Database Selection for Processing k Nearest Neighbors Queries in Distributed Environments

Clement Yu[1], Prasoon Sharma[1], Weiyi Meng[2], Yan Qin[1]

[1] Department of EECS
University of Illinois at Chicago, Chicago, IL 60607
Email: yu@eecs.uic.edu

[2] Department of Computer Science
State University of New York at Binghamton, Binghamton, NY 13902

Abstract

We consider the processing of digital library queries, consisting of a text component and a structured component in distributed environments. The text component can be processed using techniques given in previous papers such as [8, 11, 7]. In this paper, we concentrate on the processing of the structured component of a distributed query. Histograms are constructed and algorithms are given to provide estimates of the desirabilities of the databases with respect to the given query. Databases are selected in descending order of desirability. An algorithm is also given to select tuples from the selected databases. Experimental results are given to show that the techniques provided here are effective and efficient.

Keywords: k nearest neighbors, database selection, distributed databases, query processing.

1 Introduction

As pointed out in [3, 4], it is of great interest to find the k nearest neighbors, i.e., the k tuples in a database table which best match a given user query. If the table contains records that describe library items such as books, magazines and papers, then the problem becomes finding the k best matching items of a given query. Current commercial relational databases systems do not support the processing of such queries. Techniques for processing such queries in a centralized environment have recently been proposed by [3, 4]. In this paper, we examine the processing of these queries in large-scale distributed relational databases or distributed digital libraries, where hundreds of or more databases exist. Specifically, given a query which requests the k nearest neighbors from many databases, we propose a method to determine the databases which are likely to contain the desired results. This is of special interest in the Internet environment where there are numerous sites providing data about the same topic.

A straightforward way to process a nearest neighborhood query is to send the query to all databases. At the site of each database, the k nearest neighbors are returned. The returned results from the different sites are then merged at a common site to produce the overall k nearest neighbors for the query. This strategy is not efficient if there are a large number of databases because most of the them probably won't contain any of the desired k tuples. For example, if $k = 20$ and the number of databases is 200, then at least 180 of the databases won't be useful for this query. This straightforward method incurs unnecessary cost to send the query to the useless sites and unnecessary cost to process the query in these sites. Furthermore, when these sites return their retrieval results to the common site, there is further waste of communication and local processing resources.

In this paper, we propose a method to identify the databases which are likely to be useful for processing any given query and to determine the tuples from each useful site which are necessary for answering the query. In this way, both the communication cost and the local processing costs are saved. One common characteristics of these k nearest neighbors queries is that it is not necessary to obtain all the k nearest neighbors; it is often sufficient to get most of the k neighbors. Experimental results are provided to demonstrate that most of the k nearest neighbors (85% to 100%) are obtained using our approach. An average accuracy rate of 94.7% is achieved when the 20 closest neighbors are desired. To the best of our knowledge, this is the first paper on the processing of nearest neighbors queries in distributed relational databases.

Example 1 Suppose each Computer Science Department maintains an online technical report database (see [6, 13] for more information about CS technical report libraries). Each database has a table that contains information about each report published by the department. Suppose each report has a title and a publication date, among other possible information fields. Consider the query Q "Find the 10 most similar reports published around 1998 on the topic 'digital library' ". This query has two components. The first component is about the topic "digital library" and is treated as a textual query instead of a character string condition (as a textual query, it can match "libraries of digital video" and even "library" to a less extent). This component can be matched against the title field with each title being treated as a document. Methods for selecting potentially useful databases for textual queries in distributed environments are given in various papers such as [7, 8, 11]. In [11], an intermediate result of processing such a text query is a ranking of the databases in descending order of similarity. Each database is associated with a similarity (which is an inverse of distance) with respect to the query. The second component of the query is "around 1998" and can be considered as a structured condition. Reports published in 1998 satisfy this portion of the query exactly, with distance = 0. Reports which are published other than 1998 are at some distance away from the query condition; the further the year is away from 1998, the bigger the distance is.

In this paper, we provide techniques for processing this type of query conditions (i.e., structured conditions). Again, databases are ranked in descending order of distance with a distance associated with each database. By combining the techniques used for the above two types of query conditions (i.e., the textual condition and the structured condition), it is possible to process query Q, by ranking databases in ascending order of distance (or descending order of similarity). ∎

Note that query Q in the above example can be considered as a typical digital library query, containing conditions against both textual and structured data. Thus, queries used in large-scale distributed digital libraries can be processed efficiently using our method.

