Distributed Top-N Query Processing with Possibly Uncooperative Local Systems

Clement Yu, George Philip
Dept. of Computer Science
U. of Illinois at Chicago
Chicago, IL 60607
{yu,gphilip}@cs.uic.edu

Weiyi Meng
Dept. of Computer Science
SUNY at Binghamton
Binghamton, NY 13902
meng@cs.binghamton.edu

Abstract

We consider the problem of processing top-N queries in a distributed environment with possibly uncooperative local database systems. For a given top-N query, the problem is to find the N tuples that satisfy the query the best but not necessarily completely in an efficient manner. Top-N queries are gaining popularity in relational databases and are expected to be very useful for e-commerce applications. Many companies provide the same type of goods and services to the public on the Web, and relational databases may be employed to manage the data. It is not feasible for a user to query a large number of databases. It is therefore desirable to provide a facility where a user query is accepted at some site, suitable tuples from appropriate sites are retrieved and the results are merged and then presented to the user. In this paper, we present a method for constructing the desired facility. Our method consists of two steps. The first step determines which databases are likely to contain the desired tuples for a given query so that the databases can be ranked based on their desirability with respect to the query. Four different techniques are introduced for this step with one requiring no cooperation from local systems. The second step determines how the ranked databases should be searched and what tuples from the searched databases should be returned. A new algorithm is proposed for this purpose. Experimental results are presented to compare different methods and very promising results are obtained using the method that requires no cooperation from local databases.

keywords: Top-N queries, distributed databases, query processing.

Proceedings of the 29th VLDB Conference, Berlin, Germany, 2003

1 Introduction

As pointed out in [1, 3, 4, 6, 7], it is of great interest to find the N tuples in a database table which best match a given user query, for some integer N. If the table contains tuples that describe cars, then the problem becomes finding the N best matching cars of a given car description. Current commercial relational database systems do not support the processing of such type of queries. Techniques for processing such queries in a centralized environment have recently been proposed by [1, 3, 4, 6]. In this paper, we examine the processing of this type of queries in large-scale distributed relational databases. Specifically, given a query that requests the top N matched tuples (or top N tuples for short) across many databases, we study different methods to determine the databases that are likely to contain the desired results. This is of special interest in the Internet environment where numerous sites may provide data about the same type of products/services. In such an environment, most users will be satisfied with getting most of the top N results.

A straightforward way to process a top-N query in a distributed environment is to send the query to all databases. The site of each database returns the top N local tuples, which are then merged with results from other sites at a common site to produce the overall top N tuples for the query. This strategy is, however, not efficient if the number of databases is large, because most of them probably won't contain any of the desired tuples. For example, if $N = 20$ 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 at these sites. Furthermore, when these sites return their retrieval results to the common site, there will be further waste of communication and local processing resources.

In this paper, we present a two-step process to find the top N matched tuples of a given relational query in a distributed database environment. In the first step, an attempt is made to rank databases optimally with respect to the given query based on certain desirability criterion. Four different histogram based methods are utilized to estimate the desirability of each database with respect to the query. In the second step, a merge algorithm is used to determine which databases should be accessed and what tuples from accessed databases should be retrieved. Retrieved tuples are merged to form the output to the user. A new merge algorithm is proposed in this paper. Experimental results are carried out to compare the performance, in terms of both accuracy (effectiveness) and efficiency, of different approaches. The contributions of this paper are:

1. In the Internet/e-commerce environment, it is very likely that many databases (owned by different companies) are not willing to supply sufficient data for the construction of the histograms. We propose a method to construct histograms from uncooperative databases and we demonstrate that its ability in retrieving the top N tuples is significantly higher than other histogram construction techniques. To the best of our knowledge, this is the first paper on the top-N query problem in distributed uncooperative relational databases.

2. We provide a new merge algorithm and compare it both analytically and experimentally with an existing merge algorithm. It is found that the new algorithm has higher retrieval effectiveness but lower efficiency.

3. Various histogram construction and estimation techniques are compared. Experimental results indicate that our proposed histogram construction algorithm which does not require cooperation from different databases are much more accurate than three other histogram generation algorithms. Furthermore, the new algorithm gives consistently high accuracies under different query types (different dimensions, different distance functions, or a mixture of various query types).

4. We identify the type of queries where the original merge algorithm is sufficiently accurate and the type of queries where the new merge algorithm is needed for high accuracy. As a consequence, we utilize both merge algorithms to achieve both high accuracy and high efficiency.

A comparison with related literature is as follows.

1. The work reported in this paper can be considered as a generalization of the top-N query problem from centralized environments to distributed environments. The problem here is to identify the independently operated databases that are likely to contain the top N tuples for a given query; such a problem does not exist in a centralized environment.

2. Some recent techniques to construct histograms ([5, 9, 11, 13, 18, 19]) are employed here, although there is a significant difference. Histograms were traditionally used to estimate the number of tuples satisfying a certain query condition. In this paper, we modify existing techniques and propose a new technique so that they can be used to estimate the distance of the best matched tuple in a database to a given query.

3. Database (source) selection techniques discussed in [16, 17] do not consider top-N queries.

4. Preference queries have received some attention recently [10, 12]. While preference queries support richer mechanisms for a user to express preference of results than top-N queries, the evaluation techniques for these two types of queries are very different. Top-N query processing aims to evaluate only tuples that have the potential to be ranked among the top N results and uses histogram-based techniques to narrow the search space. Preference query evaluation typically ranks all tuples and uses views or materialized views to support preference capabilities [10, 12].

5. The most closely related work to this paper is [23] but there are several significant differences between it and this paper. First, a new merge algorithm is proposed, which is is compared with that in [23] in this paper. Second, [23] used just one type of histograms to estimate the distance of the best matched tuple in a database and no comparison with other methods was made. In this paper, three different types of histograms are utilized and compared with the one used in [23]. Third, the estimation algorithm in [23] is of exponential complexity, while the estimation algorithm in this paper takes low polynomial time complexity. Fourth, the new merge algorithm and the original merge algorithm are utilized jointly to achieve high accuracy and efficiency.

2 Examples of Top-N Queries

In this section, we provide a few examples to illustrate the use of the top-N queries. While such queries are widely used in text and multimedia 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 top-N 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 of how well the tuple t satisfies the query Q.

Example 1 Consider a relation about used cars. Some of the attributes in this relation are the price, p, and the mileage, m. Suppose a user is interested in finding a used car satisfying his/her requirements on these two attributes. Suppose the user submits the following SQL query.

Select C.id, C.p, C.m From Used-car C
Where C.p = 2000 and C.m = 100000

There may not be a tuple satisfying the given conditions. The remedy is to have a distance function f such that a distance can be computed between the query, $Q(p = 2000, m = 100000)$ and each tuple t. Then the tuples are ordered in ascending order of distances. Finally, the N tuples having the smallest distances are presented to the user, where N is an integer specified by the user. This may be indicated by the SQL-like query below:

Select C.id, C.p, C.m (10) From User-car C
Where C.p = 2000 and C.m = 100000
Order by Distance f

Here, the 10 best matched tuples 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 the price and the mileage given by $(2300, 110, 000)$, then the percentage differences in the two attributes are $(300/2000, 10, 000/100, 000) = (15\%, 10\%)$. For the rest of this paper, we shall use the percentage difference instead of the absolute difference. Often, different attributes may have different significance to the user. For example, price may be twice as important to the user as the mileage. Different significance can be expressed by modifying the "Where condition" in the above query to be

C.p = 2000 (I1) and C.m = 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 as the mileage.

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

(b) Generalized "distance" top-N queries
In the above case, a car costing \$1,500 is at the same distance as another car costing \$2,500 relative to the query condition of \$2,000, though the former car with the same mileage as the latter is likely to be more desirable to the user. To achieve the desired effect, the "distance" function is adjusted to permit negative values. For example, the value of \$1,500 has a negative "distance" of $(1500 - 2000)/2000 = -25\%$; while the value of \$2,000 has a distance of 0% with respect to the query condition of \$2,000. In deciding the top N tuples of a query, the tuples are sorted in ascending order of distance, with negative distances ahead of positive ones.

