On k-Nearest Neighbor Searching in Non-Ordered Discrete Data Spaces*

Dashiell Kolbe
Michigan State University
East Lansing, MI, USA
kolbedas@msu.edu

Qiang Zhu
The University of Michigan
Dearborn, MI, USA
qzhu@umich.edu

Sakti Pramanik
Michigan State University
East Lansing, MI, USA
pramanik@cse.msu.edu

Abstract

A k-nearest neighbor (k-NN) query retrieves k objects from a database that are considered to be the closest to a given query point. Numerous techniques have been proposed in the past for supporting efficient k-NN searches in continuous data spaces. No such work has been reported in the literature for k-NN searches in a non-ordered discrete data space (NDDS). Performing k-NN searches in an NDDS raises new challenges. The Hamming distance is usually used to measure the distance between two vectors (objects) in an NDDS. Due to the coarse granularity of the Hamming distance, a k-NN query in an NDDS may lead to a large set of candidate solutions, creating a high degree of non-determinism for the query result. We propose a new distance measure, called Granularity-Enhanced Hamming (GEH) distance, that effectively reduces the number of candidate solutions for a query. We have also considered using multidimensional database indexing for implementing k-NN searches in NDDSs. Our experiments on synthetic and genomic data sets demonstrate that our index-based k-NN algorithm is effective and efficient in finding k-NNs in NDDSs.

1 Introduction

There is an increasing demand for similarity searches in applications such as geographical information systems [16], multimedia databases [17], molecular biology [1], and genome sequence databases [11, 14]. There are two types of similarity searches: range queries and k-nearest neighbor (k-NN) queries. The former is to find data objects that are within a tolerant distance from a given query point/object, while the latter is to retrieve k-nearest neighbors to the query point. An example of a range query is "find the words in a document that differ from word 'near' by at most two letters". An example of a k-NN query is "find two garages that are closest to the current one".

Numerous techniques have been proposed in the literature to support efficient similarity searches in (ordered) continuous data spaces (CDS). A majority of them utilize a multidimensional index structure such as the R-tree [6], the R*-tree [2], the X-tree [3], the K-D-B tree [15], and the LSD^h-tree [8]. These techniques rely on some essential geometric properties/concepts such as bounding rectangles in CDSs.

Little work has been reported on supporting efficient similarity searches in so-called non-ordered discrete data spaces (NDDS). A d-dimensional NDDS is a Cartesian product of d domains/alphabets consisting of non-ordered elements/letters. For example, when searching genome DNA sequences, consisting of letters 'a', 'g', 't', 'c', each sequence is often divided into intervals/strings of a fixed length d. These intervals can be considered as vectors from a d-dimensional NDDS with alphabet $\{a, g, t, c\}$ for dimensions. Other examples of non-ordered discrete dimensions are color, gender and profession. Application areas that demand similarity searches in NDDSs include bioinformatics, E-commerce, biometrics and data mining.

To support efficient similarity searches in NDDSs, Gian et al. [11, 12, 13] recently proposed two indexing methods: the ND-tree and the NSP-tree. These indexing techniques were designed specifically for NDDSs. It has been shown that these techniques outperform the linear scan and typical metric trees such as the M-tree [5] for range queries in NDDSs. The reason why metric trees do not perform well in NDDSs is that they are too generic and do not take special characteristics of an NDDS into consideration. However, Gian et al.'s work in [11, 12, 13] mainly focused on handling range

*This work was supported by the National Science Foundation (under grants #II5-0414576 and #II5-0414594), Michigan State University and The University of Michigan.

1-4244-0803-2/07/$20.00 ©2007 IEEE.

queries. Although a procedure for finding the nearest neighbor (i.e., 1-NN) to a query point was outlined in [12], no empirical evaluation was given. Furthermore, the general k-NN searching in NDDSs is still an open issue.

The issue of k-NN searching in NDDSs is in fact not a trivial extension of earlier work. NDDSs raise new challenges for this problem. First, we observe that, unlike a k-NN query in a CDS, a k-NN query in an NDDS, based on the conventional Hamming distance [7], often has a large number of alternative sets/solutions, making the semantic of the k-NN query unclear. This non-determinism is mainly due to the coarse granularity of the Hamming distance. Second, existing index-based k-NN searching algorithms for CDSs cannot be directly applied to an NDDS due to lack of relevant geometric concepts/measures. On the other hand, the algorithms using metric trees are suboptimal because of their generic nature and ignorance of special characteristics of an NDDS.

To tackle the first challenge, we introduce a new extended Hamming distance, called the Granularity-Enhanced Hamming (GEH) distance. To solve the second issue, we propose a k-NN searching algorithm utilizing the ND-tree [11, 12] and employing searching heuristics based on the new distance and some characteristics of an NDDS. Our experiments demonstrate that the new GEH distance provides a greatly improved semantic discriminating power that is needed for k-NN searching in NDDSs, and that our searching algorithm is very efficient in supporting k-NN searches in NDDSs.

The issue of k-NN searching in CDSs and general metric spaces has been discussed extensively in the literature. Much work has centered around a filter and refinement process. Roussopoulos et al [16] presented a branch-and-bound algorithm for finding k-NNs to a query point. Korn et al. [10] furthered this work by presenting a multi-step k-NN searching algorithm, which was then optimized by Seidl and Kriegel [18]. In [9], a Voronoi based approach was presented to address the k-NN searching in spatial network databases. There are many other proposed k-NN searching techniques, including those utilizing metric trees [4, 5, 20]. However, to our knowledge, no work has been reported on k-NN searching in NDDSs. Our algorithm extends Roussopoulos et al.'s work in [16] to NDDSs by introducing suitable pruning metrics and relevant searching heuristics based on our new distance measure and the characteristics of NDDSs.