The rest of the paper is organized as follows. In Section 2, several examples of the k nearest neighbors queries in relational database systems are provided. In Section 3, methods for determining which databases to search for a given query are provided. Experimental results are given to demonstrate the efficiency and the effectiveness of the methods in Section 4. Conclusion and related works are provided in Section 5.

2 Motivating Examples of k-Nearest Neighbors

In this section, we provide a few examples to illustrate the use of the k nearest neighbors queries. While such queries are widely used in text databases, they are rather unusual in relational databases but are gaining importance. Their interpretations are by no means standard and are application dependent. Due to their different interpretations and applicabilities, we classify these queries into three different types. We also provide "distance" functions which may be suitable for the three different situations.

(a) Standard k-nearest neighbors queries

Given a relation $R(A_1, ..., A_m)$, where the A's are the attributes of the relation and a query $Q(q_1, ..., q_m)$, where q_i is a condition on attribute A_i, $i = 1, ..., m$, a *distance function* d such as the *Euclidean distance* or the *Manhattan distance* can be defined such that a distance $d(Q, t)$ can be computed, where $t = (t_1, ..., t_m)$ is a tuple in R. The distance is a measure on how well the tuple t satisfies the query Q.

Example 2 Consider a relation about used cars. Some of the attributes in this relation are price, p, and number of miles driven, m_d. Suppose a user is interested in finding a used car satisfying his requirements on these two attributes. He/she can then submit the following SQL-like queries.

Select C.id, C.p, C.m_d
From Used-car C
Where C.p $\approx$ 2000 and C.m_d $\approx$ 100000

There may not be a tuple satisfying the given conditions or there can be too many tuples satisfying the conditions but they are not ordered in a way that the user can easily choose the suitable ones. In the former case, the user's desire to find a suitable car is not satisfied. In the latter case, the user is overwhelmed with too many tuples. The remedy is to have a distance function f such that a distance can be computed between the query, $Q(p \approx 2000, m_d \approx 100000)$

and each tuple t. Then the tuples are ordered in ascending order of distance. Finally, the k tuples having the smallest distances are presented to the user, where k is an integer specified by the user. This may be indicated as follows.

Select C.id, C.p, C.m_d (10)
From User-car C
Where C.p $\approx$ 2000 and C.m_d $\approx$ 100000
Order by Distance f

Here, the 10 nearest neighbors are to be given to the user, where the distance function is f. ■

Although the above example is reasonable, there are rooms for better interpretations. First, if a car has an additional 10,000 miles, it may still fit the user's need. But, if the car costs an additional \$10,000, it will definitely not be suitable to the user. This can be remedied by normalizing the attribute value of each tuple by the corresponding query attribute value for each attribute. For example, if a tuple has price and number of miles driven given by $(2.3k, 110,000)$, then the percentage differences in the two attributes are $(0.3k/2k, 10,000/100,000) = (15\%, 10\%)$. For the rest of this paper, we shall use the percentage difference instead of the absolute difference. Another aspect which may better conform to the user's intention is that each attribute may affect the user differently. For example, price may be twice as important to the user than the number of miles driven. Such a desire can be expressed by modifying the "Where condition" in the above query to be

C.p $\approx$ 2000 (I1)
and C.m_d $\approx$ 100000 (I2)

where I1 and I2 are two numbers indicating the relative importance of the two attributes to the user. If I1 = 2 and I2 = 1, then price is twice as important to the user than the number of miles driven.

If the Manhattan distance is used, the "distance" due to the ith attribute, d_i, is given by $|t_i - q_i|/q_i * I_i$ and the overall distance due to multiple attributes is $\sum |t_i - q_i|/q_i * I_i$. If the Euclidean distance function is used, then $d_i = ((t_i - q_i)/q_i)^2 * I_i$ and the overall distance is $\sqrt{\sum((t_i - q_i)/q_i)^2 * I_i}$.

(b) Generalized "distance" k-nearest neighbors queries

In the above case, a car costing \$1.5k is at the same distance as another car costing \$2.5k relative to the query condition of \$2k, though the former car with the same mileage as the latter is likely to be more desirable to the user. To achieve this effect, the "distance" function is adjusted to permit negative values. For example, the value of \$1.5k has a negative "distance" of $(1.5k - 2.0k)/2.0k = -25\%$; while the value of \$2k has a distance of 0% with respect to the query condition of \$2k. In deciding the nearest neighbors of a query, the tuples are sorted in ascending order of distance, with negative distances before positive distances and highly negative distances before slightly negative distances.