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 summed "distance" is negative, then the absolute value is taken before the square root is performed and the negative sign is then added back in.

(c) Two sided generalized "distance" top-N 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 5pm. 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., 4pm 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. 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 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,

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

For the rest of this paper, we shall concentrate on these three types of top-N queries. For each type, we shall utilize the "Euclidean distance" function and the "Manhattan distance" function (in the case of the generalized distance top-N queries, negative "distance" may arise.)

The use of the top-N queries arises naturally in e-commerce applications where the specification of desired products and services is more complex. For example, when buying long term care insurance for a couple, numerous criterions may be significant. These criterions may include the number of years requiring long term care, the way the number of years are split between the couple (say the older person gets more years than the younger person), the amount of expenses allowed per day, the amount of deductible, whether the amount of allowable expenses is adjusted based on inflation, the extent the amount is applied to home care versus nursing home, and the amount the couple can afford for premium per year. It is very likely that not all of these criterions can be satisfied simultaneously. As a consequence, the best matches to the specification are needed.

3 Deciding Databases to Search

Our aim is to retrieve the top N tuples for a given query. This is equivalent to finding the N tuples with the smallest distances from the query. This needs to be done in a distributed environment across many databases. One site may be designated as a global site which assembles the tuples and returns them to the user. This site has a global distance function which is the one desired by the user. In this paper, we assume that there is one relation in each database. Each database is operated autonomously at a different site. For the remaining part of this paper, "relation" and "database" will be used interchangeably.

To facilitate the identification of the top N tuples for a given query, we store for each relation a *representative* which consists of a "histogram", taking into consideration some dependencies between the attributes. The representatives for all sites are stored at the global site where user queries are processed (if the 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 histograms on those attributes, an estimate of the desirability of each database with respect to the query, based on the global distance function, is made. The databases are then ranked with respect to the estimates of their desirabilities. Then, these databases are searched in descending order of the estimated desirability. Tuples from the selected databases are retrieved in such a way that if the databases are ranked *optimally*, then all the top N tuples will be retrieved.

In Section 3.1, we describe the constructions of four types of histograms that will be used to estimate the desirability of each database with respect to any given query. In Section 3.2, the criterion for ranking databases optimally with respect to a given query is provided. The criterion is simply that databases are ranked in ascending order of the distance of the best matched tuple in each database. This guarantees optimal ranking of databases. In Section 3.3, algorithms are given to estimate the distance of the best matched tuple in each database to the query based on different types of histograms. In Section 3.4, a new merge algorithm is provided to determine the tuples to be retrieved from the ranked databases. This algorithm is then compared with a previous merge algorithm from [23] analytically.

3.1 Histogram Construction

In this section, the constructions of four different types of histograms are presented. Due to the space limitation, only two types will be described in detail while other types will be briefly sketched.

(A) Linear Approximation (LA)

The construction of this type of histogram will be given as follows and will be illustrated by the used-car relation R on price and mileage. The general case will be sketched later in Section 3.3.

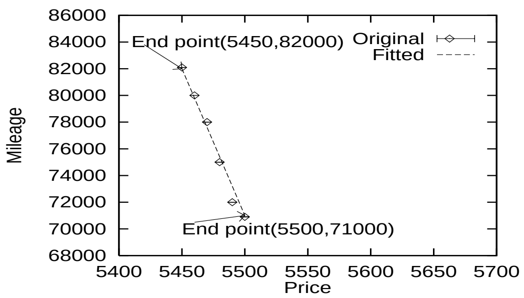

Figure 1: Linear Approximation of a Price Interval

(1). The domain of the price attribute is partitioned into fixed size intervals (each interval has a range of \$50) as follows:

Price	...	\$5400	\$5450	\$5500	...

(2). All the tuples within an interval are replaced by a straight line, which minimizes the least square error. For example in Figure 1, the line is a least square approximation to the various points in the interval [$5450, $5500]. The corresponding entry in our histogram is ($5450, 82000; $5500, 71000). In other words, each entry in the histogram consists of the co-ordinates of the two end points of a line segment. The first co-ordinate of the first point is the beginning price value; the second co-ordinate of the first point is the corresponding mileage value; the first co-ordinate of the second point is the end price value of the interval; the second co-ordinate of the second point is the corresponding mileage value.

(3). Since there may be too many intervals, requiring excessive storage, adjacent intervals are merged until a fixed number of, say r, intervals remain. The criterion to choose which adjacent intervals to merge is given by the Greedy Merge algorithm [13]. For every two adjacent intervals, the combined interval contains a straight line with a least square approximation to the tuples within it. The Greedy Merge algorithm chooses the combined interval with the smallest least square error. This process of combining adjacent intervals continues until r intervals remain. Some typical entries in the histogram are shown in the following table.

First Point	Second Point
....	....
($4000,120000)	($5000, 110000)
($5000,90000)	($6500,70000)
($6500,65000)	($7000,45000)
....	....

This method requires cooperation from local databases in the sense that it requires to have access to all data in each local database.

(B) MAXDIFF MHIST-2 (MHIST-2)

This method [19] operates as follows. Initially, there is a single region (in the case of two-dimensional data, this is a rectangle) containing all points. Then, a region (this is unique initially, but needs to be determined in later iterations of the algorithm) is chosen to be partitioned on one dimension into two regions such that the difference in density (the number of points within a region divided by the size of the region) between the two regions is the largest. (We also tried using frequency instead of density, but the result is worse.) This process is repeated until a fixed number of regions is reached. For each region in the histogram, the number of points as well as two corner points defining the region are kept. This method also needs cooperation from local databases.

(C) Frequent Queries (FQ)

As pointed out in [2], queries usually follow a Zipfian distribution and histograms should be constructed utilizing the uneven query distribution. As a consequence, it is important to provide accurate estimation for frequently submitted queries. In [2], the queries are range-queries. Here, they are top-N queries.

We now propose a method to construct a special type of "histograms" which can be constructed without the cooperation of local database systems. Note that in the Internet environment we cannot expect the needed histograms to be provided by the local systems (e.g., local e-commerce search engines). However, these engines will respond with results when a query is submitted. Thus, our approach is to submit user queries to the different databases (this has to be done anyway, so not much additional overhead is involved) and to construct the "histogram" based on the returned results of the frequently submitted queries. Our intuition is that precise information is kept for data which are frequently accessed. then accuracy of estimation will be obtained. It is well-known that in realistic situations, data in a database are unevenly accessed with small amount of data heavily accessed but most other data lightly accessed. Since heavily accessed data are likely due to frequently submitted queries, we proceed to identify such queries. Ideally, we can keep track of all submitted queries and for each distinct submitted query, we record a frequency of submission. The queries can be sorted in descending order of frequency and the k most frequently submitted queries are retained to construct a histogram. This approach may not be practical, as the number of submitted queries is very large, which requires potentially unbounded storage. In order to approximate the ideal scheme with fixed amount of storage, we utilize two lists (the Freq-Q and Infreq-Q to be described in more details later). The Infreq-Q list serves as a filter. Only queries which have been submitted twice or more within a certain period of time can enter the Freq-Q list. We want to eliminate queries, each of which is submitted more than once but not too many times from the Freq-Q list. This is accomplished by the transposition scheme [20] in which every time an element is accessed, it is interchanged with the next element. It has been shown analytically [20, 15] that the transposition schema is a very good approximation to the ideal situation where records are arranged in descending order of frequencies of access. The transposition method is used to retain the frequently submitted queries. The same principle is used to retain the frequently accessed best matched tuples of the queries. Note that it is possible multiple distinct queries access the same data. We now describe our approach in more detail as follows.

Maintain a linked list of frequent-queries, Freq-Q, a linked list of best matched tuples, BMT, to some of the queries in Freq-Q and a queue of queries, each of which occurs only once within a period of time, Infreq-Q. The three data structures are all of limited fixed sizes. Initially, all three data structures are empty. When a query Q arrives, if it is absent from both Infreq-Q and Freq-Q, and Infreq-Q is not full, it is inserted into Infreq-Q. This indicates that the query is submitted exactly once. If Infreq-Q is full, then Q is inserted and the first (i.e., the oldest) query in Infreq-Q is removed.

