BrePartition: Optimized High-Dimensional *k*NN Search With Bregman Distances

Yang Song, Yu Gu, Rui Zhang, *Senior Member, IEEE*, and Ge Yu, *Member, IEEE*

Abstract—Bregman distances (also known as Bregman divergences) are widely used in machine learning, speech recognition and signal processing, and *k*NN searches with Bregman distances have become increasingly important with the rapid advances of multimedia applications. Data in multimedia applications such as images and videos are commonly transformed into space of hundreds of dimensions. Such high-dimensional space has posed significant challenges for existing *k*NN search algorithms with Bregman distances, which could only handle data of medium dimensionality (typically less than 100). This paper addresses the urgent problem of high-dimensional *k*NN search with Bregman distances. We propose a novel partition-filter-refinement framework. Specifically, we propose an optimized dimensionality partitioning scheme to solve several non-trivial issues. First, an effective bound from each partitioned subspace to obtain exact *k*NN results is derived. Second, we conduct an in-depth analysis of the optimized number of partitions and devise an effective strategy for partitioning. Third, we design an efficient integrated index structure for all the subspaces together to accelerate the search processing. Moreover, we extend our exact solution to an approximate version by a trade-off between the accuracy and efficiency. Experimental results on four real-world datasets and two synthetic datasets show the clear advantage of our method in comparison to state-of-the-art algorithms.

Index Terms—Bregman distance, high-dimensional, *k*NN search, dimensionality partitioning

1 Introduction

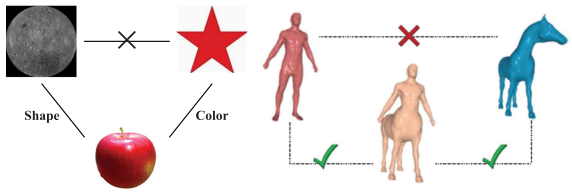
Bregman distances (also called Bregman divergences), as a generalization of a wide range of non-metric distance functions (e.g., Squared Mahalanobis Distance and Itakura-Saito Distance), play an important role in many multimedia applications such as image and video analysis and retrieval, speech recognition and time series analysis [1], [2], [3]. This is because metric measurements (such as Euclidian distance) satisfy the basic properties in metric space, such as non-negativity, symmetry and triangular inequality. Although it is empirically proved successful, the metric measurements represented by Euclidian distance are actually inconsistent with human's perception of similarity [4], [5]. Examples from [6] and [4] illustrate that the distance measurement is not metric when comparing images. As can be seen in Fig. 1a, the moon and the apple are similar in shape, the pentagram and the apple are similar in color, but there is no similarity between the moon and the pentagram. In this case, our perception of similarity violates the notion of triangular inequality and illustrates that human beings are often comfortable when deploying or using non-metric dissimilarity measurements instead of metric ones especially on complex data types [6], [7]. Likewise, as shown in Fig 1b [4], both the man and the horse are perceptually similar to their composition, but the two obviously differ from each other. Therefore, it is not appropriate to employ Euclidian distance as the distance measurement in many practical scenarios.

Since Bregman distances have the capability of exploring the implicit correlations of data features [8], they have been widely used in recent decades in a variety of applications, including image retrieval, image classification and sound processing [7], [9], [10]. Over the last several years, they are also used in many practical applications. For example, they are employed to measure the closeness between Hermitian Positive-Definite (HPD) matrices to realize target detection in a clutter [11]. They are also used as the similarity measurements of the registration functional to combine various types of image characterization as well as spatial and statistical information in image registration [12] and apply multi-region information to express the global information in image segmentation [13]. In addition, they are applied in graph embedding, matrix factorization and tensor factorization in the field of social network analysis [14]. Among the operations that employ Bregman distances, *k*NN queries are demanded as a core primitive or a fundamental step in the aforementioned applications [15], [16], [17].

In addition, data in multimedia applications such as images and videos are commonly transformed into space of hundreds of dimensions, such as the commonly used datasets (Audio, Fonts, Deep, SIFT, etc.) illustrated in our experimental evaluation. Existing approaches [6], [18] for *k*NN searches with Bregman distances focus on designing index structures for Bregman distances, but these index structures perform poorly in high-dimensional space due to either large overlaps between clusters or intensive computation.

- *Y. Song, Y. Gu, and G. Yu are with the School of Computer Science and Engineering, Northeastern University, Shenyang, Liaoning 110819, China. E-mail: ysqyw1994@163.com, {guyu, yuge}@mail.neu.edu.cn.*
- *R. Zhang is with the School of Computing and Information Systems, The University of Melbourne, Parkville, VIC 3010, Australia. E-mail: rui.zhang@unimelb.edu.au.*

Manuscript received 17 Oct. 2019; revised 15 Mar. 2020; accepted 30 Apr. 2020. Date of publication 0 . 0000; date of current version 0 . 0000. (Corresponding author: Ge Yu). Recommended for acceptance by G. Chen. Digital Object Identifier no. 10.1109/TKDE.2020.2992594



(a) Example 1 (b) Example 2

Fig. 1. Examples.

Aiming at these situations, this paper addresses the urgent problem of high-dimensional kNN searches with Bregman distances. In this paper, we propose a partition-filter-refinement framework. We first partition a high-dimensional space into many low-dimensional subspaces. Then, range queries in all the subspaces are performed to generate candidates. Finally, the exact kNN results are evaluated by refinement from the candidates. However, realizing such a framework requires solving the following three challenges:

- *Bound:* In metric space, bounds are usually derived based on triangular inequality, but in non-metric space, triangular inequality does not hold. Therefore, it is challenging to derive an efficient bound for Bregman distances which are not metric.
- *Partition:* How to partition the dimensions to get the best efficiency is a challenge. We need to work out both how many partitions and at which dimensions should we partition.
- *Index:* It is challenging to design an I/O efficient index that handles all the dimension partitions in a unified manner. Specifically, by effectively organizing the data points on disks, the index is desired to adapt to our proposed partition strategy and facilitate the data's reusability across partitions.

To address the above challenges, and make the following contributions in this paper:

- We derive the upper bounds between the given query point and an arbitrary data point in each subspace mathematically based on Cauchy inequality and the proper upper bounds are selected as the searching bounds from these subspaces. The final candidate set is the union of the candidate subsets of all the subspaces. We theoretically prove that the kNN candidate set obtained following our bound contains the kNN results.
- For dimensionality partitioning, we observe a trade-off between the number of partitions and the search time. Therefore, we derive the algorithm's time complexity and the optimized number of partitions in theory. Furthermore, a strategy named Pearson Correlation Coefficient-based Partition (PCCP) is proposed to reduce the size of the candidate set by partitioning highly-correlated dimensions into different subspaces.
- After dimensionality partitioning, we employ BB-trees in our partitioned low-dimensional subspaces and design an integrated and disk-resident index structure, named BB-forest. BB-trees can handle low-dimensional data efficiently, so they work well in our framework since the data has been partitioned into low-dimensional subspaces. In addition, the data points are well-organized on the disks based on our proposed PCCP to improve data's reusability in all the subspaces, so that the I/O cost can be reduced.
- In order to improve the search efficiency while ensuring comparable accuracy with the probability guarantee, we make a trade-off between the efficiency and the accuracy and propose a solution to approximate kNN search through the data distribution.
- Extensive experimental evaluations demonstrate the high efficiency of our approach. Our algorithm named BrePartition can clearly outperform state-of-the-art algorithms in running time and I/O cost.

The rest of the paper is structured as follows. Section 2 presents the related works. The preliminaries and overview are discussed in Section 3. We present the derivation of the upper bound in Section 4 and the dimensionality partitioning scheme in Section 5. The index structure, BB-Forest, is described in Section 6. We present the overall framework in Section 7. The extended solution to approximate kNN search is presented in Section 8. Experimental results are disscussed in Section 9. Finally, we conclude our work in Section 10.

2 Related Works

kNN search is a fundamental problem in many application domains. Here we review existing works on the kNN search problem in both metric and non-metric spaces.

2.1 Metric Similarity Search

The metric similarity search problem is a classic topic and a plethora of methods exist for speeding up the nearest neighbor retrieval. Existing methods contain tree-based data structures including KD-tree [19], R-tree [20], B+-tree variations [21], [22], [23] and transformation-based methods including Product Quantization (PQ)-based methods [24], [25], Locality Sensitive Hashing (LSH) family [26], [27], [28], [29] and some other similarity search methods based on variant data embedding or dimensionality reduction techniques [30], [31], [32]. These methods can't be utilized in non-metric space where the metric postulates, such as symmetry and triangle inequality, are not followed.

