On Metric-Space Indexing and Real Workloads

Rui Mao, Ving I. Lei, Smriti Ramakrishnan, Weijia Xu and Daniel P. Miranker
Department of Computers, the University of Texas at Austin, USA

Abstract

Contemporary technology is fostering new demands to manage large collections of complex data, including the contents of multimedia and biological databases. In many cases the similarity of the data is defined using a metric distance function. There are many competing algorithmic approaches which, off-line, create data structures materializing a hierarchical clustering of the data and leverage the triangle inequality to speed the search for similar data. In order to determine a solution of general applicability it is important to assess the variety of methods on various types of real world data.

We evaluate the performance of an algorithm from each of the three major classes of metric-space indexing methods: generalized hyper-plane, vantage point, and radius-based methods. The workloads comprise an image database, a yeast protein sequence database and a database of mass spectrometer protein signatures. For range queries of practical interest the multi-vantage point algorithm (MVP-trees) is shown to be superior. We further consider the optimization of MVP-trees. We consider a common heuristic, choosing *corners* as vantage points, and show that on real workloads choosing *centers* perform better.

1. Introduction

Digital cameras, on-line music systems and the genomic revolution are generating new large-scale database management problems. In each case the retrieval of similar objects from a database is often driven by a similarity function that is a metric distance function [15, 16, 20, 22, 23, 27, 28, 31]. Often, accurately modeling similarity is the central focus of the work. At the same time, researchers are interested in managing large databases containing these types of data by integrating metric-space indexing methods with existing relational database technology [2, 3, 7, 8]. This integration is analogous to spatial database management systems, which add extensions to relational database management systems to manage collections of 2 and 3 dimensional objects by integrating either K-D-tree or R-tree index structures [1, 11].

Definition [28]**:** A *metric space* is a pair (M, d), where M is a nonempty set and d: M*M->R is a real-valued function, called a *metric* on M, with the following properties:

(1) For all x, y ∈ M, $d(x, y) >= 0$ and $d(x, y) = 0$ if $x = y$. (*Positivity*)

(2) For all x, y ∈ M, $d(x, y) = d(y, x)$. (*Symmetry*)

(3) For all x, y, z ∈ M, $d(x, y) + d(y, z) >= d(x, z)$. (*Triangle Inequality*)

The expression d(x, y) is read "the distance from x to y". When there is no confusion, we simply say that M is a metric space, and the distance function d is a metric.

In metric-space indexing, a hierarchical clustering of the data is materialized as a tree-based data structure. Every sub-tree corresponds to a cluster. Associated with the root of every sub-tree is a predicate satisfied by every point in the cluster. Given a query, if there is no data point that satisfies both the predicate of the index node, i.e., in the sub-tree rooted at the node, and the query, the index node can be pruned, and thus distance calculations are saved. Typically the distance functions in these applications are very costly.

The value of a metric-space approach is that it is unnecessary to find a meaning for the data with respect to the axes of a coordinate system. The only way for distance calculations to be saved is by making use of the triangle inequality [6, 13]. Metric-space indexes are usually built off-line, and queries are executed on-line.

The two common query types are range query, which finds all data objects in the database that are within a certain distance to the query, and *k*-nearest neighbor query, which finds the k data objects with the smallest distance to the query. Since the k-nearest neighbor query can be systematically implemented by range query [6] and our goal is the extension of relational database systems to include metric-space relations (i.e. set-valued expressions consistent with relational algebra), we only consider range query in this paper. A range query can be expressed as a pair (q, r), where q is the query object and r is the query radius.

As technology drives practical interest in this problem, the data sets are often amply large to require external, database management system approaches. Further, it is well recognized in the science of database management that data is rarely randomly and uniformly distributed and that it is imperative to consider actual workloads [26]. In surveys by Hjaliason et. al [13] and by Chavez et. al [6], metric-space indexing methods are organized into three classes. We first assess an implementation of one algorithm from each class, modifying it, if necessary, to meet the standard characteristics of disk-based indexes as established by the work on B+-trees. In particular we consider only balanced tree structures. Nodes of the index trees are paginated, meaning the fanout of the nodes, i.e., the number of children, is set such that the storage space for the nodes corresponds to the size of a disk block.

We consider four workloads, three comprising real data, and the fourth a synthetically generated set of five dimensional vectors, typical of synthetic data in other studies. The real data are images, yeast protein sequence fragments and mass spectrometer protein signatures.

Empirical evaluation of the three methods on these data shows that the MVP method significantly outperforms the other two on meaningful range query radii. Further, we consider the heuristic on selection of vantage points in the initial construction (*bulkload*), of MVP index structures. A common heuristic exploited in the construction of MVP trees is to choose "corners" of the space for vantage points [33]. However, the arguments that originated this idea assumed that the data is uniformly distributed, and the distance function had some particular properties. None of these assumptions are true of our real world workloads. Thus, we introduce and evaluate a heuristic that seeks centers as the choice of vantage points. The key idea is that the index tree should reflect the intrinsic clustering of the data. Empirical results show that using centers as vantage points almost always outperforms corners found by a farthest-first-traversal algorithm, even for synthetic Euclidean data.

The rest of the paper is arranged as following. Three metric space index structures are discussed in Section 2, followed by an introduction to our workloads and experimental results in Section 3. Heuristics of vantage point selection of MVP methods are discussed in Section 4, and Section 5 consists of conclusions and future work.