If the query is found in Infreq-Q, then it is removed from Infreq-Q and inserted into Freq-Q. If Freq-Q is not at least $x\%$ full for some x (say 90), then the query is inserted at the end of Freq-Q else it is inserted at the $x\%$ position towards the end (allowing some time for it to be moved forward, if it is accessed again. This causes the query at the end of Freq-Q to be dropped.). The top N tuples of Q are found from the databases. From each database that contains at least one of these N tuples, the best matched tuple for the database denoted by B(Q,D) is retained. The set of B(Q,D)'s from different databases together with the corresponding database IDs are placed into the linked list BMT. In addition, the distance of the N-th best matched tuple to Q, N-distance, is attached to the query Q in the linked list Freq-Q. If BMT is not at least $x\%$ full, then {B(Q,D)} is inserted at its end else it is inserted at the $x\%$ position towards the end, possibly causing some end elements to be removed from the linked list.

If the query is found in Freq-Q, then it is moved up one position in the list (i.e., interchanged with the element in front of it). Each of the best matched tuples of Q in the linked list BMT, which can be determined using the distance of the N-th best matched tuple of Q and with a data structure such as G-tree [14], is moved up one position in BMT. The actions in Freq-Q and BMT are to move frequently submitted queries and the frequently accessed tuples towards the fronts of the two lists. The process of processing queries in Infreq-Q, Freq-Q and BMT continues until there are no substantial changes in the set of queries in the front part of Freq-Q. In other words, if the frequent queries submitted by the users become stable, the construction of the "histogram" (which is the linked list of best matched tuples, BMT, together with the frequent queries in Freq-Q) can stop. If user access pattern needs to be monitored for changes, the construction process continues.

When a query Q is submitted to be processed, it is checked against the list Freq-Q. If it is there, {B(Q,D)} (the best matched tuple from each desired database) in BMT are retrieved and the database IDs are used to decide which databases should be accessed to process the query. (It is possible that some elements in {B(Q,D)} are removed from BMT or some of the best matched tuples in various databases have been modified. That situation can be detected by processing the query and by comparing the distance of the N-th retrieved best matched tuple against N-distance. If they are different, additional databases need to be searched to obtain the actual top N tuples. The new B(Q,D)'s need to be inserted into the linked list BMT and the N-distance in Freq-Q needs to be updated.) If the top N tuples of the query are unchanged and all elements in the set {B(Q, D)} are present in BMT, then all top N tuples of Q will be retrieved.

If a query is not found in Freq-Q, the N best matched tuples from BMT are obtained. They are sorted in ascending order of distance and their corresponding database IDs are used in the merge algorithm (see Section 3.4) to retrieve the top N tuples from the databases. Here, there is no guarantee that all of the top N tuples of the query will be retrieved.

(D) Independent Linear Approximation (ILA) In this approach, there is a histogram for each attribute and the values under different attributes are assumed to be independent. This is the approach used in [23]. Again, cooperation of local systems is needed to construct the histograms.

3.2 Criterion for Selecting Databases Optimally

We now sketch how databases containing the top N tuples should be selected. First, all databases should be ranked with respect to the query (Proposition 1). Next, the proper databases (and the proper tuples) are selected using the merge algorithms to be described in Section 3.4,

Definition 1. *Suppose a user is interested in retrieving the N best matched tuples 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 N, there exists a t such that* $D_1, D_2, ..., D_t$ *collectively contain all the N best matched tuples of Q and each* $D_i, 1 \leq i \leq t$, *contains at least one of the N best matched tuples.*

By this definition, if, for a given query, the databases are optimally ranked and the user wants to retrieve the N best matched tuples, then it is sufficient to search only the first t databases. The criterion to rank databases is "for each database, obtain the distance of the best matched tuple in the database to the query; then, rank the databases in ascending order of the distances of the best matched tuples."

Example 2 Suppose there are 5 databases D_1, D_2, D_3, D_4 and D_5. Suppose that the distances of the best matched tuples 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. ■

It is shown in [22] that the above criterion for ranking databases guarantees the optimal ranking of databases. This result can be summarized into the following proposition.

Proposition 1 *For a given query Q, if databases are ranked in ascending order of the distances of the best matched tuples to Q, then they are ranked optimally with respect to Q.*

In [22], the result is stated in terms of similarity. Since similarity and distance are inverses of each other,

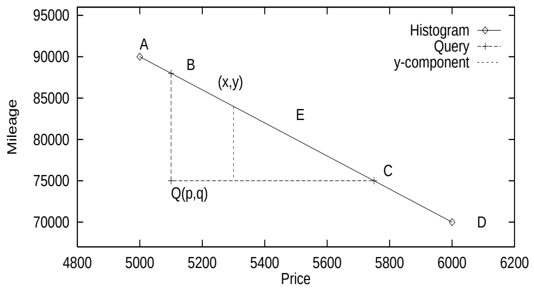

Figure 2: Shortest Distance to the query

the databases are arranged in descending order of similarities in [22].

This proposition cannot be used as is because it is impractical to actually search each database in order to find the distance of its best matched tuple for each query. Therefore, the distance will need to be estimated as described in the next subsection.

3.3 Estimate Smallest Distance to Query Point

The estimation of the distance of the best matched tuple in each database is carried out based on the histogram(s) of the database. Databases are then ranked in the ascending order of estimated distances of the best matched tuples.

First, let us consider the case when the histogram is constructed using the linear approximation (LA) approach. Consider the query (price = p, mileage = q). From the histogram of a database, find the interval I, which contains price = p. From the interval, the distance of the query point to the closest point in the straight line is computed.

To illustrate the process, suppose the Manhattan distance with relative importance w_1 on price and w_2 on mileage is desired between a query point and the closest point on the line. Suppose the least square approximation line and the query point are as shown in Figure 2. A and D are the two end points of the interval, I, and $Q(p, q)$ is the query point. First we identify a segment on the line, AD, where the closest point may lie. From Figure 2, it can be observed that for the line segment AB, B is the closest point to the query and for segment CD, C is the closest point to the query. Therefore, we need to consider only the points in the segment BC to find the closest point.

For the least square approximation line $y = mx + c$, the normalized and weighed standard Manhattan distance of any point on BC to query Q is

$$t = \frac{(mx + c - q)w_2}{q} + \frac{(x - p)w_1}{p} \qquad \frac{dt}{dx} = \frac{mw_2}{q} + \frac{w_1}{p}$$

Since the first derivative of distance is a constant, the closest point will be either B or C. Thus, the closest distance is:

$$\min\left\{\frac{(q - c - mp)w_1}{mp}, \frac{(mp + c - q)w_2}{q}\right\}$$

Other situations in which line $y = mx + c$ may not intersect with the lines $x = p$ or $y = q$ can be handled in a similar way. A similar result is obtained, if Euclidean distance is used. Other cases (generalized Euclidean distance, generalized distance, two-sided Euclidean distance and two-sided Manhattan distance) can be dealt with in a similar manner. (Note that non-linear approximation should yield higher accuracies. However, it will take more space to store histograms. Since each method has a fixed amount of space, we choose a linear approximation for both space efficiency and computation efficiency.)

The closest point to the query point may or may not appear in the interval, I. To improve the accuracy, two alternatives are chosen. The first alternative is to consider all the intervals. The second is to choose a fixed number of intervals, each of which is either I or close to I. Then the distance of the closest point is the minimum of the distances computed in the various intervals. If all the intervals are used , the time complexity is $O(|F|)$, where $|F|$ is the number of intervals, since for each interval it takes constant time to estimate the shortest distance to the query point. If a fixed number of intervals is used, then only constant time is needed. However, finding the interval containing the query point takes $O(log|F|)$ times. In summary, for a two-attribute relation, the estimation time is either $O(|F|)$ or $O(log|F|)$.

The above scheme for histogram construction and estimation can be generalized to n-dimensions for $n \geq 2$. Intervals and straight lines in 2 dimensions are replaced by hyper rectangles and hyper planes in higher dimensions.

