1) is 0.135, which is a contradiction.

Now, we will find MinT and MaxT between regions X and P. The EJ(X_2, P_1) and EJ(X_2, P_2) is 0.155 and 0.132. So MinT and MaxT should come out to be 0.116 and 0.155. The intersection and union of vectors of P and X are:

P.union=[1,50], P.intersection=[1,40]
X.union=[100,35], X.intersection=[100,30]

We will calculate MinT first. For first dimension, we find 100*100 $\geq$ 35*30 so, according to equation (3.3), w=$X.u_j$=100 and w1=$P.i_j$=1. Similarly for next dimension w=$X.i_j$=30 and w1=$P.u_j$=50. MinT comes out to be 0.135 which is incorrect. Now we will calculate MaxT. For the first dimension, we find $X.i_j > P.u_j$ i.e. 100¿1 so w=$X.i_j$=100 and w1=X.u_j=1. Similarly for second dimension we have, X.u_j ¡ P.i_j, so w=X.u_j=35 and w1=P.i_j=40.
The MaxT comes out to be 0.132 which is also incorrect.

Method to calculate MinT and MaxT: We now present a simple method to calculate MinT and MaxT between entries E and E'. The formula for MinT is given below:

$$MinT(E, E') = \frac{\sum_{j=1}^{n} E.i_j * E'.i_j}{\sum_{j=1}^{n} E.u_j^2 + \sum_{j=1}^{n} E'.u_j^2 - \sum_{j=1}^{n} E.i_j * E'i_j} \tag{3.10}$$

The idea for computing MinT is that since it is a lower bound, we want to minimize the numerator and maximize the denominator to ensure that $\forall o \in E$ and $\forall o' \in E'$, EJ(E,E') $\geq$ MinT(E,E'). We ensure that by multiplying intersection vectors in the numerator and dividing by union vectors in the denominator. Similarly formula for MaxT is given below:

$$MaxT(E, E') = \frac{\sum_{j=1}^{n} E.u_j * E'.u_j}{\sum_{j=1}^{n} E.i_j^2 + \sum_{j=1}^{n} E'.i_j^2 - \sum_{j=1}^{n} E.u_j * E'u_j} \tag{3.11}$$

3.2.2 Search Algorithm

Now we will discuss the algorithm proposed by [8] for Reverse Spatial and Textual Nearest Neighbour query and then present counter examples. The algorithm proposed by the authors is as follows.

The algorithm **RSTkNN** takes as an input Intersection Union Tree R, query Q and returns all points, i.e. database objects which are RkNN of Q. The algorithm descends the IUR Tree in a branch and bound manner, computing the upper and lower contribution list for each entry E by inheriting and updating them. Based on the $kNN_L(E)$ and $kNN_U(E)$, the algorithm decides whether to add E to the results, candidates or pruned list. The data structures used: a Priority Queue (U) sorted in decreasing order on MaxST(E,Q), result list (ROL), pruned list (PEL) and candidate list (COL). The root of IUR tree is inserted into U. While U is not empty, an entry P is popped from U. Every child entry E of the parent P inherits (copies) the contribution list of its parent in order to avoid computing contribution list from scratch and the function IsHitorDrop is invoked.

Algorithm 1 RSTkNN(R: IUR-Tree root,Q: query)

```
1: Output: All objects o, s.t o ∈RSTkNN(Q,k,R).
2: Initialize a priority queue U, and lists COL, ROL, PEL;
3: EnQueue(U,R);
4: while U is not empty do
5:     P ← DeQueue(U); //Priority of U is MaxST(P,Q)
6:     for each child entry E of P do
7:         Inherit(E.CLs, P.CLs);
8:         if IS_HIT_OR_DROP(E,Q)==false then
9:             for each entry E′ in COL,ROL,U do
10:                UPDATECL(E,E′); //update contribution lists of E;
11:                if IS_HIT_OR_DROP(E,Q)=true then
12:                    break;
13:                end if
14:                if E′ ∈ U∪COL then
15:                    UPDATECL(E′,E); //Update contribution Lists of E′ using E.
16:                    if IS_HIT_OR_DROP(E′,Q)==true then
17:                        Remove E′ from U or COL;
18:                    end if
19:                end if
20:                if E is not a hit or drop then
21:                    if E is an index node then
22:                        EnQueue(U,E);
23:                    else
24:                        COL.append(E); //a database object
25:                    end if
26:                end if
27:            end for
28:        end if
29:    end for
30: end while
31: FINALVERIFICATION(COL,PEL,Q);
```

```
32: function Is_Hit_Or_Drop(E: entry, Q:query)
33:     if kNN^L(E)≥MaxST(E,Q) then
34:         PEL.append(E);
35:         return true;
36:     else
37:         if kNN^U(E)<MinST(E,Q) then
38:             ROL.append(subtree(E));
39:             return true;
40:         else
41:             return false;
42:         end if
43:     end if
44: end function
```

An entry is rejected if its lower bound similarity with its k^{th} is greater than its upper bound similarity with query point and accepted if its upper bound similarity with its k^{th} nearest neighbour is less than its minimum similarity with query point.

If the entry still can't be pruned or added to the results, its contribution list is updated from other entries stored in the candidate list, result list, priority queue.

45: **function** UPDATECL(E: ENTRY, E': ENTRY)
46: **for** each tuple $\langle s_i, E'_i, num_i \rangle \in E^L$.CL **do**
47: **if** E'_i=E or E'_i=Parent(E) **then**
48: remove $\langle s_i, E'_i, num_i \rangle$ from E^L.CL; //Clean Conflicts
49: **end if**
50: **end for**
51: **if** kNN^U(E)< MaxST(E,E') **then**
52: E^U.CL ← TOPKMAX(E^U.CL, MaxST(E,E'),1);
53: **end if**
54: **if** kNN^L(E)< MinST(E,E') **then**
55: E^L.CL ← TOPKMAX(E^L.CL, MinST(E,E'),|E'|-1);
56: **end if**
57: **end function**

The UpdateCL function updates the contribution list of entry E with the entry E'. The contribution lists of E are scanned to find and remove the parent of E' or E' itself, if present. After the conflicts are cleaned, E' is added to the contribution lists of E. The TopkMax function returns a tuple of the form $\langle E', simST, nop \rangle$ where simST is minST or maxST with respect to E and nop is the number of points.