If the Manhattan distance function is used, $d_i = |t_i - q_i|/q_i * I_i$, if $t_i \geq q_i$; otherwise, $d_i = -|t_i - q_i|/q_i * I_i$. If the Euclidean distance is used, then $d_i = ((t_i - q_i)/q_i)^2 * I_i$, if $t_i \geq q_i$; otherwise, $d_i = -((t_i - q_i)/q_i)^2 * I_i$. Again, if the Euclidean distance is used and when all attributes are considered, the distances due to the individual attributes are summed and then the square root is taken. When a "distance" is negative, then the absolute value is taken before the square root is performed and then the negative sign is added back in.

(c) Two sided generalized "distance" k-nearest neighbors queries

Suppose we are interested in seeking an airplane ticket from Chicago to New York City. The attributes of interest could be the price and the time of departure. For the time of departure, we may specify a range of time which is acceptable, for example from 3pm to 6pm. Any time within the range will incur a distance of 0, but a time outside the range incurs a positive distance. As indicated before, we are interested in the percentage difference in each attribute. Thus, the denominator of the distance function for normalization due to time is set to be the mid-point, i.e., 4.5pm in the above example. This is to ensure that one hour deviation from either side outside the range incurs the same distance. For example, a 2pm departure time incurs a percentage distance of 1/4.5. Recall that for each attribute there is an importance factor. This can be set to eliminate the effect of where the mid-point lies. For example, the importance factor can be set to be $I_i * m_i$, where m_i is the mid-point of the interval (equal to 4.5 in the above example), and I_i is the relative importance of the ith attribute. With these parameter values, the mid-point m_i will be cancelled out from both the numerator and the denominator.

Let a range (l_i, u_i) be specified for the ith-attribute. Let m_i be the mid-point in the range, i.e.,

$(u_i - l_i)/2$. If the Manhattan distance is used,

$$d_i = \begin{cases} |t_i - u_i|/m_i * I_i, & if\ t_i > u_i \\ |l_i - t_i|/m_i * I_i, & if\ l_i > t_i \\ 0, & if\ t_i\ is\ in\ (l_i, u_i) \end{cases}$$

If the Euclidean distance is used,

$$d_i = \begin{cases} ((t_i - u_i)/m_i)^2 * I_i, & if\ t_i > u_i \\ ((l_i - t_i)/m_i)^2 * I_i, & if\ l_i > t_i \\ 0, & if\ t_i\ is\ in\ (l_i, u_i) \end{cases}$$

In the remaining part of this paper, we shall concentrate on these three types of nearest neighbors queries. For each type, we shall utilize the "Euclidean distance" function and the "Manhattan distance" function (in the case of the generalized distance neighbors queries, negative "distance" may arise.)

3 Deciding Databases to Search

Our aim is to retrieve the k nearest neighbors for a given query. This is equivalent to find the k tuples which have the smallest distance from the query. This needs to be done in a distributed environment with many databases. In this paper, we assume that there is one relation in each database and each database is located at a different site. For the remaining part of this paper, "relation" and "database" will be used interchangeably.

To facilitate the identification of the k nearest neighbors for a given query, we store for each relation a *representative* which consists of a histogram for each attribute. These representatives are stored at a central site where user queries are answered (if processing of user queries is desired at every site, then these representatives need to be replicated and stored at every site). When a user query is received, the attributes specified in the query are identified. Based on the histogram on each such attribute, an estimate is made on the desirability of each database with respect to the query. The databases are then ranked with respect to their desirability. Then, they will be searched in descending order of desirability. Tuples from the selected databases are retrieved in such a way that if the databases are ranked *optimally*, then all the k nearest neighbors will be retrieved.

In Section 3.1, the histograms are discussed. In Section 3.2, the criterion for ranking databases optimally with respect to a given query is provided. The criterion is simply that for each database, the distance of the nearest neighbor to the query in the database is obtained and databases are ranked in ascending order of the distances of the nearest neighbors in all databases. This guarantees optimal ranking of databases. In Section 3.3, a generating function is introduced to provide an estimate of the distance of the nearest neighbor for each database. In Section 3.4, an algorithm to determine which tuples from each selected database to be returned to the user is provided.

3.1 Histogram Construction

Two methods for constructing a histogram for each attribute are sketched below. They are the Simple Interval Construction method and the Greedy Merge method.