2.2 Bregman Distance-Based Similarity Search

Due to the widespread use of Bregman distances in multimedia applications, a growing body of work is tailored for the kNN search with bregman distances. The prime technique is Bregman voronoi diagrams derived by Nielsen *et al.* [33]. Soon after that, Bregman Ball tree (BB-tree) is introduced by Cayton [18]. BB-trees are built by a hierarchical space decomposition via k-means, sharing the similar flavor with KD-tree. In a BB-tree, the clusters appear in terms of Bregman balls and the filtering condition in the dual space on the Bregman distance from a query to a Bregman ball is computed for pruning out portions of the search space. Nielsen *et al.* [34] extend the BB-tree to symmetrized Bregman distances. Cayton [35] explores an algorithm to solve the range query based on BB-tree. Nevertheless, facing higher dimensions, considerable overlap between clusters will be incurred, and too

many nodes have to be traversed during a kNN search. Therefore the efficiency of BB-tree is dramatically degraded, sometimes even worse than the linear search. Zhang *et al.* [6] devise a novel solution to handle the class of Bregman distances by expanding and mapping data points in the original space to a new extended space. It employs typical index structures, R-tree and VA-file, to solve exact similarity search problems. But it's also inefficient for more than 100 dimensions, because too many returned candidates in a filter-refinement model lead to intensive computation overhead of Bregman distances.

Towards more efficient search processing, there have been increasing attentions focusing on the approximate search methods for Bregman distances [4], [18], [34], [36], [37], [38]. These approximate methods achieve the efficiency promotions with the price of losing accuracies. For example, the state-of-the-art approximate solution [34], which is designed for the high-dimensional space, exploits the data-$\bar{a}$s distribution and employs a variational approximation to estimate the explored nodes during backtracking in the BB-tree. Nevertheless, all these methods can't provide the precision guarantees, while some of them cannot be applied to the high-dimensional space [18], [36], [37], [38].

TABLE 1
Non-Metric Search Methods

Name	NM	BDS	HD	Exact
BrePartition (Our solution)	√	√	√	√
Zhang *et al.* [6]	√	√		√
BB-tree [18], [35]	√	√		√
Bregman voronoi diagram [33]	√	√		√
BB-tree variants [34]	√	√		
Ackermann *et al.* [36]	√	√		
Abdullah *et al.* [37]	√	√		
Coviello *et al.* [34]	√	√	√	
Ferreira *et al.* [38]	√	√		
Non-metric LSH [4]	√	√	√	
FastMap [39]	√			√
Wang *et al.* [40]	√			√
Boostmap [41]	√			
Athitsos *et al.* [42]	√		√	
TriGen [43]	√			√
NM-tree [44]	√			
Liu *et al.* [45]	√			
LCE [46]	√			
DBH [47]	√		√	
Dyndex [48]	√		√	
HNSW [49]	√		√	

2.3 Other Non-Metric Similarity Search Methods

There also exist many works in the context of non-metric similarity search without the limit to Bregman distances. Space-embedding techniques [39], [40], [41], [42] embed non-metric spaces into an euclidean one where two points that are close to each other in the original space are more likely close to each other in the new space. Distance-mapping techniques transform the non-metric distance by modifying the distance function itself while preserving the original distance orderings. Skopal [43] develops TriGen algorithm to derive an efficient mapping function among concave functions by using the distance distribution of the database. NM-tree [44] combines M-tree and TriGen algorithm for the non-metric search. Liu *et al.* [45] propose a simulated-annealing-based technique to derive optimized transform functions while preserving the original similarity orderings. Chen *et al.* [46] employ the constant shifting embedding with a suitable clustering of the dataset for a more effective lower-bounds. Recently, a representative technique based on Distance-Based Hashing (DBH) [47] is presented and a general similarity indexing methods for non-metric distance measurements is designed by optimizing hashing functions. In addition, Dyndex [48], as the most impressive technique based on classification performs classification of the query point to answer similarity search by categorizing points into classes. These methods degrade dramatically in performance when dealing with high-dimensional issues. There exists an approximate solution called Navigable Small World graph with controllable Hierarchy (HNSW) [49], which can be extended to non-metric space. However, it is not a disk-resient solution, while we mainly focus on disk-resident solutions in this paper.

We summarize the properties of representative non-metric search methods in Table 1. In Table 1, NM means that the method adopts the distance functions of non-metric space instead of metric space, BDS means that the method is designed specifically for Bregman distances, and HD means that the method works well in high-dimensional space (more than 100 dimensions). Our proposed solution BrePartition is the first algorithm that possesses all the four desired properties compared to existing algorithms.

3 Preliminaries and Overview

3.1 Bregman Distance

Given a d-dimensional vector space S, a query $y = (y_1, y_2, \ldots, y_d)$ and an arbitrary data point $x = (x_1, x_2, \ldots, x_d)$, the Bregman distance between x and y is defined as $D_f(x, y) = f(x) - f(y) - < \nabla f(y), x - y >$, where $f(\cdot)$ is a convex function mapping points in S to real numbers, $\nabla f(y)$ is the gradient of $f(\cdot)$ at y, and $< \cdot, \cdot >$ denotes the dot product between two vectors. When different convex functions are employed, Bregman distances define several well-known distance functions. Some representatives contain:

- *Squared Mahalanobis Distance:* The given $f(x) = \frac{1}{2}x^{\mathrm{T}}Qx$ generates $D_f(x, y) = \frac{1}{2}(x - y)^T Q(x - y)$ which can be considered as a generalization of the above squared euclidean distance.
- *Itakura-Saito Distance (ISD):* When the given $f(x) = -\sum \log x_i$, the distance is IS distance which is denoted by $D_f(x, y) = \sum \left(\frac{x_i}{y_i} - \log\frac{x_i}{y_i} - 1\right)$.
- *Exponential Distance (ED):* When the given $f(x) = e^x$, the Bregman distance is represented as $D_f(x, y) = e^x - (x - y + 1)e^y$. In this paper, we name it *exponential distance*.

In addition, our method can be applied to most measures belonging to Bregman distances, such as Shannon entropy, Burg entropy, l_p-quasi-norm and l_p-norm, except KL-divergence, since it's not cumulative after the dimensionality partitioning.

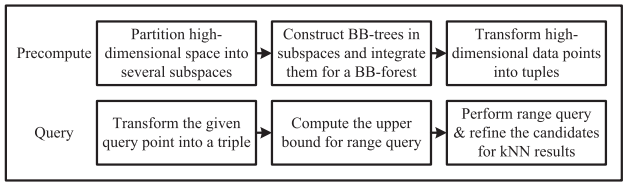


Fig. 2. Overview.

3.2 Overview

Our method consists of the precomputation and the search processing. In the precomputation, we first partition the dimensions (described in Section 5). Second, we construct BB-trees in the partitioned subspaces and integrate them to form a BB-forest (described in Section 6). Third, we transform the data points into tuples for computing the searching bound (described in Section 4). During the search processing, we first transform the query point into a triple and compute the bound used for the range query (described in Section 4). Second, we perform the range query for the candidates. Finally, the kNN results are evaluated from these candidates. The whole process is illustrated in Fig 2.

We summarize the frequently-used symbols in Table 2.

4 Derivation of Bound

For kNN queries, it is crucial to avoid the exhaustive search by deriving an effective bound as a pruning condition. We exploit the property of Bregman distances and an upper bound is derived from Cauchy inequality.

Given a data set D, suppose $x = (x_1, \ldots, x_d)^{\mathrm{T}}$ and $y = (y_1, \ldots, y_d)^{\mathrm{T}}$ are two d-dimensional vectors in D. After dimensionality partitioning, x is partitioned into M disjoint subspaces represented by M subvectors. These M subvectors are denoted by:

$$\begin{aligned} x_{i\cdot} &= \left(x_{\lceil \frac{d}{M} \rceil \times (i-1)+1}, \ldots, x_{\lceil \frac{d}{M} \rceil \times i}\right)^{\mathrm{T}} \\ &= (x_{i1}, \ldots, x_{i\lceil \frac{d}{M} \rceil})^{\mathrm{T}}, \end{aligned}$$

where $1 \leq i \leq M$. Vector y is partitioned in the same manner, while each part is denoted by $y_{i\cdot}$ $(1 \leq i \leq M)$. By Cauchy inequality, we can prove Theorem 1 below, which can be used to derive the upper bound between arbitrary $x_{i\cdot}$ and $y_{i\cdot}$ $(1 \leq i \leq M)$ in the same subspace.