58: **function** TOPKMAX(L,F(E,E'),C)
59: Return the t-th triple in contribution list L, where t is the minimum number fulfilling $\sum_{i=1}^{t} L.num_i \geq$k.
60: **end function**

After updating E with all entries of COL, ROL or U, the algorithm checks if E is an internal node of the tree or a point. If E is an internal node, it is added to the Priority Queue, otherwise to candidate list. When the priority queue becomes empty, there might be some candidate objects left. The function Final Verification is called where candidate objects are updated with elements present in pruned list to decide if they belong to results or not. The entries were not updated with elements of pruned list earlier as the algorithm adopts the idea of "Lazy travel down" to save I/O cost. In the final verification algorithm, the points which are present in the pruned lists are given priority over internal nodes. The idea behind updating the candidate object with points first is that entries in the lower level of the tree may give tighter bounds with respect to the candidate object than internal nodes of the tree. The algorithm selects an entry e from the pruned list with the lowest level and updates all candidate objects with respect to e. Finally, the children of e are added to pruned list and if candidate list is still non empty, the process continues.

```
61: function FINAL_VERIFICATION(COL,PEL,Q)
62:     while COL!=φ do
63:         Let E be an entry in PEL with the lowest level;
64:         PEL=PEL-{E};
65:         for each object o in COL do
66:             UpdateCL(o,E); //update contribution lists of o.
67:             if IsHitOrDrop(o,Q)==true then
68:                 COL=COL-{o};
69:             end if
70:         end for
71:         for each child entry E' of E do
72:             PEL=PEL∪{E'}; //access the children of E'
73:         end for
74:     end while
75: end function
```

Now we will present counter examples to prove that the above algorithm is incorrect. Our two counter examples will highlight some important properties which should be maintained for the correctness of the algorithm.

Counter Example 2: Let us consider an IUR Tree as shown in figure 3.4 and a query point Q(30,30) for α=1 and k=2. This counter example will show that locality condition has to be maintained i.e. an internal entry E should see points in itself also along with its neighbours for deciding whether query point is in its kNN or not.

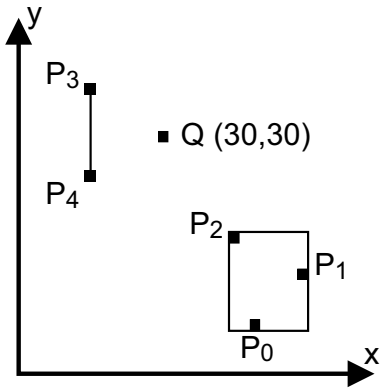
The points in the database are P_0(95,13), P_1(97,17), P_2(94,19), P_3(22,34) and P_4(22,26). The query point is (30,30). We will first present the brute force results for comparison with the RSTkNN algorithm results.ψ_s=77.88 and φ_s=3.61. The similarity of a point with other points are as follows:

P_0 $\langle P_1(0.99), P_2(0.97), P_4(0.05), P_3(0.03)\rangle$. Sim$(P_0, Q)$=0.14
P_1 $\langle P_2(1.0), P_0(0.99), P_4(0.03), P_3(0.01)\rangle$. Sim$(P_1, Q)$=0.13
P_2 $\langle P_1(1.0), P_0(0.97), P_4(0.07), P_3(0.06)\rangle$. Sim$(P_2, Q)$=0.17
P_3 $\langle P_4(0.94), P_2(0.06), P_0(0.03), P_1(0.01)\rangle$. Sim$(P_3, Q)$=0.93
P_4 $\langle P_3(0.94), P_2(0.07), P_0(0.05), P_1(0.03)\rangle$. Sim$(P_4, Q)$=0.93

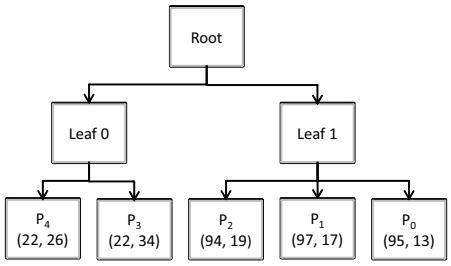
P_3 and P_4 are the RkNN of Q as their similarity with the query point is more than the similarity with k^{th} nearest neighbour. The RSTkNN algorithm proceeds as follows:

1. The root of the tree is dequeued from the priority queue. Its child Leaf 0 inherits the contribution list of its parent, which is empty, so Leaf 0 is enqueued into U.

2. Next Leaf 1 inherits contribution list of its parent and checks whether it can be hit or drop. Since the contribution list is currently empty, Leaf 1 can't be added to the results or pruned. The function UpdateCL is called which updates the contribution list of Leaf 1 by Leaf 0 which is present in the priority queue. The lower and upper contribution list of Leaf 1 is:
NN_L(Leaf 1)= $\langle$ (Leaf 0, 0.02, 2) $\rangle$
NN_U(Leaf 1)= $\langle$ (Leaf 0, 0.07, 2) $\rangle$
MinST(Leaf 1, Q)=0.12, MaxST(Leaf 1,Q)=0.17
Since the kNN^U(E) i.e. 0.07 < MinST(Leaf 1,q), Leaf 1 is added to results. However Leaf 1 would have been pruned had the entry seen points in itself along with its neighbours. The lower and upper contribution list of Leaf 1 should have been:



(a) Distribution of Points



(b) IUR Tree

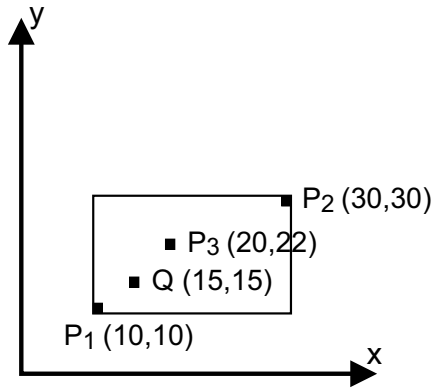
Figure 3.4: Counter Example (Locality condition)

NN_L(Leaf 1)= ⟨ (Leaf 1, 0.96, 2), (Leaf 0, 0.02, 2) ⟩
NN_U(Leaf 1)= ⟨ (Leaf 1, 1.0, 2), (Leaf 0, 0.07, 2) ⟩
Since the kNN^L(E) > MaxST(Leaf 1, Q), Leaf 1 would have been pruned.