2. Metric-Space Index Structures

In two authoritative surveys metric-space indexing methods are organized into two major classes— general hyper-plane methods and vantage point methods [13]—and an M-tree algorithm that falls outside of those classes [6]. Considering that there are now at least two variations on M-trees, we label these algorithms as a third taxonomic class called *radius-based* methods [20, 29].

Similar to answering a query of traditional data types using B+-trees, answering a range query in a metric space may be accomplished by traversing a decision tree. The traversal can be done in many styles, such as breadth-first or depth-first order. When an internal node is visited, a query predicate is compared to an index predicate parameterized by values stored in the node. If it is determined that the query predicate has no overlap with the predicate describing the points stored in a sub-tree, the sub-tree can be eliminated from further consideration. In other words, we can prune sub-trees from the traversal. The sub-trees that cannot be pruned, often more than one, are stored for future visits. When a leaf node is visited, its pre-stored information is used to decide whether each data object is a query result or not. Since there are two excellent survey articles in the literature, in the following discussion of the algorithm we introduce only the general logic used to construct the predicates for each category of metric-space indexing method, and important details of our particular implementation.

2.1 General hyper-plane method

The General Hyper-plane Tree (GHT) was first proposed by Uhlmann in [30]. A GHT is constructed recursively, top-down. In each step, two points, c_1 and c_2, are selected as centers, or pivots, for the two children, and remaining points are assigned to the closest pivot. Thus, the data is partitioned into two clusters. Then, each cluster is partitioned recursively. In the best case the tree is balanced and the construction time complexity is O(nlogn), and the space complexity is also O(nlogn).

Given a range query (q,r) and an internal node, the distances from the query object to the two pivots are first calculated. Next, if $d(q, c_1) + 2r < d(q, c_2)$, then the right child can be pruned, and only the left child is visited. If $d(q, c_2) + 2r < d(q, c_1)$, the left child can be pruned, and only the right child is visited. It is possible neither inequality is satisfied and that both children are visited.

In a leaf index node, a pivot, c, and the distances from each data point to the pivot are stored. Given a range query (q, r), d (q, c) is first calculated. Then, for each data point p, if $| d(q, c) – d(c, p)| > r$, p is pruned. Otherwise, d(q,p) is calculated. If d(q,p)<=r, p is a result, otherwise it is not.

A GHT is a main-memory binary tree structure. To use it as an external index, we arrange the size of each tree node to be equal to the size of a disk page, called the GHT-index. It can be easily extended to *m*-ary trees by select *m* points as pivots for *m* children. During the search, the pivot, c_m, which is the closest to the query object, is first found. Then, for each of the remain pivots, c, if $d(q, c_m) + 2r < d(q, c)$, the cluster with c as the pivot can be pruned. Thus, the fanout of the internal nodes is determined by the disk page size, and the number of nodes visited during the search is a measurement of I/O. We create balanced trees by using the same initialization method we developed for radius-based methods, described below.

2.2 Multiple vantage point method

Vantage Point Trees (VPT) were proposed independently by Uhlmann and by Yanilous [30, 33]. In a VP-tree, a vantage point VP and a distance r are chosen such that the bounding sphere defined by VP and r partitions the data into two evenly sized subsets. A top-down recursive construction results in a balanced binary tree.

Ozsoyoglu et. al proposed Multiple Vantage Point Trees (MVP-trees) [2,3]. The basic idea is that several vantage points are chosen. The distance to each vantage point forms a coordinate. The fanout of the internal nodes can be expanded by increasing the number of vantage points, and/or by nesting multiple bounding spheres for each vantage point. The topological structure of an MVP-tree is defined by the triple (*v, s, m*), where *v* represents the number of vantage points in each node, and *s* is the number of distance intervals as determined by $s – 1$ bounding spheres around each vantage point. Thus, the fanout of a node is s^v. *m* is the maximum number of data points in a leaf node.

To keep the partitioning balanced, an MVP-tree is constructed using a double recursion. Within a node the first vantage point is used to partition the data into s subsets. The second vantage point is used to partition each of those subsets independently. Thus, in the evaluation of a range query, a visit to an internal node results in *v* distance calculations, but considers s^v disjoint partitions of the data. The nodes size is also proportional to s^v, containing partitioning data $\{(VP_j, d_{j,i}\text{-}min, d_{j,i}\text{-}max)| i = 1, 2, …,m, j = 1, 2, …, v\}$.

Leaf nodes consist of surrogates to data points and their distances to each vantage point. Therefore, each data point in a leaf node is defined by a set of predicates, $\{(VP_i, d_i)| i = 1, 2, …,v\}$, i.e., the distance for the data point to VP_i is d_i, where i = 1, 2, …,v.

Given a range query (q, r), and an internal node, with each of its children defined by $\{(VP_j, d_{j,i}\text{-}min, d_{j,i}\text{-}max)| i = 1, 2, …,m, j = 1, 2, …, v\}$. If for any j, $d_{j,i}\text{-}min - r <= d(VP_j, q) <= d_{j,i}\text{-}max + r$ is not satisfied, the child node can be pruned. For the detail of the algorithm, interested users are referred to [2,3]. Given a leaf node, for each data point p, if for any i, $d_i - r <= d(VP_j, q) <= d_i$ is not satisfied, where i = 1, 2, …, v, the data point is pruned. If p cannot be pruned, then d(p, q) is computed. If d(p,q) <= r, then it is a result, otherwise it is not.

