Similarity Search on Bregman Divergence: Towards Non-Metric Indexing

Zhenjie Zhang[1] Beng Chin Ooi[1] Srinivasan Parthasarathy[2] Anthony K.H. Tung[1]

[1]School of Computing, National U. of Singapore, {zhenjie,ooibc,atung}@comp.nus.edu.sg
[2]Computer Science and Engineering, Ohio State University, srini@cse.osu.edu

ABSTRACT

In this paper, we examine the problem of indexing over non-metric distance functions. In particular, we focus on a general class of distance functions, namely *Bregman Divergence* [6], to support nearest neighbor and range queries. Distance functions such as KL-divergence and Itakura-Saito distance, are special cases of Bregman divergence, with wide applications in statistics, speech recognition and time series analysis among others. Unlike in metric spaces, key properties such as triangle inequality and distance symmetry do not hold for such distance functions. A direct adaptation of existing indexing infrastructure developed for metric spaces is thus not possible. We devise a novel solution to handle this class of distance measures by expanding and mapping points in the original space to a new *extended* space. Subsequently, we show how state-of-the-art tree-based indexing methods, for low to moderate dimensional datasets, and vector approximation file (VA-file) methods, for high dimensional datasets, can be adapted on this *extended space* to answer such queries efficiently. Improved distance bounding techniques and distribution-based index optimization are also introduced to improve the performance of query answering and index construction respectively, which can be applied on both the R-trees and VA files. Extensive experiments are conducted to validate our approach on a variety of datasets and a range of Bregman divergence functions.

1. INTRODUCTION

Efficient indexing for similarity search is a well studied topic in different contexts, including database [3, 13, 26], machine learning [4] and computational geometry [1]. While extensive efforts have been devoted to index structure design for similarity search, most of the existing work in this area target metric spaces, e.g. Euclidean space. The distance functions in these metric spaces strictly satisfy some important properties, including non-negativity, symmetry and triangle inequality, which are fully exploited in existing index structures.

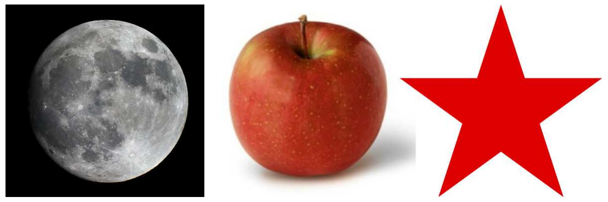


Figure 1: Examples of images for metric comparison

However, in a number of real world applications, the distance measure of choice is often not a metric. Such measures arise when comparing probability distributions, time series, images and matrices among others. To illustrate the ubiquitous nature of this concept consider Figure 1. When viewing the first and the second image (shape) and again when viewing the second and third image (color) one is able to immediately perceive a notion of similarity. On the other hand when viewing the first and third image one is forced to conclude that there is basically no similarity between the two images. Our notion of similarity in this context violates the notion of triangular inequality and illustrates that human beings are often comfortable when deploying or using non-metric dissimilarity measures instead of metric ones especially on complex data types [21]. To support such notions of similarity in a database context however is challenging.

First, as noted earlier, most indexing schemes implicitly or explicitly rely on features like triangle inequality and symmetry to deliver good performance. Second, even if one is able to design a specialized structure for a particular application it may not have general utility or interoperability across other domains. The difficulties stem from the fact that most of these non-metric spaces employ totally different representations, requiring the implementation of specialized index structures supporting each of them. Third, even if an interoperable solution is designed it may require a grounds up change to current database infrastructure that may limit its widespread adaptation and use.

In this paper we overcome these challenges, by designing a unified indexing mechanism for a general class of non-metric distance measures. The specific nature of our solution relies on effectively adapting existing indexing infrastructure in modern database systems with minimal changes, specifically to support similarity search on the general class of *Bregman divergence* measures.

First introduced by Bregman [6] as a way to solve convex programming (a subclass of nonlinear programming that

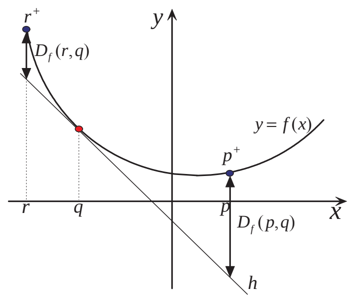


Figure 2: Geometrical interpretation of Bregman divergence

unifies and generalizes least squares, linear programming, and convex quadratic programming), Bregman divergence has attracted interest from a wide range of fields [5, 2, 11, 19, 17] because of its ease of representation and good extensibility in a number of areas such as text analysis[20], image retrieval[11, 21], motion tracking[5], graph analysis[17], and speech recognition[19, 17, 14]. We next briefly review the definition of Bregman divergence.

Assume we are given a d-dimensional space $\mathcal{S}$ on which each object p in the space is represented by a vector $p = (p[1], p[2], \ldots, p[d])$. Every Bregman divergence in $\mathcal{S}$ is associated with a convex function $f(x) : \mathcal{S} \to \mathcal{R}$, mapping objects in $\mathcal{S}$ to real numbers. Given objects p and q in $\mathcal{S}$, their dissimilarity w.r.t. the function $f(x)$ is defined as:

$$D_f(p,q) = f(p) - f(q) - \langle \nabla f(q), p - q \rangle \qquad (1)$$

Here, $\nabla f(q)$ is the gradient of the function $f(x)$ at q, and $\langle x, y \rangle$ denotes the dot product between two vectors. Intuitively, Equation (1) can be explained as the difference between $f(p)$ and the first-order Taylor expansion of the function $f(x)$ at position q. In Figure 2, the geometrical interpretation of Bregman divergence in 1-dimensional space $\mathcal{S}$ is illustrated. In this figure, the function curve $y = f(x)$ and the line $h : y = f(q) + \langle \nabla f(q), x - q \rangle$, both through the point $(q, f(q))$ in the figure, are plotted separately. The vertical difference at position $x = p$ (resp. $x = r$) is the measure of Bregman divergence, $D_f(p,q)$ (resp. $D_f(r,q)$). Bregman divergence admits several well known divergence functions when employing different convex function $f(x)$ in the definition. These include:

Example 1. ***KL-Divergence***
When $f(x) = \sum_{i=1}^{d} x[i] \log x[i]$, the distance is KL-divergence, i.e. $D_f(p,q) = \sum_{i=1}^{d} p[i] \log \frac{p[i]}{q[i]}$. KL-divergence is a common measure used for comparing distributions[17].

Example 2. ***Itakura-Saito Distance***
When $f(x) = -\log x$, the distance function becomes Itakura-Saito distance, i.e. $D_f(p,q) = \frac{x}{y} - \log \frac{x}{y} - 1$. Itakura-Saito distance is a common measure for representing the difference between two signals in frequency domain and is often applied in speech recognition [19].

Example 3. ***Von Neumann Entropy***
Given two semi-definite matrices X and Y, when $f(X) = tr(X \log X - X)$, the distance $D_f(X,Y) = tr(X \log X - X \log Y - X + Y)$. Von Neumann Entropy is used to measure the difference between two semi-definite kernel mappings [16].

In all of the examples above, despite their differences in definition, nearest neighbor and range queries play important roles in their applications. In *probabilistic databases* [23], for example, uncertain tuples with similar distributions (with KL-divergence) are of special interests since they may have stronger correlations. Similarity search on large signal and time series database (with Itakura-Saito distance) and matrix set (with Von Neumann Entropy) also dramatically decides the efficiency of speech recognition and kernel learning algorithms respectively.

To show a more specific example, in Table 1, we present three vectors generated from the DBLP data set[1] with each representing an author. Each dimension of the vector represents the probability of the author publishing in a major field of computer science. From the record of Jim Gray, it is apparent that he is closely connected to database community.

If Euclidean distance is applied on the table to search for an author who is similar to Jim Gray, the distances between Jim Gray and R1/R2 are exactly the same. However, if the KL-divergence is applied instead, the distances from Jim Gray to R1 and R2 become 0.08 and 0.014 respectively implying that R1 is more similar to Jim Gray than R2. This can be explained by the fact that both Jim Gray and R1 has higher probability of publication on System and lower probability of publication on Bioinformatics while R2 have equal probability to publish in almost all the fields except Database. KL-divergence generally captures such kinds of dissimilarities between distributions more accurately than Euclidean distance, since it gives higher weights to dimensions with larger probabilities.