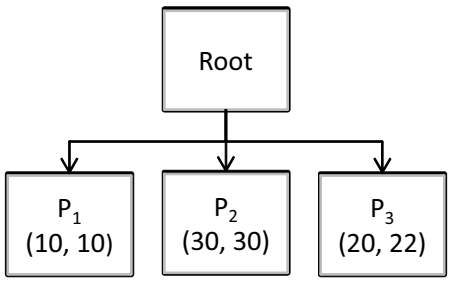
We recently observed that Ying Lu et. al[26] proposed a modified algorithm which maintains the locality condition and before updating an entry E with the objects in COL, ROL and U, sorts the objects in descending order on the basis of MaxST(E,o) ∀o ∈ COL∪ROL∪U. However, the modified algorithm still violates the completeness condition as shown in the following example.

Counter Example 3: Let us consider the distribution of points in space as shown in the figure 3.5 and a query point Q(15,15) for α=1 and k=1. This counter example will show that **completeness condition** has to be maintained i.e. an internal entry E should see all its siblings, only then a decision can be made as completion is necessary for the upper contribution list.

The points in the database are P_1(10,10), P_2(30,30), P_3(20,20). The query point is Q(15,15). We will first present the brute force results for comparison with the RSTkNN algorithm results.ψ_s=28.28



(a) Distribution of Points



(b) IUR Tree

Figure 3.5: Counter Example (Completeness condition)

and φ_s=12.80. The similarity of a point with other points are as follows:

P_1 $\langle P_3(0.90), P_2(0.0)\rangle$. Sim$(P_1, Q)$=1.0
P_2 $\langle P_3(1.0), P_1(0.0)\rangle$. Sim$(P_2, Q)$=0.45
P_3 $\langle P_1(0.90), P_2(0.0)\rangle$. Sim$(P_3, Q)$=1.0
P_1 and P_3 are the RkNN of Q as their similarity with the query point is more than the similarity with k^{th} nearest neighbour. The RSTkNN algorithm proceeds as follows:

1. The root of the tree is dequeued from the priority queue. Its first child P_1 inherits the contribution list of its parent, which is empty, so P_1 is enqueued into U since there is no element in any list(COL,ROL,U) to update contribution list of P_1.

2. P_2 inherits contribution list of its parent and checks whether it can be hit or drop. Since the contribution list is currently empty, P_2 can neither be added to the results or pruned. The function updateCL is called and the contribution list of P_2 is updated by P_1 which is in the U. The contribution list of P_2 is as follows:

$NN_L(P_2)$= ⟨ (P_1,0,1) ⟩
NN_U(Leaf 1)= ⟨ (P_1,0,1) ⟩
MinST(P_2, Q)=MaxST(P_2,Q)=0.5
Since the kNN^U < MinST(P_1, Q) i.e. 0.5, P_1 is added to results. P_1 would have been pruned had it been updated by its exact 1^{st} nearest neighbour i.e. P_3. A point to be observed is that even retrieving entries E' for updating the contribution list of E in sorted order on the basis of maximum similarity with the entry E doesn't help.

Our Algorithm for Reverse Spatial and Textual Nearest Neighbour Search: Now we present our correct algorithm.

Algorithm 2 Our_RSTkNN(R: IUR-Tree root,Q: query)

```
1: Output: All objects o, s.t o ∈RSTkNN(Q,k,R).
2: Initialize a FIFO queue U, and lists COL, ROL, PEL;
3: EnQueue(U,R);
4: while U is not empty do
5:     E ← DeQueue(U); //FIFO Queue
6:     for each tuple ⟨ s_i,E'_i,num_i ⟩ ∈ E^L.CL do
7:         if E'_i=E or E'_i=Parent(E) then
8:             remove ⟨ s_i,E'_i,num_i ⟩ from E^L.CL; //remove parent of E
9:         end if
10:    end for
11:    if ( thenE is an internal node)
12:        Additself(E) //Add E to its contribution list
13:    end if
14:    for each entry E' in U do
15:        UPDATECL(E,E'); //update contribution lists of E;
16:        UPDATECL(E',E); //update contribution lists of E';
17:    end for
18:    if E is not a hit or drop then
19:        if E is an index node then
20:            AddChild(U,E); //Add all children of E to U after each child inheriting CL of E
21:        else
22:            COL.append(E); //a database object
23:        end if
24:    end if
25: end while
26: FINAL_VERIFICATION_LAZY_TRAVEL_DOWN(COL,PEL,ROL,Q);
```

The algorithm enqueues the root of the IUR Tree in the FIFO queue. While the queue is not empty, an entry E is dequeued from the queue. The parent of E is removed from its contribution list and E is added to its contribution list to satisfy the locality condition. The contribution list of E is updated with every entry present in the queue and IsHitorDrop is invoked. If E still can't hit or drop and is an index node, i.e. an internal node, all its children inherit the contribution list from E and are added to U, else E is added to candidate object list.

We introduce two different approaches for inherit namely, Lazy and Eager. **In the lazy approach**, each entry, simply copies the contribution list of its parent in order to save computation. However, in **eager approach** each entry copies all the elements of its parent contribution list and computes its similarity from the elements present in its parent's CL to get tighter scores. Next we present two versions of our Final Verification Algorithm. Our first algorithm follows the lazy travel down approach. For each object belonging to the candidate list, the algorithm updates candidate object o with all elements in the result list and pruned list. However, even

after updating the contribution list of the candidates, it can't be hit or dropped because its contribution list contains some pruned internal nodes. The algorithm scans the upper and lower contribution list of object o, finds the first internal node and replaces it with its children. This process is repeated till its contribution list contains all points. It is possible that after running Final_Verification_Lazy_Travel_Down algorithm, there are some candidates left. These candidates are the points which should have been pruned but because of the property of the lower contribution list, the internal nodes containing k^{th} nearest neighbour were not able to survive the prefix filtering of the lower contribution list. The remaining candidates are simply added to the pruned list.

```
27: function FINAL_VERIFICATION_LAZY_TRAVEL_DOWN(COL,PEL,,ROL,Q)
28:     while COL!=φ do
29:         for each object o in COL do
30:             for each object r in ROL do
31:                 UpdateCL(o,r); //update contribution lists of o.
32:             end for
33:             for ( doeach object p in PEL)
34:                 UpdateCL(o,p); //update contribution lists of o.
35:             end for
36:             if IsHitOrDrop(o,Q)==true then
37:                 COL=COL-{o};
38:             else
39:                 while IsHitOrDrop(o,Q)!=true do
40:                     for each children e of index node E in PEL do
41:                         PEL=PEL-{E};
42:                         UpdateCL(o,child(e))
43:                     end for
44:                     if IsHitOrDrop(o,Q)==true then
45:                         break;
46:                     end if
47:                 end while
48:             end if
49:         end for
50:         Add the remaining candidate objects to Pruned List.
51:     end while
52: end function
```