Similar to the GHT, the original MVP-tree was studied as a main-memory data structure and did not account for I/O cost. We implemented a paged MVP-tree, called MVP-index, which for the purpose of this discussion amounts to setting *s* and *v* such that the size of the interior nodes corresponds to a disk page. Also, each data point in the leaf nodes of the original MVP-tree stores a list of distances from it to each of the vantage points in the nodes on the path from the leaf to the root of the tree. In our implementation we exclude this information (and the opportunity it gives for optimizing distance calculations) in preference to increasing the storage capacity of the leaf nodes and minimizing I/O.

2.3 Radius-based method

Radius-based methods, first established by the M-tree algorithm, are sometimes confused with vantage point methods [7,8]. In both cases the method relies on the choice of pivots and bounding spheres. In vantage point methods the bounding spheres form disjoint data partitions. In radius-based methods the bounding spheres may overlap. Internally, radius-based methods are more similar to GHTs than VPTs. For each cluster, a number of pivots are selected, and each remaining point is assigned to a pivot, usually (but not necessarily) the closest one. Each cluster is defined by a predicate (c, R), where c is the pivot, and R, called the covering radius, is the maximum of the distances from cluster members to the pivot. That is, the pivot and the radius define a bounding sphere.

Given a range query (q, r), an index node (c, R), and a reference point p, usually the pivot of a parent node, with pre-calculated distance d(p,c), if $| d(p, q) - d(p, c) | > r+R$, the node can be pruned, and d(q, c) doesn't need to be calculated. Otherwise, d(q,c) is first computed, then, if $d(q,c) > r + R$, the node can be pruned. Otherwise, the node cannot be pruned. The structure and search of the leaf nodes of a Radius-based index tree is the same as that of a GHT.

M-trees were inspired by R-trees, which also have overlapping predicates. The advantage is that dynamic database operations, (inserts and deletes), are simpler to implement and the data structure can be kept in balance using split-and-promote algorithms established by GiST [12]. In earlier work we developed the radius-based tree (RBT) as a refinement of M-trees [20]. The primary improvement is the initialization of the data structure [20]. We also used this initialization method to create the balance GHT in this study. M-tree's bulkload algorithm is a divide-and-conquer method.

In M-trees, the data structure is initialized through a top-down construction that selects the pivots at random and uses some rebalancing heuristics when that fails [7]. The initialization method we developed for an RBT is a bi-directional method. Briefly, data is allocated to individual disk-pages through a top-down recursive construction using farthest-first-traversal [14]. The recursion terminates when the size of the cluster fits on a disk page. The resulting tree may not be balanced. Thus, only the allocation of data to data pages is kept. The bounding predicates for each page are then treated as data. The top-down clustering is repeated, adding an interior level of index nodes above the leaves. The process is repeated until the predicates, treated as data, fit into a one disk-page sized node that roots the entire tree. The farthest-first k-center algorithm has time complexity O(kn), and is guaranteed to produce a clustering within a constant factor of two of optimal. We proved that the time complexity of bi-directional bulkload is O(nlgn) in the best case, but is $O(n^2)$ in the worst case [7,20]. We have shown good experimental execution times for the RBT initialization scheme and shown empirically that it produces significantly tighter bounding spheres than the M-tree initialization method.

In the RBT we define the radius of a bounding sphere to be the maximum distance between the pivot and each of the data objects in the sub-tree [20]. In the M-tree, the radius of the bounding sphere to the points in a sub-tree is computed from the interior nodes, per the maximum sum of the radius of a child and the distance between the centers of the child and the parent, $r = \max \{r_i + d(C, c_i)\}$. Experimental results show that RBT-index outperforms the M-tree [20].

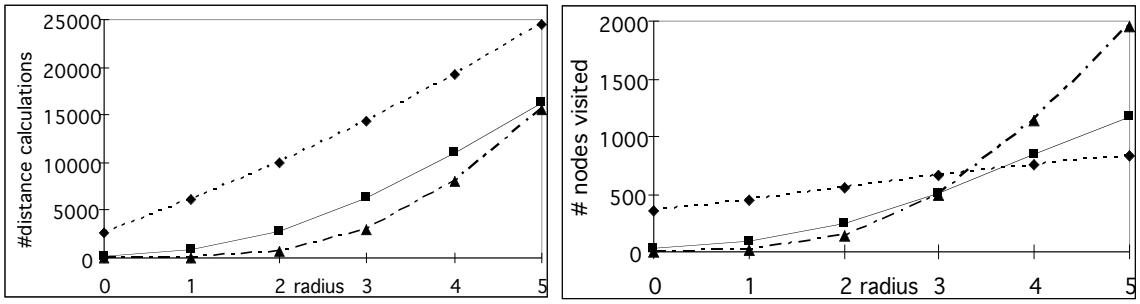
3. Experimental Results Per Algorithmic Category

