High-Dimensional Similarity Searches Using A Metric Pseudo-Grid

Christian Digout   Mario A. Nascimento
Dept. of Computing Science, Univ. of Alberta, Canada
{cdigout|mn}@cs.ualberta.ca

Abstract

Despite the proposal of numerous tree-based access structures for high dimensional similarity searches, techniques based on a sequential scan have been shown to be simple yet quite efficient alternatives. Given that random accesses to disk are expensive, a linear scan of the (smaller) pre-processed dataset is often much more efficient than even a relatively small number of random disk accesses yielded by tree-based indices. In this paper we present a technique which uses a pseudo-partition of a general metric space analog to the VA-file's partition of the vector space. The rationale is to use a number of pivot objects in the metric space, each one determining a number of hyper-rings in this space. The intersection of those rings, determine pseudo-cells analog to the VA-file cells in the vector space. In order to speedup query processing the data set is clustered (using any applicable clustering technique). Clusters not intersecting cells intersected by the query region cannot contribute to the answer set. Thus, only a few clusters are searched using an I/O efficient linear scan of the cluster's data. The proposed technique, which we call the M-GRID, *is by construction, applicable to both general metric spaces and to traditional vector spaces as long as a metric distance function is used. The* M-GRID *is robust to several parameters and experiments with synthetic and real data sets show that it is able to perform nearest neighbor queries up to 10 times faster than the VA-File.*

1 Introduction

Managing and querying high-dimensional data is important in many domains such as time sequence data [2], protein sequence data [19], density-based clustering algorithms [12] and multimedia data [20]. For instance, an image's colors and textures are typically mapped onto high-dimensional vectors for enabling similarity searches.

In the context of high-dimensional data and metric data, an important type of query is the nearest neighbor query. Given a query object q, a database O of objects o_i, and a metric distance function $d(i, j)$ a nearest neighbor query retrieves the object which is closest to the query object, i.e., $\{o_{NN} \in O | \forall o_i \in O : d(o_{NN}, q) \leq d(o_i, q)\}$. This query can be easily generalized to retrieve the K-nearest neighbors of a query object, in this paper we focus on the latter.

It is well known that effective indexing of the data is necessary for efficient query processing. Indexing high-dimensional data is a harder problem mainly due to the dimensionality curse. Often as the dimensionality of the feature space increases, the difference in the distance between the nearest and the farthest object decreases [5]. As well, it has been shown [23] that the performance of many access methods may degenerate to a sequential scan once the number of dimensions is sufficiently large. In particular, it has been argued in [23] that a sequential reading of a set of pages on disk is faster than randomly reading as few as 20% of the same set of pages, which can likely be the case for high-dimensional data.

The VA-File [23] is a conceptually simple yet very efficient access method for similarity search in high-dimensional data. It partitions the data space into a grid and creates an approximation for each data object based on the grid cell that contains the object. At query time, the VA-file sequentially scans the file containing these approximations, which is smaller than the size of the original data file. This allows most of the VA-File's disk accesses to be sequential which are much less costly than random disk accesses. One drawback of this approach is that the VA-File requires a refinement step where the original data file is accessed using random disk accesses. Experiments in [23] have shown that, in order to keep the number of random disk accesses reasonable, the file containing the approximations should be at least 25% of the size of the original data file. Also, as the number of nearest neighbors retrieved gets larger, the number of random disk accesses in the refinement step increases quickly, even if the accuracy of the approximations is finer. Another drawback of the VA-File is it can only index vector data under a metric distance, e.g., feature vectors in the Euclidean space.

In this paper we build on the VA-file's idea by presenting a general solution to high-dimensional indexing applicable to both metric data and vector data. Our main contribution in this paper is to show how to construct a pseudo-grid[1] for any metric space, including the Euclidean space, for efficient similarity searches. Initially, the data set is clustered off-line, using *any* suitable clustering algorithm. Then, at query time, the pseudo-grid is used to prune the clusters which cannot be part of the answer. Experimental results show that the our technique, which we call the M-GRID, can perform nearest neighbor query processing up to 31 times faster than a sequential scan of the original data set and up to 16 times faster than the VA-File. In addition, our method works in general metric spaces, including vector data under metric distances, can be easily implemented, and scales gracefully with respect to several of parameters, e.g., number of nearest neighbors retrieved. To the best of our knowledge, despite the large amount of research in metric indexing [10], no one has explored this idea. Moreover, often in that domain researchers are interested in minimizing computational cost, i.e., CPU cost, whereas in this work we aim at the more relevant database problem of minimizing I/O cost of similarity queries.

The rest of this paper is structured as follows. In the next section the intuition supporting the M-GRID is presented as a motivation for the next sections. Section 3 describes how to construct a pseudo-grid for metric spaces and how to use it to perform efficient nearest neighbor searches. Section 4 displays experimental results showing the efficiency of the M-GRID in performing nearest neighbor search compared to the VA-File and a sequential scan using both real and synthetic data sets. Related work to indexing high-dimensional data and metric data is described in Section 5. A conclusion and summary of our findings is given in Section 6.

2 Motivation

It is a well known result, e.g., [3, 8, 13], that when processing similarity queries in a general metric space, objects can be correctly pruned using the triangular inequality property. Given a database O of objects o_i, all one needs is a set of reference objects $P = \{p_j | p_j \in O\}$, called pivots (typically $|P| \ll |O|$), and the set of distances from each database object to all pivots objects, i.e., $D = \{d(p_j, o_i) | \forall o_i \in O \wedge \forall p_j \in P\}$. (Note that both sets P and D can be pre-computed off-line.) Given a query object q and a range r an object o_i cannot be prunned by P if and only if $|d(p_j, o_i) - d(p_j, q)| < r \; \forall p_j \in P$. That is, a large number of objects o_i can be potentially discarded without one having to compute the actual distance $d(o_i, q)$. Computing $d(.,.)$ at query time can be impractical. For instance, in [22] the authors compute the similarity between images by solving a (possibly large) network flow problem. Therefore, not having to compute $d(.,.)$ at query time in order to prune a large number of candidate objects can represent a a significant gain in query processing time.