Now consider other types of histograms. For MHIST-2, to estimate the distance of the best matched tuple to a given query, it is assumed that the points within a region are uniformly distributed. The idea of selecting databases for a given query using the FQ method is sketched in Section 3.1 (see the last two paragraphs of item (C)). For ILA, the method proposed in [23] to perform the estimate can be used and will not be repeated here. This method has an exponential complexity.

3.4 Algorithms to Select Tuples from Ranked Databases

In this section, two algorithms for selecting tuples from the ranked databases are given. The first algorithm was proposed in [23] and is included for comparison purpose; this algorithm will be referred to as Merge-1 in this paper. The second algorithm is actually a sequence of algorithms with increasing retrieval effectiveness and decreasing efficiency. This sequence of algorithms will be called MIN-p, p $\geq$ 2. (The meaning of its name will be obvious soon.) We will start off with the first algorithm of this sequence, named MIN-2. This algorithm will be slightly modified to become MOD-MIN-2. Then, Merge-1 and Mod-MIN-2 algorithms will be compared in terms of efficiency and effectiveness. It will be shown that Merge-1 is more efficient (i.e., fewer databases are likely to be accessed for a given query), while the Mod-MIN-2 is more effective (i.e., more actual top N tuples are likely to be obtained). It will also be shown that MIN-p is more effective and less efficient than MIN-(p-1), p $\geq$ 3. If the modification performed on MIN-2 is applied to MIN-p, p $\geq$ 3, then Mod-MIN-p has higher effectiveness than MIN-(p-1) which is at least as effective as Mod-MIN-2.

3.4.1 Algorithm Merge-1

Let databases be ranked in the order $[D_1, D_2, ..., D_n]$. Let N be the number of tuples the user wants to see. The databases are accessed in the order in which they are ranked, one at a time. (In practice, the first few highest ranked databases may be accessed in parallel, since they are likely to contain the desired tuples.) From each of the first two accessed databases, we obtain its best matched tuple for the query. 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 N, then sort them in ascending order of distances and return the top N results; 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 best matched tuples, one from each of these t databases. Tuples from these t databases with distance $\leq d$ are retrieved. If N or more tuples have been retrieved, then they are sorted in ascending order of distances, the first N tuples are returned to the user and the process is terminated; else the next database is accessed. Two remarks about the above merge algorithm:

1. The above description seems to indicate that some database may be accessed multiple times. In practice, the number of accesses to each such database can be restricted to one. Whenever a database is accessed for the first time, its top N tuples are retrieved and cached at the global site. All subsequent "interactions" between the accessed database and the global site will actually take place at the global site only.

2. In the future, relational databases for e-commerce will support top N queries of different types as discussed in Section 2. Thus, in this paper, we assume that local distance functions are identical to the corresponding global distance functions. If this is not the case, each invoked database is required to return N or slightly more local top N tuples. These tuples are then re-ranked using the global distance function.

The remarks apply to all the merge algorithms presented in this section.

3.4.2 Algorithm MIN-2

The Merge-1 algorithm retrieves at least one tuple from each database it accesses, because when database D_i is accessed, tuples with distances less than or equal to the maximum of the distances of the best matched tuples in the accessed databases are retrieved. If an accessed database does not contain any of the N best matched tuples of the query, some of its tuples will still be retrieved. To remedy this situation, the MIN-2 algorithm allows such a bad database to be skipped over, as long as each of its two adjacent databases contains at least one of the N best matched tuples.

As before let these databases be ordered in $D_1, D_2, ..., D_n$. MIN-2 operates as follows. Access the top two ranked databases D_1 and D_2 to obtain the best matched tuple from each of them. Let the distances of the best matched tuples be denoted by d_1 and d_2. Let the minimum of the two values be threshold $d = \min\{d_1, d_2\}$. Tuples from these two databases with distances $\leq d$ are retrieved. (If all distances are distinct, only one tuple will be retrieved in this step.) If N or more tuples are retrieved, sort them in ascending order of the actual distances; output the first N tuples and terminate; else, access the next database. Let this database be denoted D_i and let d_i be the actual distance of the best matched tuple in this database. Re-compute $d = \min\{d_i, d_{i-1}\}$, where d_{i-1} is the distance of the best matched tuple in the last accessed database. Retrieve from all accessed databases tuples with distance $\leq d$. (Tuples which have been retrieved previously will not be retrieved again.) If N or more tuples have been retrieved, sort them in ascending order of the actual distances; output the first N tuples and terminate; else, the process is repeated until N or more tuples are retrieved. Then output the N tuples with the smallest distances.

Proposition 2 Suppose N tuples are to be retrieved by algorithm MIN-2. If the first t databases contain T, the set of the top N tuples with respect to a given query, and the t-th database contains at least one tuple in T, then the algorithm MIN-2 will terminate after accessing at most $\min\{t+2, 2N\}$ databases.

Note: This result says that the algorithm must terminate after examining a certain number of databases, irrespective of whether the top N tuples are retrieved. If the estimations of the distances of the best matched tuples in the databases are reasonably accurate in the sense that the databases containing the top N tuples are ranked higher than other databases, then all of the top N tuples in T will be retrieved as shown in Proposition 3 below; otherwise, the algorithm terminates without necessarily retrieving all tuples in T. Note that both cases (case 1 with $t+2 > 2N$ and case 2 with $t+2 \leq 2N$) are possible. Furthermore, the bounds $t+2$ and $2N$ can actually be reached. Thus, the minimum of $t+2$ and $2N$ is a tight bound.

It is possible that algorithm MIN-2 needs to access $2N$ databases in order to retrieve N tuples. For example, if the following conditions are satisfied, then exactly $2N$ databases need to be accessed. The conditions are:

$d_{2i} > d_{2(i+1)}, \quad 1 \leq i \leq (N-1)$

$d_{2i-1} > d_{2i+1}, \quad 1 \leq i \leq (N-1)$

$d_{2i-1} > d_{2i} < d_{2i+1}, \quad 1 \leq i \leq (N-1)$

When these conditions are satisfied, then (with tuples properly distributed in these databases) exactly one tuple is retrieved from each database $2i, 1 \leq i \leq N$. Note that although the above situation is possible, in reality it would not happen for any reasonable estimation formula for the distance of the best matched tuple in a database with respect to any query. A reasonable estimation formula would have the distance of the best matched tuple in a database increases as the number of accessed databases increases, for most databases. In the above hypothetical situation, the distance decreases for the odd-numbered databases and for the even-numbered databases.

Proof of Proposition 2: Let the databases be accessed in the order $D_1, D_2, \ldots$. Let d_i be the distance of the best matched tuple in D_i for a given query, $i = 1, 2, \ldots$. It suffices to show that at least one tuple will be retrieved from any two consecutive databases, which are distinct from the next two databases. Consider databases D_i and D_{i+1}. When database D_{i+1} is processed, either $d_{i+1} \leq d_i$ or $d_{i+1} > d_i$. In the former case, one or more tuples will be retrieved from database D_{i+1}. In the latter case, at least one tuple will be retrieved from database D_i. Thus, at least one tuple will be retrieved from these two databases. Consequently, for every two non-overlapping consecutive databases $[D_1, D_2]$, $[D_3, D_4]$, ..., $[D_{2N-1}, D_{2N}]$, at least one tuple is retrieved. This shows that the algorithm must terminate after accessing at most $2N$ databases.

Let N_d be the distance of the N-th best matched tuple in the first t databases. Consider database D_{t+1}. Since this database does not contain any tuple in T (but may have one or more tuples having the same distance as N_d), $d_{t+1} \geq N_d$. Similarly, for database D_{t+2}, $d_{t+2} \geq N_d$. When database D_{t+2} is examined, threshold $d = \min\{d_{t+1}, d_{t+2}\} \geq N_d$ and this will cause T to be retrieved and the algorithm to be terminated. ■

Proposition 3 Let the distance of the N-th best matched tuple, N_d, be distinct. If the first t databases collectively contain T, the set of the top N tuples and each of these databases contains at least one tuple in T, then algorithm MIN-2 will retrieve all tuples in T by accessing at least $t+1$ databases and at most $t+2$ databases.

The following example shows that the condition "the distance of the N-th best matched tuple is distinct" is necessary for retrieving all of the top N tuples.