Final_Verification_Eager_Travel_Down algorithm is the eager version of the first algorithm where the algorithm eagerly traverses down the IUR Tree and update all candidate objects, with all the points in the results and the pruned list. The pruned list contains only points, i.e. all the internal nodes are replaced by the points before updating candidate objects.

```
53: function FINAL_VERIFICATION_EAGER_TRAVEL_DOWN(COL,PEL,,ROL,Q)
54:     PEL= SubTree(PEL) //Replace all index nodes with points so that PEL contain points
    only
55:     while COL!=φ do
56:         for each object o in COL do
57:             for each object r in ROL do
58:                 UpdateCL(o,r); //update contribution lists of o.
59:             end for
60:             for ( doeach object p in PEL)
61:                 UpdateCL(o,p); //update contribution lists of o.
62:             end for
63:             if IsHitOrDrop(o,Q)==true then
64:                 COL=COL-{o};
65:             end if
66:         end for
67:     end while
68: end function
```

We will now illustrate our algorithm with an example. Consider the IUR Tree shown in the figure 3.6. We have five points P_0(95,13), P_1(97,17), P_2(94,19), P_3(22,34) and P_4(22,26). Query

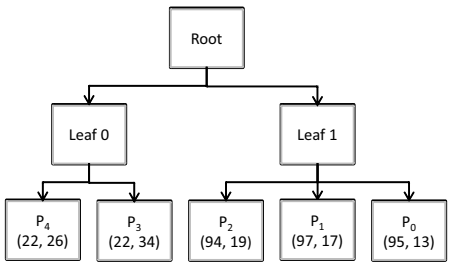


Figure 3.6: IUR Tree

point Q(94,18), α=1 and k=2. We will first present the brute force results for comparison with the RSTkNN algorithm results.ψ_s=77.88 and φ_s=3.61. The similarity of a point with other points are as follows:

P_0 $\langle P_1(0.99), P_2(0.97), P_4(0.05), P_3(0.03)\rangle$. Sim$(P_0, Q)$=0.98
P_1 $\langle P_2(1.0), P_0(0.99), P_4(0.03), P_3(0.01)\rangle$. Sim$(P_1, Q)$=1.0
P_2 $\langle P_1(1.0), P_0(0.97), P_4(0.07), P_3(0.06)\rangle$. Sim$(P_2, Q)$=1.0
P_3 $\langle P_4(0.94), P_2(0.06), P_0(0.03), P_1(0.01)\rangle$. Sim$(P_3, Q)$=0.06
P_4 $\langle P_3(0.94), P_2(0.07), P_0(0.05), P_1(0.03)\rangle$. Sim$(P_4, Q)$=0.07

P_0,P_1 and P_2 are the RkNN of Q as their similarity with the query point is more than the similarity with k^{th} nearest neighbour. The RSTkNN algorithm proceeds as follows:

1. The root of the IUR Tree is dequeued from FIFO Queue. The upper and lower contribution list of the root is:

NN_L(Root)=⟨ (Root,0.0,4) ⟩
NN_U(Root)=⟨ (Root,1.0,4) ⟩
MinST(Root,Q)=MaxST(Root,Q)=1.0
The root can't be hit or dropped, so its children Leaf 0 and Leaf 1 inherit the contribution list of their parent and added to queue.

2. Leaf 0 is popped from the queue and it adds itself to its contribution list.Leaf 0 updates its contribution list with Leaf 1 and vice versa. The upper and lower contribution list of Leaf 0 is:
NN_L(Leaf 0)=⟨ (Leaf 0,0.94,1),(Leaf 1,0.0,3) ⟩
NN_U(Leaf 0)=⟨ (Leaf 0,1.0,1),(Leaf 1,0.07,3) ⟩
MinST(Leaf 0,Q)=0.06,MaxST(Leaf 0,Q)=0.07
The function IsHitorDrop is called and since Leaf 0 can't be pruned or added to the results, its children P_3 and P_4 inherit contribution list of Leaf 0 and are added to the queue.

3. Leaf 1 is popped from the queue and it adds itself in its contribution list. Leaf 1 updates its contribution list with the elements P_3 and P_4 present in the queue and vice versa. The contribution list of Leaf 1 after updation is:
NN_L(Leaf 1)=⟨ (Leaf 1,0.96,2) ⟩
NN_U(Leaf 1)=⟨ (Leaf 1,1.0,2) ⟩
MinST(Leaf 1,Q)=0.97,MaxST(Leaf 1,Q)=1.0
The points P_3 and P_4 don't survive the prefix filtering of upper and lower contribution lists and are removed from the respective lists. Since Leaf 1 can't be hit or dropped, its children P_0,P_1 and P_2 inherit the contribution list of Leaf 1 and are added to the queue.

4. P_4 is popped from the queue and it updates its contribution list with all other points present in U and vice versa. The upper and lower contribution list of P_4 after updation is:
$NN_L(P_4)$=⟨ (P_3,0.94,1),(P_2,0.07,1) ⟩
$NN_U(P_4)$=⟨ (P_3,0.94,1),(P_2,0.07,1) ⟩
MinST(P_4,Q)=MaxST(P_4,Q)=0.07
Since $kNN^L(P_4) \geq$ MaxST(P_4, Q), it is pruned.

5. P_3 is popped from the queue and it updates its contribution list with all other points present in U and vice versa. The upper and lower contribution list of P_3 after updation is:
$NN_L(P_3)$=⟨ (P_4,0.94,1),(P_2,0.06,1) ⟩
$NN_U(P_3)$=⟨ (P_4,0.94,1),(P_2,0.06,1) ⟩
MinST(P_3,Q)=MaxST(P_3,Q)=0.05
Since $kNN^L(P_3) \geq$ MaxST(P_3, Q), it is pruned.

6. P_2 is popped from the queue and it updates its contribution list with all other points present in U and vice versa. The upper and lower contribution list of P_2 after updation is:
$NN_L(P_2)$=⟨ (P_1,1.0,1),(P_0,0.97,1) ⟩
$NN_U(P_2)$=⟨ (P_1,1.0,1),(P_0,0.97,1) ⟩
MinST(P_2,Q)=MaxST(P_2,Q)=1.0
Since $kNN^U(P_2) \leq$ MinST(P_2, Q), it is added to the results.