(a) Simple Interval Construction Method

For each attribute, the range of values is partitioned into subranges of equal width. For each subrange, a frequency count which gives the number of tuples which have values within the subrange is kept. For example, a histogram for the mileages of cars can be as shown in Table 1. If a subrange has too many tuples, then it can be divided into smaller subranges of equal width. For example, the subrange [40k, 50k) may be partitioned into smaller subranges [40k, 45k) and [45k, 50k) and within each smaller subrange, the number of tuples is kept.

Miles	#tuples
[0,10k)	200
[10,20k)	150
[20k,30k)	170
[30k,40k)	500
[40k,50k)	1000
......	......

Table 1: A Histogram for Car Mileages

From the histogram in Table 1, it can be seen that if there are 10,000 tuples in this relation, then the probability that a tuple with mileage in the range [0,10k) is 200/10000 = 0.02.

(b) Greedy Merge Method

This method was proposed in [10]. Initially, the range of values is partitioned into a large number of subranges of equal width. As in the simple interval construction method, the number of tuples which have values within each subrange is counted. Associated with each subrange, an error of estimation can be computed. For a subrange $[b, e)$, the error of estimation is given by $E = \sum_{i=b}^{e-1}(c_i - ac)^2$, where c_i is the count (the number of tuples) at attribute value i and ac is the average count per attribute value in the subrange. In the Greedy Merge method, the counts within a subrange are approximated by a linear function of the form $s(i) = a_0 + a_1 * i$. It can be shown that the error using the linear function, $E' = (1 - r^2)E$, where r is in $[-1, 1]$ and is the linear correlation between the counts and the attribute values within the subrange. This implies that using a linear approximation function yields a smaller estimation error than using the mean.

The Greedy Merge method merges 2 adjacent subranges with the smallest estimation error. This is repeated until a certain number of subranges is reached. At that point, the counts for the different subranges are kept. If proper statistics are kept for each subrange, then determining which adjacent subranges to be merged can be carried out efficiently. The details can be found in [10].

3.2 Criterion for Selecting Databases Optimally

Definition 1 *Suppose a user is interested in retrieving the k nearest neighbors to a submitted query Q. Databases $\{D_i, 1 \leq i \leq n\}$ are optimally ranked in the order $D_1, D_2 ..., D_n$, if for every k, there exists a t such that $D_1, D_2, ..., D_t$ collectively contain all the k nearest neighbors of Q and each $D_i, 1 \leq i \leq t$, contains at least one of the k nearest neighbors.*

The criterion to rank databases is "for each database, obtain the distance of the closest neighbor in the database to the query; then, databases are ranked in ascending order of the distance of the closest neighbor."

Example 3 Suppose there are 5 databases D_1, D_2, D_3, D_4 and D_5. Suppose that the distances of the nearest neighbors in these databases to query Q are 0.8, 0.6, 0.3, 0.7 and 0.5, respectively. Then, for query Q, the databases should be ranked in the order D_3, D_5, D_2, D_4, D_1. ■

This guarantees that databases are ranked optimally, as shown in the following proposition.

Proposition 1 *For a given query Q, if databases are ranked in ascending order of the distance of the nearest neighbor to Q, then they are ranked optimally with respect to Q.*

Proof: In [14], a proposition was proved for ranking text databases (i.e., search engines) optimally in the context of retrieving the k most similar documents to a given text query across multiple text databases. The proposition in [14] states that if databases are ranked in descending order of the similarity of the most similar document in each database, then the databases are optimally ranked. The proof of the new proposition can essentially follow that given for the proposition in [14]. The only major difference is that in [14], similarities are used while in this paper, distances are used. Since similarity and distance are inverses, the result holds. ■

3.3 Generating Function to Provide the Estimate

For each attribute, say attribute A_i, specified in the user query, a polynomial is constructed for the attribute of each relation in the distributed database. This polynomial essentially gives the probabilities that tuples in this database are at various "distances" from the query condition specified on attribute A_i. Specifically, let $Q = (q_1, ..., q_i, ..., q_m)$ be the query where q_i is the value of attribute A_i. Let I_i be the degree of importance of the attribute in the relation. Then a polynomial g_i is constructed. It is given by $p_1X^{e_1} + p_2X^{e_2} + ... + p_sX^{e_s}$, where s is the number of subranges of the ith attribute, X is a dummy variable, p_j is the probability that a tuple in the database has a value within the subrange whose mid-point is at weighted distance e_j (given by $d_j * I_i$) away from query condition q_i, where d_j is the "distance" of the mid-point of the subrange from the query condition q_j in the ith attribute. Let the subrange be (l_{ji}, u_{ji}). Then, in the computation of d_j, it is assumed that each tuple in that subrange takes on the mid-point value, i.e., $(u_{ji} + l_{ji})/2$. For example, in the car example, for the first subrange, the mid-point is 5k miles. If the query condition is 3k, then the "generalized distance" due to this attribute is $(5k - 3k)/3k * I_i$, if the Manhattan distance is used. The e's are in ascending order.