3.1 Workloads

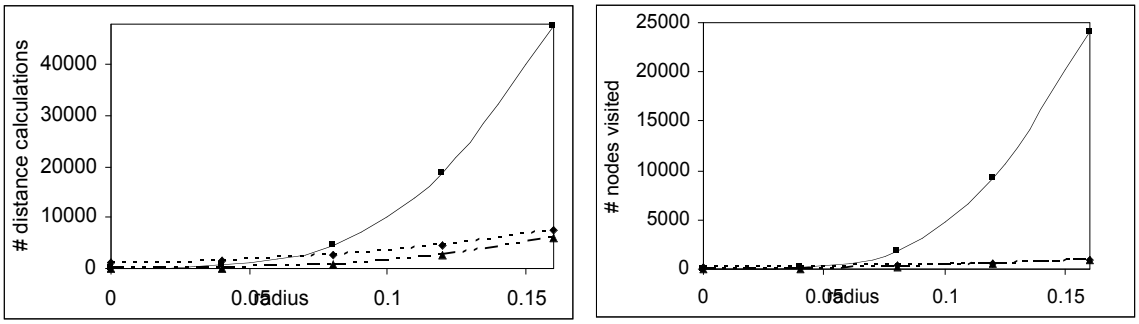
The experiments involve four datasets: yeast protein sequence fragments, mass spectrometer protein signatures, images, and randomly generated vectors of a 5-dimensional Euclidean space. All the datasets are publically available [9].

The yeast protein sequence dataset represents the entire yeast proteome as curated by NCBI [25]. The dataset contains FASTA formatted amino acid translations extracted from GenBank/EMBL/DDBJ annotated records. We split the sequences into overlapping fixed length fragments, or q-grams. The fragment length is set to 5. The distance between two fragments is their weighted edit distance using weights determined by the mPAM amino-acid substitution model [32]. The distance values are integers between 0 and 30.

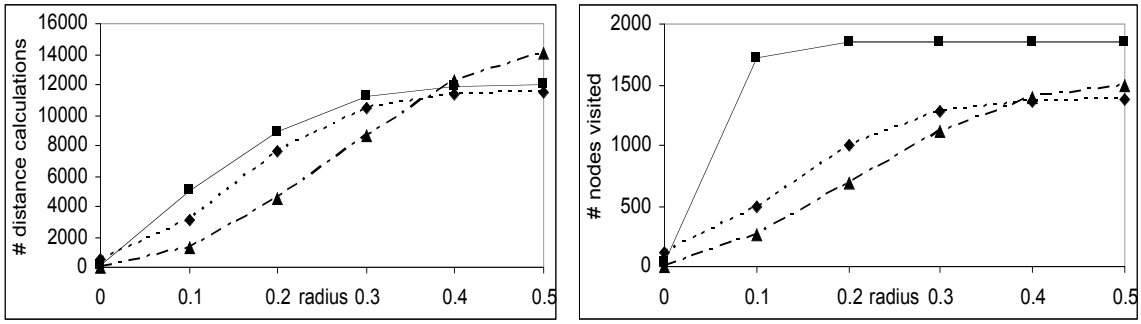
Our mass spectrometer data are theoretic spectra computed from the contents of the SWISS-PROT



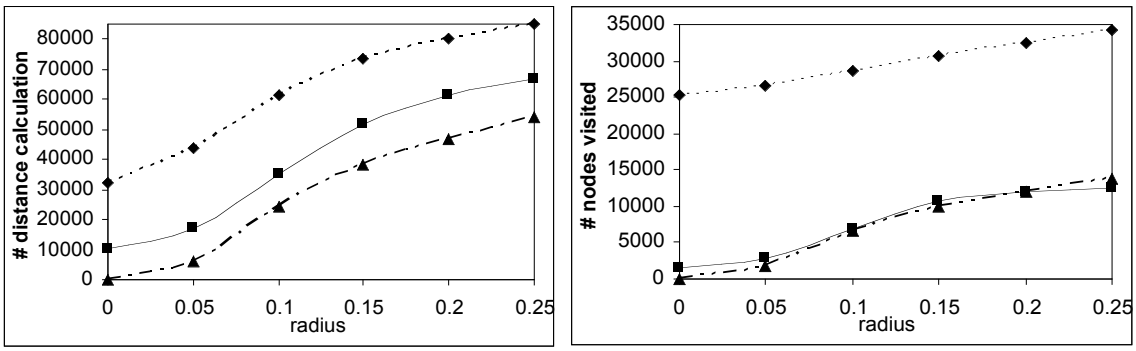
(a) Protein sequence fragment, #distance calculations / #nodes visited VS radius, size: 30000



(b) Vector, #distance calculations / #nodes visited VS radius, size: 1 million



(c) Images, #distance calculations / #nodes visited VS radius, size: 10221



(d) Mass spectrometer data. #distance calculations / #nodes visited VS radius, size: 60567

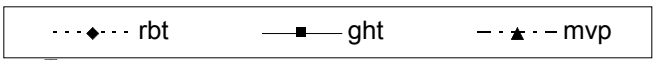


Figure 1. Comparison of GHT, RBT and MVP indexes

database downloaded on 27 April 2004 [24]. Each spectra is represented as a very high dimensional binary vector, one dimension for each resolvable mass. We defined a pseudo-metric, fuzzy cosine distance, to

approximate the behavior of a similarity measure called the shared peaks count (SPC) commonly used to "identify proteins by database look-up of mass-spec signatures" [22]. Fuzzy cosine distance is cosine distance conditioned on a tolerance, t units, of resolvable mass. That is, in the inner product term, a one is computed for each pair of ones, one in each vector, that are within t units (dimensions) of a resolvable mass. Thus, the distance values are continuous in $[0, \pi/2]$.