7. P_0 is popped from the queue and it updates its contribution list with all other points present in U and vice versa. P_0 is updated with P_2 present in the result list. The upper and lower contribution list of P_0 after updation is:

$NN_L(P_0)$=⟨ (P_1,0.99,1),(P_2,0.97,1) ⟩
$NN_U(P_0)$=⟨ (P_1,0.99,1),(P_2,0.97,1) ⟩
MinST(P_0,Q)=MaxST(P_0,Q)=0.98
Since $kNN^U(P_0) \leq$ MinST(P_0, Q), it is added to the results.

8. P_1 is popped from the queue and it updates its contribution list with all other points present in U and vice versa. P_1 is updated with P_2 and P_0 present in the result list. The upper and lower contribution list of P_1 after updation is:
$NN_L(P_1)$=⟨ (P_2,1.0,1),(P_0,0.99,1) ⟩
$NN_U(P_1)$=⟨ (P_2,1.0,1),(P_0,0.99,1) ⟩
MinST(P_1,Q)=MaxST(P_1,Q)=1.0
Since $kNN^U(P_1) \leq$ MinST(P_1, Q), it is added to the results.
After the termination of our algorithm the lists are as follows:
Result list(ROL) = $\langle P_0, P_1, P_2 \rangle$
Pruned list(PEL) = $\langle P_3, P_4 \rangle$
Candidate list(COL) = $\langle \phi \rangle$

Theorem 3. *Given an integer k, a query point Q, and an index tree R, the algorithm 2 correctly returns all RSTkNN points.*

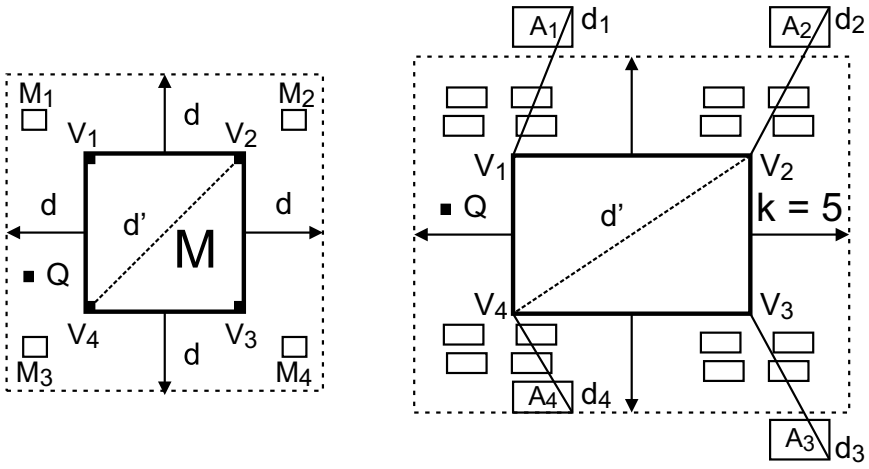
The proof is similar to that given by Jiaheng et al.[8] and is omitted here to avoid repetition. The only difference in the proof is the lemmas for MinST and MaxST proposed by [8] were incorrect and our proposed method for calculating MinST and MaxST can be plugged in.

3.3 Public Query over Private Data

In this section, we formalize the problem of performing Reverse Nearest Neighbour Query when the database objects are not points but regions such that user can be anywhere within the region and the exact location of the querying user is known to the location-based server. We consider spatial distance only as the measure of similarity between two objects. In deciding whether an entry E is in the RkNN of the query point or not, E has to compute the similarity with k^{th} nearest neighbour either exactly or approximately by computing a lower and upper bound of the exact similarity. Let us consider the figure below. The region M wants to find out its k^{th} nearest neighbour and then decide about the query point. The idea is that region M will expand its cloaked area in every direction, like done in the range query until it finds its k nearest neighbours as shown in the figure 3.7(a). We will now propose pruning and accept rules for a region R in space and how to decide the optimal range(d) for the range query.

The MBR M has four corner points named as V_1,V_2,V_3 and V_4. The idea is to compute the maximum distance of each vertex V to its k^{th} nearest neighbour and then decide about the distance for range query around MBR M. We now present the accept and reject rules and then present a solution for finding the optimal range.

A region E can be the RkNN of Q if the extended region around the MBR contains less than k fully overlapped and partial overlapped regions. This ensures that query point is one of the k^{th} nearest neighbour of the region E since the distance of entry E to its query point is less than the distance to its k^{th} nearest neighbour.



(a) (b)

Figure 3.7: Public Query over Private Data

If the entry E contains less than k fully overlapped regions, but the number of fully and partially overlapped regions are greater than k, the entry E can be added to the candidate set. In the regions belonging to the candidate set, exact distance computations have to be made by the querying user on the client side and the location-based server will simply return candidates and result set to the querying user.

We can only prune a region E with respect to Q only if the following condition is satisfied:
$\forall$ p $\in$ E, if the distance(p,Q) > distance(p,$k^{th}NN$), then region E an be pruned. The distance d by which the region will be extended should be such that we can guarantee that every point in the region will find its k^{th} nearest neighbour within that distance. The distance d is defined as follows. We find the maximum distance of the k^{th} nearest neighbour for each vertex(V_1,V_2,V_3,V_4) of the region. Let us assume as shown in figure 3.7(b), that k=5 and the k^{th}NN for vertices V_1, V_2, V_3 and V_4 are A_1, A_2, A_3 and A_4 respectively. Let these distances be denoted as d_1, d_2, d_3 and d_4 respectively. Let d' be the distance between the antipodal corners (i.e. two opposite corners) of the region. Therefore, distance d can be defined as d=d'+ m, where m=min{ $d_{max}kNN(V_1), d_{max}kNN(V_2), d_{max}kNN(V_3), d_{max}kNN(V_4)$ }.In figure 3.7(b), m=d_4. The definition of d guarantees that every point p in the region will find its k^{th} nearest neighbour within the distance d.