To illustrate this argument, consider the objects in Figure 1, where there is a single pivot object p_1. For the query object q and range r, a pruning ring for p_1 is defined by two radii, $(d(p_1, q) - r)$ and $(d(p_1, q) + r)$ where $d(x, y)$ is a metric distance function that measures the distance between any two database objects x and y. Any object outside the pruning ring can be safely discarded. In Figure 1, o_1 lies outside p_1's pruning ring. As such, the triangular inequality says that $d(q, o_1) + d(p_1, q) \geq d(p_1, o_1)$, hence $d(q, o_1) > r$ and therefore o_1 cannot be a possible answer to query q. This way, by using $d(p_1, q)$ and $d(p_1, o_1)$ (which is precomputed), the cost of computing $d(q, o_1)$ is avoided.

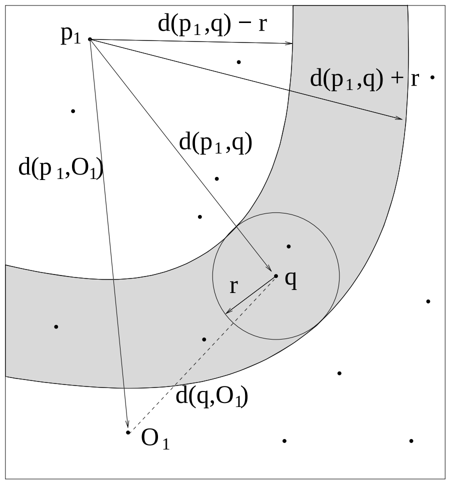

Figure 1: Pruning objects in Metric Space using the triangular inequality

The VA-File partitions the data space by dividing it into a grid of cells. This allows objects to be approximated by the cells where they lie. The idea of creating a pseudo-grid can be generalized to a metric data space. Instead of dividing each dimension into slices, pivots in the data set can be selected and rings can be formed based on a set of range distances to the pivots. The "cells" in this partitioned space are then considered as the intersection of those rings. For instance, in Figure 2 the pivots are p_1 and p_2 and the rings are created based on a set of range distances to those pivots.

Similarly to what was done in other works, e.g., [13], each object in the the data set can be mapped into a vector space, i.e., $o_i = \langle (d(o_i, p_1), d(o_i, p_2), \cdots, d(o_i, p_j) \rangle$. Typically, P is of low cardinality, which means that the mapped vector space is of low dimensionality, and can be clustered efficiently and effectively using a number of clustering methods.

[1]It is not possible to create an actual grid in general metric space because there may be no actual data space with dimensions for partitioning the space.

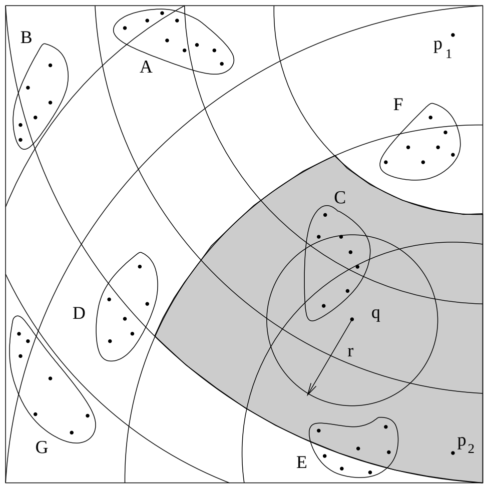

Figure 2: The rings splitting up the data space into cells

Clusters that do not touch pruning rings intersected by the query range can now be safely discarded. Those that are not discarded have to be further searched. Assuming that the objects in the clusters are physically clustered in the disk as well (which can be enforced when assembling the clusters), one can take advantage of the fact that a linear scan can be performed efficiently in order to prune objects within the clusters. In summary, there are two search phases at query time, in the first clusters are pruned and consequently all objects lying within them, and in the second objects within non-pruned clusters are investigated. Figure 2, illustrates this idea for a given q and radius r. The shaded cells intersect the query region, i.e., the region defined by the circle around q. By employing the notion of a grid, a cluster can be pruned if it does not touch any cells which also intersect the query region. In the figure only clusters C and E touch cells intersecting the query region and therefore clusters A, B, D, F and G can be pruned from the answer set without computing any of the actual distances between objects in those clusters and the query object. However, all objects within clusters C and E would have to be verified. Since they should be, by construction, clustered in the disk as well, that can be done efficiently.

3 The M-GRID

Given the rationale above we can now propose a new access method for both vector data (under a metric distance) and general metric data. We call this method the M-GRID. The M-GRID is a dynamic, metric access structure and is designed to reduce the processing time of similarity search queries for high-dimensional vector data sets and metric data sets. The M-GRID clusters objects by their positions in a pseudo-grid based on their distances to pivot objects. Cells in the pseudo-grid are determined by the distance the rings are from the pivots (Figure 2) which allows clusters to be pruned if the cluster does not touch any cells intersecting the query region. In order to take advantage of sequential disk accesses, the M-GRID performs most of its disk accesses sequentially within the clusters, being an I/O efficient metric access method. In addition the M-GRID is computationally inexpensive as the majority of the distances computed at query time are in a low dimensional space. The actual, and potentially expensive, distance functions are typically only computed for a small number of candidates objects within few clusters.