The image dataset consists of 10221 images. Each image is represented by 3 vectors corresponding to its properties in structure, color, and texture. For each feature vector, a metric distance function is defined [15, 16]. We use a linear combination of the 3 metric distance functions as the distance function in the experiments. The distance values are continuous in [0,1].

Table 1. Statistics of data

	Maximum database size in records	#query	Expected maximum application radius	Max radius used	Average size of result set for maximum radius query
Vector	1,000,000	1000	N/A	0.16	439
Protein fragment	30,000	300	5	5	2000
Mass-spec	60,567	600	0.2	0.25	4
Image	10,221	100	0.4	0.5	9900

The uniform vector dataset consists of 1 million vectors randomly selected from the 5-d [0,1] hyper-cube. Euclidean distance is used.

For each dataset, a number of data points are randomly selected as query objects. Data sets of different size were derived randomly sampling the largest data set.

To evaluate the performance, the pre-selected query objects are executed with a number of radii on each index**.** Only data of a reasonable radius is used, per the anticipated application. For a large radius, a major part of the database is returned as a result. Normally a user only wants a small number of results, whether he or she executes a range query or a k-nearest neighbor query. Moreover, for a large radius, the number of distance calculations will be close to or even greater than the database size, which makes the search degrade to a linear scan, and is thus not of much interest.

We are interested in the algorithmic characteristics of the indexing methods. Thus, only implementation-independent statistics are desired as performance measurements. Distance calculations in metric space are usually based on the content of data objects, and are thus usually costly. Therefore, as other researchers have done, we use as performance measurements the average number of distance calculations and the average number of I/O operations needed to answer the query. In our implementations, index node size is equal to the disk page size, 4096 bytes, thus, we simply use the number of nodes visited during the search as the number of I/O operations.

Table 2. Characteristics of the Data Structures

	GHT		MVP		RBT	
	Fanout	**Height**	**(v, s, m)**	**Height**	**Fanout**	**Height**
Vector	16	5	(4,2, 20)	5	16	5
Protein fragment	20	3	(3, 3, 50)	4	20	3
Mass-spec	16	4	(4,2,20)	6	16	4
Image	8	4	(3,2,15)	5	8	4

Some statistics of the datasets are listed in Table 1. Note that the expected maximum application radius is the maximum radius that is used in real applications.

3.2 Structure of the Indexes

For each kind of data, the data of different size indexes shows similar relationships among the three index structures. Thus, we only show the results of the largest databases of each kind of data (Figure 1). Statistics about the bulkload parameters and index structure are listed in Table 2.

3.3 Results

The data is shown in Figure 1. The figure shows that for all the data types, MVP-index always yields the smallest number of distance calculations and number of index nodes visited for small radii for almost all cases. As the radius increases, MVP index is outperformed by either GHT or RBT. However, as we can see, the number of distance calculations or nodes visited at the radii where MVP-index is outperformed is relatively large, comparing to the database size. Note, these radii or larger radii are not of much interest, as discussed before.

It can also be observed that GHT outperforms RBT in most of the cases except for the synthetic uniformly distributed random vector data. Since RBT is an optimized version of M-tree, and yields out better performance than M-tree [20], we can expect that MVP-index to always outperform M-tree, and even GHT-index will outperform M-tree in most cases.

4. Refining the MVP-Index

The query performance of an MVP-index is affected greatly by its selection of vantage points. It is a NP hard problem to decide the optimal set of vantage points. Many heuristics have been proposed, among them random selection (random-vp) and farthest points (fp-vp) are popular choices. Random choice has the advantage of speed and it may be combined with sampling techniques. A further advantage of random selections relative to other heuristics, is that it usually leads to more balanced data structures, especially when the data displays little skew**.**

The use of farthest-first-traversal dates to Yianilos original paper on VPT [33]. Yianilos considered the use of Euclidean metrics on uniformly distributed data. In this context there is a persuasive argument that the surface area of the partitioning spheres should be minimized and that is achieved by finding "corners" of the space. The farthest-first-traversal k-center algorithm satisfies an intuitive notion of corner finding. It is fast, O(nk) and as been shown to produce clusters within a factor of 2 of optimal [14]. In its pure form, the first point in the algorithm is chosen at random. Since choosing the first point at random provides no assurance that it will be in a corner, Yianilos proposed a sampling technique to select the first point [33]. Two random samples are first generated. Then, for each point in the first sample, its 2^{nd}-moment of the distances to points in the second sample is computed. Finally, the point with the largest 2^{nd}-moment is selected as the first farthest point.

```
selectFirstPoint(D: dataset) {
   r = a random point;
   f1 = a set of farthest points return by farthest-first-
      traversal ( ) with r as the first farthest point;
   for (each point pi in f1) find pi' in D that is the farthest
      to p1;
   compute all the pair-wise distances of {r} U f1 U {pi'};
   let d(u,v) be the largest pair-wise distance;
   return u;
}
```

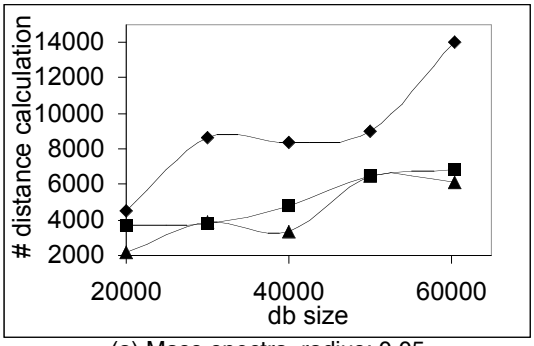
Figure 2. Algorithm to select first farthest point

In our work, we first select a random point as the first farthest point. Then run farthest-first-traversal on it to get a number of farthest points. Next, for each farthest point, we find the point with the largest distance to it in the whole dataset. Finally, we compute the pair-wise distances among all the points selected out previously, and the select the point with the largest pair-wise distance as the first farthest point. The algorithm to select the first farthest point is shown in Figure 2.