Chapter 4

Experiments and Results

In this chapter, we present our experimental results and study the effect of different parameters on our algorithm. We implemented our proposed algorithm in Java on an Intel Xeon (R) CPU=26500@2.00 GHz with 64 GB of RAM. We used two real world data sets namely, SimpleGeoPlaces[27] dataset with 30k points of interest and GeographicNames[28] dataset with 25k points of interest. SimpleGeoPlaces is a dataset containing points of interest with spatial coordinates and textual descriptions. We extracted our textual descriptions from its tags, category and subcategory elements. GeographicNames is a real life dataset from the United States Board on Geographic Names. We extracted the locations of points and textual vectors from the feature name, feature class and county name elements. The statistics of our dataset are shown in figure 4.1: The GeographicNames dataset has more number of unique keywords, textual descriptions

Statistics Shop CD	SGP	GN
total # of objects	30,000	25,000
total unique words in dataset	2110	20,757
average # words per object	4	4

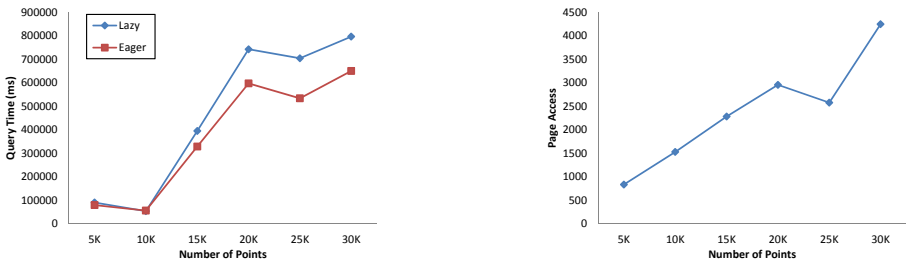
Figure 4.1: Datasets Statistics

are more sparse compared to SimpleGeoPlaces dataset. We generated for every dataset, a set of 25 queries randomly. We ran our experiments on different range of parameters as shown in figure 4.2:

Parameter	Range	Default
k	2-10	10
Alpha	0-1	0.4
Number of Query Words	1-4	4
Number of Points	1-30,000	15,000

Figure 4.2: Range of Parameters

Performance of Eager vs. Lazy RSTkNN: We implemented two version of our algorithm Lazy RSTkNN and Eager RSTkNN. We compared the two algorithms on the basis of query time and page access.



(a) Query Time

(b) PageAccess

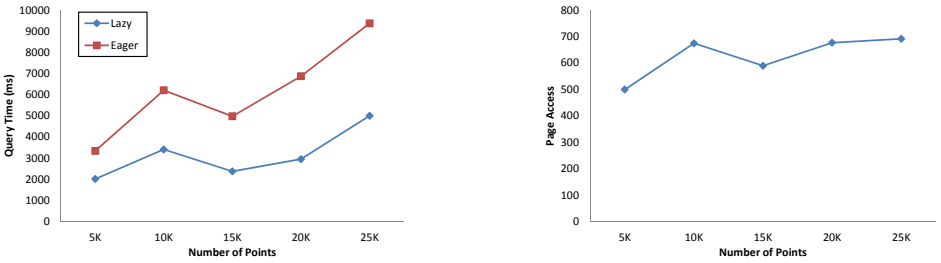
Figure 4.3: SimpleGeoPlaces Dataset

There are two factors responsible for deciding which approach performs better. The eager approach computes the exact score of an entry with the elements present in its parent contribution list during inherit. So, eager approach takes more time as compared to lazy approach as it simply copies the list of parent. Another factor is the similarity scores are tighter with the elements in the contribution list after inheriting, which leads to a difference in the size of upper contribution list, as upper contribution list contains elements whose upper bound score is at least equal to the k^{th} lower bound score. In the SimpleGeoPlaces dataset, eager approach performs better and in the GeographicNames dataset lazy performs better.

In GeographicNames dataset, the textual vectors are more sparse and even computing the exact scores doesn't tighten the lower bound values much. Therefore, lazy approach performs better than the eager approach. The results also show that the page access is same for both Lazy and Eager approach and query time increases with the number of points.

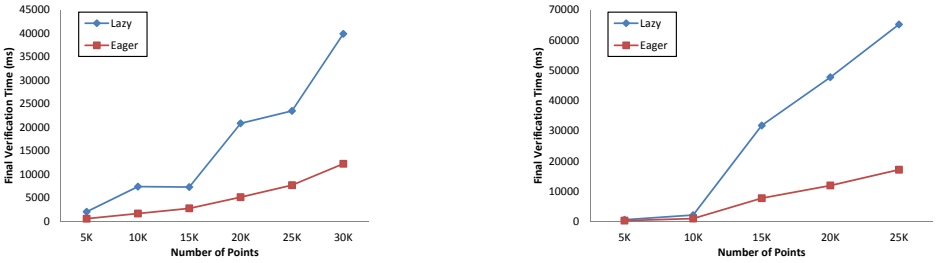
Performance of Eager vs. Lazy Final Verification: We implemented two versions of our algorithm Lazy RSTkNN and Eager Final Verification. We compared the two algorithms on the basis of query time. The results for both the datasets are shown below.

The results show that eager final verification is better compared to the lazy approach. In the lazy approach, for every candidate object, the algorithm tries to avoid visiting the children of a pruned entry, if a decision can be made by updation with the parent itself. However, doing a linear scan to find an internal node, replacing it with its children and again invoking IsHitorDrop



(a) Query Time (b) PageAccess

Figure 4.4: GeographicNames Dataset



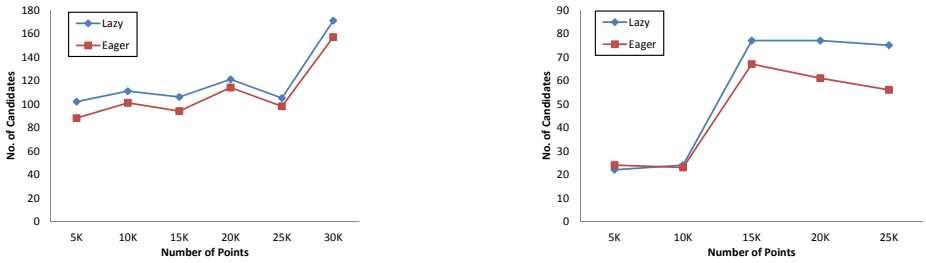
(a) SimpleGeoPlaces (b) GeographicNames

Figure 4.5: Final Verification Time

is the reason for more time required in lazy final verification.