Theorem 1. *The upper bound between $x_{i\cdot}$ and $y_{i\cdot}$ $(1 \leq i \leq M)$ is derived:*

$$D_f(x_{i\cdot}, y_{i\cdot}) \leq \alpha_x^{(i)} + \alpha_y^{(i)} + \beta_{yy}^{(i)} + \sqrt{\gamma_x^{(i)} \times \delta_y^{(i)}}.$$

For simplicity, $\alpha_x^{(i)}$, $\alpha_y^{(i)}$, $\beta_{yy}^{(i)}$, $\gamma_x^{(i)}$ and $\delta_y^{(i)}$ are used to mark these formulas:

$$\alpha_x^{(i)} = \sum_{j=1}^{\lceil d/M \rceil} f(x_{ij}), \alpha_y^{(i)} = -\sum_{j=1}^{\lceil d/M \rceil} f(y_{ij}),$$

$$\beta_{xy}^{(i)} = -\sum_{j=1}^{\lceil d/M \rceil} \left(x_{ij} \times \frac{\partial f(y)}{\partial y_{ij}}\right), \beta_{yy}^{(i)} = \sum_{j=1}^{\lceil d/M \rceil} \left(y_{ij} \times \frac{\partial f(y)}{\partial y_{ij}}\right),$$

$$\gamma_x^{(i)} = \sum_{j=1}^{\lceil d/M \rceil} x_{ij}^2, \delta_y^{(i)} = \sum_{j=1}^{\lceil d/M \rceil} \frac{\partial f(y)}{\partial y_{ij}}^2.$$

TABLE 2
Frequently Used Symbols

Symbol	Explanation
S	dataset
n	number of data points
d	dimensionality of each data point
k	number of returned points that users require
x	data point
y	query point
$P(x)$	transformed data point
$Q(y)$	transformed query point
$D_f(x, y)$	the Bregman distance between x and y
$UB(x, y)$	the upper bound of Bregman distance between x and y
M	number of partitions

Proof. Please see Section 1 in the supplementary file, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TKDE.2020.2992594. □

Based on Theorem 1, we can derive the upper bound between two arbitrary data points in each subspace. Furthermore, we can obtain the upper bound between two data points in the original high-dimensional space by summing up the M upper bounds from all the subspaces. Theorem 2 proves it.

Theorem 2. *The Bregman distance between x and y is bounded by the sum of each upper bound from its corresponding subspace. Formally, $D_f(x, y) \leq UB(x, y)$, where $UB(x, y) = \sum_{i=1}^{M} UB(x_{i\cdot}, y_{i\cdot})$.*

Proof. Please see Section 2 in the supplementary file, available online. □

Therefore, when dealing with a kNN search, we first compute the upper bounds between the given query points and every data point in the dataset. And then the kth smallest upper bound is selected and its components are selected as the searching bounds to perform range queries in their corresponding subspaces as the filter processing. The union of the searching result sets of all the subspaces is the final candidate set for the refinement processing. Theorem 3 proves that all the kNN points are in the final candidate set.

Theorem 3. *The candidate set of each subspace is denoted by C_i $(1 \leq i \leq M)$, and the final candidate set:*

$$C = C_1 \cup C_2 \cup \cdots \cup C_M.$$

The kNN points of the given query point y exist in C.

Proof. Please see Section 3 in the supplementary file, available online. □

Based on the upper bound derived in Theorem 1, two components α_x and γ_x of data points in all the subspaces can be computed offline. It can be considered as the precomputation that each partitioned multi-dimensional data point in each subspaces is transformed into a two-dimensional tuple denoted by $P(x) = (\alpha_x, \gamma_x)$ (See Fig. 3 for an illustration). Once the query point is given, we only need to compute the values of α_y, β_{yy} and δ_y, which can be considered as a triple denoted by $Q(y) = (\alpha_y, \beta_{yy}, \delta_y)$, to obtain the upper

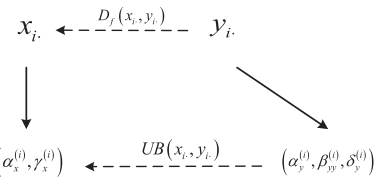


Fig. 3. Transformation.

bounds while the computation overhead is very small. Relying on the precomputation, the search processing will be dramatically accelerated.

Algorithm 1 describes the process of computing the upper bound between two arbitrary data points by their corresponding transformed vectors. Algorithms 2 and 3 describe the transformations of the partitioned data point and the partitioned query point, respectively. Algorithm 4 describes the process of determining the searching bound from each subspace.

Algorithm 1. UBCompute $(P(x), Q(y))$

Input: the transformed vectors of x and y, $P(x) = (\alpha_x, \gamma_x)$ and $Q(y) = (\alpha_y, \beta_{yy}, \delta_y)$.
Output: the upper bound of x and y, ub.
1: $ub = \alpha_x + \alpha_y + \beta_{yy} + \sqrt{\gamma_x \times \delta_y}$;
2: **return** ub;

Algorithm 2. PTransform (X)

Input: the subvectors' set of a partitioned data point $X = \{x_1, x_2, \ldots, x_M\}$.
Output: the transformed tuples' set of a partitioned data point $P = \{P(x_1), P(x_2), \ldots, P(x_M)\}$.
1: **for** $i = 1$ to M **do**
2: Compute $\alpha_x^{(i)}$ and $\gamma_x^{(i)}$;
3: $P(x_1) = (\alpha_x^{(i)}, \gamma_x^{(i)})$;
4: **end for**
5: $P = \{P(x_1), P(x_2), \ldots, P(x_M)\}$;
6: **return** P;

Algorithm 3. QTransform (Y)

Input: the subvectors' set of the partitioned query point $Y = \{y_1, y_2, \ldots, y_M\}$.
Output: the transformed tuples' set of the partitioned query point $Q = \{Q(y_1), Q(y_2), \ldots, Q(y_M)\}$.
1: **for** $i = 1$ to M **do**
2: Compute $\alpha_y^{(i)}$, $\beta_{yy}^{(i)}$ and $\delta_y^{(i)}$;
3: $Q(x_i) = (\alpha_y^{(i)}, \beta_{yy}^{(i)}, \delta_y^{(i)})$;
4: **end for**
5: $Q = \{Q(y_1), Q(y_2), \ldots, Q(y_M)\}$;
6: **return** Q;

5 Dimensionality Partitioning

According to Cauchy inequality, we can prove:

$$\sum_{i=1}^{M_1} \sqrt{\gamma_x^{(i)} \times \delta_y^{(i)}} > \sum_{i=1}^{M_2} \sqrt{\gamma_x^{(i)} \times \delta_y^{(i)}},$$

when $M_1 < M_2$. It indicates that more partitions bring tighter bounds. Furthermore, we can prove that there is an exponential relationship between the upper bound and the number of partitions. However, more partitions lead to more online computation overhead. Therefore, it's an unresolved issue to determine the number of partitions. Besides, how to partition these dimensions may also affect the efficiency. In this section, the optimized number of partitions which contributes to the efficiency is derived theoretically and a heuristic strategy is devised for the purpose of reducing the candidate set by partitioning.

Algorithm 4. QBDetermine (S_t, Q)

Input: the transformed dataset $S_t = \{P_1, P_2, \ldots, P_n\}$, the transformed query point Q.
Output: the set containing M subspaces' searching bounds QB.
1: $UB = \emptyset$, $QB = \emptyset$;
2: **for** $i = 1$ to n **do**
3: **for** $j = 1$ to M **do**
4: $ub_{ij} = \mathsf{UBCompute}(P_{ij}, Q_j)$; // Algorithm 1
5: $ub_i += ub_{ij}$;
6: $QB_i = QB_i \cup \{ub_{ij}\}$;
7: **end for**
8: $UB = UB \cup ub_i$;
9: **end for**
10: Sort UB and find the kth smallest ub_t $(1 \le t \le n)$;
11: $QB = QB_t = \{ub_{t1}, ub_{t2}, \ldots, ub_{tM}\}$.
12: **return** QB;

5.1 Number of Partitions

More partitions lead to a tighter bound, which indicates a smaller candidate set intuitively. But more partitions also incur more computation overhead when calculating the upper bounds from more subspaces at the same time. It's a trade-off between the number of partitions and the efficiency. In this part, we will theoretically analyze the online processing to derive the optimized number of partitions for the optimized efficiency. The online processing mainly includes two parts, one is to compute the upper bounds between the given query point and an arbitrary data point in the dataset and sort these upper bounds to determine the searching bound. The other is to obtain the candidate points by performing range queries and refine the candidate points for the results. Below we will derive the time complexities of the online kNN search.

As a prerequisite, we transform the given query into a triple:

$$Q(y) = \left(-\sum f(y), \sum \frac{\partial f(y)}{\partial y}, \sum y \times \frac{\partial f(y)}{\partial y} \right).$$

When it comes to computing upper bounds, these triples in the M partitions can be computed in $O(d)$ time. Since we have already transformed the multi-dimensional data points in each subspace into tuples, we can compute the upper bounds in $O(Mn)$ time in all M subspaces. The time complexity of summing up all n points' upper bounds is also $O(Mn)$ and the time complexity of sorting them to find the kth smallest one is $O(n \log k)$. Therefore, the whole time complexity of transforming the query point, computing the upper bounds, summing up them and sorting them to find the kth smallest one is $O(d + Mn + n \log k)$.