In this paper, we propose a framework for supporting Bregman divergence in an effective way with existing index structures that are well tested and commonly supported by current DBMS. To overcome the non-metric dilemma we rely on an important observation about Bregman divergences and transform data points in the original d-dimensional space into an extended $d+1$ dimensional space. Subsequently we show how existing indexing infrastructure (R-trees and VA-files) can be leveraged with minimal change on this extended space to efficiently support nearest neighbor and range queries on the general class of Bregman divergences. A key element here is a novel query transformation method to facilitate effective pruning on both the R-Tree and VA file. Finally, we perform detailed analysis to show that contrary to metric space, the performance of indexes under Bregman divergence can be very sensitive to the query workload. We show that some knowledge on the query distribution can be exploited to improve indexing structure dramatically.

In short, the contributions of this paper are as follow:

1. We present a general framework to index Bregman divergence with existing index structures, such as the R-tree and VA file.
2. We re-formulate the problem of Bregman divergence as a projection problem in an extended space, and derive better pruning strategies for existing index structures.
3. We propose a distortion model with query distribution and show how it can be utilized in the construction of both the R-Tree and VA file.

[1]Details on how these vectors are computed can be found in Section 6

Author	AI	Application	Bioinformatics	Database	Hardware	Software	System	Theory
Jim Gray	0.109	0.109	0.059	0.314	0.0987	0.091	0.123	0.093
R1	0.141	0.101	0.069	0.276	0.094	0.089	0.123	0.103
R2	0.1	0.1	0.1	0.299	0.1	0.1	0.1	0.1

Table 1: An example data set. Like Jim Gray, R1 has higher chance of being associated with System than Bioinformatics while R2 have almost equal distribution in all dimensions except Database.

Notation	Explanation
$\mathcal{S}$	the space for data points
d	dimensionality of $\mathcal{S}$
x, p, r	data points
q	query point
$f(x)$	a convex function defined in $\mathcal{S}$
$f'(x)[i]$	partial derivative of $f(x)$ on $x[i]$, i.e. $\frac{\partial f(x)}{\partial x[i]}$
$D_f(x, q)$	Bregman divergence defined on $f(x)$
$\nabla f(x)$	gradient of $f(x)$ at x, i.e. $\nabla f(x) = (f'(x)[1], \ldots, f'(x)[d])$
$\langle x, y \rangle$	dot product between two vectors x and y
$\mathcal{S}^+$	extended space with $d+1$ dimensions
x^+, p^+, r^+	corresponding mapping point of x, p, r in $\mathcal{S}^+$
b_i	number of representation bits on dimension i in VA-File
R	bounding rectangle in $\mathcal{S}^+$
$LB(R, q)$	lower bound on distance from R to q
$UB(R, q)$	upper bound on distance from R to q
$v^i_{\max}$	maximal value of all points on dimension i
$v^i_{\min}$	minimal value of all points on dimension i
A^P_i	the expectation on the positive values of $v_q[i]$
A^N_i	the expectation on the negative values of $v_q[i]$

Table 2: Notation Table

4. We analyze the performance of the proposed methods with extensive experimental studies.

To improve the readability, we summarize the notations in Table 2.

2. RELATED WORK

Applications of Bregman Divergence: KL-divergence, is utilized frequently in text analysis [20], image classification [21] and content-based image retrieval problems [22]. Recently, Long *et al.*[17], propose a new graph partitioning method, that relies on KL-divergence as a distance measure. In the context of kernel methods, Kulis *et al.*[16] present a new technique that relies on Von Neumann Entropy. The Itakura-Saito distance [14], is often employed to measure the difference between signals [12], with applications in speech recognition.

Recently, researchers have examined the use of Bregman divergence for generalized clustering problems [2]. In a related context [19], Nielsen and Rock focus on discovering the Bregman ball with the minimal radius to cover the given point set for a better summarization (clustering) of speech data.

In computer vision and multimedia systems, KL-divergence is usually adopted to compare objects. For example, in [21], different color and textures are compared using different measures, among which KL-divergence shows promising performance. In [11], Goldberger et al. modeled the images with Gaussian Mixture Models, and measured the difference between images by computing the KL-divergence between Gaussian components. In [5], Boltz proposed a region-of-interest (ROI) tracking method that relied on a distribution-based representation. The target can be efficiently discovered in videos by searching for k-th nearest neighbor of objects in the frames, with KL-divergence as the underlying distance measure. The work presented in this article *can be effectively utilized to speed up such applications.*

Indexing for Similarity Search: Indexing is a well studied topic in the database community. KD-Tree [10] is one of the first index structures proposed for nearest neighbor and range search. In database systems, R-Tree and its variants [13, 3] are well recognized as an effective index structure for multi-dimensional indexing and can dramatically reduce the disk I/O operations for query processing when the dimensionality of the domain is low to moderate. In high dimensional spaces, VA file and its variants [26, 9], are typically the indexing methods of choice and are typically shown to outperform tree-based index structures. As noted earlier, most previous proposals on indexing in the database community implicitly rely on the notion of a *metric*. In iDistance [15], B$^+$-tree is adopted to index high-dimensional data by mapping the objects onto some space filling curve, such as Hilbert curve and Z-curve.

Some specific non-metric distances have been studied individually in database research community. The k-match distance [25], for example, measures the distance between two d-dimensional records depending on the minimal k out of d dimensions. In [8], some embedding method was introduced to index distance function without triangle inequality. Both of the studies mentioned above cannot be implemented directly in commercial database system and not applicable to Bregman divergence.

The work most closely related to the theme of this paper is recent work by Cayton on Bregman Ball (BB-)Trees[7]. BB-tree is a main-memory baseed data structure where every node is associated with a Bregman ball that covers all data points indexed in the subtree rooting at the node. The tree is constructed from root to leaves recursively. K-means clustering is employed to generate children nodes from the parent. For nearest neighbor query, despite the lack of triangle inequality, the minimal and maximal distances from a Bregman ball to the query point can be estimated by a binary search on the line connecting the ball center and the query. This facilitates the pruning of nodes that will not contain any potential nearest neighbor points.

This approach has several drawbacks. First, the construction of Bregman ball on large datasets is expensive. The BB-tree relies on a clustering step, which can be very slow and often unstable. Second, updating the BB-tree is extremely expensive with respect to both insertion and deletion operations. Third, complete infrastructure modification is needed if any commercial database needs to incorporate Bregman ball tree within existing system. In this work we provide a *scalable solution that addresses these limitations.*

3. GENERAL INDEXING FRAMEWORKS

Our objective is to provide efficient access methods to support range and k-nearest neighbor queries for *different* instantiations of Bregman divergence i.e. our solution should be applicable regardless of the convex function selected.

In this paper, we first present a simple and general scheme that enables the employment of traditional Euclidean space indexing techniques, such as R-tree and VA file to be directly applied for *any Bregman divergence*. Specifically, our scheme involves a vector transformation which creates a mapping from the original d-dimensional space $\mathcal{S}$ to some new $(d+1)$-dimensional space $\mathcal{S}^+$. For every point $x = (x_1, x_2, \ldots, x_d)$ in $\mathcal{S}$, it is mapped to another point $x^+ = (x_1, x_2, \ldots, x_d, f(x))$ in the new space $\mathcal{S}^+$.

Intuitively, the new space $\mathcal{S}^+$ includes an additional dimension containing information on the function $f(x)$. Using the example in Table 1 and adopting KL-divergence, the record of Jim Gray will be transformed to a 9-dimensional vector with value $\sum_{i=1}^{8} x[i] \log x[i] = -0.841$ on the additional dimension. In Figure 2, p^+ and q^+ are 2-dimensional points on the curve $y = f(x)$.

All the index structures proposed in this paper are based on the extended space $\mathcal{S}^+$. Both the R-tree and VA file rely on some *bounding rectangle* to prune points that cannot be in the query results. A bounding rectangle R usually consists of $d \times 2$ boundaries, where d is the dimensionality of the space. For each dimension, an upper boundary $R.u[i]$ and lower boundary $R.l[i]$ indicate the maximal and minimal values of the points stored in R. For example, the *Minimum Bounding Rectangle*(MBR) in the R-tree and the *Cell* in the VA file are two special cases of the general bounding rectangle. Given a query point q, the effectiveness of these index structures depend on the estimation of the distance from points in a bounding rectangle to the query point q. In particular, the computation of lower and upper bound on the distance must be efficiently supported. In the following lemma, we will show that Bregman divergence on bounding rectangles in the extended space $\mathcal{S}^+$ can also be computed easily like in Euclidean space.