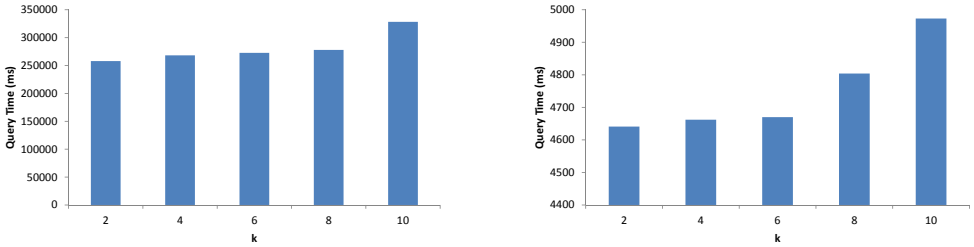
Number of Candidates Eager vs Lazy: The results for both the datasets are shown in the figure 4.6. The results show that eager approach results in less number of candidates compared to lazy approach. This is primarily because of the tight similarity score in eager approach. The results also show that the number of candidates is very less compared to the dataset size. The percentage of the number of candidates is less than 0.6 for both the datasets.

In the next set of experiments, we analyse how the performance of our algorithm depends on three parameters, namely, α, k and the number of query words.
Effect of k: We fix α=0.4, qw=4 and vary k from 2 to 10. The results show that the query time increases with the increase in k as the size of lower and upper bound contribution list depends on the value of k.
The reason for the increase in the size of contribution lists is with the increase in k, an entry/node has to see more number of entries/nodes to find its k^{th} nearest neighbour.



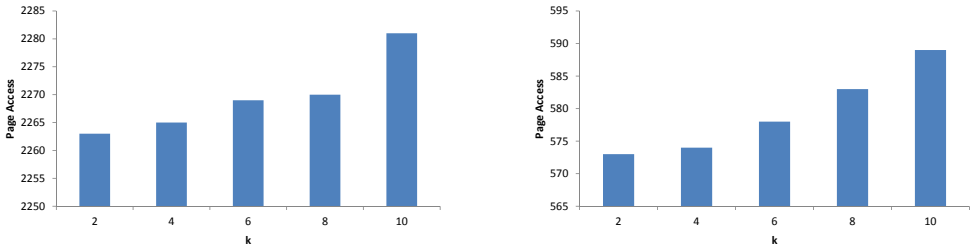
(a) SimpleGeoPlaces (b) GeographicNames

Figure 4.6: Number of Candidates



(a) SimpleGeoPlaces (b) GeographicNames

Figure 4.7: Effect of k on query Time

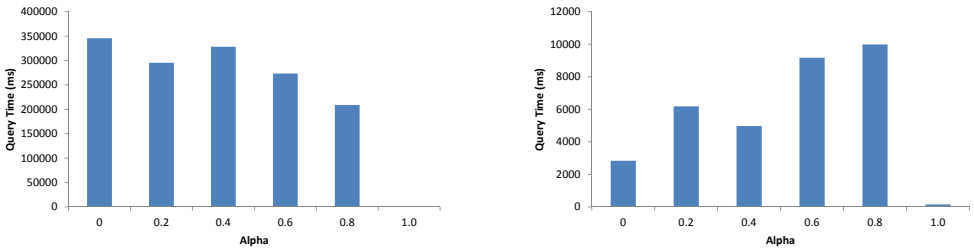


(a) SimpleGeoPlaces (b) GeographicNames

Figure 4.8: Effect of k on Page Access

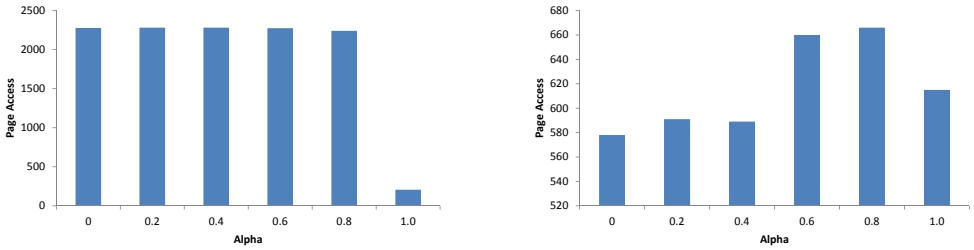
Effect of alpha: We fix k=10, qw=4 and vary α from 0 to 1. The results show that the

algorithm is insensitive to α. The runtime is obviously less in space compared to text, but there is no relationship between alpha and the performance of our algorithm.



(a) SimpleGeoPlaces (b) GeographicNames

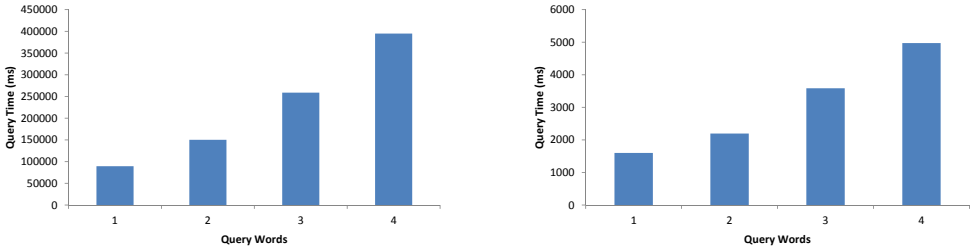
Figure 4.9: Effect of alpha on query Time



(a) SimpleGeoPlaces (b) GeographicNames

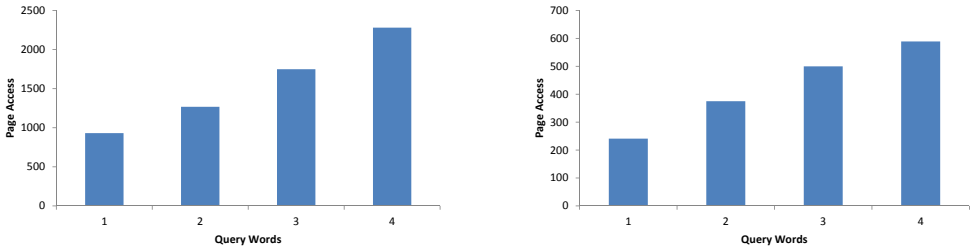
Figure 4.10: Effect of alpha on Page Access

Effect of query words (qw): We fix k=10, α=0.4 and vary the number of query words from 1 to 4. The results show that our algorithm runs slower with the increase in the number of query words. The reason is with the increase in number of query words, the textual similarity of the query increases with other entries in IUR tree which increases the query time as more entries/nodes in the IUR tree needs to be visited.



(a) SimpleGeoPlaces (b) GeographicNames

Figure 4.11: Effect of qw on query Time



(a) SimpleGeoPlaces (b) GeographicNames

Figure 4.12: Effect of qw on Page Access

Our experimental results show that our algorithm is sensitive to k and the number of query words and there is a trade-off between the performance of eager and lazy RSTkNN algorithm. We also show that lazy travel down technique in final verification is better compared to eager final verification.