3.1 Building the M-GRID

Given a database of object O and a metric distance $d(.,.)$, there are four main steps in creating the M-GRID:

- **Selecting the set P of pivots objects.** Since the only information one can use to prune objects at search time is the distance between the query and the pivots and the pre-computed distances between the database objects and the pivots, it is important to choose pivots that can use those distances to prune objects effectively. A good pivot selecting algorithm selects pivots that are able to discriminate well between the distances to pairs of randomly chosen objects from the data set. Pivots can be selected from the data set in a variety of ways, e.g., using the HF algorithm [13], using the Incremental Selection (IS) algorithm [9], or even randomly. We have found, through experimental results, that the IS algorithm provides a very effective way to select good pivots for general metric spaces.

- **Pre-computing the set of distances D between every data object and all pivots.** These are computed in order to allow the use of the triangular inequality at query time. Note that this can be done off-line without any impact on query processing time.

- **Creating the rings, hence creating the cells in a pseudo-grid.** The rings are created so there is an equal number of objects in each ring for each pivot. This allows more dense regions of the data space to be covered by more rings and increases the number of objects pruned during similarity search. The intersection of such rings form the “cells” in a low-dimensional mapped space, the so-called pseudo-grid for metric data (see Figure 2 for a two-dimensional example).

- **Clustering the data objects.** The technique chosen to clustering the data objects is orthogonal to the M-GRID’s usage. Even though in this paper we use the K-means clustering algorithm [15] for simplicity and ease of implementation, this is not a requirement—any other clustering technique applicable to vector space could be used. Naturally, different techniques are likely to yield different clusters which may improve (or not) the M-GRID's performance. Since the number of pivots tends to be small, the data set can be clustered efficiently. Objects are clustered based on their distances to pivots so objects occurring in the same rings should be placed in the same clusters.

The main structure of the M-GRID consists of three logical levels. The first level contains the pivots, the distances the rings are from each pivot and the mean of each cluster. The number of pivots, k, tends to be small, in the range of δ to 2δ where δ is equal to the fractal (intrinsic) dimensionality of the data set [13]. The storage space for this level is small and assumed to be stored in main memory.

The second level consists of the Cluster-Cell Array (CC-Array). Each cell is represented in the CC-Array by two elements, (C_{ptr}, C_{empty}), where C_{ptr} is a pointer to the cluster closest to the cell and C_{empty} is one bit which indicates whether the cell is empty or contains objects (Figure 3(a)). This level is also assumed to fit in main memory. Our experiments show that good performance can be obtained using four pivots and ten rings per pivot. This yields a CC-Array requiring only about 20 Kbytes.

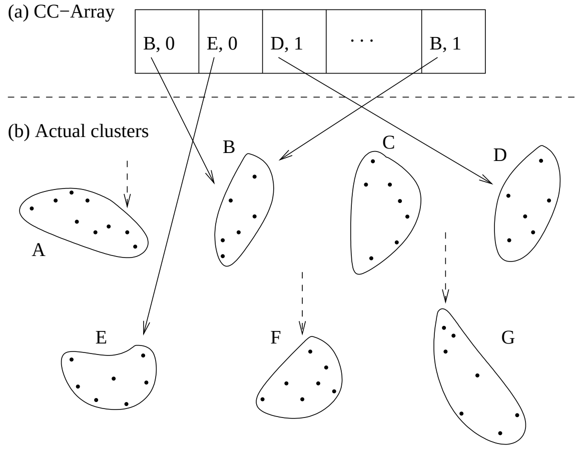

Figure 3: Two levels of the M-GRID's structure: (a) CC-Array, stored in main memory, and (b) the clusters, whose contents are physically clustered on disk

The closest cluster to each cell is determined by computing the distance from the center of each cell to the mean of each cluster. The CC-Array structure for an empty cell, i.e., one which does not intersect any cluster, also point to the nearest cluster because the first cluster to visit is determined by the cell the query object is in, even if the cell is empty. Also, the pointers are used for the insertion and deletion algorithms to determine the cluster to insert or delete the object. Storing whether each cell contains any objects is useful to prevent visiting clusters based on a cell that does not contain any objects but intersects the query region and points to a cluster that would not otherwise be visited. Figure 3(a) illustrates this discussion. Cell 1 contains zero objects but points to cluster B, while the C_{empty} for Cell 3 is equal to 1, indicating Cell 3 contains at least one object and the pointer for the cell points to cluster D. For instance, in Figure 2, the dark-shaded cell to the right of cluster D is empty but intersects the query region and its pointer will also point to Cluster D (its nearest cluster). By using 1 bit for the cell to indicate the cell is empty, we can avoid visiting Cluster D in this example. It should be noted that each cell can only point to one cluster, i.e., the cluster it is closest to in the metric grid. While a cell can only point to one cluster, several different cells can point to the same cluster. The actual number of objects in each cell is stored separately on disk and is only used to aid inserting and deleting objects from the M-GRID.

The third level of the M-GRID (Figure 3(b)) contains the clustered objects in the data set. It is very important to enforce that each object in the same cluster is physically clustered on disk. This allows clusters to be accessed efficiently using sequential disk accesses during query processing.