In the filter-refinement processing, we specify a parameter λ $(0 < \lambda < 1)$ to describe the pruning effect of the

searching bound for the complexity analyses. We suppose that data points are stored in BB-trees and each leaf node is full. And we set the capacity of each leaf node in a BB-tree as C and the number of leaf nodes is n/C. Thus the number of accessed leaf nodes during the range queries can be estimated as $\lambda n/C$. According to the proposed method in [35], the secant method is employed to determine whether a cluster and the searching range intersect, or whether one contains another when performing the range queries. For the time consumption of the determination is negligible, the time complexity of searching in the M BB-trees is equivalent to that of searching in a binary tree and the time complexity is $O(M \times \frac{\lambda n}{C} \log \frac{n}{C})$. In BB-trees, the capacity of each leaf node increases with the size of the corresponding dataset since we have to restrict the height of the tree, and the value of $\frac{n}{C}$ can be considered as a constant. Consequently, the time complexity of searching in the BB-trees is ignored.

Once we have identified the clusters that intersect the given searching range, all the data points in these clusters will be loaded from the disk into the memory for processing as the candidate points which will be refined to obtain the kNN points. Thus we should estimate the size of the candidate set primarily. There is an exponential relationship between the upper bound UB and the number of partitions M and we describe it as $UB = A\alpha^M$, where $0 < \alpha < 1$. In addition, we assume the parameter λ which describes the pruning effect is proportional to the upper bound UB and is represented as $\lambda = \beta UB$. Based on these, the number of the candidate points is $\beta A\alpha^M n$. Actually, the final candidate set is the union of all the subspaces' candidate subsets. Here we directly consider the candidate set obtained by searching in the original space with the summing searching bound as the final candidate set, since it is a very good approximation of the union of all the subspaces' candidate subsets which is verified by experiments. Therefore, the refinement process takes $O(\beta A\alpha^M nd + \beta A\alpha^M n \log k)$. The first item represents the time complexity of computing the Bregman distances between each candidate point and the query, while the second represents the time complexity of evaluating the kNN points.

Therefore, the online time complexity is

$$O(d + Mn + n\log k + \beta A\alpha^M nd + \beta A\alpha^M n\log k),$$

in total and we target at minimizing the total time cost.

Theorem 4. *For any user-specified k $(0 < k \leq n)$, the time complexity can be minimized by setting the number of partitions*

$$M = \log_\alpha \frac{2n}{-\mu \ln \alpha (d + \log k)},$$

where $\mu = \beta An$.

Proof. Please see Section 4 in the supplementary file, available online. □

Especially, since the number of partitions requires to be determined offline, we set the value of k to 1 which will not impact too much on the number of partitions since k is negligible compared to the value of n. Moreover, the result computed by Theorem 4 may not be an integer, hence we compute the time costs in both cases of rounding up and down and choose the best value of M. A and α can be determined by fitting the function $UB = A\alpha^M$ through two arbitrary points' UB and the corresponding M, while the points are randomly selected from the dataset. And β can be determined by computing the proportion of the points within each sample's UB to n.

As M increases, the I/O cost decreases exponentially, and the corresponding time consumption can be estimated as $\frac{\beta A\alpha^M n}{Bv}$, where B denotes the disk's page size and v denotes the disk's IOPS. Since the IOPS of current mainstream SSD is very high, the loss of the I/O's time consumption can be negligible compared to the gains in the CPU's running time for the optimized partition number. When using the low-level storage device, this time consumption incurred by I/O operations can be added to the above cost model for deriving the optimized number of partitions.

5.2 Partitioning Strategy

Initially, we simply choose an equal and contiguous strategy for the dimensionality partitioning. Since the final candidate set is the union of the candidate subsets of all the subspaces, the size of the candidate set depends on the size of the candidate subset of each partition. Therefore, we attempt to develop a strategy for the dimensionality partitioning to further reduce the size of the final candidate set.

Intuitively, when the number of the candidate points generated from each partition is constant, if the intersection between these partitions is small, the candidate set will be large. In the worst case, the final candidate set reaches the maximum in the case that their intersection is empty. Conversely, large portions of overlap among these partitions' candidate sets may lead to a smaller candidate set. At best, each of them is equivalent to the final candidate set. Therefore, our objective is to reduce the size of the entire candidate set by making the partitions' candidate set overlap as large and as possible.

Simply, if there exist two identical partitions, their corresponding candidate sets will overlap completely leading to a smaller candidate set. Therefore, it is crucial to measure and compare the similarities between different partitions. However, there is no reasonable indicator satisfying our requirements. To address this issue, we simplify partitions to dimensions and employ Pearson Correlation Coefficient to measure the correlations between different dimensions and the correlations are utilized to indicate the similarities. The dimensions with strong correlations will be assigned to different partitions to make dimensions in each partition uniformly distributed. It is a heuristic algorithm and named Pearson Correlation Coefficient-based Partition (PCCP) for the dimensionality partitioning.

Given a d-dimensional dataset, we attempt to divide d dimensions into M partitions while each partition contains $\lceil d/N \rceil$ dimensions. The correlation between two arbitrary dimensions X and Y is measured by the Pearson correlation coefficient $r(X,Y) = cov(X,Y)/\sqrt{var(X)var(Y)}$, where $cov(X,Y)$ represents the covariance between X and Y, and $var(X)$ and $var(Y)$ represent the variances of X and Y, respectively. Specifically, we only consider the level of dimensions' correlations and ignore whether it is positive or negative, so we only consider the absolute values of the Pearson correlation coefficients. Our proposed strategy consists of two steps:

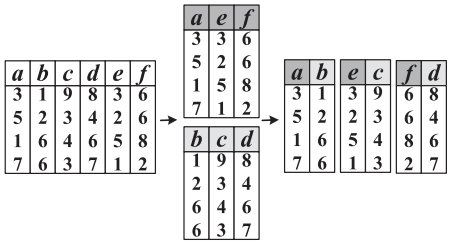


Fig. 4. An example of PCCP.

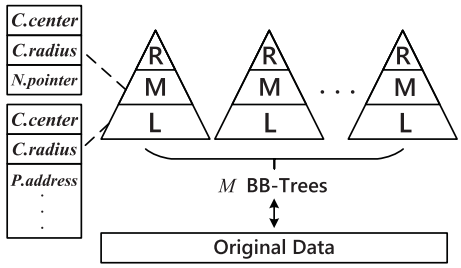


Fig. 5. BB-forest.

Assignment. We assign d dimensions to M groups and attempt to ensure high similarities between the dimensions in each group. In detail, we first select a dimension randomly and insert it into a group, find the dimension having the largest Pearson correlation coefficient with the selected dimension and insert it into the same group. Then, the dimension which has the largest absolute Pearson correlation coefficient with an arbitrary inserted dimension is selected and inserted into the current group. Following these steps, we continue searching and inserting until M dimensions have been inserted and a group is formed. The above steps are repeated until all d dimensions are assigned to $\lceil d/M \rceil$ different groups.

Partitioning. We select a dimension from every group and insert it into the current partition so that each partition has $\lceil d/N \rceil$ dimensions. The above steps are repeated until the M partitions are obtained.

Fig. 4 illustrates the process by an example. There exists a six dimensional dataset whose dimensions are denoted by a, b, c, d, e and f. In the assignment, we randomly select a dimension such as a, and compute the correlations between a and other dimensions. We find $|r(a, e)|$ is the largest, so we assign a and e to the first group. Whereafter, we compute the correlations between e and other dimensions since we have computed the correlations between a and other dimensions. We find $|r(e, f)|$ is the largest among the correlations between a and other dimensions except e, as well as the correlations between e and other dimensions except a. So we insert f into the group containing a and e. The others, b, c and d, are assigned to the second group. After assignment, we randomly select a dimension from each of the two groups and insert them into a partition. The above procedure is repeated three times and the final partitions are obtained as Fig. 4.

6 Indexing Subspaces for *k*NN Search

After partitioning, the original high-dimensional space is divided into many low-dimensional subspaces. This enables us to use existing indexing techniques which are efficient for low-dimensional spaces such as [6], [18]. Although the previous BB-tree [18] is only designed for the *k*NN search, an algorithm is explored to specifically solve the range query based on BB-tree in [35], which shows good performance. Therefore, we employ BB-tree for the range queries in each partition's filtering process and adopt the algorithm in [35] in this paper. We construct a BB-tree for each subspace and all the BB-trees form a *BB-forest*. Note that directly conducting *k*NN queries in each partition cannot obtain a correct candidate. Thus the *k*NN algorithm and the filtering condition proposed in [18] cannot be simply extended using the partition strategy. On the other hand, the range query algorithm proposed in [35] cannot directly solve the *k*NN problem. Therefore, our framework is essentially different from the existing BB-tree based filtering algorithms.