It should be noted that the distance d_j from the query condition q_j, which is associated with the subrange, is by no means unique nor most reasonable. Instead of using the mid-point of each subrange, the *mean* can be utilized. In the Greedy Merge method, a linear function is used to estimate the distribution of the attribute values within each subrange. From the linear function, it is possible to estimate the *mean* of the attribute values within the subrange. However, this requires storing the coefficients of the linear function. For simplicity, we use the mid-point of each subrange.

To summarize, for each attribute, say the ith attribute, g_i gives the distribution of the tuples of the relation which are in increasing weighted distances from the query specification on the ith attribute (and due to the ith attribute only). Each tuple is assumed to take on the mid-point value of the subrange where it resides. In the polynomial, for each term involving X, the coefficient of X is the probability that a tuple in the relation is at a weighted distance given by the exponent of X away from the query condition q_i. Suppose that the query has specifications on a set of attributes S. Then, these polynomials are multiplied together to yield $\prod_i g_i$, where the product is over all i in S. This product polynomial, after arranging the terms in ascending order of the exponent of X, gives the distribution of the tuples of the relation in ascending order of weighted distance from the query, taking into consideration all attributes specified by the query. Again, in the case of Euclidean distance, the actual distances should be the square root of the exponents of X. When a "distance" is negative, then the absolute value is taken before the square root is performed and then the negative sign is added back in.

Proposition 2 *If the values of the tuples of a relation R are distributed independently in the attributes specified by the query and each value takes on the mid-point value of the subrange where it resides, then $\prod_i g_i$ gives the probability distribution of the tuples of the relation R in increasing distances from the query, after the product is arranged in ascending order of the exponent of X.*

Proof: Recall that $p_j * X^{e_j}$ in g_i gives the probability of a tuple which is at "weighted distance" e_j from the query condition q_i, due to the ith attribute only. For another attribute, say the kth attribute, $p_t * X^{e_t}$ gives the probability of a tuple which is at "weighted distance" e_t from q_k, due to the kth attribute only. By the independence assumption of the attributes, the probability that a tuple is at "distance" $e_j + e_t$ from the query based on the ith and kth attributes only is $p_j * p_t$. The corresponding term in the product of g_i and g_k is $p_j * p_t * X^{e_j+e_t}$. Thus, after all polynomials associated with the query are multiplied, a term of the form $a*X^e$ gives the probability, a, that a tuple in the relation is at distance e away from the query. (In the case of Euclidean distance, the actual distance is the square root of e.) All terms with the same exponent are added together. After the terms are arranged in ascending order of the exponent, the distribution of the tuples is given in ascending order of distance from the query, since the exponents are the distances. ■

Observations:

1. Usually attribute values are not distributed independently. In our experimental results in Section 4, the attributes are dependent but very reasonable results are obtained.

2. The assumption that each attribute value takes on the mid-point of a subrange is not realistic. However, it does not seem to affect our experimental results significantly.

3. The "distance" function that we use, say the "Euclidean distance" (actually the square of Euclidean distance) or the "Manhattan distance" functions, assumes that the distance is the sum of the weighted "distances" due to the individual attributes specified in the query. This allows us to add the exponents of X to arrive at the total distance.

4. The complexity of the algorithm (to multiply the polynomials) is exponential to the number of attributes which are specified in the query and are needed in the distance computation. Usually, the number of attributes involved in a query is very small. For example, in searching for a car having restrictions on price and mileage, only two attributes are involved. Other specification such as the make or the model of the car are exact conditions and are not involved in distance computations.

Example 4 Consider the query Q(miles $\leq$ 10k, price $\leq$ 15k) using the generalized Manhattan distance. Suppose the histogram for mileage is that given in Table 1. Then, the mid-points of the subranges are $5k, 15k, 25k$, etc. The generalized distances of 10k from the mid-points of the subranges are $-5k, 5k, 15k$, etc. After the normalization by 10k, the percentage differences are $-0.5, 0.5, 1.5$, etc. Assuming that there are 10,000 tuples, the probabilities of the subranges are $0.02, 0.015, 0.017$, etc. As a result, the polynomial associated with the mileage attribute is $g_{mileage} = 0.02X^{-0.5} + 0.015X^{0.5} + 0.017X^{1.5} + ...$

Suppose the histogram for price is given by Table 2.