LEMMA 3.1. *Given a bounding rectangle R in $\mathcal{S}^+$ and a query point q in $\mathcal{S}$, lower bound and upper bound on Bregman divergence from any point in R to q can be computed in $O(d)$ time, where d is the dimensionality of $\mathcal{S}$.*

PROOF. Assume that the lower bound and upper bound of R on dimension i are $R.l[i]$ and $R.u[i]$ respectively. For each point $x \in \mathcal{S}$ and its corresponding mapping location $x^+ \in \mathcal{S}^+$, x^+ is covered by R, if and only if $R.l[i] \le x^+[i] = x[i] \le R.u[i]$ for $1 \le i \le d$ and $R.l[d+1] \le x^+[d+1] = f(x) \le R.u[d+1]$.

Given the query point $q = (q[1], \ldots, q[d])$ and the gradient $\nabla f(x)|_{x=q} = (f'(q[1]), \ldots, f'(q[d]))$ with respect to the function $f(x)$, we have

$$\begin{aligned} D_f(x,q) &= f(x) - f(q) - \sum_{i=1}^{d} f'(q[i]) * (x[i] - q[i]) \\ &= x^+[d+1] - f(q) - \sum_{i=1}^{d} f'(q[i]) * (x^+[i] - q[i]) \\ &\le R.u[d+1] - \sum_{i=1}^{d} \min\{f'(q[i])R.l[i], f'(q[i])R.u[i]\} - \\ &\quad f(q) + \sum_{i=1}^{d} f'(q[i])q[i] \end{aligned}$$

Algorithm 1 K-Nearest Neighbor Search (R-tree T, Query q)

```
1: Set the k-nearest neighbor set $S$ as empty
2: Set threshold distance $\theta = \infty$
3: Clear a priority queue $Q$
4: Enqueue the root of $T$ into $Q$
5: while $Q$ is not empty do
6:   Dequeue the head node $N$ from $Q$
7:   if $N$ is a leaf node then
8:     for each point $p$ stored in $N$ do
9:       if $D_f(p,q) < \theta$ then
10:        Insert $p$ into $S$ and update $\theta$
11:  else
12:    for each child node $M$ of $N$ do
13:      Retrieve the MBR $R$ of $M$
14:      if $LB(R,q) < \theta$ then
15:        Enqueue $M$ into $Q$ with $LB(R,q)$
16: Return points in $S$
```

Since $f(q) - \sum_i f'(q[i])q[i]$ is a constant and independent of x, the upper bound only involves finding the smaller value between $f'(q[i])R.l[i]$ and $f'(q[i])R.u[i]$ for each dimension, which takes $O(d)$ time. Similarly for the lower bound we have:

$$\begin{aligned} D_f(x,q) \ge\ & R.l[d+1] - \\ & \sum_{i=1}^{d} \max\{f'(q[i])R.l[i], f'(q[i])R.u[i]\} - \\ & f(q) + \sum_{i=1}^{d} f'(q[i])q[i] \end{aligned}$$

This completes the proof of the lemma. □

In the rest of paper, we use $LB(R,q)$ and $UB(R,q)$ to denote the lower bound and upper bound on the distance from R to q respectively, as derived in the lemma above. We next show how the above lemma facilitates the employment of the R-tree and VA file as the underlying index structure for nearest neighbor and range query search.

3.1 R-tree Index

To handle Bregman divergence, the R-tree is built on the extended $(d+1)$-dimensional space $\mathcal{S}^+$ by transforming each point $x \in \mathcal{S}$ to the new point $x^+ \in \mathcal{S}^+$ as described earlier. The insertions and deletion of points in the tree follow the traditional strategies in the literature of the R-tree. For each intermediate node N in the tree, an MBR is constructed to approximate the locations of all points stored in the subtree, while leaf node is linked to a disk page containing all original points for the index. In Algorithm 1, we present the details on how to discover the k nearest neighbor to the given query point q. The algorithm essentially follows the branch-and-bound strategy in traditional query processing with one exception that in line 14 we leverage the lower bound estimation from Lemma 3.1. Due to limited space, we skip the details on range query search with the R-tree structure since it requires only some simple extensions to the above nearest neighbor search algorithm.

3.2 VA file Index

The VA file is another popular index structure, that is shown to be effective, especially on high dimensional data. The general idea of the VA file is to partition the indexed

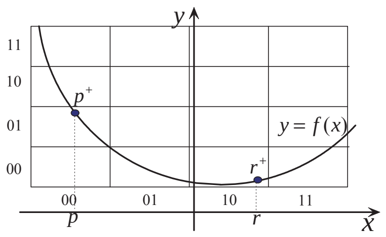


Figure 3: Example of VA file

space into small cells. Points in the same cell are represented by the a bit string called *Vector Approximation* which can approximate the location of the cell. On each dimension i, a positive integer b_i is specified, dictating that dimension i should be partitioned into 2^{b_i} intervals of equal length. Each of these interval can then be uniquely represented by a bit string of length b_i based on their ordering. Given any point, its vector approximation can be computed by concatenating the bit string of the interval that it falls in for each dimension.

For example, in Figure 3, given the 2-dimensional space $\mathcal{S}^+$ and $b_i = 2$ for both dimensions, the space with the indexed points is divided into $\prod 2^{b_i} = 16$ cells. Point p^+ and r^+ are assigned with bit string "0001" and "1000" respectively, according to the definition above.

The vector approximations of all points are stored in a single file, called a VA file, in the order as in its original storage structure. A reverse mapping from the bit string to the cell is maintained by keeping the boundary values on all dimensions. When answering queries, the vector approximations are loaded from the VA file into main memory. By retrieving the cell containing the point p^+ with the reverse mapping, the system decides if p^+ is likely to be included in the result set by using the lower bound and upper bound distance from the cell to the query q. If the lower bound distance is already larger than the current threshold, this point p^+ is pruned, otherwise p^+'s identifier is stored as a candidate point. After scanning the complete VA file, random accesses to the original storage structure is conducted to verify the remaining candidate points.

In particular, the exact values of these candidate points are retrieved in ascending order of their lower bound distance to the query and the search stops when minimal lower bound distance is larger than the current maximal distance of k-nearest neighbor result set. The details of the query answering algorithm for k-nearest neighbor are summarized in Algorithm 2.

Similar to the R-tree index structure, the lower bound and upper bound computation depends on the correctness of Lemma 3.1. We also skip the details of range search algorithm with VA file here.

4. PROBLEM REFORMULATION AND IMPROVED BOUNDS

We next present a new geometrical interpretation of the Bregman divergence which allows us to transform the distance measure into the dot product of two vectors in the extended space $\mathcal{S}^+$ [2]. Based on this interpretation, we derive a new formulation of the nearest neighbor and range query problems over Bregman divergence. We will then derive tighter lower and upper bound estimation for the distance between a query point and a bounding rectangle based on this new formulation.

[2]Incidentally, transforming the distance measure into dot product also implies that our studies can be extended to the domain of kernel methods in the future. We will only discuss this briefly in Section 7 due to lack of space.

Algorithm 2 K-Nearest Neighbor Search (VA file F, Original Database D, query q)

```
1: Clear Candidate Set $CS$
2: Set threshold $\theta = \infty$
3: for each vector approximation $a_i \in F$ do
4:   Retrieve the cell $R$ covering $a_i$
5:   Calculate $LB(R,q)$ and $UB(R,q)$
6:   if $LB(R,q) < \theta$ then
7:     Insert $a_i$ into the $CS$
8:     Find the new $k$th smallest upper bound as new $\theta$
9: Sort the points in $CS$ with non-descending order of lower
   bound distance
10: Clear Result Set $RS$
11: Set threshold $\theta = \infty$
12: for each $a_i$ in $CS$ do
13:   Retrieve the original point $p$ of $a_i$ in $D$
14:   Calculate $D_f(p,q)$
15:   if $D_f(p,q) < \theta$ then
16:     Update result set $RS$ and threshold $\theta$
17: Return points in $RS$
```

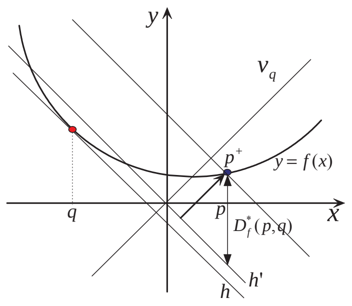


Figure 4: Geometrical Interpretation

We will use Figure 4 to illustrate our query reformulation. Given the query point q in the figure, the distance between a point p and q is defined be the vertical difference between the point $(p, f(p))$ on curve $y = f(x)$ and the point $(p, f(q) + \langle \nabla f(q), p - q \rangle)$ on h, which is tangent to the curve $y = f(x)$ at $(q, f(q))$.