The rest of this paper is organized as follows. Section 2 formally defines the problem of k-NN searching, introduces the new GEH distance in NDDSs, and discusses its properties. Section 3 presents our index-based k-NN searching algorithm for NDDSs, including its pruning metrics and heuristics. Section 4 discusses experimental results. Section 5 summarizes our conclusions and gives some future research directions.

2 *k*-Nearest Neighbors in NDDS

To study the issue of k-NN searching in NDDSs, we need to introduce essential concepts, identify relevant problems and develop solutions to the problems. In this section, we first review some concepts related to NDDSs, including the ND-tree that our searching algorithm is based on. We then formally define a k-NN search and identify a major problem associated with k-NN searches in NDDSs. To overcome the problem, we introduce a new extended Hamming distance and discuss its properties.

2.1 The ND-Tree

The ND-tree has some similarities, in structure and function, to the R-tree [6] and its variants (the R*-tree [2] in particular). As such, the ND-tree is a balanced tree with leaf nodes containing the indexed vectors. The vectors can be reached by traversing a set of branches starting at the root and becoming more refined as one traverses towards the leaves.

The ND-tree is built using similar methods to those for the R-tree. Each vector is inserted into the tree after an appropriate position is found in the tree and the relevant minimum bounding rectangles may need to be split to accommodate the insertion.

A key difference between the ND-tree and its continuous cousins is the way in which a minimum bounding rectangle is defined and utilized. In the ND-tree, minimum bounding rectangles are discrete. A *discrete minimum bounding rectangle* (DMBR) for a set $G = \{R_1, R_2, \ldots, R_n\}$ of discrete rectangles $R_i = S_{i1} \times S_{i2} \times \ldots \times S_{id}$ $(1 \le i \le n)$ is defined as follows:

$$DMBR(G) = (\cup_{i=1}^{n} S_{i1}) \times (\cup_{i=1}^{n} S_{i2}) \times \ldots \times (\cup_{i=1}^{n} S_{id}) ,$$

where S_{ij} $(1 \le j \le d)$ is a set of elements/letters from the j-th dimension of the given d-dimensional NDDS. Such a DMBR allows the ND-tree to utilize a non-Euclidean method of measurement for calculating the distance between a vector $\alpha = (\alpha[1], \alpha[2], \ldots, \alpha[d])$ and a DMBR $R = S_1 \times S_2 \times \ldots \times S_d$:

$$Dist(\alpha, R) = \sum_{i=1}^{d} \left\{ \begin{array}{ll} 0 & \text{if } \alpha[i] \in S_i \\ 1 & \text{otherwise} \end{array} \right\} .$$

1-4244-0803-2/07/$20.00 ©2007 IEEE.

This distance can be interpreted as saying that, for each dimension in query vector α, if the element represented therein occurs anywhere in the subtrees covered by DMBR R, then nothing will be added to the current distance, otherwise a one will be added.

The ND-tree was successfully utilized for supporting range queries [11, 12]. In this paper, its applicability of supporting k-NN searches in NDDSs is studied.

2.2 Definition of k-Nearest Neighbors in NDDS

In general, the set/solution of k-nearest neighbors for a given query may not be unique because more than one neighbor can have the same distance from the query point (vector). We define a candidate set/solution of k-nearest neighbors for a query point q as follows:

Definition 1 (Candidate Set of k-Nearest Neighbors): *Let the universe of discourse for variables A_i and B_i ($1 \leq i \leq k$) be the set of all objects in the database. Let k-NNS denote a candidate set/solution of k-nearest neighbors in the database for a query point q and $D(x, y)$ denote the distance between objects x and y. Then k-NNS is defined as follows:*

$$k\text{-}NNS \in \{ \{A_1, A_2, ..., A_k\} : \forall_{B_1,B_2,...,B_k}[\sum_{i=1}^{k} D(q, A_i) \leq \sum_{j=1}^{k} D(q, B_j)] \wedge \forall_{m,n\in\{1,2,...,k\}}[(m \neq n) \rightarrow (A_m \neq A_n)] \} . \quad (1)$$

Formula (1) essentially says that k objects/neighbors A_1, A_2, ..., A_k in k-*NNS* have the minimum total distance to q. This definition is in fact valid for both continuous and discrete data spaces. Since k-*NNS* is a set of neighbors, there is no ordering implied among the neighbors. In the following recursive definition we provide a procedural semantic of a candidate k^{th}-nearest neighbor, which is based on an ordered ranking of the neighbors in the database for a query point q.

Definition 2 (Candidate k^{th}-Nearest Neighbor): *Let the universe of discourse for variables A and B be the set of all objects in the database. Let A_k denote a candidate k^{th}-nearest neighbor in the database for a query point q. We recursively define A_k as follows:*

$$A_1 \in \{A : \ \forall_B(D(q, A) \leq D(q, B))\} ,$$
$$A_k \in \{A : \ \forall_B(D(q, A) \leq D(q, B) \wedge B \notin \{A_1, A_2, ..., A_{k-1}\} \wedge A \notin \{A_1, A_2, ..., A_{k-1}\})\} , \quad for\ k \geq 2 .$$

Definition 2 can be used to produce all the candidate k-*NNS*s given by Definition 1, as stated in the following proposition.

Proposition 1 *All the candidate k-NNSs given by Definition 1 can be produced by Definition 2.*

PROOF. Omitted. ∎

From the above definitions, we can see that there are multiple possible k-*NNS*s for a given query. Therefore, k-*NNS* is generally not unique. The non-uniqueness of k-*NNS* has an impact on the semantics of the k-nearest neighbors. We define the degree of non-uniqueness of k-nearest neighbors by the number of possible k-*NNS*s that exist for the query. This degree of non-uniqueness is computed by the following proposition.

Proposition 2 *The number Δk of candidate k-NNSs is given by:*

$$\Delta k = N!/(t!(N - t)!) , \quad (2)$$

where t is defined by $D(q, A_{k-t}) \neq D(q, A_{k-t+1}) = D(q, A_{k-i+2}) = ... = D(q, A_k)$; A_j ($1 \leq j \leq k$) denotes the jth-nearest neighbor; N is the number of nearest neighbors in the database that have the same distance as $D(q, A_k)$.