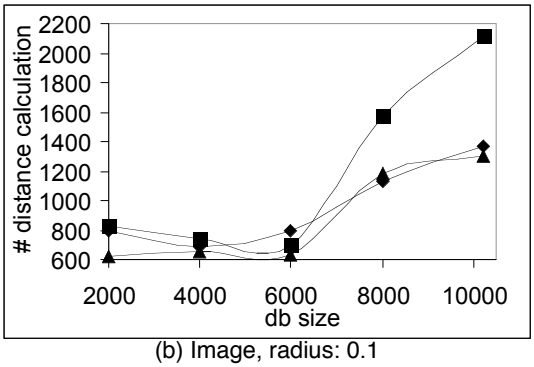
After the first farthest point is selected, in each step, the point with the largest distance to the current set of farthest point is selected as the next farthest point. The distance from a point to a set of points is defined as the smallest distance from the point to all the points in the set. The time and space complexity of farthest-first-traversal is O(n), where the number of farthest points is considered as a constant [14].

Even though the integration of farthest-first-traversal clustering with MVP is widespread, including in our own work, we encountered situations that caused us to question this. Further, even in his own account, Yianilos' is explicit about the limits of his argument. For example, he assumed the zero probability spheres (ZPS) property, i.e, $P(y \in S | d(y,x) = r, \forall x \in S, r >= 0) = 0$, where S is the whole dataset, which implies "The discrete metric is thus excluded along with many other cases" [33]. The metric on the yeast data set is

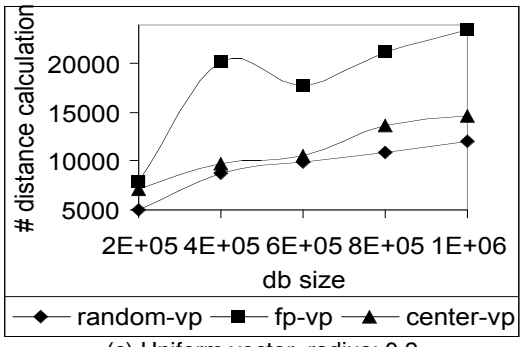
discrete. The other two practical workloads violate this or some other assumption underlying the corner argument.



(a) Mass-spectra, radius: 0.05

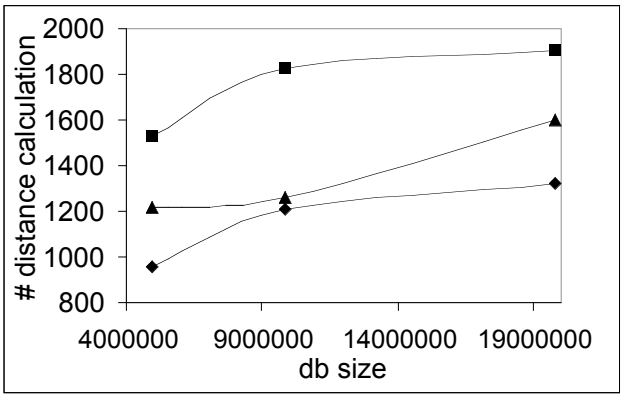


(b) Image, radius: 0.1

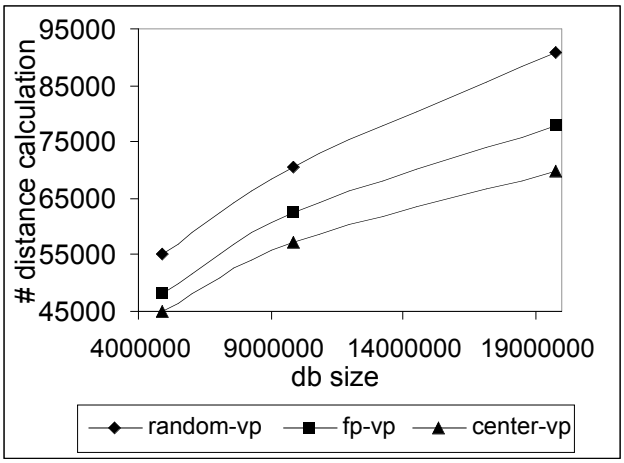


(c) Uniform vector, radius: 0.2

Figure 3. Comparison of VP selection heuristics



(a) radius: 1



(b) radius: 3

Figure 4. Comparison of VP selection heuristics of protein fragments

Thus, we investigate three heuristics concerning the choice of vantage-points. In particular we introduce the use of intrinsic cluster centers of the data as vantage-points. We implement a minimum-sum of diameters (MSD) clustering algorithm derived from CLARA (Clustering LARge Applications) [18]. CLARA is a simple k-median algorithm based on sampling and iteration. The time complexity of the algorithm is O(n) [18]. K-means algorithm can not be applied to metric space directly because algebra operations in metric space are not defined. K-center algorithms are also not applicable because its objective only considers the maximum cluster diameter. Therefore, k-median and MSD (Minimize the Sum of cluster Diameter) algorithms are our only choices. K-median algorithms seek to minimize the sum of distances from all data points to their cluster centers, while MSD algorithms try to minimize the sum of cluster diameters. Both are NP hard problems and there are some primal-dual constant factor approximation algorithms [5, 17]. However, these algorithms usually have high time complexity, which makes them not applicable to the case of indexes, where the datasets are large.