The generalized distances of 15k from the mid-points of the subranges are $-14.5k, -13.5k, -12.5k$, etc. After normalization by 15k, the percentage differences are $-0.967, -0.9, -0.833$, etc. The probabilities of the subranges are 0.003, 0.008, 0.005, etc. Thus,

Price	#tuples
[0,1k)	30
[1k,2k)	80
[2k,3k)	50
[3k,4k)	35
[4k,5k)	100
......	......

Table 2: A Histogram for Price

the polynomial associated with the price attribute is $g_{price} = 0.003X^{-0.967}+0.008X^{-0.9}+0.005X^{-0.833}+...$

The polynomial representing both the mileage and the price attributes is the product of the above two polynomials and is given by

$$0.00006X^{-1.467} + 0.00016X^{-1.4} + ...$$

■

The result of the product polynomial will now be used to estimate the distance of the nearest neighbor in a given relation. Suppose the product polynomial is

$$c_1 * X^{d_1} + c_2 * X^{d_2} + ... + c_k * X^{d_k} \quad (1)$$

where the exponents of X are in ascending order. Let N be the number of tuples of the relation. Then, $N*c_1$ is the expected number of tuples which are at distance d_1 away from the query. If $N * c_1 \geq 1$, then the distance of the nearest neighbor in this relation is estimated to be d_1; else, we compare $N*(c_1+c_2)$ with 1. If the former is as least as large as 1, then the distance is estimated to be d_2. In general, if t is the smallest integer such that $N*(c_1+c_2+...+c_t) \geq 1$, then the distance is estimated to be d_t.

Example 5 Continue on the Example 4. The expected number of tuples having generalized distance -1.467 from the user query is $10000 * 0.00006 = 0.6$. Since it is less than 1, we consider the next term. The expected number of tuples having generalized distance -1.4 from the user query is 1.6. Since $0.6 + 1.6 \geq 1$, the generalized distance of the nearest neighbor is estimated to be -1.4. ■

3.4 Algorithm to Select Tuples from Ranked Databases

Let databases be ranked in the order $[D_1, D_2, ..., D_m]$. Let k be the number of tuples the user wants to see. In order to improve the accuracy, the algorithm retrieves an additional s tuples (i.e. retrieve $k+s$ tuples), but return the k closest neighbors to the user. The databases are accessed in the order in which the databases are ranked, one at a time. (In practice, the first few highest ranked databases may be accessed in parallel, since it is expected that they will contain the desired tuples.) From the first and the second accessed databases, we obtain the nearest neighbor from each of these databases. Let the distances of these tuples be d_1 and d_2, respectively. Let $d = \max\{d_1, d_2\}$. Tuples from these two databases with distances $\leq d$ are gathered. If the number of such tuples is greater than or equal to $(k+s)$, then the k nearest neighbors are returned to the user; otherwise, the next ranked database is accessed. In general, suppose the first t databases have been accessed and d is the maximum value of the distances of the nearest neighbors, one from each of these t databases. If $(k+s)$ tuples have been retrieved, then the k retrieved nearest neighbors are returned to the user; else the next database is accessed. Let d_{t+1} be the distance of the nearest neighbor in database D_{t+1}. If $d_{t+1} > d$, then retrieve the nearest neighbor from database D_{t+1} and tuples which have not been retrieved but with distance $\leq d_{t+1}$ from the first t databases else retrieve from database D_{t+1} tuples with distance $\leq d$. In either case, d is updated to be $\max\{d, d_{t+1}\}$. If the total number of retrieved documents retrieved is $(k+s)$ or more, then return the k nearest neighbors to the user and terminate.

This algorithm guarantees that if the databases are ranked optimally, then all the desired k nearest neighbors of the query will be retrieved. For a proof, see [14], where similarities which are the inverses of distances are used.

4 Experiments

In Section 4.1, we describe the data and query collection used in the experiments. In Section 4.2, two measures of retrieval, one reflecting the quality (i.e., accuracy) and the other reflecting the efficiency are provided. Experimental results are provided in Section 4.3.