PROOF. Omitted. ∎

Note that t denotes the number of objects with distance $D(q, A_k)$ that have to be included in a k-*NNS*. If $t = k$, all the neighbors in a k-*NNS* are of the same distance as $D(q, A_k)$; that is, A_0 is a dummy object in this case. The values of N and t depend on parameters such as the dimensionality, database size, data distribution in the database, and query point.

For a k-NN query on a database in a continuous data space based on the Euclidean distance, k-*NNS* is typically unique (i.e., $\Delta k = 1$) since the chance for two objects having the same distance to the query point is usually very small. As a result, the non-uniqueness is usually not an issue for k-NN searches in continuous data spaces.

However, non-uniqueness is a common occurrence in an NDDS. As pointed out in [11, 12], the Hamming distance is typically used for NDDSs. Due to the insufficient semantic discrimination between objects provided by the Hamming distance and the limited number of elements available for each dimension in an NDDS, Δk for a k-NN query in an NDDS is usually large. For example, for a data set of 2M vectors in a 10-dimensional NDDS under the uniform distribution, the average Δk values for 100 random k-NN queries with k=1, 5, 10 are about

8.0, 19.0K, 45.4M, respectively (see Figure 1 in Section 4). This demonstrates a high degree of non-determinism for k-NN searches based on the Hamming distance in an NDDS, especially for large k values.

To mitigate the problem, we extend the Hamming distance to provide more semantic discrimination between the neighbors of a k-NN query point in an NDDS.

2.3 A New Extended Hamming Distance

The Hamming distance [7] between two vectors $\alpha = (\alpha[1], \alpha[2], \ldots, \alpha[d])$ and $\beta = (\beta[1], \beta[2], \ldots, \beta[d])$ is defined as follows:

$$D_{Hamming}(\alpha, \beta) = \sum_{i=1}^{d} \left\{ \begin{array}{ll} 1 & \text{if } \alpha[i] \neq \beta[i] \\ 0 & \text{otherwise} \end{array} \right\} .$$

Intuitively, the Hamming distance indicates the number of dimensions on which the corresponding components of α and β differ.

Although the Hamming distance is very useful for exact matches and range queries in NDDSs, it does not provide an effective semantic for k-NN queries in NDDSs due to the high degree of non-determinism, as mentioned previously. We notice that the Hamming distance does not distinguish equalities for different elements. For example, it treats '$a = a$' as the same as '$b = b$' by assigning 0 to the distance measure in both cases. In many applications such as genome sequence searches, some matches (equalities) may be considered to be more important than others. Based on this observation, we extend the Hamming distance to capture the semantics of different equalities in the distance measure.

Several constraints have to be considered for such an extension. First, the extended distance should enhance the granularity level of the Hamming distance so that its semantic discriminating power is increased. Second, the semantic of the traditional Hamming distance needs to be preserved. For example, from a given distance value, one should be able to tell how many dimensions are distinct (and how many dimensions are equal) between two vectors. Third, the extended distance should possess a triangular property so that pruning during an index-based search is possible.

We observe that matching two vectors on a dimension with a frequently-occurred element is usually more important than matching the two vectors on the dimension with an uncommon (infrequent) element. Based on this observation, we utilize the frequencies of the elements to extend the Hamming distance as follows:

$$D_{GEH}(\alpha, \beta) = \sum_{i=1}^{d} \left\{ \begin{array}{ll} 1 & \text{if } \alpha[i] \neq \beta[i] \\ \frac{1}{d} f(\alpha[i]) & \text{otherwise} \end{array} \right\}, \quad (3)$$

$$where \quad f(\alpha[i]) = 1 - frequency(\alpha[i]) .$$

This extension starts with the traditional Hamming distance by adding one to the distance measure for each dimension that does not match between the two vectors. The difference is that, when the two vectors match on a particular dimension, the frequency of the common element (i.e., $\alpha[i] = \beta[i]$) occurring in the underlying database on the dimension is obtained from a lookup table. This frequency value is then subtracted from one and then added to the distance measure. Thus, the more frequently an element occurs, the more it will subtract from one and thus the less it will add to the distance, thereby indicating that the two vectors are closer than if they had matched on a very uncommon element.

The factor of $\frac{1}{d}$ is used to ensure that the frequency based adjustments to the distance measurement do not end up becoming more significant than the original Hamming distance. This is a key factor in maintaining the triangular property.

From the distance definition, we can see that, if $m \leq D_{GEH}(\alpha, \beta) < m + 1$ ($m = 0, 1, ..., d$), then vectors α and β mis-match on m dimensions (i.e., match on $d - m$ dimensions). Within each interval, the smaller the distance value, the larger the frequency(ies) of the element(s) shared by α and β on the matching dimension(s). Clearly, unlike the traditional Hamming distance, which has at most d (integer) values — resulting in a quite coarse granularity, this new extended distance allows many more possible values — leading to a refined granularity. We call this extended Hamming distance the *Granularity-Enhanced Hamming* (*GEH*) distance. Due to its enhanced granularity, the GEH distance can dramatically reduce Δk in Proposition 2, leading to more deterministic k-NN searches in NDDSs. For example, for the forementioned data set of 2M vectors in a 10-dimensional NDDS under the uniform distribution, the average Δk values for 100 random k-NN queries with k=1, 5, 10 are about 1.09, 1.11, 1.06, respectively (see Figure 1 in Section 4).

In fact, the Euclidean distance measurement can be considered to have the finest (continuous) granularity at one end, while the Hamming distance measurement has a very coarse (discrete integers) granularity at the other end. The GEH distance measurement provides an advantage in bringing discrete and continuous distance measurements closer to each other.