To compare the effect of the three heuristics, we build MVP-indexes with each heuristics of a series of sizes for each kind of data. Then, we used the same method as in Section 4 to collect the statistics as performance measurements. Due to page limit, only data of typical radii are presented here. The data of vectors, images and mass spectrometer is shown in Figure 3, and data of protein sequence fragments of two radii is shown in Figure 4. Note that larger-size sequence fragment data, up to 20 million, is used.

From these figures, first we can see that center-vp performs the best or close to the best for most of the cases. It always outperforms fp-vp except for just a few points. Second, center-vp and random-vp have similar performance for uniform vector and image. As discussed before, random-vp usually leads to balance tree. When the radius is very small, only a few index nodes are visited at each level, and the query performance is dominated by the height of the index tree. Therefore, random-vp has good performance for very small radii. Figure 4 shows this more clearly. We can see that for radius 1, random-vp outperforms the other two, while be outperformed by other two at radius 3.

We also notice that there are several dips when databases get larger. We owe this to the randomness in selection of data, vantage-points and queries.

5. Conclusions and Future Work

Empirical studies of metric space indexing methods seldom consider real data in analyses and evaluation. This is surprising in light of the broad range of emerging database applications. If the history of spatial database is a guide, we can anticipate a proliferation of specialized metric-space indexing methods, each of which showing strength in a particular application area. If the goal is to field a general solution, then we, as a community, need to find algorithms that are good, or very good, on a broad range of applications. Avoiding very poor performance on an occasional application is more important the performing optimally on a few applications.

A goal of this study is to set a baseline moving forward. Due to their similarity with R-trees, radius-based methods may excel in supporting dynamic database operations. Our experience here is that the vantage-point and hyper-plane methods are much better at retrieval. At search radii typical of real queries on these data MVP-index is always better. Our results do demonstrate crossovers at larger radii. Thus, we do not make a definitive claim of superiority of the MVP methods over GHT-indexes. Such a claim would require a much broader test suite.

In regards to future research, the algorithms needed to support dynamic operations on MVP-trees have been published [10]. Further optimization of the search characteristics of MVP-tree would still be beneficial. The I/O results we report may yield acceptable performance. However, they do suggest that formal, adversarial arguments that conclude that good index structures for metric-spaces may be infeasible have some merit. In our experience, also voiced by Brin, the success of a metric-space indexing method is connected to ability of the method to accurately model the natural hierarchical clustering that may be displayed in the data [4]. Per effort to improve M-trees, when packing data on disk-pages, outliers in clusters may severely impact the ability of the index predicates to form easily distinguishable, (small and prunable), clusters. We believe that this should be a primary focus in further efforts. For example, CLARA is based on random sampling and thus is very likely not accounting for outliers. In highly skewed data sets, this effect may be more pronounced. But highly skewed data sets can display much more structure than uniform data sets. Consequently, we believe much more opportunity lay ahead [21].

Last but not least, lessons learned from static clustering and bulk-loading need to be translated to split-and-promote policies to support dynamic database operations. Considering that there were over 100 papers describing multidimensional indexing methods before commercial enterprises settled on a choice of R-trees of K-D trees as the basis of spatial databases, we are optimistic that a general purpose distance-based database index will be found to support general purpose metric-space index.

Reference

[1] J. L. Bentley, "Multidimensional binary search trees used for associative searching", *Communications of the ACM*, 18(9):509--517, September 1975

[2] T. Bozkaya, and M. Ozsoyoglu, "Distance-based indexing for high-dimensional metric spaces", In *Proc. ACM SIGMOD International Conference on Management of Data*, 1997, pp. 357-368.

[3] T. Bozkaya, and M. Ozsoyoglu, "Indexing Large Metric Spaces for Similarity Search Queries", *Association for Computing Machinery Transactions on Database System*, 1999, pp. 11-34.

[4] S. Brin, "Near Neighbor Search in Large Metric Spaces". In *Proc. 21st. Int. Conf. Very Large Data Bases (VLDB)*, 1995, pp. 574-584.