Intuitively, if we store each node's corresponding cluster's center and radius, and their pointers to their children or the addresses of the data points in this cluster, BB-tree can be simply extended to the disks to process large-scale datasets. Fig. 5 illustrates the disk-resident integrated index structure which consists of M BB-trees constructed in the M partitioned subspaces. In each BB-tree, the intermediate nodes store their corresponding clusters' centers and radii, denoted by $C.center$ and $C.radius$, respectively. And the leaf nodes store not only clusters' centers and radii, but the addresses of the data points in their corresponding clusters, denoted by $P.address$, which are used for reading the original high-dimensional data points from the disks.

According to [35], which develops an algorithm for efficient range search with Bregman distances, only each cluster's center and radius are required to determine whether two clusters intersect or one cluster contains another during the search processing. Therefore, the disk-resident version of BB-tree still works for efficient range search with Bregman distances. However, since our proposed BB-forest is an integrated index structure, how to organize the data on the disk is a major issue to address. If the clusters in these subspaces are significantly different, different BB-trees' candidate nodes obtained by ranges queries will index different data points which causes more I/O accesses when reading high-dimensional data points from the disks.

Benefitting from our devised PCCP, similar clusters are obtained in different subspaces. Therefore, after the dimensionality partitioning, we construct a BB-tree in a randomly selected subspace and the original high-dimensional data points indexed by each leaf node are organized according to the order of these leaf nodes. At the same time, the address of each point containing the disk number and offset is recorded. During the construction of other BB-trees, the recorded addresses of these data points are stored in leaf nodes for indexing. Since the candidate points in different subspaces are the same at best, we will read the same part of the disks when performing range queries in different subspaces to reduce I/O accesses.

7 Overall Framework

In this paper, we solve the *k*NN search problem with Bregman distances in high-dimensional space by our proposed partition-filter-refinement framework. The framework consists of the precomputation in Algorithm 5 and the search processing in Algorithm 6. In Algorithm 5, we first

determine the number of partitions (Line 2), which is introduced in Section 5.1. Then, we perform the dimensionality partitioning in the dataset based on our proposed strategy PCCP denoted as PCCP(S) (Line 3), which is introduced in Section 5.2. Finally, we transform the original data points in each subspace into tuples used for computing the upper bounds later (Lines 4-7), which is introduced in Section 4. At the same time, the partitioned data points will be indexed by the BB-trees of all the subspaces forming the BB-forest (Line 8), and the construct of the BB-forest is denoted as BBFConstruct(S_p) which is introduced in Section 6.

Algorithm 5. BrePartitionConstruct ($S, n, d, A, \alpha, \beta$)

Input: S, n, d, A, α, β.
Output: S_t, BBF.
1: $S_t = \emptyset$, $BBForest = \emptyset$;
2: $M = \log_{\frac{2n}{\alpha - \beta And \ln \alpha}}$;
3: $S_p =$ PCCP(S);
4: **for** each $X \in S_p$ **do**
5: $P =$ PTransform(X); // Algorithm 2
6: $S_t = S_t \cup P$;
7: **end for**
8: $BBF =$ BBFConstruct(S_p); **return** S_t, BBF;

In Algorithm 6, we first rearrange the query point according to the partitioned data points (Line 2). Second, we transform the query point into triples in each subspace (Line 3) and compute the searching bounds from all the subspaces (Line 4), which is introduced in Section 4. Third, we perform range queries with the computed searching bounds over all the BB-trees in the BB-forest for retrieving the candidates (Lines 5-7). Finally, we evaluate the candidate points for the kNN points and return the result set (Lines 8-9).

Algorithm 6. BrePartitionSearch (S_p, y, k, S_t, BBF)

Input: S_p, y, k, S_t, BBF.
Output: Res.
1: $Cand = \emptyset$, $Res = \emptyset$;
2: Rearrange the query point to obtain Y;
3: $Q =$ QTransform(Y); // Algorithm 3
4: $QB =$ QBDetermine(S_t, Q); // Algorithm 4
5: **for** each $T_j \in BBF$ **do**
6: $Cand = Cand \cup T_j$.rangeQuery($Y[j], QB[j]$);
7: **end for**
8: Evaluate the kNN result set $Res = \{x_1, x_2, \ldots, x_k\}$;
9: **return** Res;

8 Extension to Approximate kNN Search

In previous sections, we mainly concentrate on the exact retrieval of kNN searches. However, through our research, we can tighten the bounds derived above for comparable approximate results with probability guarantees to improve the efficiency. We present an approximate solution with probability guarantees. By the solution, given a query point q and a probability guarantee p ($0 < p \leq 1$), the retrieval k points are the exact kNN points of q with the probability guarantee p.

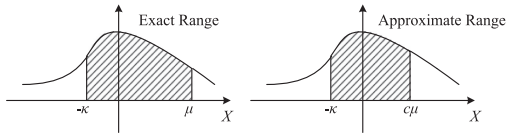


Fig. 6. Exact and approximate cases.

According to the previous description in Theorem 1, we relax the item β_{xy} by employing Cauchy-inequality and an exact bound for the kNN search can be derived. The exact searching bound can be simply represented in the form of $\kappa + \mu$, where

$$\kappa = \sum_{i=1}^{d} f(x_i) - \sum_{i=1}^{d} f(y_i) + \sum_{i=1}^{d} \left(y_i \times \frac{\partial f(y)}{\partial y_i} \right),$$

which isn't affected when computing the upper bound, and

$$\mu = \sqrt{\sum_{i=1}^{d} x_i^2 \times \sum_{i=1}^{d} \frac{\partial f(y)}{\partial y_i}^2},$$

which is obtained by relaxing β_{xy}. Therefore, the searching bound is actually determined by μ. In our proposed approximate kNN search solution, μ is tightened by multiplying a coefficient denoted as c ($0 < c \leq 1$) in the context that the distribution of each dimension's data is known. With this tighter bound, we can obtain the kNN results more efficiently within a smaller bound with the probability guarantee. Therefore, we mainly focus on how to derive the value of c when the value of μ is known.

Proposition 1. *We set the following two events with respect to A and B, which describe the exact condition and the approximate condition, respectively. And C indicates that Bregman distances are non-negative.*

$$A : \beta_{xy} < \mu, B : \beta_{xy} \leq c\mu (0 < c \leq 1), C : \beta_{xy} \geq -\kappa.$$

When the distribution of each dimension in the dataset is known[1] and the given probability guarantee is p, the value of c is:

$$c = \Psi_{\beta_{xy}}^{-1}(p\Psi_{\beta_{xy}}(\mu) + (1-p)\Psi_{\beta_{xy}}(-\kappa))/\mu,$$

where $\Psi_{\beta_{xy}}$ and $\Psi_{\beta_{xy}}^{-1}$ are used to denote the variable β_{xy}'s cumulative distribution function (cdf) and its inverse function, respectively.

Proof. Please see Section 5 in the supplementary file, available online. □

Proposition 1 illustrates that the exact bound is tightened by multiplying an approximate coefficient to obtain the approximate results with probability guarantees in the condition that the distribution of each dimension in the dataset is known. Fig. 6 shows the exact and approximate cases intuitively. When given the exact bound, we first compute the approximate coefficient in the original space and the exact bounds obtained in all partitions are multiplied by the approximate coefficient to be each partition's new approximate bound.

1. Histograms can be used to describe each dimension's distribution and a known distribution similar to the histogram (such as Normal distribution) can be chosen to fit the dimension's distribution by the least squares method.

TABLE 3
Parameters

Parameter	Varying Range
Result Size k	20, 40, 60, 80, 100
Dimensionality (Fonts)	100, 200, 300, 400
Data Size (Sift)	2M, 4M, 6M, 8M, 10M

TABLE 4
Datasets

Parameter	n	d	M	Page Size	Measure
Audio	54387	192	28	32 KB	ED
Fonts	745000	400	50	128 KB	ISD
Deep	1000000	256	37	64 KB	ED
Sift	11164866	128	22	64 KB	ED
Normal	50000	200	25	32 KB	ED
Uniform	50000	200	21	32 KB	ISD

9 Experiments

We conduct extensive experiments on four real datasets and two synthetic datasets to verify the efficiency of our methods. In this section, we present the experimental results.

9.1 Experiment Setup

9.1.1 Benchmark Method

According to Table 1, we select two state-of-the-art techniques [6], [18], which are designed for the exact kNN search for Bregman distances. We denote them by "VAF" and "BBT", respectively. Our method is denoted by "BP" (the abbreviation of "BrePartition") and its approximate versions are denoted by "ABP" (the abbreviation of "Approximate BrePartition"). Three key parameters are shown in Table 3. We measure the index construction time, I/O cost and CPU time to verify the efficiency of the benchmarks and our proposed method. These methods are implemented in Java. All experiments were done on a PC with Intel Core i7-2600M 3.40 GHz CPU, 8 GB memory and 1 TB WD Blue 3D NAND SATA SSD, running Windows x64.