Example 3 Let $N = 5$. Suppose for a given query, databases are ranked in the order $[D_1, D_2, D_3, D_4, \ldots]$; D_1 has a tuple with distance 0.5; D_2 has 3 tuples, all having the distance 0.6; D_3 has a tuple with distance 0.6; D_4 has a tuple with distance 0.4; all other tuples, if any, have distances higher than 0.6. Clearly, the top 5 tuples have distances (0.4, 0.5, 0.6, 0.6, 0.6). When algorithm MIN-2 is invoked, after database D_3 is processed, the 5 tuples with distances (0.5, 0.6, 0.6, 0.6, 0.6) are retrieved, without retrieving the best matched tuple with distance 0.4. This happens because several tuples, including the 5-th best matched tuple, have the same distance 0.6. ■

Proof of Proposition 3: Suppose the N-th best matched tuple is from database D_i, for some $i \leq t$. Since each of these t databases contains at least one tuple from T, $d_j \leq N_d, 1 \leq j \leq t$. Since N_d is distinct, $d_j < N_d, 1 \leq j \leq t, j \neq i$. When processing databases $D_1, \ldots, D_j, j < i$, the thresholds for these databases $\leq \max\{d_1, \ldots, d_j\} < N_d$. When processing database D_i, the threshold $= \min\{d_{i-1}, d_i\} < N_d$. When processing databases $D_{i+1}, \ldots, D_j, i < j \leq t$, the thresholds $\leq max\{d_{i+1}, \ldots, d_j\} < N_d$. There are fewer than N tuples with distance $< N_d$. Thus, the algorithm cannot stop after processing the first t databases only. When processing the $(t+1)$th database, $d_{t+1} > N_d$. The threshold $= min\{d_t, d_{t+1}\} = d_t$. If $d_t = N_d$, then T will be retrieved and the algorithm terminates else since there are fewer than N tuples with distances $< N_d$, database D_{t+2} needs to be examined. The threshold is $min\{d_{t+1}, d_{t+2}\} > N_d$, implying that all tuples in T will be retrieved. ■

Although in general the MIN-2 algorithm has higher retrieval effectiveness than the Merge-1 algorithm, the following example shows that it is possible that MIN-2 has a lower effectiveness in some situation.

Example 4 Suppose site 1 has a single tuple with distance 0.6, site 2 has a single tuple with distance 0.55, site 3 has a tuple with distance 0.52, site 4 has 3 tuples with distances 0.4, 0.45 and 0.5 and site 5 has a single tuple with distance 0.3. Let the number of tuples to be retrieved be 4. The original algorithm, Merge-1, after visiting site 4, retrieves the tuples with distances 0.6, 0.55, 0.52, 0.4, 0.45, and 0.5. It then returns the 4 tuples with distances 0.4, 0.45, 0.5 and

0.52. After visiting site 4, MIN-2 retrieves the tuples with distances 0.55, 0.52 and 0.4 only. Therefore, it needs to proceed further. After visiting site 5, it retrieves the tuples with distances 0.55, 0.52, 0.4 and then 0.3. Clearly, the 4 best matched tuples are those with distances 0.3, 0.4, 0.45 and 0.5. The original algorithm, Merge-1, retrieves 3 of them, while the MIN-2 algorithm retrieves only 2 of them. ■

One possible way for a remedy is as follows. We proceed exactly the same as MIN-2. At the same time, we also consider the set of tuples, P, which is retrievable by the original algorithm, Merge-1. At any site, the number of tuples retrieved by the modified MIN-2 algorithm is identical to that of MIN-2. However, if there are tuples in P which have lower distances than those of tuples retrievable by MIN-2, replace the latter tuples by the former tuples.

Example 5 Continue on the last Example. After visiting the first 4 sites, the new algorithm retrieves the tuples with distances 0.55, 0.52 and 0.4. But, it also considers the set of tuples retrievable by Merge-1. This includes the set of tuples with distances 0.45 and 0.5. Since the tuples having distances 0.55 and 0.52 are not as good as tuples with distances 0.45 and 0.5, they are replaced by the latter two tuples. Thus, after visiting 4 sites, the set of tuples retrieved by the modified MIN-2 have distances 0.4, 0.45 and 0.5. After visiting site 5, we get the four tuples with distances 0.3, 0.4, 0.45 and 0.5. This set of tuples has higher retrieval effectiveness than the set of tuples retrieved by the original algorithm, Merge-1. ■

The modified algorithm, MOD-MIN-2, is described below. The details of the modification of the algorithm are as follows.

When database i is visited, two sets of tuples are considered. Let min = $\min\{d_{i-1}, d_i\}$. All tuples from databases 1 to i with distances $\leq$ min are definitely retrieved when database i is visited. Let this set be DRet_i. Let max = $max\{d_1, ..., d_n\}$. Let the set of tuples with distances $\leq$ max be DPot_i. This set of tuples will be used to ensure that Dr_i, the set of retained tuples up to and including database i are the tuples with the smallest distances within those databases. Specifically, let Dr_2 be the set of tuples retrieved from databases 1 and 2 having distance $\leq \min\{d_1, d_2\}$; for $i > 2$, Dr_{i-1} be the set of tuples retained by the Mod-MIN-2 algorithm after visiting databases 1, 2 ,..., $i-1$. Then, Dr_i is obtained as follows.

1. Get $\text{PR}_i = \text{Dr}_{i-1} \cup \text{DRet}_i$; these tuples consist of newly retrieved tuples after visiting database i plus the accumulated tuples after visiting databases 1, ..., $i-1$. Let the cardinality of PR_i be r and the cardinality of DRet_i be m.
2. Get $\text{DUC}_i = \text{DPot}_i - \text{PR}_i$; these tuples are retrievable by the original algorithm, Merge-1, after visiting database i but may not be retrieved by the MIN-2 algorithm;
3. Find $(r-m)$ tuples in $\text{DUC}_i \cup (\text{Dr}_{i-1} - \text{DRet}_i)$ having the smallest distances. Let this set of $(r-m)$ tuples be AD_i
4. $\text{Dr}_i = \text{DRet}_i \cup \text{AD}_i$. This ensures that the new algorithm retrieves the same number of tuples as MIN-2.

If N or more tuples are retrieved, then sort the tuples in descending order of similarity, return the top N tuples and terminate; else, the next database is visited and the same process is applied.

Lemma 1 *After visiting database i, the r tuples retained by the new algorithm, MOD-MIN-2, are the r tuples with the smallest distances within the i databases.*

Proof: All tuples with distances $\leq$ min within the i databases are included in DPot_i. These are the tuples with the smallest distances within the i sites. DPot_i can be partitioned into DRet_i, $\text{Dr}_{i-1} - \text{DRet}_i$ and DUC_i (see Steps 1 and 2). The first set contains tuples with the smallest distances (distances $\leq$ min), while the second and third sets have tuples with distances between min and max. If r is exactly the cardinality of DRet_i, m, then the tuples with the smallest distances are retrieved. If $r > m$, let $f = r - m$. The f tuples outside DRet_i and having the lowest distances among tuples in $(\text{Dr}_{i-1} - \text{DRet}_i) \cup \text{DUC}_i$ are found in Step 3 and retained in Step 4. ■

Proposition 4 *The new algorithm, MOD-MIN-2, has effectiveness at least as high as that of algorithm Merge-1, when the number of databases is at least 2N.*

Proof: Let the original algorithm terminates after visiting t databases. Let j be the number of tuples retrieved by the new algorithm after t databases are visited. If $j \geq N$, then by Lemma 1, the N best matched tuples within the t databases are retrieved. Thus, it is at least as effective as the original algorithm. If $j < N$, then additional tuples need to be searched to obtain N or more tuples. By Proposition 2, the MIN-2 algorithm will terminate after searching at most 2N databases. The new algorithm retrieves exactly the same number of tuples as that retrieved by MIN-2. Thus, it must retrieve N or more tuples, after visiting at most 2N databases. Let the number of databases visited be s. Clearly, $s > t$. After these s databases are visited, the number of tuples retained is N. Again, by Lemma 1, these are the N tuples having the smallest distances within the first s databases. Since $s > t$, these tuples are at least as good as the N tuples retrieved by the original algorithm. ■

We now generalize the MIN-2 algorithm to MIN-p algorithm, with $p \geq 2$. MIN-p takes the minimum of the distances of the best matched tuples (tuples with smallest distances) in the current and the preceding $(p-1)$ databases as the threshold. All tuples with distances $\leq$ the threshold from the current database to all preceding databases are retrieved. If N or more tuples are retrieved, then sort the tuples in descending order of similarity, take the top N tuples and terminate else the process is continued for the next database.