4.1 Data Collection and Query Collection

Used car data were collected from Excite's Classification 2000 website with the following conditions: Make = "any", Model = "all models", Year = "1900 to 2000", Price = "$500 to $27,000". There are more than 50,000 tuples. The tuples are arbitrarily assigned to 29 databases, without any duplication of tuples. The queries are two attribute queries involving price and mileage. The values associated with the two attributes are chosen to reflect reality. Specifically, if the mileage is high, the price is low; if the mileage is low, the price is high. The relative degrees of importance associated with the two attributes are arbitrarily chosen. 35 queries are generated. For each query, the three interpretations as discussed in Section 2 are applied and they are: the standard k nearest neighbors queries; the generalized k nearest neighbors queries and the two sided-generalized k nearest neighbors queries. For each interpretation, the two "distance functions", namely, the Euclidean distance and the Manhattan distance functions are used.

4.2 Criterion of Performance

Performances are measured in two ways: the quality of the retrieved tuples and the efficiency of retrieval. The former is measured by the number of the tuples which are retrieved and are among the actual k nearest neighbors divided by k. If the quantity is 100%, then all of the k nearest neighbors are retrieved. The higher the percentage, the higher the quality of retrieval is achieved. The second quantity measures the efficiency and is the ratio of the number of databases searched to the actual number of databases containing the k nearest neighbors. If the percentage is 100%, then the number of databases accessed is the same as the number of databases containing the k nearest neighbors although not necessarily the same set of databases is accessed. A value exceeding 100% indicates inefficiency. The lower the percentage, the higher the efficiency is achieved. However, any value in this measure below 100% indicates the quality of retrieval is also below 100%. Ideal retrieval would have both measures equal to 100%.

4.3 Experimental Results

The two methods of constructing histograms, i.e., the Simple Interval Construction method and the Greedy Merge method are employed. For each method, the parameter $s = 20\%k$, i.e., whenever the user wants to obtain the k closest neighbors, an additional 20% of the tuples are retrieved by our algorithm but only k tuples are returned to the user.

Tables 3-14 give the experimental results of the three types of query interpretations, each with the Euclidean distance and the Manhattan distance functions. A summary of the results is given as follows.

# tuples desired	Accuracy	Efficiency
10	0.86	0.94
20	0.90	0.88
30	0.90	0.90

Table 3: Results based on Standard Manhattan distance, Simple Interval

# tuples desired	Accuracy	Efficiency
10	0.86	0.93
20	0.92	0.92
30	0.93	0.94

Table 4: Results based on Standard Manhattan distance, Greedy Merge

# tuples desired	Accuracy	Efficiency
10	0.85	0.93
20	0.88	0.90
30	0.90	0.90

Table 5: Results based on Standard Euclidean distance, Simple Interval

# tuples desired	Accuracy	Efficiency
10	0.86	0.94
20	0.90	0.92
30	0.91	0.92

Table 6: Results based on Standard Euclidean distance, Greedy Merge

# tuples desired	Accuracy	Efficiency
10	0.93	1.12
20	0.93	1.24
30	0.96	1.06

Table 7: Results based on Generalized Manhattan distance, Simple Interval

# tuples desired	Accuracy	Efficiency
10	1.00	1.00
20	1.00	1.14
30	0.99	0.98

Table 8: Results based on Generalized Manhattan distance, Greedy Merge

# tuples desired	Accuracy	Efficiency
10	0.95	1.05
20	0.95	1.23
30	0.97	1.14

Table 9: Results based on Generalized Euclidean distance, Simple Interval

# tuples desired	Accuracy	Efficiency
10	1.00	1.00
20	1.00	1.21
30	0.99	1.05

Table 10: Results based on Generalized Euclidean distance, Greedy Merge

# tuples desired	Accuracy	Efficiency
10	0.97	1.11
20	0.98	1.12
30	0.97	1.09

Table 11: Results based on Two-sided Manhattan distance, Simple Interval

# tuples desired	Accuracy	Efficiency
10	0.94	1.06
20	0.93	1.04
30	0.92	1.07

Table 12: Results based on Two-sided Manhattan distance, Greedy Merge

# tuples desired	Accuracy	Efficiency
10	0.96	1.08
20	0.97	1.12
30	0.97	1.10

Table 13: Results based on Two-sided Euclidean distance, Simple Interval

# tuples desired	Accuracy	Efficiency
10	0.94	1.03
20	0.93	1.08
30	0.92	1.08

Table 14: Results based on Two-sided Euclidean distance, Greedy Merge

1. For the Simple Interval Construction Method, the accuracy ranges from 85% to 98%. For the Greedy Merge Method, it ranges from 86% to 100%. The average accuracy rate for retrieving the top 20 tuples by the Simple Interval method, averaged over the 6 interpretations of the "distance" functions, is 93.5%. The corresponding accuracy rate for the Greedy Merge method is 94.7%. Thus, the Greedy Merge method yields slightly higher accuracy than the Simple Interval Construction Method. However, there are slight deteriorations for the two-sided Manhattan distance and the two-sided Euclidean distance. Clearly, if both the histograms constructed by the Simple Interval Construction Method and the histograms constructed by the Greedy Merge Method are kept, then we can apply the former histogram for the two-sided Manhattan distance queries and for the two-sided Euclidean distance queries and the latter histogram for the other 4 types of queries to yield the best results.