2.4 Triangular Property of the GEH Distance

An efficient indexing method is vital to the performance of k-NN searches. A key to achieving high

searching efficiency via an index tree is its ability to prune useless subtrees. However, pruning requires the underlying distance measure to satisfy the triangular property. The following proposition claims that the above GEH distance possesses this crucial property.

Proposition 3 (Triangular Property): *The GEH distance possesses the triangular property. In other words, for any three vectors V_A, V_B, and V_C, the addition of the distances of any two pairs is greater than or equal to the distance of the third pair, namely:*

$$D_{GEH}(V_A, V_B) + D_{GEH}(V_B, V_C) \geq D_{GEH}(V_A, V_C) .$$

PROOF. The proof of this proposition is divided into two steps: first, a base case is established where the property holds. Second, the base case is evolved into a generic case where the property still holds.

Step 1: Assume that $D_{GEH}(V_A, V_C)$ is maximum. Thus every pair of corresponding elements in V_A and V_C must be different. From equation (3), we have:

$$D_{GEH}(V_A, V_C) = d ,$$

where d is the number of dimensions of the underlying NDDS. Now assume that $D_{GEH}(V_A, V_B)$ is minimal. Thus every element in V_B must equal the corresponding element in V_A, i.e., $V_A = V_B$. From equation (3), we have:

$$D_{GEH}(V_A, V_B) = x, \text{ where } 0 \leq x < 1 .$$

The term x represents the summation of all the frequency values obtained using equation (3) and then dividing by d. Since $V_A = V_B$ and $D_{GEH}(V_A, V_C) = d$, we have:

$$D_{GEH}(V_B, V_C) = d .$$

Since $x + d \geq d$, we have the following inequality;

$$D_{GEH}(V_A, V_B) + D_{GEH}(V_B, V_C) \geq D_{GEH}(V_A, V_C) .$$

Step 2: The second step involves three sub-steps; one for each vector that needs to become generic.

Step 2.1: The first step is to evolve V_B into a generic vector, starting with the initial vectors from Step 1: $D_{GEH}(V_A, V_B) = x$, where $0 \leq x < 1$; $D_{GEH}(V_B, V_C) = d$; $D_{GEH}(V_A, V_C) = d$.

To make V_B generic, we apply n changes. Each change is done by switching an element in V_B away from its original element. After these n changes have been done, you are left with the following distances:

$$D_{GEH}(V_A, V_B) = x + n - c_1 ,$$
$$D_{GEH}(V_B, V_C) = d - n_1^* + c_2 ,$$
$$D_{GEH}(V_A, V_C) = d ,$$

where c_1 represents the culmination of adjustment values from each of the n elements switched; n_1^* represents the number of elements switched that now equal their corresponding element in V_C; c_2 represents the culmination of the adjustment values to be added due to these newly matching elements.

Here, we know that $n \geq n_1^*$, $x \geq c_1$, and $c_2 \geq 0$. Using these inequalities it can be shown that the distance measures still satisfy the triangular property.

Step 2.2: The next step is to evolve V_C into a generic vector, starting with the final vectors from Step 2.1 and applying j changes to V_C. This leaves you with the following distances:

$$D_{GEH}(V_A, V_B) = x + n - c_1 ,$$
$$D_{GEH}(V_B, V_C) = d - n_1^* + c_2 + (j_1^* - c_3) - (j_2^* - c_4),$$
$$D_{GEH}(V_A, V_C) = d - j_3^* + c_5 ,$$

where j_1^* and j_2^* represent the integer values that $D_{GEH}(V_B, V_C)$ increases and decreases, respectively, as elements are switched; c_3 and c_4 represent the adjustment values due to those changes; j_3^* and c_5 represent the integer and adjustment changes to $D_{GEH}(V_A, V_C)$, respectively, due to the element changes. Note that every time j_2^* is incremented there are two possibilities: either the element being switched in V_C becomes an element in V_B that still matches V_A, in which case j_3^* is incremented by one and both c_4 and c_5 are incremented by the same amount; or it becomes an element in V_B that does not match V_A, which means that $n \geq n_1^* - 1$ (note that, if this situation arises twice, $n \geq n_1^* - 2$, and so on), and only c_4 is incremented. This leaves us to note that $j_2^* + n_1^* \leq j_3^* + n$ and that $c_4 \geq c_5$. Finally, with $j_1^* \geq c_3$, it can be shown that these distance measures still satisfy the triangular property.

Step 2.3: The final step is to evolve V_A into a generic vector. This is actually a trivial step, because V_A was only defined in relation to the original vectors V_C and V_B, and because V_C and V_B can be transformed into any general vectors from their starting points, we can start V_A as any vector we wish. Thus V_A is a generic vector and the triangular inequality holds true for any three vectors V_A, V_B, and V_C. ∎

3 A *k*-NN Algorithm for NDDS

To efficiently process k-NN queries in NDDSs, we introduce an index-based k-NN searching algorithm. This algorithm utilizes the ND-tree recently proposed by Qian, et al. [11, 12] for NDDSs. The basic idea of this algorithm is as follows. It descends the ND-tree from

the root following a depth-first search strategy. When k possible neighbors are retrieved, the searching algorithm uses the distance information about the neighbors already collected to start pruning search paths that can be proven to not include any vectors that are closer to the query vector than any of the current neighbors. The details of this algorithm are discussed in the following subsections.

3.1 Heuristics

In the worst case scenario, the search would encompass the entire tree structure. However, our extensive experiments have shown that the use of the following heuristics is able to eliminate most search paths before they need be traversed.

MINDIST Pruning: Similar to [16], we utilize the minimum distance (MINDIST) between a query vector q and a DMBR $R = S_1 \times S_2 \times \ldots \times S_d$, denoted by $mdist(q, R)$, to prune useless paths. Based on the GEH distance, MINDIST is formally defined as follows:

$$mdist(q, R) = \sum_{i=1}^{d} \left\{ \begin{array}{ll} 1 & \text{if } q[i] \notin S_i \\ \frac{1}{d} f(q[i]) & \text{otherwise} \end{array} \right\} ,$$

$$f(q[i]) = 1 - frequency(q[i]) . \quad (4)$$

This calculation is then used in conjuncture with the $Range$ of the current k-nearest neighbors (with respect to the query vector) to prune subtrees. Specifically, the heuristic for pruning subtrees is:

H_1: If $mdist(q, R) \geq Range$, then prune the subtree associated with R .