Let h' be the line that is parallel to h and passes through the origin. h' is a distance of $f(q) - \langle \nabla f(q), q \rangle$ above h along the extended dimension y in $\mathcal{S}^+$. Correspondingly, the distance between the point $(p, f(p))$ and h' can be computed as $D^*_f(p,q) = f(p) - \langle \nabla f(q), p \rangle$.

It is easy to verify that $D^*_f(p,q)$ is proportional to the projection length of the point $p^+ = (p, f(p))$ on the vector v_q, where $v_q = (-\nabla f(q), 1)$ is orthogonal to h and h'. This projection length can be computed as the dot product between p^+ and v_q i.e. $\langle p^+, v_q \rangle$. In the rest of the paper, we will refer to v_q as a *query vector* as there is a one-to-one mapping between a query point q and a query vector v_q. Based on this observation, we will reformulate the nearest neighbor and range query problem using $\langle p^+, v_q \rangle$. In the following, we formally present and prove the reformulation using the next two lemmas.

LEMMA 4.1. *Given a data set P, a query point q and a*

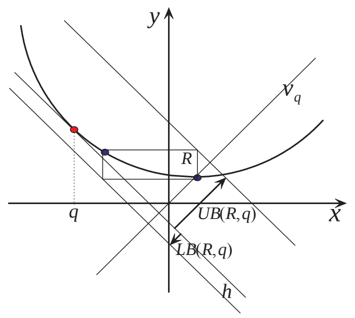


Figure 5: Example of naive lower and upper bounds

Bregman divergence D_f, the nearest neighbor can be represented as the minimizer of

$$NN(P, q, D_f) = \min_{p \in P} \langle p^+, v_q \rangle$$

Proof. Based on the definition of nearest neighbor, the nearest neighbor aims to minimize $D_f(p, q)$ for any $p \in P$. The distance between p and q can be re-formulated as:

$$\begin{aligned} D_f(p,q) &= f(p) - f(q) - \langle \nabla f(q), p - q \rangle \\ &= f(p) - \langle \nabla f(q), p \rangle + \langle \nabla f(q), q \rangle - f(q) \end{aligned}$$

Since $\langle \nabla f(q), q \rangle - f(q)$ is a constant for any $D_f(p, q)$, p is the nearest neighbor to q iff p can minimize $f(p) - \langle \nabla f(q), p \rangle$. By the definitions of p^+ and v_q, we have $f(p) - \langle \nabla f(q), p \rangle = \langle p^+, v_q \rangle$, which completes the proof of the lemma. □

Lemma 4.2. *Given a data set P, a query point q, a Bergman divergence D_f and a threshold θ, any point p is in the range query result if and only if*

$$\langle p^+, v_q \rangle \leq \theta + f(q) - \langle \nabla f(q), q \rangle$$

The proof of Lemma 4.2 is similar to that of Lemma 4.1.

The benefits of this reformulation include the fact that projections between vectors are easier to visualize and the fact that vectors are easier for distribution analysis (covered in Section 5).

Revisiting the Rectangular Bounds: In the previous section, we derived a lower bound and upper bound on the distance between a point in some bounding rectangle to some query point q. Here, we will first analyze the tightness of the bound within the context of the new geometrical interpretation introduced above. We then demonstrate how better bounds can be derived to improve the effectiveness of pruning for both the R-tree and the VA file structures.

As mentioned before, the Bregman divergence between a point p and a query point q can be interpreted as the projection length from p^+ to the vector v_q. The lower and upper bound derived in Lemma 3.1 can be regarded as finding the minimum and maximum projection length on v_q from the rectangle R. In Figure 5, for example, we plot a bounding rectangles covering a group of points. The lower bound and upper bounds of the rectangle can be found by projecting the whole rectangle onto the query vector v_q. Note that the lower bound can be a negative value, such as the lower bound of the rectangle R in Figure 5.

These bounds however can be improved by considering that any point in the rectangle must ultimately stay on the curve $y = f(x)$ in the extended space. The improved scheme of bound estimation stems from the observation that $f(x)$ is usually the sum of some function $g(x[i])$ on each dimension i. If such function $g(x[i])$ exists, given a bounding rectangle R with its lower and upper boundaries on all dimensions, we can analytically calculate the exact minimal and maximal values of $g(x[i]) - q[i]x[i]$ for each i. By summing up all these extreme values, the exact minimal and maximal distances can be found for the bounding rectangle R. In other words:

$$LB^*(R, q) = \sum_{i=1}^{d} \min_{R.l[i] \leq x[i] \leq R.u[i]} (g(x[i]) - q[i]x[i])$$

and,

$$UB^*(R, q) = \sum_{i=1}^{d} \max_{R.l[i] \leq x[i] \leq R.u[i]} (g(x[i]) - q[i]x[i])$$

Taking KL-divergence as an example, the convex function is defined as $f(x) = \sum_i x[i] \log x[i]$. Thus, the function $g(x[i]) = x[i] \log x[i]$ is able to satisfy our condition above. This works for Itakura-Saito distance as well with $g(x[i]) = -\log x[i]$.

Leveraging some simple calculus, we know that the derivative of $g(x[i]) - q[i]x[i]$ is $g'(x[i]) - q[i]$. Since $f(x)$ is a convex function, $g'(x[i])$ must be some monotonic function on $x[i]$, as well as $g'(x[i]) - q[i]$. Thus, the extreme values of $g(x[i]) - q[i]x[i]$, for both maximum and minimum, can be attained on three possible locations along the interval of R on dimension i. These locations include the two ending points of the interval, $R.l[i]$ and $R.u[i]$, as well as the location with zero derivative, i.e. some $x[i]$ satisfying $g'(x[i]) - q[i] = 0$. While the ending points can be found directly from the interval information, the zero-derivative location will require the use of a binary search operation along the interval $[R.l[i], R.u[i]]$. Fortunately, for most of the popular convex functions, this location can be pre-computed explicitly in constant time. For KL-divergence, for example, with the function $g(x[i]) = x[i] \log x[i]$, it is easy to verify that $x[i] = \exp(q[i] - 1)$ is the location to satisfy $g'(x[i]) - q[i] = 0$. For Itakura-Saito distance, as another example, with the function $g(x[i]) = -\log x[i]$, the zero-derivative location is $x[i] = -\frac{1}{q[i]}$. Note that there exists only one location with zero derivative on the whole dimensions. Therefore, this location may not be covered by every bounding rectangle. After testing the three (or two) candidate locations, the maximal and minimal values on dimension i can be directly computed.

Using this strategy, the lower and upper bound computed is much tighter than the naive bounding method. In Figure 6, we present the new bounds for the same bounding rectangle R in Figure 5. The new bounds $LB^*(R, q)$ and $UB^*(R, q)$ are achieved at the ending points of the interval on x-axis, respectively. While the naive lower bound in Figure 5 is negative, the new lower bound $LB^*(R, q)$ becomes positive. Thus, the tightness on the bound from points in R to query q is dramatically improved.

5. OPTIMIZATION WITH QUERY DISTRIBUTION

In traditional indexing problems over Euclidean distance, query distribution often plays a limited role. Basically, adaptive optimization with query distribution does not greatly enhance the query processing efficiency, but actually incurs

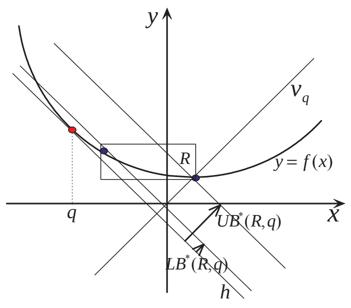


Figure 6: Example of tighter bounds

huge costs on index construction and maintenance. However, when indexing over Bregman divergence, we argue that query distribution can play a greater role in index structure optimization. We make our case using analysis that depends on the distance distortion of bounding rectangles and through the maintenance of relevant summary statistics of size $O(d)$. Before delving into the details of our analysis and method, some definition on the distortion of bounding rectangles is stated below first.