[5] M. Charikar and R. Panigrahy, “Clustering to minimize the sum of cluster diameters”, *Proceedings of the thirty-third annual ACM symposium on Theory of computing*, July 2001.
[6] E. Chavez, G. Navarro, R. Baeza-Yates, and J.L. Marroquin, “Searching in metric spaces”, *ACM Computing Surveys*, 2001, Vol. 33(3), pp. 273-321, 2001
[7] P. Ciaccia and M. Patella, “Bulk loading the M-tree”. *In Proceedings of the 9th Australasian Database Conference (ADC'98)*, Perth, Australia, February 1998 15--26.
[8] P. Ciaccia, M. Patella and P. Zezula, “M-tree: an efficient access method for similarity search in metric spaces”. *Proc. 23rd Int. Conf. Very Large Databases (VLDB)*, 1997.
[9] Datasets used in this paper: ttp://mobios.csres.utexas.edu/internal/icde05-mvpdata/
[10] A. Fu, Y. L. Cheung, and Y. S. Moon, "Dynamic VP-Tree Indexing for NNearest Neighbor Search Given Pair-Wise Distances", in VLDB Journal, Springer, volume 9, Issue 2, pages 154--173, 2000. 133
[11] A. Guttman, “R-trees: A Dynamic Index Struc- ture for Spatial Searching”. Proc. of SIGMOD, 1984.
[12] J.M. Hellerstein, J.F. Naughton, and A. Pfeffer, “Generalized Search Trees for Database Systems”, *Proc. 21st Int'l Conf. on Very Large Data Bases*, Zürich, September 1995, pp. 562-573.
[13] G. R. Hjaltason and H. Samet, “Index-driven similarity search in metric spaces”, *ACM Transactions on Database Systems (TODS)*, Volume 28 Issue 4, December 2003.
[14] D.S. Hochbaum, and D.B. Shmoys, “A best possible heuristic for the k-center problem”, *Mathematics of Operational Research*, 1985, Vol. 10(2), pp.180-184.
[15] Q. Iqbal, and J.K. Aggarwal, “Perceptual Grouping for Image Retrieval and Classification”, *3rd IEEE Computer Society Workshop on Perceptual Organization in Computer Vision*, Vancouver Canada, July 8, 2001, pp. 19.1-19.4.
[16] Q. Iqbal, and J.K. Aggarwal, “Image Retrieval via Isotropic and Anisotropic Mappings”, *Pattern Recognition Journal*, December 2002, Vol. 35, no. 12, pp. 2673-2686.
[17] K. Jain and V.V. Vazirani, “Primal-dual approximation algorithms for metric facility location and k-median problems”, In *Proceedings of the 40th Annual IEEE Symposium on Foundations of Computer Science*, October 1999.
[18] L. Kaufman and J.R. Peter, “Finding groups in data: An introduction to cluster analysis”, John Wiley & Sons, 1990
[19] E. Keogh, "Exact Indexing of Dynamic Time Warping", *Proc. 28th International Conference on Very Large Database (VLDB), V*LDB Endowment, 2002.
[20] R. Mao, W. Xu, N. Singh, and D.P. Miranker, “An Assessment of a Metric Space Database Index to Support Sequence Homology”, In *the proceeding of the 3rd IEEE Symposium on Bioinformatics and Bioengineering*, March 10-12, 2003, Washington D.C.
[21] R.R. Mettu and C.G. Plaxton C. G.. Optimal Time Bounds for Approximate Clustering. In Proceedings of the 18th Conference on Uncertainty in Artificial Intelligence, August 2002, pages 344-351.
[22] .P. Miranker, S. Ramakrishnan and W. Briggs, “Fast coarse filtering method for protein identification by peptide mass
fingerprinting”, Technical report, Department of Computer Sciences, University of Texas at Austin, August, 2004.
[23] D.P. Miranker, W. Xu, and R. Mao, “Architecture and Application of MoBIoS, a Metric-Space DBMS to Support Biological Discovery”, *15th International Conference on Scientific and Statistical Database Management. (SSDBM03)*, 2003, pp. 241-244.
[24] NCBI Mass Spectrometer data website: ftp://ftp.ncbi.nih.gov/blast/db/FASTA/swissprot.gz
[25] NCBI protein data website: ftp://ftp.ncbi.nih.gov/genbank/genpept.fsa.z
[26] B. J. Oommen and M. Thiyagarajah, “The Rectangular Attribute Cardinality Map: A New Histogram-like Technique for Query Optimization”. Technical Report TR-99-01, School of Computer Science, Carleton University, Ottawa, Canada, Jan 1999.
[27] S.B. Needleman and C.D. Wunsch, “A general method applicable to the search for similarities in the amino acid sequence of two proteins”, *Journal of Molecular Biology*, 1970, Vol. 48, pp. 443-453.
[28] S. Roman, “Advanced Linear Algebra”. Springer-Verlag, Graduate Texts in Mathematics Vol. 135, 1992.
[29] C. Traina , A. J. M. Traina, B. Seeger, and C. Faloutsos, “Slim-Trees: High Performance Metric Trees Minimizing Overlap Between Nodes”, *EDBT 2000*: 51-65J.K.
[30] Uhlmann, “Satisfying General Proximity/Similarity Queries with Metric Trees”, *Information Processing Letter*, November 25, 1991, Vol. 40(4), pp.175-179.
[31] W. Xu, W.J. Briggs, J. Padolina, W. Liu, C.R. Linder, and D.P. Miranker, “Using MoBIoS' Scalable Genome Joins to Find Conserved Primer Pair Candidates Between Two Genomes”, to appear in *proceedings of 12th International Conference on Intellegent system for Molecular Biology*, Galsgow, UK, July31-Aug05, 2004..
[32] W. Xu, and D.P. Miranker, “A metric model for amino acid substitution”, in press *Bioinformatics*, 2003.
[33] P. Yianilos, “Data structures and algorithms for nearest neighbor search in general metric spaces”, In *Proc. 4th ACM-SIAM. Symposium on Discrete Algorithms (SODA'93)*, 1993, pp. 311-32