Given the size of the pseudo-grid, i.e., number of pivots and number of rings, the object data set, and a metric distance $d(.,.)$, the steps for building the M-GRID can be summarized as follows:

1. Compute and store the set of pivots ($p_j \in P$), e.g., using the IS algorithm [9].

2. For each pair (o_i, p_j), $o_i \in O$ and $p_j \in P$, compute and store $d(o_i, p_j)$.

3. Determine the ring distances for each pivot such that each ring contains approximately the same number of objects

4. Using $\langle (d(o_i, p_1), d(o_i, p_2), \cdots, d(o_i, p_j) \rangle$ as the $|P|$-dimensional coordinates for each data object o_i, cluster them using any suitable clustering algorithm. Compute and store the mean object of each cluster

5. For each entry in the CC-Array (there are as many as the resulting number of cells after partitioning the pseudo-grid), set C_{ptr} to the nearest cluster and set $C_{empty} = 0$ if the cell is empty, otherwise set it to 1.

3.2 Similarity Search (KNN) in the M-GRID

K-Nearest Neighbor (KNN) algorithms work by initially finding a range to query which is further tightened as better (closer) neighbors are found during query processing. For the M-GRID, the idea is to find an initial range query and prune away clusters which cannot be part of the answer. Only those cells that intersect the current query range and whose clusters have objects may be searched using an efficient linear scan of the data set.

The first step is to compute the distance between the query object q and each pivot p_j. The cell containing the query object (C_q) is determined by calculating which ring for each pivot the query object belongs to using its distance to each of the pivots. The CC-Array can then be used to obtain the cluster pointer associated to C_q. This cluster is, by construction, a cluster close to q, and as such can be used as the first cluster to be visited and provide an initial query radius. Recall that since clusters are stored in a way that their objects are physically clustered on disk, they can be searched efficiently. At this point the distance between q and the Kth nearest neighboor found thus far can be used to tighten the query radius. Any clusters not intersecting cells which intersect the updated query region, can be safely pruned. Using Figure 2 as an example for a 5 NN search, one can see that cluster C would be the first one to be visited, determining the query radius r. All other clusters but E could then be discarded.

After the first cluster is visited and some clusters are pruned, the distance from C_q to the mean of each cluster not yet pruned or visited is computed, sorted in ascending order and stored in the Active Cluster List (ACL). The order of the clusters in the ACL determines the order they may be visited. Similarly to how it was done when visiting the first cluster, each cluster (in the ACL order) is searched, and the distances between q and all objects in that cluster are computed. If a closer object than the current Kth nearest neighbor is found, this object is used to update the current set of KNNs as well as the current KNN distance. The ACL needs to be recomputed since the query radius is tightened by the update on the KNN distance. This process continues until all clusters in the ACL have been visited or pruned.

In summary, the search procedure works are follows:

1. Find an initial search radius by searching for candidate objects within the cluster closest to the cell where the query belongs.

2. Prune all clusters which do not not intersect the initial query range.

3. Search through all clusters not yet pruned, by increasing distance to the query object, updating the current KNN as closest neighbors are found.

4. Since the query radius (KNN distance) may have been updated, check again for clusters which may now be pruned and repeat the last step above until there are no more clusters to investigate.

It is easy to see that, by its very design, the search procedure described above does not dismiss any object which could be part of the correct answer, i.e., there is no false-negatives. As well, since a cluster or objects inside cluster are only discarded if they cannot be possibly be within the closest KNN, it is safe to state that the search yield correct results.

The M-GRID aims primarily at reducing the number of disk accesses by avoiding to search within clusters unless necessary. Clusters are stored sequentially on disk, so the objects in each cluster are accessed sequentially. Only one random disk access is needed to find the beginning of the cluster on disk, therefore the number of random disk accesses is equal to the number of clusters visited, which the M-GRID aims at minimizing. The M-GRID is designed for K-nearest neighbor queries but can be easily modified to process range queries. To process range queries, the algorithm can be changed so the K-NN radius is equal to the radius of the range query and, unlike K-NN queries, the radius is fixed. All clusters intersecting the radius of the range query will be retrieved and all objects closer than the radius will be returned as the answer to the range query.

3.3 Inserting and Deleting Objects

As mentioned earlier, the M-GRID is a dynamic indexing structure, and therefore can handle the insertion and deletion of objects in the database. To insert an object into the M-GRID, the cell containing the object (C_{in}) is identified using its distances to each of the pivots. The cluster to insert the object is determined by looking up in the CC-Array the cluster C_{in} points to. If $C_{empty} = 0$, i.e., then that bit must be set to 1 to indicate the cell contains at least one object. The number of objects in the cell is also incremented by 1. The object is inserted into the closest cluster to C_{in} and the mean of the cluster is recomputed.

Objects can be deleted from the M-GRID by removing the object from its cluster. Similar to an insert, the cell C_{del} containing the object o_d to be deleted is identified and the cell is looked up in the CC-Array to determine the cluster containing o_d. Note that since the cell contains at least one object, o_d itself, the cluster containing o_d is the closest to that cell. The number of objects in the cell is decremented by 1 and if no other objects reside in the cell, C_{empty} is set to 0. The mean of the cluster is then recomputed.

4 Experimental Results

We tested the performance of the M-GRID using a variety of synthetic data sets. The data sets are designed to test the scalability of the M-GRID with respect to varying the cardinality of the data set, varying the number of clusters in the data set, varying the percentage of noise in the data set, varying the number of dimensions of the data set, varying the maximum distance objects in clusters can be from the seeds of the clusters, varying the number of pivots and the number of rings used in the M-GRID and, finally, varying the number of nearest neighbors retrieved during similarity search. The values used for the generated data sets are shown in Table 1.