Definition 5.1. *Given a bounding rectangle R, the expected distortion of R with respect to some query distribution* [3] *$\mathcal{Q}$ is the expected difference between the lower bound and upper bound on the distance from R to the query q following $\mathcal{Q}$, i.e.*

$$D(R, \mathcal{Q}) = E(UB(R, q) - LB(R, q))$$

Intuitively, the expected distortion, measures the variance on the distance from points in the bounding rectangle to the queries. When the distortion is small, the points are prone to having similar distance with respect to the queries. This can improve the pruning efficiency of our algorithms. To reiterate, in Euclidean space, the expected distortion with respect to the query distribution is not that important, since the value of the expected distortion can be bounded by the boundaries of the rectangle itself. The following lemma proves this statement rigorously.

Lemma 5.1. *In Euclidean space, the expected distortion of a bounding rectangle R can be bounded by*

$$D(R, \mathcal{Q}) \leq \sqrt{\sum_{i=1}^{d} (R.u[i] - R.l[i])^2}$$

Proof. Since Euclidean space obeys triangle inequality, the difference between the lower bound and upper bound w.r.t. a query q is no larger than the maximum distance between any two points in R,

$$UB(R, q) - LB(R, q) \leq \sqrt{\sum (R.u[i] - R.l[i])^2}$$

Assuming the probability density function $p(x)$ for the query distribution $\mathcal{Q}$, the expected distortion can be measured by employing basic calculus as follows:

[3] $\mathcal{Q}$ can either be represented by some continuous statistical function or by discrete samples of the query points.

$$\begin{aligned} D(R, \mathcal{Q}) &= \int_x (UB(R, x) - LB(R, x)\, dp(x) \\ &\leq \int_x \sqrt{\sum (R.u[i] - R.l[i])^2}\, dp(x) \\ &= \sqrt{\sum (R.u[i] - R.l[i])^2} \end{aligned}$$

This completes the proof of the lemma. □

The above lemma implies that query distribution does not have great impact on the expected distortion of the bounding rectangle in Euclidean spaces. Unfortunately, this property does not hold for Bregman divergence. Given some Bregman divergence $D_f(x, q)$ and some bounding rectangle R for data points, there does not exist a constant to bound the expected distortion. In KL-divergence, for example, the distance tends to be infinity between a query point q and a bounding rectangle R if $q[i]$ is close to zero for some dimension i and $R.l[i] > 0$. This highlights the difficulty in constructing and maintaining an index structure for Bregman divergence. However, it also enables us to reason about improving performance with some knowledge of the query distribution. We next describe the summary statistics we maintain in order to estimate the distortion.

Given a query distribution $\mathcal{Q}$, we use A_i^P (resp. A_i^N) to denote the average value of the query vector v_q on dimension i, with q following the distribution $\mathcal{Q}$ and $v_q[i] > 0$ (resp. $v_q[i] \leq 0$). Therefore, the query statistic set contains $d+1$ pairs of (A_i^P, A_i^N) for $1 \leq i \leq d+1$. The following lemma proves that these statistics are sufficient to estimate the expected distortion of a bounding rectangle R.

Lemma 5.2. *Given $\cup_{i=1}^{d+1} \{A_i^P, A_i^N\}$ and a bounding rectangle R, the expected distortion of R is bounded by the following inequality*

$$D(R, \mathcal{Q}) \leq \sum_{i=1}^{d+1} (R.u[i] - R.l[i])(A_i^P - A_i^N)$$

Proof. From the definition of expected distortion, if the query distribution $\mathcal{Q}$ has probability density function $\sigma(x)$ for query vector x, we have

$$D(R, \mathcal{Q}) = \int_x UB(R, x)\sigma(x)\, dx - \int_x LB(R, x)\sigma(x)\, dx$$

In the following, we use $\sigma_i(x_i)$ to denote the probability density function of $\mathcal{Q}$ on dimension i. Simply employing the naive bound derived in Section 3, we have the expected upper bound as follows.

$$\begin{aligned} & \int_x UB(R, x)\, dx \\ \leq & \sum_{i=1}^{d+1} \left(\int_{x_i > 0} R.u[i] x_i \sigma_i(x_i)\, dx_i + \int_{x_i \leq 0} R.l[i] x_i \sigma_i(x_i)\, dx_i \right) \\ = & \sum_{i=1}^{d+1} \left(R.u[i] \int_{x_i > 0} x_i \sigma_i(x_i)\, dx_i + R.l[i] \int_{x_i \leq 0} x_i \sigma_i(x_i)\, dx_i \right) \\ = & \sum_{i=1}^{d+1} \left(R.u[i] A_i^P + R.l[i] A_i^N \right) \qquad (2) \end{aligned}$$

The last equality is because $A_i^P = \int_{x_i > 0} x_i \sigma_i(x_i)\, dx_i$ and $A_i^N = \int_{x_i \leq 0} x_i \sigma_i(x_i)\, dx_i$. Similarly, we can derive the inequality on the lower bound

$$\int_x LB(R,x)\,dx \geq \sum_{i=1}^{d+1}\left(R.l[i]A_i^P + R.u[i]A_i^N\right) \quad (3)$$

By (2)-(3), we arrive at the conclusion of the lemma with simple algebra. □

The last lemma implies that the average positive and negative values on the query vectors contain enough information to help us estimate the distortion on bounding rectangles. The maintenance on these averages are affordable for most database systems, since both the storage and computation cost is negligible. This enables the system to organize the data either into bounding rectangles or cells such that query performance over these index can be improved substantially. In the rest of the section, we will specifically detail how we apply this technique on the R-tree and VA file respectively.

5.1 Application on the R-tree

In traditional R-tree and its variants, an important operation is the selection of nodes when inserting a new point into the index tree. The selection is optimized by picking up the node with the minimal expansion of the MBR. Different implementations employ different measures on the expansion of MBR, such as the volume and the sum of the edge length etc.

To incorporate our new distortion measure into the construction and maintenance of the R-tree, we just need to replace the existing expansion measure with the new expected distortion according to the query distribution statistics, as summarized above. The node, whose MBR needs the minimal increase of expected distortion to include the newly added point, is chosen as the insertion node.

When dealing with split operation, the seeds of the splitting nodes are also selected according to the expected distortion. In our experimental studies, the tree randomly picks up pairs of children nodes (or points), and finally selects the pair whose minimal bounding rectangle has the maximal expected distortion with respect to query distribution.

5.2 Application on VA file

The cell structure in the VA file is fairly different from the MBRs in the R-tree. While MBRs are frequently updated during the insertion and deletion operations, the boundaries on the cells in VA file are pre-defined before the computation of vector approximations. Updates on such boundaries are very expensive since the vector approximations of the whole data set have to be re-computed.

However, if the statistical information on the query distribution is available before the construction of the index structure, the system can then leverage this by selecting better boundaries potentially improving query processing performance. The main idea of this optimization technique is to minimize the maximal expected distortion of all cells by carefully selecting the number of bits on each dimension as well as the boundaries on the dimensions. The following lemma shows that equal division is optimal if the numbers of bits on all dimensions are fixed.

LEMMA 5.3. *If every dimension i is approximated with b_i bits, the optimal interval length on dimension i is $(v^i_{\max} - v^i_{\min})2^{-b_i}$.*

PROOF. If summing up the expected distortion of all cells, the following equation holds with Lemma 3.1, when iterating each cell R in the VA file.

$$\begin{aligned}\sum_R D(R,\mathcal{Q}) &= \sum_R\sum_{i=1}^{d+1}(R.u[i]-R.l[i])(A_i^P - A_i^N)\\ &= \sum_{i=1}^{d+1}\prod_{j\neq i}2^{b_j}(v^i_{\max}-v^i_{\min})(A_i^P - A_i^N)\end{aligned}$$

Leveraging the *pigeonhole principle*, the maximal expected distortion is no smaller than $\sum_R D(R,\mathcal{Q})\prod_{i=1}^d 2^{-bi}$. Combining this with the equality above, we obtain:

$$\max_R D(R,\mathcal{Q}) \geq \sum_{i=1}^{d+1}2^{-b_i}(v^i_{\max}-v^i_{\min})(A_i^P - A_i^N)$$

By applying equal division on all dimensions, the expected distortion of all cells can be computed as: $\sum_{i=1}^{d+1}2^{-b_i}(v^i_{\max}-v^i_{\min})(A_i^P - A_i^N)$. Therefore, equal division must be the optimal boundary selection. □

While the above lemma justifies the selection of equal width division, the next lemma implies how to optimally select number of bits on the dimensions.

LEMMA 5.4. *If the total number of bits on all dimensions is fixed, i.e. $\sum_i b_i = B$, the optimal number of bits b_i on dimension i is*

$$b_i = \frac{B - \sum_{i=1}^{d+1}\log_2 w_i}{d+1} + \log_2 w_i$$

with $w_i = (v^i_{\max}-v^i_{\min})(A_i^P - A_i^N)$.