By taking the closest distance between a DMBR and q, we are guaranteeing that no vectors that are included in the DMBR's subtree are closer than the current $Range$ and thus need not be included in the continuing search.

MINMAXDIST Pruning: We also utilize the minimum value (MINMAXDIST) of all the maximum distances between a query vector q and a DMBR R along each dimension, denoted by $mmdist(q, R)$, for pruning useless paths. In simple terms, $mmdist(q, R)$ represents the shortest distance from vector q that can guarantee a vector in R/subtree can be found. For vector q and DMBR $R = S_1 \times S_2 \times \ldots \times S_d$, MINMAXDIST is formally defined as follows:

$$mmdist(q, R) = \min_{1 \leq k \leq d} \{ f_m(q[k], S_k) + \sum_{i=1, i \neq k}^{d} f_M(V[i], S_i) \} ,$$

$$f_m(q[k], S_k) = \left\{ \begin{array}{ll} \frac{1}{d} f(q[k]) & \text{if } q[k] \in S_k \\ 1 & \text{otherwise} \end{array} \right\} ,$$

$$f_M(q[i], S_i) = \left\{ \begin{array}{ll} \frac{1}{d} f(q[j]) & \text{if } \{q[i]\} = S_i \\ 1 & \text{otherwise} \end{array} \right\} ,$$

where $f()$ on the right hand side of the last two formulas is defined in equation (4).

To process k-NN searches in our algorithm, $mmdist()$ is calculated for each non-leaf node of the ND-tree using query vector q and all the DMBRs (for subtrees) contained in the current node. Once each of these MINMAXDIST values (for subtrees) have been calculated, they are sorted in the ascending order and the k-th value is selected as $MINMAXDIST_k$ for the current node.

The k-th MINMAXDIST value is selected to guarantee that *at least k* vectors will be found in searching the current node. This selected $MINMAXDIST_k$ is then used in the following heuristic:

H_2: If $MINMAXDIST_k(node) < Range$, then let Range = $MINMAXDIST_k(node)$.

Search Ordering: For those subtrees that are not pruned by heuristic H_1 or H_2, we need to decide an order to access them. Two search orders were suggested in [16]: one is based on the ordering of MINDIST values, and the other is based on the ordering of MINMAXDIST values. For the MINDIST ordering, it optimistically assumes that a vector with a distance of the MINDIST value exists in the relevant DMBR, which is typically not the case in an NDDS. The set of elements on each dimension from different vectors often yields a combination that is not an indexed vector in the DMBR. On the other hand, the MINMAXDIST ordering is too pessimistic to be practically useful. Accessing the subtrees based on such an ordering is almost the same as a random access in NDDSs.

These observations led us to seek another search ordering for NDDSs. For a query vector q and the DMBR $R = S_1 \times S_2 \times \ldots \times S_d$ of a subtree T, we call the i-th dimension ($1 \leq i \leq d$) is a promising one of R (or T) with respect to q if $q[i] \in S_i$. In such a case, $q[i]$ is called a promising element of R (or T). The frequency of $q[i]$ occurring in the indexed vectors in T on the i-th dimension is called the tree frequency $tree_freq(q[i])$ of $q[i]$ in T. Note that $tree_freq(q[i])$ is different from $frequency(q[i])$ in equation (4) (or in (3)). $frequency(q[i])$ is the frequency of $q[i]$ occurring in the entire database on the i-th dimension, which represents a property for the underlying data set rather than for a specific subtree. From an extensive empirical study, we found that the following heuristic to access

subtrees during a k-NN search in an NDDS provided the most promising results:

H_3: Access subtrees in the descending order of the number N of promising dimensions of each subtree and access subtrees with the same N in the descending order of the total tree frequency of promising elements of each subtree.

This heuristic essentially gives a higher priority to a subtree that may contain an indexed vector matching the query vector on more dimensions.

3.2 Algorithm Description

Our k-NN algorithm adopts a depth first traversal of the ND-tree and applies the forementioned heuristics to prune non-productive subtrees and determine the access order of the subtrees. The description of the algorithm is given in the following subsections.

3.2.1. *k*-NN Query Algorithm

Given an ND-tree with root node T, the algorithm finds a set of k-nearest neighbors to query vector q that satisfies condition (1) in Definition 1. It invokes two functions: $ChooseSubtree$ and $RetrieveNeighbors$. The former chooses a subtree of a non-leaf node to descend, while the latter updates a list of k-nearest neighbors using vectors in a leaf node.

Algorithm 1 : k-**NNQuery**
Input: (1) query vector q; (2) the desired number k of nearest neighbors; (3) an ND-tree with root node T for a given database.
Output: a set k-NNS of k-nearest neighbors to query vector q.

1. let k-$NNS = \emptyset$, $N = T$, $Range = \infty$;
2. **while** $N \neq NULL$ **do**
3.   **if** N is a non-leaf node **then**
4. $[NN, Range] = ChooseSubtree(N, q, k, Range)$;
         // NN is the root of the chosen subtree
5.   **if** $NN \neq NULL$ **then**
6. $N = NN$;
7.   **else** $N = N.Parent$; **end if**;
8.   **else**
9. $[k\text{-}NNS, Range] =$
           $RetrieveNeighbors(N, q, k, Range, k\text{-}NNS)$;
10. $N = N.Parent$;
11.   **end if**;
12. **end while**;
13. **return** k-NNS.