We assigned objects to clusters non-uniformily, i.e., some clusters in the data set contain much more objects than others to simulate more realistic data sets. Note that changing the number of objects or the number of clusters in the data set, does not affect the distribution of the number of objects in each cluster, it just changes the average number of objects in each of the clusters. The distance the objects are from the seeds of the clusters is also non-uniform. We use the L_1 metric distance in our experiments because it has been shown to be more effective in high-dimensional spaces at distinguishing the distance between objects than the Euclidean distance (L_2 norm) [1]. All data sets are generated in a unit hyper-cube. For each data set, we used a sample of 100 random queries, and the page size for all experiments was set to 4 Kbytes. Our measure of efficiency is the average actual time required to process a K-nearest neighbor query. Using only number of I/Os may be a misleading measure since the cost of a random I/O is much higher than a sequential I/O due to the extra seek time within the disk. Such an overhead, on the other hand, is captured by measuring the actual query processing time. To evaluate the efficiency of the M-GRID, we compare it to the VA-File and a sequential scan (denoted by SeqScan) of the data set. For all experiments, 10 data sets are generated the same way (following the same distribution) and the average number of disk accesses is used for the M-GRID and the VA-File. For the sequential scan, the number of disk accesses is obviously constant for each experiment.

The first experiment is designed to test the scalability of the M-GRID while varying the cardinality of the data set. In Figure 4 we see the speedup of the M-GRID compared to a sequential scan of the data set is stable at approximately 22 times with a slight increase in performance of the M-GRID as the number of objects in the data set is increased. This occurs because as the number of objects in the data set increases, the number of objects in each cluster increases. This results in an increase in performance of the M-GRID because the NN distance is quickly reduced for most queries, resulting in accessing fewer clusters. For the same reason, the speedup of the M-GRID compared to the VA-File increases slightly to greater than 10.

When decreasing the number of clusters (Figure 5), the number of objects per cluster increases. For the M-GRID, if any part of the cluster intersects the query region, the entire cluster is retrieved from disk, and consequently slower performance. Dividing a cluster into smaller clusters will likely result in retrieving less objects as only some of the resulting clusters may now intersect the query region. However, this

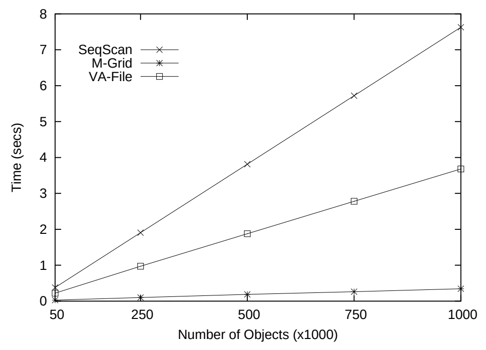

Figure 4: Varying the number of objects in the data set

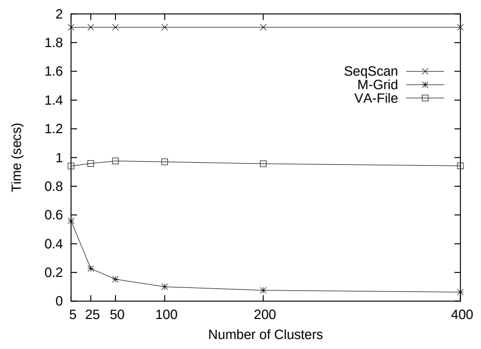

Figure 5: Varying the number of clusters in the data set

is beneficial only to a certain extent as can be clearly seen in the figure. At some point the same objects will still need to be retrieved, just more clusters (with fewer objects per cluster) will be visited, yielding more random I/Os. Figure 5 shows the M-GRID may be more than 30 times faster than a sequential scan of the data set and 15 times faster than the VA-File. Note that even when the number of clusters is very small, the M-GRID is still about twice as fast as the VA-File.

We consider as noise in the data set objects which are created randomly and uniformly in the data set and are not clearly part of any “natural” cluster. By adding noise, one can investigate the resiliency of the access structure. Figure 6 shows the M-GRID is resistant to increasing the percentage of noise in the data set and the performance of the M-GRID only decreases slightly when the amount of noise in the data set is increased up to to 80% of the total number of objects in the data set. This means the M-GRID can perform nearest neighbor search efficiently even if the clusters are not well-defined. When 100% of the data set is noise, the distribution of the data set is basically uniform. Since the M-GRID is designed to take advantage of some sort of underlying clustered structure it cannot improve on the performance yielded by a sequential scan, and it is out-

Table 1. Values used for generating synthetic data sets

Parameter	Default Value	Min. Value	Max. Value
Number of objects	250,000	50,000	1,000,000
Number of clusters	100	5	400
Ratio noise	20%	0%	100%
Data dimensionality	64	32	512
Maximum Distance [% of maximum distance]	1%	0.1%	10%
Number of Pivots	4	2	8
Number of hyper-rings	10	5	20
Number of nearest neighbors	10	1	50

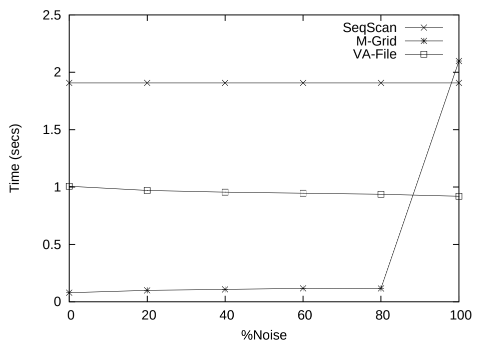

Figure 6: Varying the percentage of noise in the data set

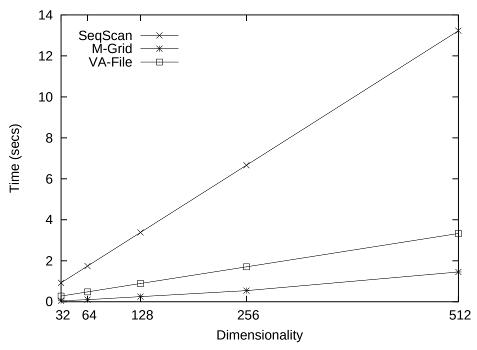

Figure 7: Varying the dimensionality of the data set

performed by the VA-File. This occurs because the VA-File approximates the actual distance of every object in the data set and actually performs the best for uniform data sets because the objects in the data set are farther from each other. This reduces the likelihood of objects occurring in the same cell for the VA-File and thus having the same approximations. For a up to a fair amount of noise, e.g., 80%, the M-GRID is about 8 times faster than the VA-file and 16 times faster than the sequential scan.