PROOF. In last lemma, we have shown that expected distortion over all cells is no better than $\sum_{i=1}^{d+1}2^{-b_i}w_i$ with $w_i = (v^i_{\max}-v^i_{\min})(A_i^P - A_i^N)$. To optimize it with constraint $\sum b_i = B$, we apply the method of Lagrange multipliers on the expected distortion. With λ as the Lagrange operator on the constraint, the new objective function can be re-written as.

$$F = \sum 2^{-b_i}w_i + \lambda(\sum b_i - B)$$

To reach the optimal solution, each partial derivative on b_i must be zero, as

$$\frac{\partial F}{\partial b_i} = -\ln 2\cdot w_i 2^{-b_i} + \lambda = 0$$

Solving the equation array with $(d+1)$ equality, for each i we have

$$\log_2 w_i - b_i = \frac{\sum\log_2 w_i - B}{d+1}$$

Thus, we prove the lemma. □

This lemma can thus be directly applied on the selection of appropriate number of bits for each dimension, depending on $w_i = (v^i_{\max}-v^i_{\min})(A_i^P - A_i^N)$.

6. EXPERIMENTAL STUDIES

To evaluate the effectiveness of the methods proposed in this paper, we conduct extensive experimental studies on both synthetic and real data sets. We begin by describing the data sets.

KDD99 Data Set: This is a probability vector data set with 72 dimensions and 489794 tuples, constructed from the

KDD Cup 1999 dataset which has been used for research on intrusion detection and network attack analysis. The original data set contains 4898431 TCP packet records on 41 dimensions. In the pre-processing step, we transform the original records into a probability vector by recording the number of packets that belong to each of the 72 different types of network connections over a sliding window of 500 consecutive packets. By continuously moving the sliding window, a snapshot distribution is recorded and stored as a data point, after 10 new packets replace the 10 oldest packets. KL-divergence is the dissimilarity measure applied, which is helpful in discovering points in time with similar packet distributions.

DBLP Data Set: This is a probability vector data set with 8 dimensions and 279124 records, generated from the DBLP database retrieved in October 2007. The original data set contains a connected graph of 279124 authors with edges representing co-authoring relationship between the authors, as well as bipartite graph between authors and papers. To analyze the research area that the authors are working on, the number of top-tier conference papers in 8 different fields are counted for each author according to the conference ranking retrieved online[4]. To smoothen the vectors, we further calculate the average on the probabilities over the neighborhood of each author, which consists of authors within 10 hops on the co-author graph, as his final probability vector related to the fields. Similar to KDD99 data, we apply KL-divergence on this data set. [5]

Synthetic Data: Uniform Distribution: This dataset is generated in a manner such that the positions of the objects follow uniform distribution on a d-dimensional unit hyper-rectangle in domain $[0, 1]$ on each dimension.

Synthetic Data: Clustered Distribution This dataset is generated from a 4-component Gaussian Mixture Model [18]. The center of the components are uniformly selected in the unit hyper-rectangle $[0, 1]^d$ of dimensionality d. The variances on the dimensions are independently chosen in the interval $[0, 0.5]$. A noise uniformly distributed in the whole domain is also generated, which consists of 1% of the data set. The queries on the synthetic data are generated at random and we employ both the KL-divergence and the Itakura-Saito(IS) for experiments on this data.

In our experiments, we evaluate performance of the baseline *R-tree* and *VA file* solutions described in Section 3, as well as the optimized variants, namely *Improved Bounding* for better pruning (Section 4) and *Workload Optimization* for better index construction (Section 5). In the rest of the section, we use 'R' and 'V' to denote the baseline R-tree and VA file respectively, and use 'B' and 'Q' to denote the two techniques for improved bounding and workload optimization. Based on this notation, an algorithm marked "R-BQ" would refer to an R-tree combined with both optimizations.

We also compare the proposed methods against Bregman Ball tree[6] (denoted by "BBT") which is a main-memory algorithm for processing nearest neighbor search under Bregman divergence. Another baseline algorithm we compare with is the linear scan algorithm denoted "LS".

To verify the efficiency of these index structures, we measure the *Index Construction Time*, *Querying CPU Time* and *Querying I/O cost* in our experiments. The query points are randomly selected from the data set. The experiments are repeated 10 times to obtain the average results. Since BBT is a main-memory based algorithm that assumes all records are loaded into the memory, the Querying I/O Time is not measured for BBT.

[4]http://www3.ntu.edu.sg/home/ASSourav/crank.htm
[5]Both constructed datasets are available at: http://www.comp.nus.edu.sg/ zhangzh2/bregman-data.zip
[6]http://lcayton.ucsd.edu/bbtree-code.zip

Parameter	Varying Range
number of bits	4,6,8,10,**12**,14
dimensionality	2,4,**8**,16,32
result size k	5,**10**,20,40,80
search range θ	0.005,**0.01**,0.02,0.04,0.08
data size	**1M**,2M,3M,4M,5M
page size	**4KB**, 8KB, 16KB, 32KB

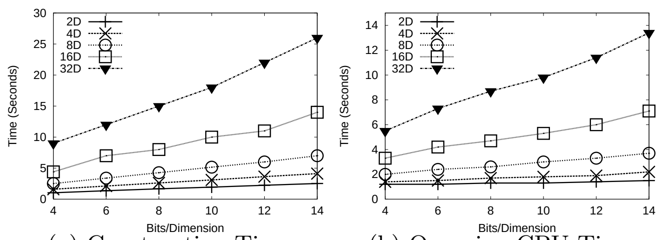
Table 3: Varying parameters

In Table 3, we summarize the parameters tested in our experiments, as well as their default values (in bold font) and ranges that we tested on. Note that the first parameter, number of bits is only applicable to the VA file, which is studied separately in the following as a form of parameter tuning before we compare VA file with other methods.

All the programs are compiled by gcc 3.4.3 in Red hat Linux Operating system and run on IBM x255 server with four Intel Xeon MP 3.0 GHz CPU, 18G DDR memory and six 73.4GB Ultra320 SCSI hard disks.

6.1 Effect of Number of Bits on K-NN Queries

In this subsection, we test the impact of number of bits on the efficiencies of VA file, and justify our selection of the default value on the number of bits in Table 3.

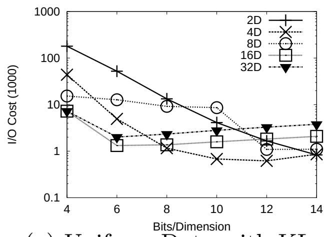


(a) Construction Time (b) Querying CPU Time

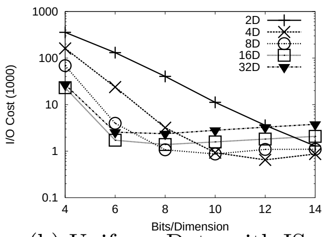
Figure 7: Effect of number of bits on Clustered Data with KL

Since the results on different distributions and distance functions are similar, in Figure 7, we only report the results of construction time and querying CPU cost of VA file on clustered data with KL. The figure shows that both construction cost and querying CPU cost increases linearly with respect to the number of bits on each dimension. This is because most of the CPU cycles are spent on calculating and interpreting the approximation bit string during index construction and query processing respectively.

Different from the construction and querying CPU costs, the I/O cost of VA file is no longer linear to the number of bits. In Figure 8, we show the I/O costs of VA file with different number of bits on data sets with various dimensionality. The results in the figure show that increase in the number of bits will help to reduce I/O cost when the dimensionality is low. However, when more bits are added into the vector approximation, the size of the bit string file



(a) Uniform Data with KL

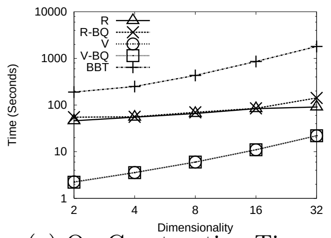


(b) Uniform Data with IS

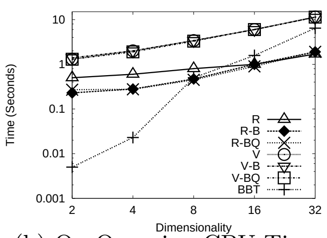
Figure 8: Effect of Bit Number on I/O Cost

grows substantially while the improvement on the pruning efficiency is limited. Thus, the I/O cost on the scan of the vector approximation file dominates the total I/O cost for k-nearest neighbor query on high dimensional space. This can be verified by the results in Figure 8(a) on uniformly distributed data, in which linear increase of I/O cost can be observed when there are more than 6 bits used on 16 and 32 dimensional space. Based on the results observed, we decided to use 12 bits for each dimension in the rest of our experiments, since it can achieve optimal or near-optimal performance on almost all data sets under evaluation.