In the algorithm, step 1 initializes relevant variables. Steps 2 - 12 traverse the ND-tree. Steps 3 - 7 deal with non-leaf nodes by either invoking $ChooseSubtree$ to decide a descending path or backtracking to the ancestors when no more subtree to explore. Steps 8 - 11 deal with leaf nodes by invoking $RetrieveNeighbors$ to update the list of current k-nearest neighbors. Step 13 returns the result. Note that $ChooseSubtree$ not only returns a chosen subtree but also may update $Range$ using heuristic H_2. If no more subtree to choose, it returns $NULL$ for NN at step 4. Similarly, $RetrieveNeighbors$ not only updates k-NNS but also may update $Range$ if a closer neighbor(s) is found.

3.2.2. Function $ChooseSubtree$

The effective use of pruning is the most efficient way to reduce the I/O cost for finding a set of k-nearest neighbors. To this end, the three heuristics discussed in Section 3.1 are employed in function $ChooseSubtree$.

Function 1 $ChooseSubtree(N, q, k, Range)$

1. **if** list L for active subtrees of N not exist **then**
2.   use heuristic H_2 to update $Range$;
3.   use heuristic H_1 to prune subtrees of N;
4.   use heuristic H_3 to sort the remaining (active) subtrees and
         create a list L to hold them;
5. **else**
6.   use heuristic H_1 to prune subtrees from list L;
7. **end if**;
8. **if** $L \neq \emptyset$ **then**
9.   remove the 1st subtree NN from L;
10.   **return** $[NN, Range]$;
11. **else**
12.   **return** $[NULL, Range]$;
13. **end if**.

In the above function, steps 1 - 4 handle the case in which the non-leaf is first time visited. In this case, the function applies heuristics H_1 - H_3 to update $Range$, prune useless subtrees, and order active subtrees (their root nodes) in a list. Step 6 applies heuristic H_1 and current $Range$ to prune useless subtrees for a non-leaf node that was visited before. Steps 8 - 12 return a chosen subtree (if any) and the updated $Range$.

3.2.3. Function $RetreiveNeighbors$

The main task of $RetrieveNeighbor$ is to examine the vectors in a given leaf node and update the current k-nearest neighbors and $Range$.

Function 2 $RetrieveNeighbors(N, q, k, Range, k\text{-}NNS)$

1. **for** each vector v in N **do**
2.   **if** $D_{GEH}(q, v) < Range$ **then**
3. k-$NNS = k$-$NNS \cup \{v\}$;
4.   **if** $|k\text{-}NNS| > k$ **then**
5.   remove vector v' from k-NNS such that
           v' has the largest $D_{GEH}(q, v')$ in k-NNS;
6. $Range = D_{GEH}(q, v'')$ such that v'' has the
           largest $D_{GEH}(q, v'')$ in k-NNS;
7.   **end if**;
8.   **end if**;
9. **end for**;
10. **return** $[k\text{-}NNS, Range]$.

A vector is collected in k-NNS only if its distance to the query vector is smaller than current $Range$ (steps 2 - 3). A vector has to be removed from k-NNS if it has more than k neighbors after a new one is added (steps 4 - 7). The vector to be removed has the largest distance to the query vector. If there is a tie, a random furthest vector is chosen.

4 Experimental Results

To evaluate the effectiveness of our GEH distance and the efficiency of our k-NN searching algorithm, we conducted extensive experiments. The experimental results are presented in this section. Our k-NN searching algorithm was implemented in the C++ programming language. For the comparison purpose, we also implemented the algorithm using the original Hamming distance. All experiments were ran on a PC under OS Windows XP. The I/O block size was set at $4K$ bytes. Both synthetic and genomic data sets were used in our experiments. The synthetic data sets consist of uniformly distributed 10-dimensional vectors with values in each dimension of a vector drawn from an alphabet of size 6. The genomic data sets were created from the Ecoli DNA data (with alphabet: $\{a, g, t, c\}$) extracted from the GenBank. Each experimental data reported here is the average over 100 random queries.

4.1 Effectiveness of GEH Distance

The first set of experiments was conducted to show the effectiveness of the GEH distance over the Hamming distance, by comparing their values of Δk as defined in Proposition 2 of Section 2.2.

Figure 1 gives the relationship between Δk and database sizes for both the GEH and Hamming distances, when k=1, 5 and 10. The figure shows a significant decrease in the values of Δk using the GEH distance over those using the Hamming distance. This significant improvement in performance for the GEH distance is observed for all the database sizes and k values considered. Figure 1 shows that when the GEH distance is used, Δk values are very close to 1, indicating a promising behavior close to that in CDSs.

4.2 Efficiency of k-NN Algorithm

One set of experiments was conducted to examine the effects of our three heuristics, presented in Section 3.1, on the performance of our k-NN searching algorithm presented in Section 3.2. We considered the following three versions of our pruning strategies in the experiments.

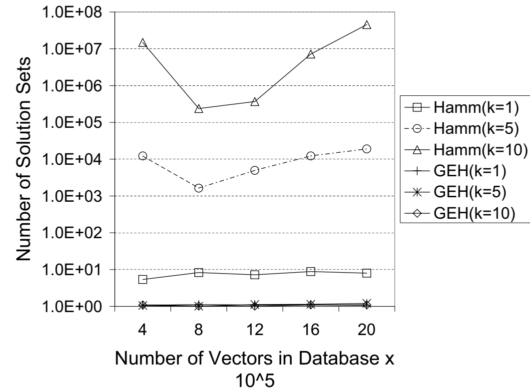

Figure 1. Comparison of Δk values for the GEH and Hamming distances

Version $V1$: only heuristic H_1 is used.
Version $V2$: both heuristics H_2 and H_1 are used.
Version $V3$: all three heuristics H_1 - H_3 are used.

Figure 2 shows that $V2$ provides a little improvement in the number of disk accesses over $V1$. However, $V2$ does make good performance improvements over $V1$ for some of the queries (note that the performance improvement presented is an average over 100 queries). Thus, we have included H_2 in version $V3$. As seen from the figure, $V3$ provides the best performance improvement among the three versions for all database sizes tested. Hence $V3$ is adopted in our k-NN searching algorithm and used in all the remaining experiments.