The results for varying the dimensionality between 32 and 512 are shown in Figure 7. Note that the maximum distance objects can be from the seeds of the clusters, i.e., their centers. also remains constant at 1%. However, when the dimensionality of the data set is increased, 1% of the maximum possible distance between two objects will also increase. The figure shows the M-GRID is more than 3(3) times faster than the VA-File when the number of dimensions is 256, while for dimensionalities of 32 and 64, the M-GRID is 5 times faster than the VA-File. The relative performance gain of the M-GRID decreases as the dimensionality increases compared to a sequential scan of the data set and the VA-File. One reason for this is that the maximum distance objects in clusters can be is, by default, 1%. Therefore the size of the clusters increases when increasing the dimensionality and the total hyper-volume of the clusters increases faster than the maximum distance. Although the total hyper-volume of the data space also increases, each dimension is still in the range zero to one. This causes the chance of clusters overlapping to be higher and increases the chance of visiting additional clusters, diminishing the M-GRID's advantage.

Figure 8 shows the performance of the M-GRID decreases as maximum distance the objects are from the center of the clusters increases. The main reason for this is as this distance increases, the distribution of the objects in the data set becomes more uniform. This causes the K-nearest neighbor distance to increase significantly, reducing the ability of the pivots to prune clusters (refer to the previous discussion of Figure 6). It should be noted that if the data set can be clustered more effectively, the M-GRID could be able to prune more clusters and increase its efficiency. (In this paper we are not concerned with the clustering algorithm itself since it is a mere accessory to the proposed M-GRID.) Even when *max_d* reaches 10%, the VA-File is only slightly faster than the M-GRID, while both are faster than a sequential scan of the data set.

The M-GRID has two main input parameters for building the metric grid, the number of pivots and the number of rings. Both of these parameters affect the performance of the M-GRID. Figure 9 shows the affect of varying the

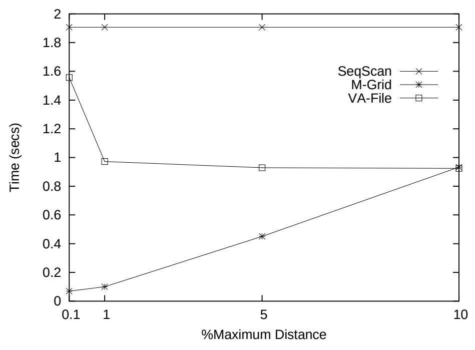

Figure 8: Varying the maximum distance objects in a cluster can be from the seed of the cluster

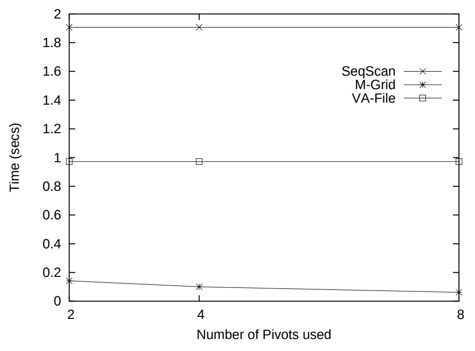

Figure 9: Varying the number of pivots used by the M-GRID

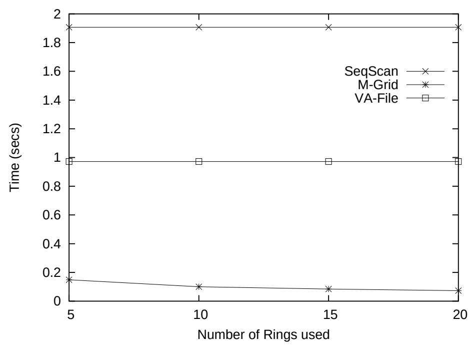

Figure 10: Varying the number of rings used by the M-GRID

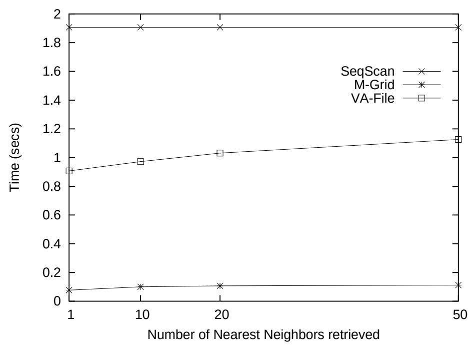

Figure 11: Varying the number of nearest neighbors retrieved M-GRID

number of pivots. Increasing the number of pivots results in more clusters being pruned. This occurs because the more pivots we use, the greater the probability that clusters will not intersect rings which the query region intersects, for at least one pivot and can be pruned. This results in visiting less clusters and retrieving less objects from disk. Increasing the number of pivots used by the M-GRID can only increase the performance up to a certain extent, at some point using more pivots will not result in pruning more clusters. This occurs because there is always going to be at least one cluster for every query which must be visited and clusters which intersect the query region can not be pruned no matter how many pivots are used. The speedup of the M-GRID compared to a sequential scan of the original data set increases to 31 and to 6 for the VA-File when the number of pivots is equal to 8.