6.2 Effect of Dimensionality on KNN Queries



(a) On Construction Time



(b) On Querying CPU Time

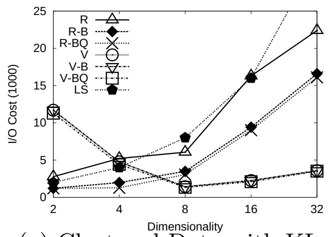
Figure 9: Effect of Dimensionality on Clustered Data with IS

Figure 9 summarizes the experimental results on construction time and querying CPU time on R-trees, VA files and BBTree. VA file is the most efficient method for index construction, which only takes one scan of the whole data set. On the other hand, BBTree is always at least one magnitude slower than any other method in our tests because of its employment of clustering algorithm as a splitting operator, leading to low effectiveness and efficiency particularly in high dimensional spaces.

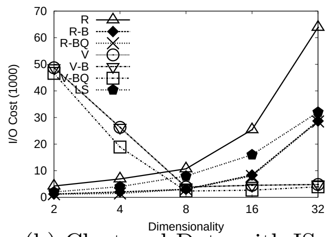
On querying CPU time, BBTree turns out to be CPU efficient when dimensionality is low. In 2-dimensional space, it is better than the others by almost a magnitude of two. However, the efficiency of BBTree drops quickly with the increase of dimensionality. When the dimensionality reaches 16, it is clearly worse than the R-tree in terms of CPU time. On 32 dimensional space with clustered distribution, almost all methods proposed in this paper is competitive to BBTree on CPU time. The R-tree and its variants always save more CPU cycles than VA file based data structures on this set of experiments. The optimization techniques, including improved bounding and adaptive index construction, enables the R-tree to achieve better efficiency when dimensionality is not very high. But these techniques do not affect the CPU cost of VA files. This is because VA files spend most of the CPU cycles in calculating the bounding boxes on the points based on their bit string representations and is thus oblivious to our optimization proposal.

Data Set	R	R-BQ	V	V-BQ	BBT
DBLP	14.5	15	1.5	1.5	141
KDD	91	232	9	9	2862

Table 4: Index Construction Time on Real Data Sets (Seconds)



(a) Clustered Data with KL



(b) Clustered Data with IS

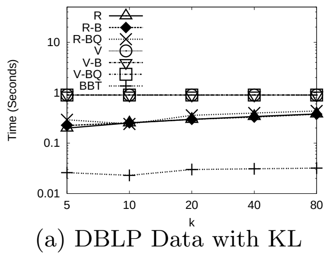
Figure 10: Effect of Dimensionality on I/O Cost

In Figure 10, we show the impact of dimensionality on I/O cost. For dimensionality lower than 8, the R-tree and its variants perform better than VA files on I/O cost. As dimensionality increases, the advantage of the R-trees diminishes while VA file presents dramatic improvement on I/O cost. The results in the figure also show that optimization techniques can greatly improve the performance of the R-tree. The improved bounding method reduces the I/O cost by half in low dimensional space, while workload optimization technique can further improve it for certain distribution of the data. For example, on clustered data with KL divergence (Figure 10(a)), workload optimization cuts 10% of I/O cost on the R-tree structure under improved bounding. We note that performance of basic R-tree is worse than the naive linear scan (LS) on very high dimensional spaces. For VA file, improved bounding does not show obvious improvements on the performance. This is because the cells are sufficiently small after we divided them using large number of bits. Workload optimization, on the other hand, is able to significantly improve the performance of VA file if the distribution is skewed. Such improvement can be observed in Figure 10(b) on the clustered data set on the IS distance.

6.3 Effect of Result Size k on KNN Queries

In Table 4, we list the CPU time for index construction, which are similar to the results on synthetic data sets. The construction of VA file is the fastest among all methods on KDD data with 72 dimensions, while the R-trees with workload optimization takes about 4 minutes to finish. BBTree takes almost 50 minutes to construct its index on KDD data set, making it much less attractive in any real system.

In Figure 11, we provide results that vary k, which is the number of nearest neighbors that must be returned. On the 8 dimensional DBLP data set, BBTree shows an advantage on CPU time. On the 72 dimensional KDD data set, however, the advantage of BBTree no longer exists. These results are consistent with the observations in Figure 9(b). While no optimization techniques is able to claim improve-


(a) DBLP Data with KL

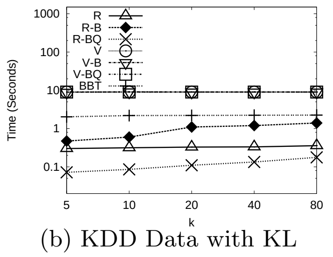
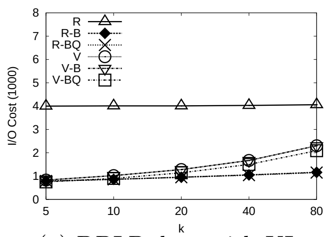

(b) KDD Data with KL

Figure 11: Result Size k vs. Querying CPU Time

ment upon the basic R-tree structure on DBLP data set (Figure 11(a)), we observe efficiency lift of R-BQ on KDD data set (Figure 11(b)). This improvement is due to the characteristic of KDD data set, which consists of huge clusters with highly similar points in the space.


(a) DBLP data with KL

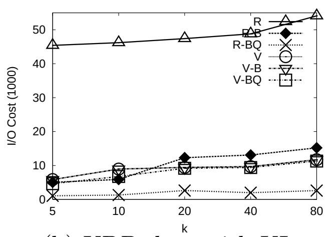
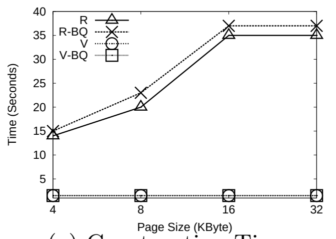

(b) KDD data with KL

Figure 12: Result Size k vs. Querying I/O Time

The results on I/O cost with varying k in Figure 12 match our expectation. In general, the basic R-tree has the worst performance among all methods. However, our proposed optimization techniques are able to significantly bridge this gap in performance. On the two real data sets, R-BQ enables the R-tree to achieve the best performance with minimal I/O cost compared to any other index structure. On clustered data set over IS distance, VA file works better when k is small. With larger k, VA file has to randomly access the original data points more frequently, incurring large I/O cost. Thus, VA file degrades linearly when k increases. Also, V-BQ on KDD data set is able to utilize the distribution information with about 10% savings in I/O costs.

6.4 Effect of Page Sizes on KNN Queries


(a) Construction Time

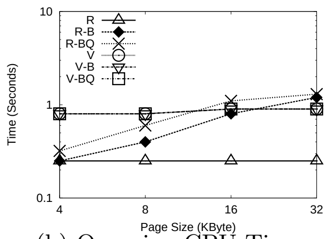
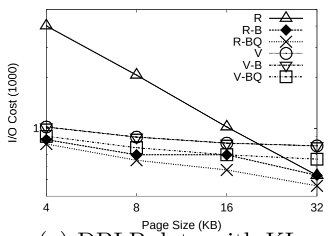

(b) Querying CPU Time

Figure 13: Effect of Page Sizes on DBLP for KNN

Page size is another important parameter of the system. The results in Figure 13(a) imply that the construction time of VA file does not depend on the specified page size, since the bit transformation is the dominant cost in index construction, which is irrelevant to page size. However, the construction time of the R-trees increases quickly with larger page size, since the intermediate nodes in the R-tree can store more detailed summarization information on its children, given the larger page sizes. Updates on the R-trees, such as split, take more time with larger page, because it needs to iterate the entries to select the best splitting plan.

From Figure 13(b), we observe that the R-tree without any optimization is the method spending the minimal CPU cycles. With larger pages, VA file has very limited room to improve its performance, because VA file usually accesses the pages randomly when searching for nearest neighbor results. The optimization techniques on the R-tree cost more CPU cycles, since the algorithm iterates every entry in the the node page, leading to more computation.