Definition 1 *A database is bad, if it does not contain any of the N best matched tuples.*

Proposition 5 *Suppose the N best matched tuples are contained in the first t databases with the t-th database containing at least one of the N best matched tuples. Suppose N_d is the distance of the N-th best matched tuple and it is distinct. Suppose for the first t databases, for every $p \geq 2$ consecutive ordered databases, there are at most p-1 bad databases. Then, after searching at most the first $(t+p)$ databases, all the N most similar tuples will be retrieved.*

Proof: Consider a bad database d' among the first t databases. At the time when database d (which may or may not be d') among the first t databases is visited, the threshold is the minimum among the distances of the tuples with smallest distances in database d and in the previous $(p-1)$ databases. By the hypothesis, one of these p databases must contain one of the N best matched tuples. Thus, the threshold $\leq N_d$. Thus, no tuple is retrieved from d' when any database among the first t databases is accessed. When database t is accessed, if N_d is the distance of the best matched tuple in database t, then all of the N best matched tuples will be retrieved and the algorithm terminates. Otherwise, the distance of the best matched tuple in database t is $f < N_d$. When database $(t+i), 1 \leq i \leq p-1$, is accessed, the threshold is no larger than f. Thus, no tuple will be retrieved from these $p-1$ databases. When database $(t+p)$ is accessed, the threshold is larger than N_d, implying all of the N best matched tuples will be retrieved and will be ahead of all other tuples. ■

The above proposition says that the MIN-p merge algorithm allows up to (p-1) bad databases for every p consecutive databases and gets optimal retrieval. The bigger the value of p, the more tolerant the algorithm is. However, it becomes less efficient.

It is obvious that when the modification which is applied to MIN-2 is applied to MIN-p, $p \geq 2$, then the modified MIN-p algorithm will have better retrieval effectiveness than the modified MIN-(p-1) algorithm. Since the Mod-MIN-2 algorithm has better retrieval effectiveness than Merge-1, the Mod-MIN-p algorithm has better effectiveness than Merge-1 as well.

4 Experiments

The data and query collections used in the experiments are described in Section 4.1. In Section 4.2, two measures of retrieval, one reflecting the effectiveness (accuracy) and the other reflecting the efficiency, are provided. Experimental results are provided in Section 4.3.

4.1 Data Sets and Query Sets

Two data sets are used: used car data and forest cover data.

(1). 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 32,449 tuples. The tuples are arbitrarily assigned to 28 databases, without any duplication of tuples. Two types of queries are used and they are 2-D queries and 3-D queries. 2-D queries are two-attribute queries involving *price* and *mileage* and 3-D queries are three-attribute queries involving *price*, *mileage* and *age*. The values associated with the two attributes are chosen to reflect reality and some diversification. Ideally, if the mileage is high, the price must be low; if the mileage is low, the price must be high. A set of 50 queries is generated on price and mileage. This query set is used in all the experimental runs. For each query, the three interpretations as discussed in Section 2 are applied and they are: the standard top-N queries; the generalized top-N queries and the two sided-generalized top-N queries. For each interpretation, the two "distance functions", namely, the Euclidean distance and the Manhattan distance functions are used.

(2). Forest Cover data were obtained from University of California at Irvine (ftp://ftp.ics.uci.edu/pub/machine-learning-databases/covtype/). There are 581,012 tuples and they are randomly assigned to 50 databases. Three types of queries are utilized. 2-D queries involve the attributes *elevation* and *slope*; 3-D queries uses an additional attribute *horizontal_distance_to_road*; 5-D queries use two more attributes *horizontal_distance_to_hydro* and *horizontal_distance_to_fire_points*.

A set of 10,000 distinct queries that satisfy the Zipfian distribution are randomly generated, i.e., the i-th most frequently submitted query has probability $\frac{1/i}{\sum_{j=1}^{K} 1/j}$, where $K = 10,000$ is the number of distinct queries. This is used for the FQ method. For all other methods, the 50 queries having the highest frequencies of submission are used. (If the 50 queries with highest frequencies were used by the FQ method, then 100% accuracy will be achieved.) Each method is assigned the same amount of space for histogram construction. It is about 2% of the size of the data for the forest-cover data set (for example, for the FQ method, space is allocated for up to 12,000 best matched tuples, 1,000 frequent queries and 1,000 in-frequent queries in Infreq-Q) and also about 2% of the size of the data for the used-car data set.

4.2 Performance Measures

Performances are measured in two ways: the effectiveness of the retrieval and the efficiency of the retrieval. The former is measured by the number of the retrieved tuples that are among the actual N best matched tuples divided by N. If the quantity is 100%, then all of the N best matched tuples are retrieved. Higher percentages indicate higher retrieval effectiveness. The latter (i.e. efficiency) is measured by two factors: the first factor is the ratio of the number of databases accessed to the actual number of databases containing the N best matched tuples. If the ratio is 100%, then the number of databases accessed is the same as the number of databases containing the N best matched tuples although not necessarily the same set of databases is accessed. A ratio exceeding 100% indicates inefficiency. Lower ratios indicate higher efficiency. However, any ratio below 100% indicates that the retrieval effectiveness is also below 100%. The second factor is the time to determine the ordering of the databases with respect to the query by executing against the histogram. (This is an overhead versus sending a query to all databases.) Since the FQ method seems to be the only viable method (see the next subsection), we shall report that factor for the FQ Method only. Ideal retrieval is achieved when accuracy is 100%, efficiency is 100% and the time to order the databases for each query is small.

4.3 Experimental Results

In Section 4.3.1, we use the used car data set to show that the new merge method, MOD-MIN-2, used in conjunction with Linear Approximation Method (LA) is much better than the previous merge method used in conjunction with the Independent Linear Approximation Method (ILA). (Experimental results showing that the new merge method by itself yields substantial improvement over the previous merge method are obtained, but they are not shown here for lack of space.) Thus, we eliminate the ILA method from further consideration. In Section 4.3.2, the three methods LA, MHIST-2 and FQ are compared with respect to the used-car data set and the forest-cover data set. It will be shown that in most situations, the FQ method achieves significantly higher accuracy than the other two methods for both 2-D and 3-D queries and has high accuracy in all situations. 5-D queries using the forest-cover data set are executed by the FQ method and superior results are also obtained. (The used car data set does not have 5 attributes allowing approximation. Attributes such as the model and the make of a car require exact satisfaction, not approximate satisfaction.) In the above experiments, a histogram is constructed for each combination of distance function and query dimension. Having so many histograms is simply not practical. Thus, in Section 4.3.3, we also experiment with the situation that a single histogram is used for all types of queries (having different dimensions and/or having different distance functions). The FQ-method continues to perform well when a single histogram is used for queries of various types. The original algorithm and the new merge algorithm, MOD-MIN-2, are used for different query types to yield both high accuracy and high efficiency. The average time to order the databases for each query for the FQ method is reported to be reasonable.

	Method LA		**Method ILA**		
N	Accu.	Effic.	Accu.	Effic.	#Q
5	70.4	153.7	28.4	75.6	50
10	74.2	135.2	31.6	68.9	50
20	83.6	118.9	37.3	59.9	50
30	86.7	116.5	42.3	60.3	50

Table 1: Standard Manhattan Distance

	Method LA		**Method ILA**		
N	Accu.	Effic.	Accu.	Effic.	#Q
5	69.6	156.9	27.6	73.8	50
10	77.6	137.6	30.4	67.1	50
20	86.5	124.4	36.5	58.3	50
30	88.8	119.6	39.9	60.9	50

Table 2: Standard Euclidean Distance

4.3.1 Comparing ILA with LA Using the Used Car Data Set

A previous solution [23] consists of the Merge-1 algorithm and a method (a generating function approach with an exponential time complexity) to estimate the distance of the best matched tuple based on the ILA histograms. The LA method consists of the new merge algorithm, MOD-MIN-2, the LA histograms and a new algorithm (with low polynomial time complexity, as given in Section 3.3) for estimation.