Figure 10 shows that increasing the number of rings also increases the performance of the M-GRID compared to a sequential scan of the data set and the VA-File. Increasing the number of rings divides the data space into a finer grid which makes approximating clusters by the cells they intersect more accurate. This allows some clusters which would otherwise border the query region to be pruned and not be visited. Similar to increasing the number of pivots, increasing the number of rings also increases the number of clusters pruned and improves the efficiency of the M-GRID. As noted above, for every data set there will be some limit at which increasing the number of rings cannot result in pruning more clusters as some clusters will have to visited regardless.

Next, we test the effect of varying the number of nearest neighbors K retrieved. Figure 11 shows the largest advantage for the M-GRID is achieved when $K = 1$. The reason for this is for small values of K, only 1 or 2 clusters have to be retrieved (or $1\% - 2\%$ of the data set). Increasing the number of nearest neighbors retrieved past 10 has very little impact on the performance of the M-GRID because often the additional nearest neighbors retrieved are found in the clusters already accessed and additional clusters do not have to be visited. The K-nearest neighbor distance usually does not increase rapidly when K is larger. This causes the number of disk accesses to increase slowly with increasing K. It should be noted that as K is increased, the additional disk accesses for the VA-File are random disk accesses. As K is increased, more objects have to be accessed randomly from the original data file in the refinement step. This means the actual time required for the VA-File to perform NN search will increase rapidly as K is increased, even if the total number of disk accesses does not increase as quickly.

5 Related Work

There have been many indexing structures proposed in the literature to speedup similarity search processing in indexing high-dimensional and metric spaces. These are divided into two main categories: those that index the Euclidean space and those that index all metric spaces. Note that although the later can also index objects mapped onto the Euclidean space, since a (metric) L_p norm is often used, they typically do not explore properties of the embedded data space.

An early Metric Access Method (MAM) is the BK-tree [8]. It is based on the triangular inequality property and attempts to minimize the number of distance calculations to answer a similarity query. A pivot is arbitrarily selected from the data set and the objects are partitioned into subtrees based on their distance from the pivot. This process continues recursively with each sub-tree being broken into smaller and smaller groups. During similarity search, branches which do not intersect the query region are pruned, and the final answer is determined by computing the actual distance from each object in branches not pruned to the query object. Only discrete distance functions are considered for the BK-tree and the number of disk accesses is considered insignificant[2].

A MAM which also uses a tree approach for indexing metric data is the Fixed-Queries tree [3]. Its structure is similar to the BK-tree except each node at the same level of the tree uses the same pivot, this reduces the cost of following more than one path in the tree because additional distances from the query object to pivots in each sub-tree do not have to be computed. Fixed-Queries trees prune branches which do not contain any candidate points in the answer set using the triangular inequality property. To determine the final answer, the distance between objects in branches not pruned and the query object are computed. The Fixed-Queries tree also uses a discrete metric distance function and only considers the number of distance calculations in the cost of similarity search.

Another MAM which uses a similar technique to the Fixed-Queries tree is the MVP-tree [6]. The MVP-tree is a static, height-balanced tree that builds a hierarchical structure which partitions the data space into spherical regions using pivots. Precomputed distances between pivots and data objects are utilized to prune objects using the triangular inequality property. The MVP-tree uses more than one pivot at each node and each sub-tree uses different pivots.

Two additional MAMs are the M-tree [11] and the Slim-tree [16]. Both the M-tree and the Slim-tree are dynamic indexing structures which allow insertions of new data objects without completely rebuilding the tree. Both partition objects such that each node contains a pivot object, and all objects in its sub-tree are within a certain distance of the pivot object. The authors of the Slim-tree also define a measure of the amount of overlap between two nodes in a metric space and present a "slim-down" algorithm to try and minimize the overlap of nodes. These two methods consider the number of disk accesses and distance calculations as an efficiency measure as both can be expensive for metric data.

Two MAMs similar to each other and designed to reduce the number disk accesses and distance calculations for similarity queries are proposed in [7, 13]. Precomputed distances from the data objects to the pivots are used to prune objects for similarity searches. The precomputed distances are read sequentially from disk as sequential disk accesses are much less expensive than random disk accesses. These methods can be used on their own or be incorporated into another index structure such as the R-tree [14] to further increase similarity query performance.

Many indexing structures have been proposed for data in the vector space. Two partitioning strategies used with these indexing structures are data partitioning and space partitioning strategies. Indexing structures which use the data partitioning strategy are the R-tree [14], SR-tree [17] and the X-tree [4]. These methods attempt to index the data by partitioning the data into minimum bounding regions. An indexing technique which uses the space partitioning strategy is the VA-file [23], discussed earlier. A method which partitions the data space into a grid and clusters the data set is Clindex [18]. It is only shown to work in vector space, i.e., it cannot be used for general metric spaces. Also, Clindex answers approximate nearest neighbor queries so it does not guarantee to find the exact answer. A method which uses both the data partitioning and the space partitioning strategy is the A-tree [21]. The A-tree employs the notion of approximate bounding regions to improve similarity query performance. One limitation of these indexing structures is they cannot index metric data, which is often the case with multimedia data.

6 Conclusion

In this paper, we presented the M-GRID, a new index structure for high-dimensional data and metric data. The M-GRID builds a pseudo-grid structure for any metric space, clusters the data (using any clustering algorithm), and uses the pseudo-grid and the clusters to answer KNN

[2]A hardly reasonable assumption nowadays.

queries. Most of the disk access by the M-GRID are sequential, thus fast. The M-GRID can handle dynamic data and can be used with vector and non-vector data, which makes it a very general access structure.