2. For the generalized and the two-sided queries, the accuracy rates are over 90% and there is not much room for improvement; for the standard queries with the Greedy Merge Method, the accuracy rates range from 86% to slightly above 90%. Thus, it may be desirable to have a 5% improvement.

3. For efficiency, the average worst case is 1.24, meaning that in the worst case an additional 24% of databases need to be accessed. For most situations, the efficiency rates are between 90% and

100%. Thus, there is not much room for improvement.

5 Conclusion and Related Works

The work reported here are extensions from the following works:

1. It extends the processing of the top-k queries from centralized relational databases [3, 4] to distributed relational databases. We also utilize "distance" functions which are suitable for different applications.

2. It modifies the technique of processing text queries in distributed document databases to be applicable to distributed relational databases. In document processing environment, the number of keywords or terms is very large and usually exact matching of terms is required. In relational databases, the number of attributes in a relation is usually rather small. Two distinct values of the same attribute are separated by a "distance"; the further the separation of the two values the larger the distance. Due to these differences, the "generating function" technique in [12] are modified to be applicable in this environment.

The histograms that we utilize to select databases (sites) to search for a given query are rather primitive. But it has the advantage of being simplistic and space efficient. It may be possible to have slightly higher accuracies by utilizing the linear estimation function within each subrange to estimate the mean of attribute values within the subrange [10] and then use the mean to estimate the distance of a tuple in the subrange from the query condition.

The experimental results provided here show that the methods we employ in retrieving the k nearest neighbors for a given query in a distributed database environment are effective and are efficient. We also sketch how the technique given here and our earlier technique [11] can be combined to process digital library queries involving both text and structured data.

Acknowledgement

This work is supported in part by the following NSF grants: IIS-9902792, IIS-9902872.

References

[1] M. Carey and D. Kossmann. *On Saying "Enough Already!" in SQL*, ACM SIGMOD, 1997.

[2] M. Carey and D. Kossmann. *Reducing the Braking Distance of an SQL Query Engine*, VLDB 1998.

[3] S. Chaudhuri and L. Gravano. *Evaluating Top-k Selection queries*, VLDB 1999.

[4] D. Donjerkovic and R. Ramakrishnan. *Probabilistic Optimization of Top N Queries*, VLDB 1999.

[5] R. Fagin. *Combining fuzzy Information from Multiple Systems*, ACM Symposium on Principles of Database Systems, 1996.

[6] J. French, E. Fox, K. Maly, and A. Selman. *Wide Area Technical Report Service: Technical Report Online.* ACM CACM, 38:4, April 1995, 45-46.

[7] S. Gauch, G. Wang, and M. Gomez. *Profusion: Intelligent Fusion from Multiple, Distributed Search Engines*, Journal of Universal Computer Science, 1996.

[8] L. Gravano and H. Garcia-Molina. *Generalizing GlOSS to Vector-Space databases and Broker Hierarchies*, VLDB, 1995.

[9] Y. Ioannidis and V. Poosala. *Histogram-based Approximation of Set-valued Query Answers*, VLDB 99.

[10] A. Konig and G. Weikum. *Combining Histograms and Parametric Curve Fitting for Feedback-Driven Query Result-Size Estimation*, VLDB, 1999.

[11] K. Liu, C. Yu, W. Meng, W. Wu and N. Rishe, *A Statistical Method for Estimating the Usefulness of Text databases* , IEEE Transactions on Knowledge and Data Engineering, (to appear).

[12] W. Meng, K. Liu, C. Yu, X. Wang, Y. Chang, N. Rishe. *Determine Text Databases to Search in the Internet.* VLDB 98.

[13] Networked Computer Science Technical Reference Library (http://cs-tr.cs.cornell.edu/).

[14] C. Yu, W. Meng, K. Liu, W. Wu, and N. Rishe. *Efficient and Effective Metasearch for a Large Number of Text Databases.* Eighth ACM International Conference on Information and Knowledge Management (CIKM'99), Kansas City, November 1999.