Tables 1 and 2 give the accuracies and the efficiencies of the two methods for 2-D queries when the standard Manhattan and the standard Euclidean distance functions are employed. (In all tables, #Q stands for "number of queries used".) It clearly demonstrates that the LA method is much better than the ILA method. Furthermore, the ILA has such low accuracy that it is not useful in practice. (In [23], much higher accuracy was reported for the ILA method. However, much more space was allocated for histogram construction in [23] than in here.)

	Method LA		Method ILA		
N	Accu.	Effic.	Accu.	Effic.	#Q
5	100	193.5	42.0	147.0	50
10	100	169.2	48.8	181.0	50
20	100	153.0	49.7	189.0	50
30	99.3	141.8	52.9	158.5	50

Table 3: Generalized Manhattan Distance

	Method LA		Method ILA		
N	Accu.	Effic.	Accu.	Effic.	#Q
5	83.6	141.7	49.6	98.5	50
10	83.4	124.2	54.6	84.8	50
20	86.7	113.2	57.1	76.7	50
30	89.6	110.1	60.2	76.2	50

Table 4: Two-sided Manhattan Distance

Tables 3 and 4 show the corresponding results for the methods LA and ILA for the Generalized Manhattan Distance function and the Two-sided Manhattan Distance. When an estimation method, a histogram construction method and a merge algorithm are fixed, the differences in accuracy between the Manhattan distance and the Euclidean Distance functions are usually 1 to 2 percentages. As a consequence, we do not show the corresponding results for the Generalized Euclidean Distance function and the Two-sided Euclidean Distance function. Again, for all these distance functions, the LA method with the new merge algorithm has much higher accuracy than the ILA method with the original merge algorithm. Based on the results identified in Tables 1-4, it is clear that significantly higher accuracies can be achieved by having a histogram construction process which incorporates the dependencies between attributes and utilizing the new merge algorithm.

4.3.2 Comparing Methods LA, MHIST-2 and FQ

In the second set of experiments, the following issues are addressed: (1) Are there significant differences in accuracies among the methods: LA, MHIST-2 and FQ; (2) How do these methods perform when the number of dimensions increases. It is known that maintaining accuracies in high dimensional space is very challenging. For both issues, both the used-car data set and the forest-cover data set are used. For issue 1, only 2-D queries are submitted and answered. For issue 2, the number of dimensions is increased to 3 and then to 5.

From Tables 1-5 involving the used-car data set, the best performance in accuracy is achieved by the FQ method; it is followed by the LA method and then the MHIST-2 method. The differences in accuracy between the FQ method and the LA method are very significant for the Standard Manhattan distance and the Standard Euclidean distance functions, ranging from 20% for 5 best matched tuples to about 6-9% for 30 best matched tuples. The high accuracy of the FQ method for the Standard Manhattan and the Standard Euclidean distance functions is also confirmed when the Forest Cover data set is used, as seen in Tables 9-10. While the MHIST-2 method is inferior to the LA method for the used-car data, it is better than the latter method for the forest cover data set (see Tables 9-11). For both data sets, the FQ method is much better than the other methods for the two standard distance functions. For both data sets, the same is true in efficiency.

For the Generalized Manhattan distance function, the 3 methods LA, MHIST-2 and FQ all averaged higher than 95% accuracy (see Tables 3 and 7). These methods also perform similarly for the generalized Euclidean distance function (the results are not shown here due to space limitation). Thus, all these methods perform extremely well for these two generalized distance functions, although the FQ method performs slightly worse. In terms of efficiency, the three methods are comparable for the used-car data set. For each of the Two-Sided distance functions, each method performs better than its corresponding Standard distance function but worse than the corresponding Generalized distance function (See for example, the results of FQ in Tables 5, 7 and 8). For these Two-sided distance

	Method MHIST-2			Method FQ		
N	Accu.	Effic.	#Q	Accu.	Effic.	#Q
5	48.0	162.5	50	90.9	139.2	10000
10	58.6	137.5	50	93.3	136.3	10000
20	72.7	122.9	50	95.0	122.4	10000
30	81.3	123.3	50	95.5	118.9	10000

Table 5: Standard Manhattan Distance, 2-D Used-Car Data Set

	Method MHIST-2			Method FQ		
N	Accu.	Effic.	#Q	Accu.	Effic.	#Q
5	49.2	154.0	50	89.3	139.7	10000
10	56.6	134.8	50	91.8	137.6	10000
20	69.4	123.0	50	93.6	120.5	10000
30	76.1	121.0	50	94.2	121.9	10000

Table 6: Standard Euclidean Distance, 2-D Used-Car Data Set

	Method MHIST-2			Method FQ		
N	Accu.	Effic.	#Q	Accu.	Effic.	#Q
5	97.2	200.6	50	100	196.1	10000
10	96.2	172.5	50	93.8	163.9	10000
20	94.7	156.5	50	98.2	153.4	10000
30	98.7	159.0	50	99.0	144.1	10000

Table 7: Generalized Manhattan Distance, 2-D Used-Car Data Set

	Method MHIST-2			Method FQ		
N	Accu.	Effic.	#Q	Accu.	Effic.	#Q
5	65.2	159.3	50	96.1	141.3	10000
10	71.6	129.8	50	96.0	134.1	10000
20	74.8	118.5	50	95.4	117.2	10000
30	77.0	116.4	50	94.4	120.7	10000

Table 8: Two-Sided Manhattan Distance, 2-D Used-Car Data Set

	Method MHIST-2			Method FQ		
N	Accu.	Effic.	#Q	Accu.	Effic.	#Q
5	62.8	245.8	50	95.8	167.5	10000
10	67.0	195.0	50	96.8	152.9	10000
20	70.4	169.4	50	97.4	140.9	10000
30	78.5	160.0	50	97.8	131.0	10000

Table 9: Standard Manhattan Distance, 2-D Forest Cover Data Set

functions, the FQ method has superior performance (see Table 8. Results for the FQ method involving the Two-sided Euclidean distance function are similar and are not shown.). In summary, the FQ method has excellent accuracies for both data sets and for all 6 distance functions.

We now report the results for the 3 methods involving 3 attributes using the Standard Manhattan Distance and the Standard Euclidean Distance only. The accuracy results for the Two-Sided distance function and the Generalized distance function will be higher, as observed earlier. We first report the results of the three methods on the forest-cover data set. As can be seen in Tables 12-13, the accuracies of the two methods LA and the MHIST-2 are too low to be acceptable, while the FQ method has accuracies above 95%. Only the efficiency of the last method is reported. It is reasonably efficient. The results of the FQ method for the used-car data set are reported in Table 14. Although the results are worse than those reported for the same method with 2-D attributes, the accuracies remain high, from 83% to 91%.

The results for the FQ method involving 5-D attributes are reported in Table 15. (For the used-car data set, there are no 5-D top-N queries, as complete satisfaction is required for attributes such as model and make for cars.) In comparing the results in Table 15 to those in Tables 12-13, it is clear that there is essentially no difference in accuracy between 5-D attributes and 3-D attributes, although there is a deterioration in retrieval efficiency. Accuracies stay above 95% on the average. Based on the results reported here, the FQ method is the only method that produces high accuracy for 2-D, 3-D and 5-D attributes for both data sets. Its efficiency is also very reasonable.