9.1.2 Datasets

Four real-life datasets Audio,[2] Fonts,[3] Deep,[4] Sift[5] and two synthetic datasets are summarized in Table 4. Normal is a 200-dimensional synthetic dataset which has 50000 points generated by simulating the random number of standard normal distribution. Uniform is a 200-dimensional synthetic dataset which has 50000 points generated by simulating the random number of uniform distribution between [0,100]. Normal and Uniform are only used for evaluating our proposed approximate solution. For all these datasets, 50 points are randomly selected as the query sets. And we randomly select 50 samples for computing A, α and β.

9.2 Index Construction Time

In Fig. 7, we illustrate the index construction time of all the three testing methods on six testing datasets. The construction of VA-file is the fastest among all the methods on all the datasets, while the index structures constructed based on Bregman balls, such as BB-forest and BB-tree, increase the construction time by at least one magnitude because of the time-consuming clustering process. And the construction of BB-tree is slower than our proposed BB-forest since the clustering becomes less effective with the increase of dimensionality. In addition, it's indicated that the construction of VA-file becomes slower in high-dimensional spaces as well.

2. http://www.cs.princeton.edu/cass/audio.tar.gz
3. http://archive.ics.uci.edu/ml/datasets/Character+Font+Images
4. http://yadi.sk/d/I_yaFVqchJmoc
5. https://archive.ics.uci.edu/ml/datasets/SIFT10M

9.3 Impact of Dimensionality Partitioning

In our method, we develop the dimensionality partitioning technique to solve the kNN search problem in the high-dimensional space. Through theoretical analysis, the optimized number of partitions M can be derived and we obtain the optimized numbers of partitions on the four datasets according to Theorem 4, which are shown in Table 4. In addition, a novel strategy named PCCP are explored. In this part, we will evaluate the impact of the parameter M and PCCP on the efficiency of our method and validate that the derived value of M is optimized.

9.3.1 Impact of the Number of Partitions M

We show the results on four real datasets to demonstrate the impact of M on the efficiency. We vary k to 20, 40 and 60 to evaluate the I/O cost and the running time when the value of M is set within a certain range and the results are shown in Figs. 8 and 9. From Fig. 8, the I/O cost decreases as M increases in all the cases. Moreover, the I/O cost decreases more and more slowly as M increases. In Fig. 9, the trend of running time is different from that of the I/O cost, which generally has a trend of descent first and then ascent.

9.3.2 Validation of the Derived Optimized Value of M

As shown in Fig. 8, the I/O cost decreases more and more slowly as M increases. This is because the number of candidate points decreases at a slower and slower rate as the searching range decreases at a slower and slower rate. As can be seen in Fig. 8, the I/O cost is reduced at a low speed after M reaches the optimized value. Meanwhile, from Fig. 9, when the values of M are set to our derived optimized number of partitions on the four datasets, the CPU's running time is minimum in all cases which brings up to 30 percent gains. In addition, the random reads of the current mainstream SSD are about 5k IOPS to 500k IOPS. In this case, the efficiency gains achieved by the number of the I/O reductions can be negligible compared to the gains in the CPU's running time. Therefore, the experimental results validate that our derived value of M is optimized.

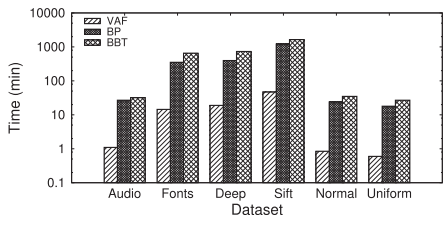


Fig. 7. Index construction time.

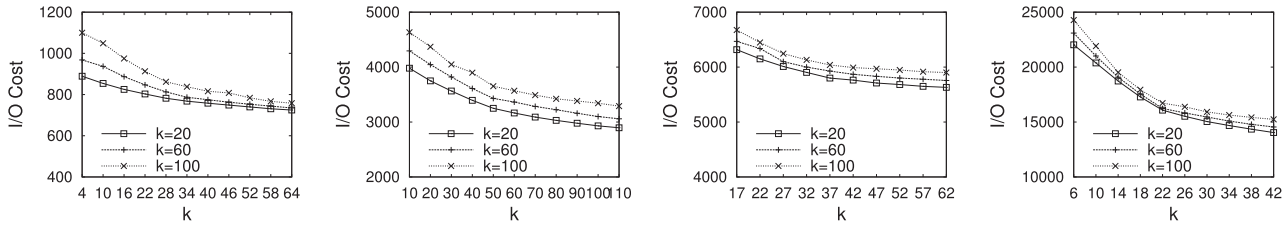


Fig. 8. I/O Cost (Impact of M).

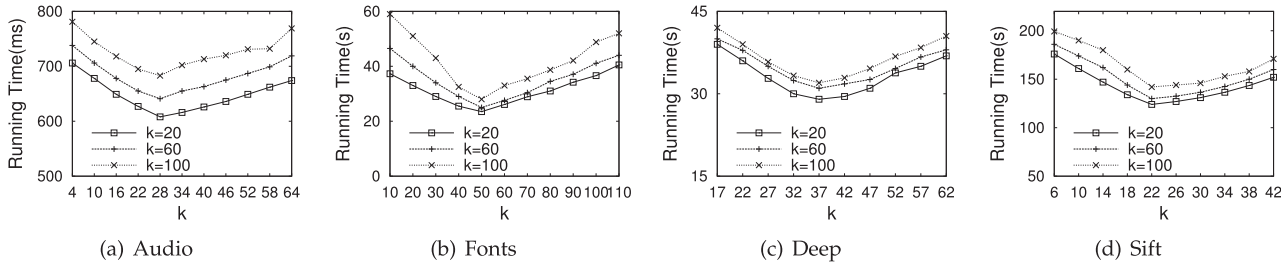


Fig. 9. Running time (impact of M).

9.3.3 Impact of PCCP

In this section, we evaluate how PCCP influences the efficiency of our method by testing it in both cases with and without PCCP and the results are shown in Fig. 10. The default value of k is 20. From the experimental results, both the I/O cost and the running time are reduced by 20 to 30 percent when PCCP is applied, which indicates that PCCP can reduce the number of candidate points leading to lower I/O costs and time consumption. Moreover, the experimental results demonstrate our proposed index structure, BB-forest, can avoid some invalid disk accesses relying on PCCP. We also evaluate the impact of randomly selecting the first dimension in PCCP on the performance. The experimental results are shown in the Section 7 of the supplementary file, available online.

9.4 I/O Cost

In this part, we evaluate the I/O cost by varying k from 20 to 100 and show the experimental results in Fig. 11. We extend the memory-resident BB-tree to a disk-resident index structure following the idea of our proposed BB-forest. As k increases, BrePartition outperforms the other two methods in I/O cost almost in all testing cases. It benefits from the derived bound's good pruning effect, which leads to a small quantity of candidate points. Moreover, BrePartition possesses good performance on all datasets, which shows that our proposed method can be applied to both large-scale and high-dimensional datasets.

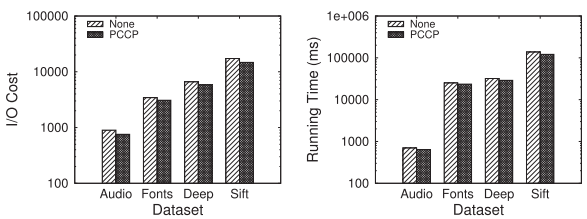


Fig. 10. Impact of PCCP.

From Fig. 11, we find that the I/O cost of VA-file is lower than that of BB-tree owing to two aspects. On the one hand, the lower and upper bounds computed by their proposed search algorithm are reasonable, even though less tighter than ours. On the other hand, the whole space has been compactly divided using large number of bits via VA-file which can reduce the size of the candidate set and fewer points will be loaded into main memory. Even so, loading more candidate points and all the disk-resident vector approximations causes more extra I/O costs. Moreover, BB-tree performs the worst among all the testing methods. This is because large overlaps among clusters in the high-dimensional space causes more candidates, which should be checked for the kNN results.

9.5 Running Time

In Fig. 12, we evaluate the CPU's running time by varying k to test the efficiency of our method. In all testing cases, BrePartition performs consistently the best regardless of the dimension and the size of datasets, which shows our method's high scalability. This is mainly because we partition the original high-dimensional space into several low-dimensional subspaces and build our proposed integrated index structure, BB-forest to accelerate the search processing.