(a) DBLP data with KL

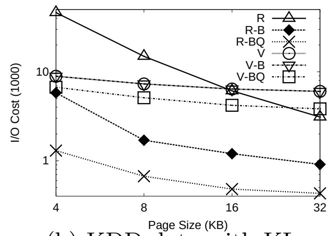
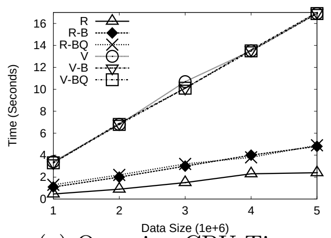

(b) KDD data with KL

Figure 14: Effect of Page Sizes on I/O Cost

Based on the results in Figure 14, we make several important observations. First, the R-tree performs much better if more space is given for a single page on disk. When the page size is 32KB, R-BQ is even better than VA files by two magnitudes on I/O costs, on highly clustered KDD data. For VA file, workload optimization technique is also able to utilize the large page size and show a larger gap in performance over VA files without adaptive construction.

6.5 Effect of Data Size on KNN Queries

We next study the effect of data size. In Figure 15. Here we evaluate the querying CPU time and I/O cost of the proposed methods. In a nutshell, all the methods show linear increase on both CPU cycles as well as I/O cost, proving that they are scalable to large data sets.


(a) Querying CPU Time

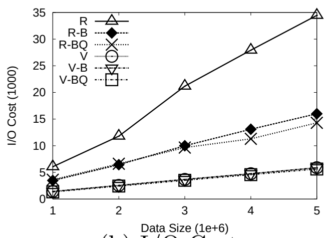

(b) I/O Cost

Figure 15: Effect of Data Size on KNN Queries

6.6 Effect of Search Range on Range Queries

Due to the limited space, we skip most of the experimental results on range query. Instead, we only provide partial results with varying search range θ in Figure 16. While the search range does not affect the CPU time for VA file, the performance of the R-tree is sensitive to θ. With a large search range, the R-tree degrades quickly, since the bounding techniques distinguish the distance better when points are close to the query. Finally, when testing I/O cost with varying search range, we find that the R-tree structures is worse when θ increases. However, the R-tree is stable on I/O costs on KDD data, because very few points are returned even when the threshold on the search range increases.

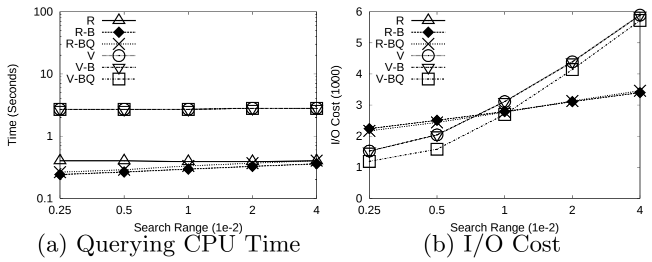


Figure 16: Effect of Search Range on Range Queries

Finally. we also conducted some experiments comparing conventional R-tree for Euclidean distance and our Bregman R-tree for KL divergence, on KDD99 and DBLP data sets. The performance of Bregman R-tree only deteriorates with a very small margin on efficiency. These results will be available in the extended version of the paper.

7. CONCLUSIONS AND FUTURE WORK

In this paper we investigate the problem of efficiently supporting similarity search on a general class of non-metric (Bregman) distance measures. We present a general purpose solution relying on a novel mapping strategy and also one which effectively leverages existing indexing infrastructure present in modern database systems. We present several important optimizations based on improved bounding strategies and query workload information to efficiently support k-nearest neighbor and range queries. We find that both the mapping and optimization strategies can be achieved with minimal changes to existing database infrastructure. A detailed experimental study verifies the efficacy and efficiency of the proposed techniques on real and synthetic data.

In the future, we will conduct more detailed studies on indexing objects in the kernel space which our framework can be easily extended to handle. Kernel-based methods have been shown to be effective in a number of applications with nearest neighbor and range query being one of the principal bottlenecks to realizing scalable solutions. For example, it is interesting to investigate how to improve the performance of LLE [24].

8. ACKNOWLEDGMENT

Zhenjie Zhang, Anthony K. H. Tung and Beng Chin Ooi were supported by Singapore NRF grant R-252-000-376-279. Srinivasan Parthasarathy was supported by NSF IIS-0347662 (CAREER) and NSF CCF-0702587.

9. REFERENCES

[1] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Y. Wu. An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *J. ACM*, 45(6):891–923, 1998.

[2] A. Banerjee, S. Merugu, I. S. Dhillon, and J. Ghosh. Clustering with bregman divergences. *Journal of Machine Learning Research*, 6:1705–1749, 2005.

[3] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The r*-tree: An efficient and robust access method for points and rectangles. In *SIGMOD Conference*, pages 322–331, 1990.

[4] A. Beygelzimer, S. Kakade, and J. Langford. Cover trees for nearest neighbor. In *ICML*, pages 97–104, 2006.

[5] S. Boltz, E. Debreuve, and M. Barlaud. High dimensional statistical distance for region-of-interest tracking: Application to combining a soft geometric constraint with radiometry. In *CVPR*, 2007.

[6] L. Bregman. The relaxation method of finding the common point of convex sets and its application to the solution of problems in convex programming. *USSR Computational Mathematics and Mathematical Physics*, 7(3):200–217, 1967.

[7] L. Cayton. Fast nearest neighbor retrieval for bregman divergences. In *ICML*, pages 112–119, 2008.

[8] L. Chen and X. Lian. Efficient similarity search in nonmetric spaces with local constant embedding. *IEEE Trans. Knowl. Data Eng.*, 20(3):321–336, 2008.

[9] H. Ferhatosmanoglu, E. Tuncel, D. Agrawal, and A. E. Abbadi. Vector approximation based indexing for non-uniform high dimensional data sets. In *CIKM*, pages 202–209, 2000.

[10] J. H. Friedman, J. L. Bentley, and R. A. Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Trans. Math. Softw.*, 3(3):209–226, 1977.

[11] J. Goldberger, S. Gordon, and H. Greenspan. An efficient image similarity measure based on approximations of kl-divergence between two gaussian mixtures. In *ICCV*, pages 487–493, 2003.

[12] R. M. Gray, A. Buzo, A. H. Gray, and Y. Matsuyama. Disotrotion measures for speech processing. *IEEE Transaction on Acoustics, Speech, and Signal Processing*, 28(4).

[13] A. Guttman. R-trees: A dynamic index structure for spatial searching. In *SIGMOD Conference*, pages 47–57, 1984.

[14] F. Itakura and S. Saito. A statistical method for estimation of speech spectral density and formant frequencies. *Electronics and Communications in Japan*, 53:36–43, 1970.

[15] H. V. Jagadish, B. C. Ooi, K.-L. Tan, C. Yu, and R. Zhang. idistance: An adaptive b$^+$-tree based indexing method for nearest neighbor search. *ACM Trans. Database Syst.*, 30(2):364–397, 2005.

[16] B. Kulis, M. Sustik, and I. Dhillon. Learning low-rank kernel matrices. In *ICML*, pages 505–512, 2006.

[17] B. Long, Z. M. Zhang, and P. S. Yu. Graph partitioning based on link distributions. In *AAAI*, pages 578–583, 2007.

[18] G. McLachlan and D. Peel. *Finite Mixture Models.* Wiley-Interscience, 2000.

[19] F. Nielsen and R. Nock. On approximating the smallest enclosing bregman balls. In *Symposium on Computational Geometry*, pages 485–486, 2006.

[20] F. C. N. Pereira, N. Tishby, and L. Lee. Distributional clustering of english words. In *ACL*, pages 183–190, 1993.

[21] J. Puzicha, Y. Rubner, C. Tomasi, and J. M. Buhmann. Empirical evaluation of dissimilarity measures for color and texture. In *ICCV*, pages 1165–1172, 1999.

[22] N. Rasiwasia, P. Moreno, and N. Vasconcelos. Bridging the gap: Query by semantic example. *IEEE Transaction on Multimedia*, 9(5).

[23] D. Suciu and N. N. Dalvi. Foundations of probabilistic answers to queries. In *SIGMOD Conference*, 2005.

[24] J. Tang, X.-S. Hua, G.-J. Qi, Y. Song, and X. Wu. Video annotation based on kernel linear neighborhood propagation. *IEEE Transaction on Multimedia*, 4(10):620–628, 2008.

[25] A. K. H. Tung, R. Zhang, N. Koudas, and B. C. Ooi. Similarity search: A matching based approach. In *VLDB*, pages 631–642, 2006.

[26] R. Weber, H.-J. Schek, and S. Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *VLDB*, pages 194–205, 1998.