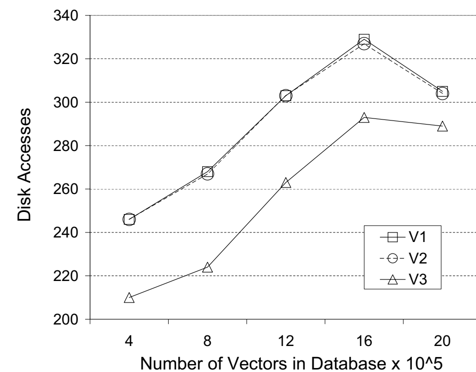

Figure 2. Effects of heuristics in the k-NN algorithm

Another set of experiments was conducted to compare the disk I/O performance of our index-based k-NN searching algorithm with that of the linear scan for databases of various sizes. Figure 3 shows the performance comparison for the two methods on synthetic data sets. From the figure, we can see a significant reduction in the number of disk accesses for our k-NN searching algorithm over the linear scan. In fact, this performance improvement gets increasingly larger as the

database size increases. Figure 4 shows the performance comparison for the two methods on genomic data sets. Since a genome sequence is typically divided into intervals of length 11 for searching, genomic data sets of 11-dimensional vectors were considered for the experiments. This figure demonstrates that our k-NN searching algorithm is superior to the linear scan in performance on real-world genomic data sets as well. In fact, our method performs better on the genomic data sets than on the synthetic data sets. The main reason for this phenomenon is probably due to the difference in data distributions for the data sets.

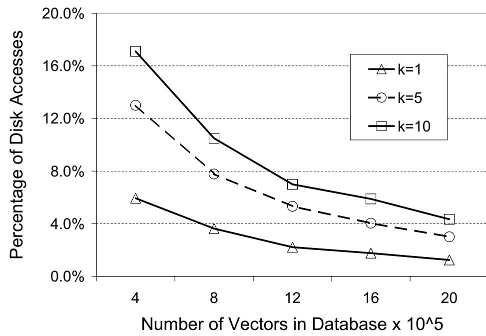

Figure 3. Performance of the *k*-NN algorithm vs. the linear scan on synthetic data sets with various sizes

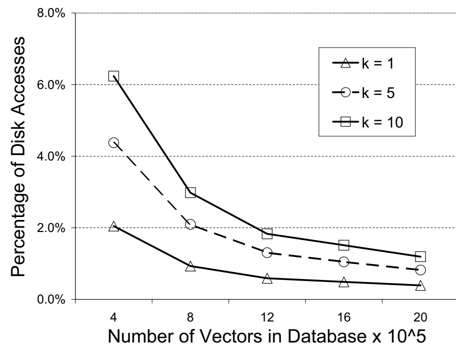

Figure 4. Performance of the *k*-NN algorithm vs. the linear scan on genomic data sets with various sizes (*k*=10)

To examine the effect of dimensionality on the performance of our *k*-NN searching algorithm, we ran random k-NN queries (with $k = 10$) on a series of genomic data sets with various numbers of dimensions, each of which contains 1 million vectors. As seen from Figure 5, the performance gain of our algorithm over the linear scan is quite significant for lower dimensions. However, the amount of this improvement decreases with an increasing number of dimensions. This phenomenon of deteriorating performance with an increasing number of dimensions is also true in continuous data spaces due to the well-known dimensionality curse problem. Since the performance gain of our searching algorithm increases with the database size, our algorithm is expected to perform well for more dimensions when the database size is larger.

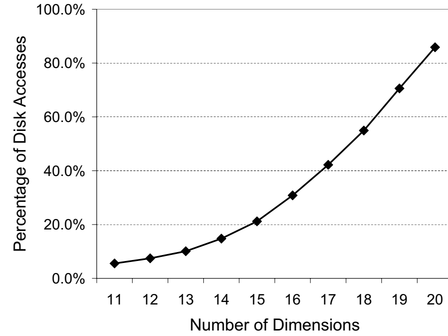

Figure 5. Performance of the *k*-NN algorithm vs. the linear scan on genomic data sets with various dimensions (*k*=10)

Additionally, we have compared the disk I/O performance of the k-NN searching algorithm using the GEH distance with that of the k-NN searching algorithm using the Hamming distance. Figure 6 shows the percentage I/Os for the GEH distance versus the Hamming distance for various database sizes and k values. From the figure, we can see that the number of disk accesses is significantly reduced for all test cases when the GEH distance is used as opposed to the Hamming distance. We feel that this is due to an increase in the pruning power of heuristics H_1 and H_2 using the GEH distance. These results indicate that the use of the GEH distance will cost less in disk accesses while providing a far more deterministic result than that using the Hamming distance for a *k*-NN query.

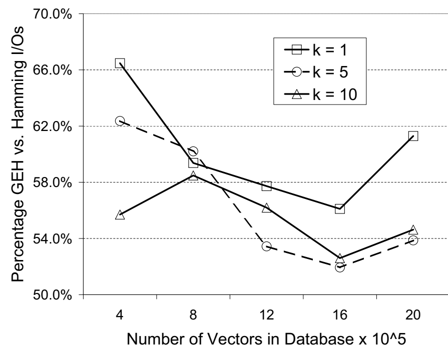

Figure 6. Performance comparison for the GEH and Hamming distances

The above results show that, for both synthetic and genomic data, our *k*-NN searching algorithm based on the GEH distance far outperforms the linear scan. Only when the dimensionality of the underlying NDDS begins to grow excessively, does the benefits of our algorithm start to become less significant. This is a result of the well-known dimensionality curse problem.