Similarly, compared to BrePartition, VA-file doesn't show satisfying results on the CPU's running time. This is because it's required to scan all the vector approximations representing the data points when pruning the useless points for the candidate points. Besides, the reverse mapping from the vector approximations to their corresponding original points when verifying the candidate points is time-consuming as well. Similar to the I/O cost, BB-tree shows the worst results, because too many candidates to be checked cause performance degradation in the high-dimensional space.

9.6 Impact of Dimensionality

In this section, we evaluate how the dimensionality of a dataset influences the efficiency of these methods. We select Fonts

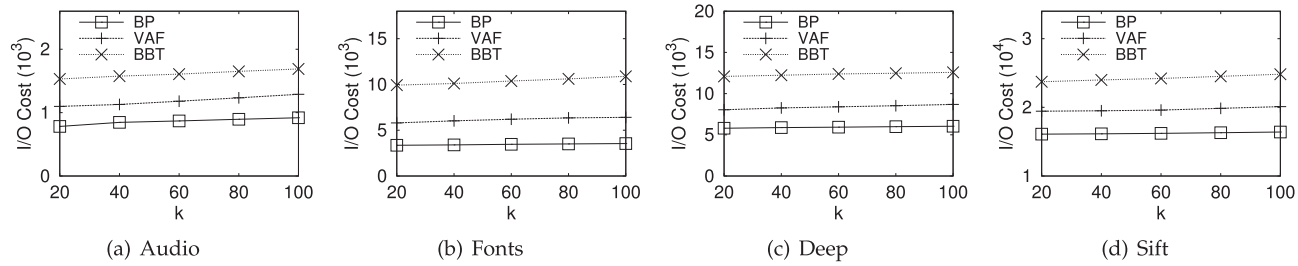


Fig. 11. I/O cost.

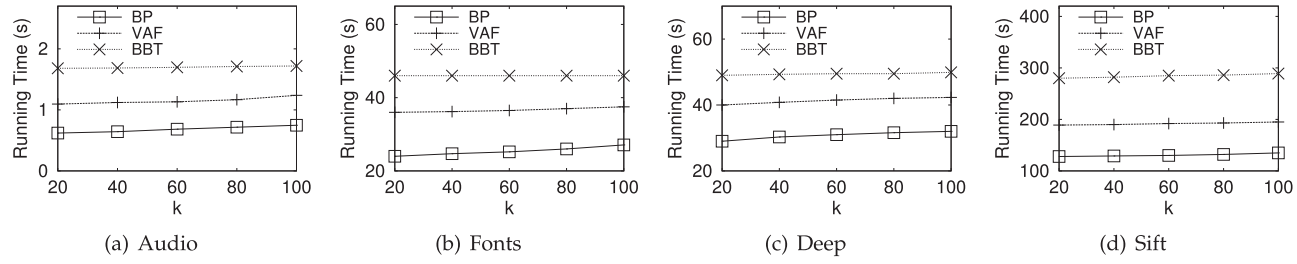


Fig. 12. Running time.

as the testing dataset and evaluate the I/O cost and the CPU's running time in different dimensions of the dataset. We vary the dimensionality from 10 to 400 and the experimental results are shown in Fig. 13. In BrePartition, when the dimensionality of testing changes, the number of partitions changes as well. According to Theorem 4, we set the computed optimized number of partitions to 3, 9, 13, 29 and 50, when the dimensionality is 10, 50, 100, 200 and 400, respectively.

The experimental results illustrate that both the I/O cost and the running time increase monotonically as the dimensionality of the dataset increases. BrePartition demonstrates the best performance as before, and the indicators show slighter increases as the dimensionality increases, which illustrates that it's applicable to various dimensional spaces. This is because BrePartition can derive appropriate bounds for different dimensional spaces. The growth rates of I/O cost and running time in VA-file increase as the dimensionality increases, which shows its poor scalability in high-dimensional space. BB-tree illustrates good performances in 10-dimensional space, but its I/O cost and running time increase significantly, which indicates it's only an efficient method in the low-dimensional space.

9.7 Impact of Data Size

In this part, we select the dataset Sift and set the data size from 2000000 to 10000000 to test these algorithms' I/O cost and CPU's running time in different scales of datasets. The results are illustrated in Fig. 14. Based on the above Theorem 4, the value of data size n impacts little on the value of partition's number M, so we consistently select 22 as the number of partitions regardless of changes in the data size.

From the experimental results, the I/O cost and the running time almost increase linearly as the size of the dataset increases. BrePartition requires the lowest I/O cost and running time indicating its good scalability with the increasing data amount. In addition, VA-file demonstrates comparable performances in I/O cost and running time as well. Compared to BrePartition, BB-tree's I/O cost and running time are multiplied since it isn't suitable for high-dimensional spaces as explained in Sections 9.4 and 9.5.

9.8 Evaluations of Approximate Solution

We evaluate our proposed approximate solution in this section. Our solution is compared with the state-of-the-art approximate solution proposed in [34] denoted as "Var" in our paper. We first define overall ratio denoted as $OR = \frac{1}{k}\sum_{i=1}^{k} \frac{D_f(p_i,q)}{D_f(p_i^*,q)}$, where p_i is the ith point returned and p_i^* is the exact ith NN point. Overall ratio describes the accuracy of the approximate solutions. A smaller overall ratio means a higher accuracy. Since "Var" is based on the traditional BB-tree, we also extend the memory-resident BB-tree to a disk-resident index structure following the idea of our proposed BB-forest to test the I/O cost. In addition, the running time is also tested for evaluating its efficiency. The results are shown in Fig. 15.

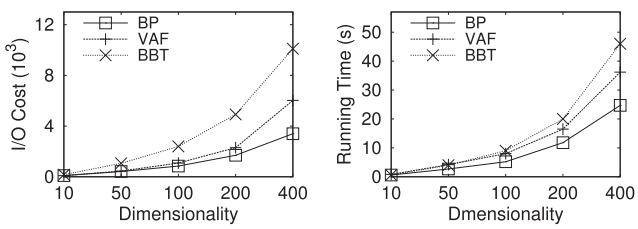


Fig. 13. Impact of dimensionality (fonts).

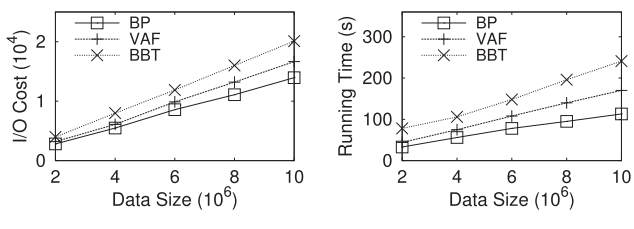
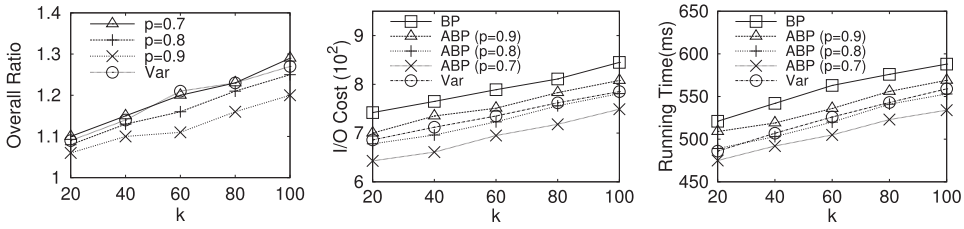


Fig. 14. Impact of data size (sift).



(a) Overall Ratio (b) I/O Cost (c) Running Time

Fig. 15. Evaluations of approximate solution (normal).

Fig. 15a shows the overall ratios by varying k from 20 to 100. Generally, a larger k can lead to a larger overall ratio. Moreover, we evaluate the overall ratio when the probability guarantee is set to 0.7, 0.8 and 0.9, respectively. From the experimental results, the overall ratio decreases as p increases which indicates that a higher probability guarantee leads to a higher accuracy. Compared with "Var", our solution almost performs better in all cases on Normal. Similarly, we evaluate the I/O cost and the running time by varying k and p. The experimental results are shown in Figs. 15b and 15c. Generally, the I/O cost and the running time increase as k increases. And we observe reverse trends from the results when varying p because the searching range will be extended as p increases. Since "Var" reduce the number of nodes checked in the searching process using data's distributions, the I/O cost and the running time are reduced. Even though, the I/O cost and the running time of "Var" are larger than our solution in most cases. It indicates that our approximate solution can yield the higher efficiency while ensuring the accuracy. The experimental results on Uniform are similar to those on Normal, they are shown in Section 6 in the supplementary file, available online.