To demonstrate the performance of our technique, we ran a variety of experiments to using various synthetic data sets. The M-GRID can perform nearest search up to 42 times faster than a sequential scan of the data set and up to 10 times faster than the VA-File.

Future work should be done on investigating an optimal number of pivots and the number of rings to use within the M-GRID. Further work should also be done on investigating how the distribution of the data set, after updates, affects the performance of the M-GRID and how can we detect and address this issue. Finally, tests using real data sets are underway, and may also provide further insight into M-GRID's properties

Acknowledgements

This work was supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC), the Alberta Informatics Circle of Research Excellence (iCORE) and the Canadian Cultural Content Management Research Network, a Network financed through Heritage Canada's New Media Research Networks Fund.

References

[1] C. C. Aggarwal, A. Hinneburg, and D. A. Keim. On the surprising behavior of distance metrics in high dimensional space. In *Proc. of the 8th Intl. Conf. on Database Theory*, volume 1973, pages 420–434, 2001.

[2] R. Agrawal, C. Faloutsos, and A. Swami. Effi cient Similarity Search In Sequence Databases. In *Proc. of the 4th Intl. Conf. of Foundations of Data Organization and Algorithms*, pages 69–84, 1993.

[3] R. Baeza-Yates, W. Cunto, U. Manber, and S. Wu. Proximity matching using fi xed-queries trees. In *Proc. of the 5th Symposium on Combinatorial Pattern Matching*, pages 198–212, 1994.

[4] S. Berchtold, D. Keim, and H.-P. Kriegel. The X-tree: An index structure for high-dimensional data. In *Proc. of the 22nd Intl. Conf. on Very Large Databases*, pages 28–39, 1996.

[5] K. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft. When is "nearest neighbor" meaningful? In *Proc. of the 7th Intl. Conf. on Database Theory*, volume 1540, pages 217–235, 1999.

[6] T. Bozkaya and M. Ozsoyoglu. Distance-based indexing for high-dimensional metric spaces. In *Proc. of the 1997 ACM SIGMOD Intl. Conf. on Management of Data*, pages 357–368, 1997.

[7] B. Braunmuller, M. Ester, H.-P. Kriegel, and J. Sander. Multiple similarity queries: A basic DBMS operation for mining in metric databases. *IEEE Transactions on Knowledge and Data Engineering*, 13(1):79–95, 2001.

[8] W. A. Burkhard and R. M. Keller. Some approaches to best-match fi le searching. *Communications of the ACM*, 16(4):230–236, 1973.

[9] B. Bustos, G. Navarro, and E. Chavez. Pivot selection techniques for proximity searching in metric spaces. In *Pattern Recognition Letters*, volume 24, pages 2357–2366, 2003.

[10] E. Chvez, G. Navarro, R. Baeza-Yates, and J. L. Marroqun. Searching in metric spaces. *ACM Computing Surveys*, 33(3):273–321, 2001.

[11] P. Ciaccia, M. Patella, and P. Zezula. M-tree: An effi cient access method for similarity search in metric spaces. In *Proc. of 23rd Intl. Conf. on Very Large Data Bases*, pages 426–435, 1997.

[12] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proc. of the 2nd Intl. Conf. on Knowledge Discovery and Data Mining*, pages 226–231, 1996.

[13] R. S. Filho, A. Traina, C. T. Jr., and C. Faloutsos. Similarity search without tears: The OMNI family of all-purpose access methods. In *Proc. of the 17th IEEE Intl. Conf. on Data Engineering*, pages 623–630, 2001.

[14] A. Guttman. R-trees: A dynamic index structure for spatial searching. In *Proc. of ACM SIGMOD Intl. Conf. on Management of Data*, pages 47–57, 1984.

[15] J. Han and M. Kamber. *Data Mining: concepts and techniques*. Morgan Kaufmann Publishers, 2001. Page 349–351.

[16] C. T. Jr., A. Traina, C. Faloutsos, and B. Seeger. Fast indexing and visualization for metric data sets using slim-trees. *IEEE Transactions on Knowledge and Data Engineering*, 14(2):244–260, 2002.

[17] N. Katayama and S. Satoh. The SR-tree: An index structure for high-dimensional nearest neighbor queries. In *Proc. of the ACM SIGMOD Intl. Conf. on Management of Data*, pages 369–380, 1997.

[18] C. Li, E. Chang, H. Garcia-Molina, and G. Wiederhold. Clindex: Approximate similarity queries in high-dimensional spaces. *IEEE Transactions on Knowledge and Data Engineering*, 14(4):792–808, 2002.

[19] L. Liao and W. S. Noble. Combining pairwise sequence similarity and support vector machines for remote protein homology detection. In *Proc. of the 6th Intl. Conf. on Research in Computational Molecular Biology*, pages 225–232, 2002.

[20] G. Lu. *Multimedia Database Management Systems*. Artech House, 1999.

[21] Y. Sakurai, M. Yoshikawa, S. Uemura, and H. Kojima. The A-tree: An index structure for high-dimensional spaces using relative approximation. In *Proc. of the 26th Intl. Conf. on Very Large Data Bases*, pages 516–526, 2000.

[22] R. Stehling, M. Nascimento, and A. X. Falcão. MiCRoM: A metric distance to compare segmented images. In *Proc. of the 2002 Visual Information Systems Conf.*, pages 12–23, 2002.

[23] R. Weber, H.-J. Schek, and S. Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *Proc. of the 24th Intl. Conf. on Very Large Databases*, pages 194–205, 1998.