5 Conclusions

Similarity searches in NDDSs are becoming increasingly important in application areas such as bioinformatics, biometrics, E-commerce and data mining. Unfortunately, the prevalent searching techniques based on multidimensional indexes such as the R-tree and the K-D-B tree in CDSs cannot be directly applied to NDDSs. On the other hand, recent work [11, 12, 13, 14] on similarity searching in NDDSs focused on range queries. Nearest neighbor searches were not developed to the same extent. In particular, the k-nearest neighbor searching is still an open issue. We observe that the issue of k-NN searching in NDDSs is not a simple extension of its counterpart in CDSs.

A major problem with a k-NN search in NDDSs using the conventional Hamming distance is the non-determinism of its solution. That is, there usually is a large number of candidate solutions available. This is mainly caused by the coarse granularity of measurement offered by the Hamming distance. To tackle this problem, we introduce a new extended Hamming distance, i.e., the GEH distance. This new distance takes the semantics of matching scenarios into account, resulting in an enhanced granularity for its measurement. It is proven that the GEH distance possesses the triangular property that is useful in an index-based searching algorithm.

To support efficient k-NN searches in NDDSs, we propose a searching algorithm utilizing the ND-tree [11, 12]. Based on the characteristics of NDDSs, three effective searching heuristics are incorporated into the algorithm. The metrics for the heuristics using the GEH distance are carefully defined.

Our extensive experiments demonstrate that our GEH distance measure provides an effective semantic discriminating power among the vectors to mitigate the non-determinism for k-NN searches in NDDSs. Experiments also show that the k-NN searching algorithm is efficient in finding k-NNs in NDDSs, compared to the linear scan method. The algorithm is scalable with respect to the database size. However, when the number of dimensions is high, our algorithm seems to suffer the same dimensionality curse problem as the similar techniques in continuous data spaces.

Our future work includes investigating other useful extensions of the Hamming distance such as the one capturing the semantics of mismatching scenarios, developing more efficient searching heuristics and algorithms, studying the dimensionality curse problem in NDDSs, and exploring techniques for k-NN searches in hybrid data spaces which include both continuous and discrete dimensions.

Acknowledgments

The authors of this paper wish to extend their gratitude towards Gang Qian, Changqing Chen and Chad Klochko for their help in developing experimental programs and conducting some experiments.

References

[1] A. Badel, J. P. Mornon and S. Hazout. Searching for Geometric Molecular Shape Complementarity Using Bidimensional Surface Profiles. In *J. of Molecular Graphics*, 10: 205–211, 1992.

[2] N. Beckmann, H.P. Kriegel, R. Schneider and B. Seeger. The R*-tree: an efficient and robust access method for points and rectangles. In *Proc. of ACM SIGMOD*, pp. 322–331, 1990.

[3] S. Berchtold, D. A. Keim and H.-P. Kriegel. The X-tree: an index structure for high-dimensional data. In *Proc. of VLDB*, pp. 28–39, 1996.

[4] S. Brin. Near neighbor search in large metric spaces. In *Proc. of VLDB*, pp. 574–584, 1995.

[5] P. Ciaccia, M. Patella and P. Zezula. M-tree: an efficient access method for similarity search in metric spaces. In *Proc. of VLDB*, pp. 426–435, 1997.

[6] A. Guttman. R-trees: a dynamic index structure for spatial searching. In *Proc. of ACM SIGMOD*, pp. 47–57, 1984.

[7] R. Hamming Error-detecting and Error-correcting Codes. In *Bell System Technical Journal* 29(2): 147-160, 1950.

[8] A. Henrich. The LSDh-tree: an access structure for feature vectors. In *Proc. of IEEE ICDE*, pp. 362–369, 1998.

[9] M. Kolahdouzan and C. Shahabi. Voronoi-Based K Nearest Neighbor Search for Spatial Network Databases. In *Proc. of VLDB*, pp. 840–850, 2004.

[10] F. Korn, N. Sidiropoulos, C. Faloutsos, et al. Fast Nearest Neighbor Search in Medical Image Databases. In *Proc. of VLDB*, pp. 215–225, 1996.

[11] G. Qian, Q. Zhu, Q. Xue and S. Pramanik. The ND-tree: A dynamic indexing technique for multidimensional non-ordered discrete data spaces. In *Proc. of VLDB*, pp. 620–631, 2003.

[12] G. Qian, Q. Zhu, Q. Xue and S. Pramanik. Dynamic indexing for multidimensional non-ordered discrete data spaces using a data-partitioning approach. In *ACM TODS*, 31(2): 439–484, 2006.

[13] G. Qian, Q. Zhu, Q. Xue and S. Pramanik. A space-partitioning-based indexing method for multidimensional non-ordered discrete data spaces. In *ACM TOIS*, 23(1): 79–110, 2006.

[14] G. Qian. Principles and applications for supporting similarity queries in non-ordered-discrete and continuous data spaces. *Ph.D. Dissertation*, Michigan State Univ., 2004.

[15] J. T. Robinson. The K-D-B-Tree: a search structure for large multidimensional dynamic indexes. In *Proc. of ACM SIGMOD*, pp. 10–18, 1981.

[16] N. Roussopoulos, S. Kelley and F. Vincent. Nearest neighbor queries. In *Proc. of SIGMOD*, pp. 71–79, 1995.

[17] T. Seidl and H.-P. Kriegel. Efficient User-Adaptable Similarity Search in Large Multimedia Databases. In *Proc. of VLDB*, pp. 505–515, 1997.

[18] T. Seidl and H. P. Kriegel. Optimal Multi-Step k-Nearest Neighbor Search. In *Proc. of SIGMOD*, pp. 154–164, 1998.

[19] R. Weber, H. Schek and S. Blott. A Quantitative Analysis and Performance Study for Similarity-search Methods in High Dimensional Spaces. In *Proc. of VLDB*, pp. 194–205, 1998.

[20] X. Zhou, G. Wang, J. X. Yu and G. Yu. M^+-tree: a new dynamical multidimensional index for metric spaces. In *Proc. of Australasian Database Conf.*, pp. 161–168, 2003.