	Method MHIST-2			Method FQ		
N	Accu.	Effic.	#Q	Accu.	Effic.	#Q
5	61.6	241.3	50	95.8	170.2	10000
10	64.8	208.4	50	97.0	157.7	10000
20	71.1	177.4	50	97.5	141.9	10000
30	79.5	163.7	50	97.8	132.8	10000

Table 10: Standard Euclidean Distance, 2-D Forest Cover Data Set

	Standard Euclidean			Standard Manhattan		
N	Accu.	Effic.	#Q	Accu.	Effic.	#Q
5	64.4	197.3	50	66.4	198.0	50
10	67.4	158.2	50	67.0	154.0	50
20	67.3	134.7	50	67.6	133.3	50
30	69.7	120.0	50	67.9	114.4	50

Table 11: Method LA, 2-D Forest Cover Data Set

	Method LA	Method MHIST-2	Method FQ	
N	Accu.	Accu.	Accu.	Effic.
5	34.0	46.0	95.8	170.2
10	36.8	50.8	97.0	157.7
20	43.1	51.9	97.5	141.9
30	45.2	57.1	97.8	132.8

Table 12: Standard Euclidean Distance, 3-D Forest Cover Data Set

4.3.3 A single histogram for a mixture of queries

In reality, users submit different types of queries (different dimensions, different distance functions) at different times. Our last experiment involves 2-D, 3-D and 5-D queries with the 6 different distance functions. Each of the 2-D, 3-D and 5-D type of queries has approximately the same number of occurrences; each of the 6 distance functions involved in the queries also has approximately the same number of occurrences. 10,000 such distinct queries satisfying the Zipfian distribution are generated. 20,000 such queries, generating from the same distribution are used to construct the histogram using no more than 2% of the space of the data. Then, 1,000 queries are submitted to test the accuracy and the efficiency of the FQ method. As indicated in the left part of Table 16, accuracy ranges from 95.7% to 97.7%. However, efficiency ranges from 205% to 146%. After examining the accuracies of the queries, it is found that for the Standard Euclidean

	Method LA	Method MHIST-2	Method FQ	
N	Accu.	Accu.	Accu.	Effic.
5	45.6	41.6	95.8	167.5
10	49.0	45.0	96.8	152.9
20	57.0	50.8	97.5	140.7
30	60.9	54.9	97.8	131.0

Table 13: Standard Manhattan Distance, 3-D Forest Cover Data Set

	Standard Manhattan		Standard Euclidean	
N	Accu.	Accu.	Accu.	Effic.
5	83.0	144.7	83.4	144.7
10	86.7	127.9	87.2	128.5
20	90.5	120.6	90.1	118.4
30	92.6	118.7	91.7	118.6

Table 14: Method FQ, 3-D Used-Car Data Set

	Standard Manhattan		Standard Euclidean	
N	Accu.	Accu.	Accu.	Effic.
5	95.2	214.1	94.4	233.4
10	96.2	182.0	95.4	202.7
20	97.1	162.7	96.7	174.1
30	97.2	152.5	96.6	161.0

Table 15: Method FQ, 5-D Forest Cover Data Set

Distance queries and the Standard Manhattan Distance queries, the use of the new merge algorithm gives approximately 3-4% improvement in accuracy over the original merge algorithm, while for all other queries, there is negligible difference in accuracy between the merge algorithms. Thus, the experiment is repeated in which the new merge algorithm is used for the former two types of queries and the original merge algorithm is used for the remaining four types of queries. The right part of Table 16 shows that such a strategy yields essentially the same accuracy, while efficiency improves significantly. The average time to determine the ordering of the databases for a query using the FQ method is 37.8 msec. Thus, the FQ method can order the databases accurately in reasonable time.

	MOD-MIN-2		Merge-1 & MOD-MIN-2 combined	
N	Accu.	Effic.	Accu.	Effic.
5	95.7	205.1	95.7	153.0
10	96.9	169.7	96.7	133.8
20	97.5	156.0	97.4	126.3
30	97.7	146.4	97.6	120.1

Table 16: Method FQ

5 Conclusion

In this paper, we presented a two-step solution for finding the N best matched tuples for a given query in a distributed relational database environment. Four different methods for implementing the first step, i.e. ranking databases based on the estimated distance of the best matched tuple in each database, were compared. In addition, the new merge algorithm (MOD-MIN-2) was proposed for the second step. This algorithm was shown to achieve higher effectiveness than the merge algorithm (Merge-1) proposed in [23]. Our experimental results show that the FQ method when used in conjunction with the new merge algorithm achieve high accuracies in all cases. Furthermore, the method does not require cooperation from various databases, in contrast to all other histogram construction methods [21]. Thus, the approach suggested here is very promising.

We believe that the proposed solution will be useful in large scale distributed systems (e.g., the Internet) where the same or similar goods and services are offered by numerous vendors and the users need to seek the best results among these offerings without exerting excessive burden on him/her and on the computing and communication resources. An example application can be the long term care insurance where there are about 120 companies selling such insurance and the specification of the desired care required by an individual/couple can be complicated (i.e., require top-N query processing).

Acknowledgment: This work is supported in part by the following grants: IIS-9902872, IIS-9902792, EIA-9911099, IIS-0208574, IIS-0208434 and ARO-2-5-30267. We are also grateful to Wensheng Jia who implemented some of the programs.

References

[1] N. Bruno, S. Chaudhuri and L. Gravano. *Performance of Multiattribute Top-k queries on Relational Systems.* Technical report, Columbia University, Computer Science Dept, 2000.

[2] N. Bruno, S. Chaudhuri and L. Gravano. *STHoles: A Multidimensional Workload-Aware Histogram.* ACM SIGMOD Conference, 2001.

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

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

[5] A. Deshpande, M. Garofalakis and R. Rastogi. *Independence is good: Dependency-Based Histogram Synopses for High Dimensional Data.* SIGMOD Conference, 2001.

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

[7] A. Fu, P. Chan, Cheung Y and Moon, Y. *Dynamic VP tree indexing for N -nearest neighbor search given pairwise distances* , VLDB Journal, 2000.

[8] L. Gravano, and Garcia-Molina, H. *Merging ranks from heterogeneous internet sources.* VLDB Conference, 1997.

[9] D. Gunopulos, G. Kollios, V.J. Tsotras and C. Domeniconi. *Approximating multi-dimensional aggregate range queries over real attributes.* ACM SIGMOD Conference, 2000.

[10] V. Hristidis, N. Koudas, Y. Papakonstantinou. PREFER: A System for the Efficient Execution of Multi-parametric Ranked Queries. ACM SIGMOD Conference, 2001.

[11] Y. Ioannidis and V. Poosala. *Histogram-based Approximation of Set-valued Query Answers*. VLDB Conference, 1999.

[12] W. Kiessling, G. Koestler. Preference SQL - Design, Implementation, Experiences. VLDB Conference, 2002.

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

[14] A. Kumar. G-Tree: A New Data Structure for Organizing Multidimensional Data. IEEE TKDE, 6(2), April 1994.

[15] K. Lam, M. Siu, and C. Yu. A Generalized Counter Scheme. J. of Theoretical Computer Science, Sept. 1981, pp. 271-278.

[16] A. Levy, A. Rajaraman, and J. Ordille. *Querying Heterogeneous Information Sources Using Source Descriptions*. VLDB Conference, 1996.

[17] L. Liu. *Query Routing in Large-scale Digital Library Systems*. 15th International Conference on Data Engineering (ICDE'99), March 1999.

[18] M. Muralikrishna and D. J. DeWitt. *Equi-Depth Histograms For Estimating Selectivity Factors For Multi-Dimensional Queries*. ACM SIGMOD Conference, 1988.

[19] V. Poosala and Y. Ioannidis. *Selectivity Estimation Without the Attribute Value Independence assumption*. VLDB Conference, 1997.

[20] R. Rivest. On Self-Organizing Sequential Search Heuristics. CACM 19(2): 63-67, 1976.

[21] N. Thaper, S. Guha, P. Indyk, and N. Koudas. Dynamic Multidimensional Histogram. ACM SIGMOD Conference, 2002.

[22] C. Yu, W. Meng, W. Wu, K. Liu. *Efficient and Effective Metasearch for Text Databases Incorporating Linkages among Documents*. ACM SIGMOD Conference, May 2001.

[23] C. Yu, P. Sharma, W. Meng and Y. Qin. *Databases Selection for Processing k Nearest Neighbors Queries in Distributed Environments*. 1st ACM/IEEE-CS joint conf. on DL, 2001.