Chapter 5

Conclusion and Future Work

We presented a solution for performing Public Reverse Nearest Neighbour Query over Private data in two dimensional spaces considering only spatial similarity. We proposed a generic framework for Reverse Nearest Neighbour Search independent of the index structure and the type of data. We also highlighted the importance of maintaining locality and completeness condition and presented two different approaches, Lazy and Eager. Our experimental study highlights the parameters which affect the performance of our algorithm and the factors responsible for trade-off in performance of Lazy and Eager approach.

In this thesis, we focus only on location privacy, but maintaining query privacy is also important. It will be interesting to develop efficient algorithms for performing the three Privacy Preserving Reverse Nearest Neighbour queries while maintaining both location and query privacy. We would further like to study the impact of textual clustering on the performance of our algorithm and further optimize our algorithm. We would like to extend our RSTkNN algorithm for performing bichromatic reverse nearest neighbour queries.

Bibliography

[1] Gruteser, Marco, and Dirk Grunwald. *Anonymous usage of location-based services through spatial and temporal cloaking* Proceedings of the 1st international conference on Mobile systems, applications and services. ACM, 2003.

[2] Liu, Fuyu, Kien A. Hua, and Ying Cai. *Query l-diversity in location-based services* Mobile Data Management: Systems, Services and Middleware, 2009. MDM'09. Tenth International Conference on. IEEE, 2009.

[3] Korn, Flip, and S. Muthukrishnan. *Influence sets based on reverse nearest neighbor queries*, ACM SIGMOD Record 29.2 (2000): 201-212.

[4] Foxphilly. *Former New Jersey Police Officer Convicted of Stalking Woman* http://www.myfoxphilly.com/story/22687207/former-new-jersey-police-officer-convicted-of-stalking

[5] Krumm, John. *A survey of computational location privacy* Personal and Ubiquitous Computing 13.6 (2009): 391-399.

[6] Mokbel, Mohamed F., Chi-Yin Chow, and Walid G. Aref. *The new Casper: query processing for location services without compromising privacy* Proceedings of the 32nd international conference on Very large data bases. VLDB Endowment, 2006.

[7] Salton. *Term-weighting approaches in automatic text retrieval* In Information Processing and Management, pages 513523, 1988.

[8] Lu, Jiaheng, Ying Lu, and Gao Cong. *Reverse spatial and textual k nearest neighbour search* Proceedings of the 2011 ACM SIGMOD International Conference on Management of data. ACM, 2011.

[9] P-N. Tan, M. Steinbach, and V. Kumar. *Introduction to Data Mining* Addison- Wesley, 2005.

[10] Yang, Congyun, and King-Ip Lin. *An index structure for efficient reverse nearest neighbor queries* Data Engineering, 2001. Proceedings. 17th International Conference on. IEEE, 2001.

[11] Achtert, Elke, et al. *Efficient reverse k-nearest neighbor search in arbitrary metric spaces* Proceedings of the 2006 ACM SIGMOD international conference on Management of data. ACM, 2006.

[12] Achtert, Elke, et al. *Reverse k-nearest neighbor search in dynamic and general metric databases.* Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology. ACM, 2009.

[13] Tao, Yufei, Dimitris Papadias, and Xiang Lian. *Reverse kNN search in arbitrary dimensionality* Proceedings of the Thirtieth international conference on Very large data bases,Volume 30. VLDB Endowment, 2004.

[14] Wu, Wei, et al. *Finch: Evaluating reverse k-nearest-neighbor queries on location data* Proceedings of the VLDB Endowment 1.1 (2008): 1056-1067.

[15] Cheema, Muhammad Aamir, et al. *Influence zone: Efficiently processing reverse k nearest neighbors queries* Data Engineering (ICDE), 2011 IEEE 27th International Conference on. IEEE, 2011.

[16] De Felipe, Ian, Vagelis Hristidis, and Naphtali Rishe. *Keyword search on spatial databases* ICDE 2008. IEEE 24th International Conference on. IEEE, 2008.

[17] Cong, Gao, Christian S. Jensen, and Dingming Wu. *Efficient retrieval of the top-k most relevant spatial web objects* Proceedings of the VLDB Endowment 2.1 (2009): 337-348.

[18] Cao, Xin, Gao Cong, and Christian S. Jensen. *Retrieving top-k prestige-based relevant spatial web objects* Proceedings of the VLDB Endowment 3.1-2 (2010): 373-384.

[19] Li, Guoliang, Jing Xu, and Jianhua Feng. *Keyword-based k-nearest neighbor search in spatial databases* Proceedings of the 21st ACM international conference on Information and knowledge management. ACM, 2012.

[20] Vlachou, Akrivi, et al. *Reverse top-k queries* Data Engineering (ICDE), 2010 IEEE 26th International Conference on. IEEE, 2010.

[21] Vlachou, Akrivi, et al. *Branch-and-bound algorithm for reverse top-k queries* Proceedings of the 2013 international conference on Management of data. ACM, 2013.

[22] Lian, Xiang, and Lei Chen. *Efficient processing of probabilistic reverse nearest neighbor queries over uncertain data* The VLDB JournalThe International Journal on Very Large Data Bases 18.3 (2009): 787-808.

[23] Cheema, Muhammad Aamir, et al. *Probabilistic reverse nearest neighbor queries on uncertain data.* Knowledge and Data Engineering, IEEE Transactions on 22.4 (2010): 550-564.

[24] Bernecker, Thomas, et al. *Efficient probabilistic reverse nearest neighbor query processing on uncertain data.* Proceedings of the VLDB Endowment 4.10 (2011): 669-680.

[25] Li, Jiajia, Botao Wang, and Guoren Wang. *Efficient Probabilistic Reverse k-Nearest Neighbors Query Processing on Uncertain Data.* Database Systems for Advanced Applications. Springer Berlin Heidelberg, 2013.

[26] LU, YING, et al. *Efficient Algorithms and Cost Models for Reverse Spatial-Keyword k-Nearest Neighbor Search*

[27] DataSet *SimpleGeoPlaces*, http://datahub.io/dataset/simplegeo_places_dump_20110628 /resource/1f76900b-5238-47b1-92ea-4e265bbc3956

[28] DataSet *Geographic Names*, http://geonames.usgs.gov/domestic/download_data.htm.