10 Conclusion

In this paper, we address the important problem of high-dimensional kNN search with Bregman distances and propose a dimensionality partitioning approach named BrePartition. BrePartition follows a partition-filter-refinement framework. We have proposed a number of novel techniques to overcome the challenges of the problem. First, we derive an effective upper bound based on Cauchy inequality as the pruning condition to significantly reduce the number of candidates we need to check closely. Second, we optimize the dimensionality partitioning by computing the optimized number of partitions to reduce the running time and devising a strategy called PCCP to further reduce the size of the candidate set. Third, we design an integrated index structure, named BB-forest, which consists of BB-trees for all the individual subspaces. In addition, we extend our exact solution to an approximate version via data's distribution. Experimental results demonstrate that our method can yield significant performance improvement in CPU's running time and I/O cost. In the future work, we will further improve the existing brief approximate solution and propose a more efficient solution by converting the Bregman distance into Euclidian distance and employing traditional metric searching methods to solve the high-dimensional kNN search with Bregman distances. In addition, we will also improve our designed BB-forest so that it can support inserting or deleting large-scale data more efficiently.

Acknowledgments

This work was supported by the National Key Research and Development Program of China (2018YFB1003400), the National Natural Science Foundation of China (U1811261) and Liaoning Revitalization Talents Program (XLYC1807158).

References

[1] J. Goldberger, S. Gordon, and H. Greenspan, "An efficient image similarity measure based on approximations of kl-divergence between two gaussian mixtures," in *Proc. 9th IEEE Int. Conf. Comput. Vis.*, 2003, pp. 487–493.

[2] F. Nielsen and R. Nock, "On approximating the smallest enclosing bregman balls," in *Proc. 22nd ACM Symp. Comput. Geometry*, 2006, pp. 485–486.

[3] B. Long, Z. M. Zhang, and P. S. Yu, "Graph partitioning based on link distributions," in *Proc. 22nd AAAI Conf. Artif. Intell.*, 2007, pp. 578–583.

[4] Y. Mu and S. Yan, "Non-metric locality-sensitive hashing," in *Proc. 24th AAAI Conf. Artif. Intell.*, 2010, pp. 539–544.

[5] J. Laub, J. H. Macke, K. Müller, and F. A. Wichmann, "Inducing metric violations in human similarity judgements," in *Proc. 20th Annu. Conf. Neural Inf. Process. Syst.*, 2006, pp. 777–784.

[6] Z. Zhang, B. C. Ooi, S. Parthasarathy, and A. K. H. Tung, "Similarity search on bregman divergence: Towards non-metric indexing," *Proc. VLDB Endowment*, vol. 2, no. 1, pp. 13–24, 2009.

[7] J. Puzicha, Y. Rubner, C. Tomasi, and J. M. Buhmann, "Empirical evaluation of dissimilarity measures for color and texture," in *Proc. Int. Conf. Comput. Vis.*, 1999, pp. 1165–1172.

[8] X. Li *et al.*, "Learning bregman distance functions for structural learning to rank," *IEEE Trans. Knowl. Data Eng.*, vol. 29, no. 9, pp. 1916–1927, Sep. 2017.

[9] N. Rasiwasia, P. J. Moreno, and N. Vasconcelos, "Bridging the gap: Query by semantic example," *IEEE Trans. Multimedia*, vol. 9, no. 5, pp. 923–938, Aug. 2007.

[10] R. Gray, A. Buzo, A. Gray, and Y. Matsuyama, "Distortion measures for speech processing," *IEEE Trans. Acoust. Speech Signal Process.*, vol. ASSP-28, no. 4, pp. 367–376, Aug. 1980.

[11] X. Hua, Y. Cheng, H. Wang, Y. Qin, and D. Chen, "Geometric target detection based on total bregman divergence," *Digital Signal Process.*, vol. 75, pp. 232–241, 2018.

[12] D. P. L. Ferreira, E. Ribeiro, and C. A. Z. Barcelos, "A variational approach to non-rigid image registration with bregman divergences and multiple features," *Pattern Recognit.*, vol. 77, pp. 237–247, 2018.

[13] D. Cheng, D. Shi, F. Tian, and X. Liu, "A level set method for image segmentation based on bregman divergence and multiscale local binary fitting," *Multimedia Tools Appl.*, vol. 78, no. 15, pp. 20 585–20 608, 2019.

[14] A. Okuno and H. Shimodaira, "Hyperlink regression via bregman divergence," *CoRR*, vol. abs/1908.02573, 2019.

[15] K. Q. Weinberger, J. Blitzer, and L. K. Saul, "Distance metric learning for large margin nearest neighbor classification," in *Proc. Advances Neural Inf. Process. Syst.*, 2005, pp. 1473–1480.

[16] S. Ramaswamy and K. Rose, "Fast adaptive mahalanobis distance-based search and retrieval in image databases," in *Proc. Int. Conf. Image Process.*, 2008, pp. 181–184.

[17] T. Dong, Y. Ishikawa, and C. Xiao, "Top-k similarity search over gaussian distributions based on kl-divergence," *J. Inf. Process.*, vol. 24, no. 1, pp. 152–163, 2016.
[18] L. Cayton, "Fast nearest neighbor retrieval for bregman divergences," in *Proc. 25th Int. Conf. Mach. Learn.*, 2008, pp. 112–119.
[19] J. L. Bentley, "Multidimensional binary search trees used for associative searching," *Commun. ACM*, vol. 18, no. 9, pp. 509–517, 1975.
[20] A. Guttman, "R-trees: A dynamic index structure for spatial searching," in *Proc. Annu. Meeting*, 1984, pp. 47–57.
[21] H. V. Jagadish, B. C. Ooi, K. Tan, C. Yu, and R. Zhang, "idistance: An adaptive b^+-tree based indexing method for nearest neighbor search," *ACM Trans. Database Syst.*, vol. 30, no. 2, pp. 364–397, 2005.
[22] A. Arora, S. Sinha, P. Kumar, and A. Bhattacharya, "Hd-index: Pushing the scalability-accuracy boundary for approximate KNN search in high-dimensional spaces," *Proc. VLDB Endowment*, vol. 11, no. 8, pp. 906–919, 2018.
[23] R. Zhang, B. C. Ooi, and K. Tan, "Making the pyramid technique robust to query types and workload," in *Proc. 20th IEEE Int. Conf. Data Eng.*, pp. 313–324, 2004.
[24] J. Heo, Z. L. Lin, and S. Yoon, "Distance encoded product quantization for approximate k-nearest neighbor search in high-dimensional space," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 41, no. 9, pp. 2084–2097, Sep. 2019.
[25] Y. Liu, H. Cheng, and J. Cui, "PQBF: I/O-efficient approximate nearest neighbor search by product quantization," in *Proc. ACM Conf. Inf. Knowl. Manage.*, 2017, pp. 667–676.
[26] Y. Liu, J. Cui, Z. Huang, H. Li, and H. T. Shen, "SK-LSH: An efficient index structure for approximate nearest neighbor search," *Proc. VLDB Endowment*, vol. 7, no. 9, pp. 745–756, 2014.
[27] J. Gao, H. V. Jagadish, W. Lu, and B. C. Ooi, "DSH: Data sensitive hashing for high-dimensional K-NN search," in *Proc. Int. Conf. Manage. Data*, 2014, pp. 1127–1138.
[28] L. Chen, Y. Gao, B. Zheng, C. S. Jensen, H. Yang, and K. Yang, "Pivot-based metric indexing," *Proc. VLDB Endowment*, vol. 10, no. 10, pp. 1058–1069, 2017.
[29] W. Liu, H. Wang, Y. Zhang, W. Wang, and L. Qin, "I-LSH: I/O efficient c-approximate nearest neighbor search in high-dimensional space," in *Proc. 35th IEEE Int. Conf. Data Eng.*, 2019, pp. 1670–1673.
[30] Y. Hwang, B. Han, and H. Ahn, "A fast nearest neighbor search algorithm by nonlinear embedding," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2012, pp. 3053–3060.
[31] Y. Sun, W. Wang, J. Qin, Y. Zhang, and X. Lin, "SRS: Solving C-approximate nearest neighbor queries in high dimensional euclidean space with a tiny index," *Proc. VLDB Endowment*, vol. 8, no. 1, pp. 1–12, 2014.
[32] Y. Gu, Y. Guo, Y. Song, X. Zhou, and G. Yu, "Approximate order-sensitive K-NN queries over correlated high-dimensional data," *IEEE Trans. Knowl. Data Eng.*, vol. 30, no. 11, pp. 2037–2050, Nov. 2018.
[33] F. Nielsen, J. Boissonnat, and R. Nock, "On bregman voronoi diagrams," in *Proc. 18th Annu. ACM-SIAM Symp. Discrete Algorithms*, 2007, pp. 746–755.
[34] E. Coviello, A. Mumtaz, A. B. Chan, and G. R. G. Lanckriet, "That was fast! speeding up NN search of high dimensional distributions," in *Proc. 30th Int. Conf. Mach. Learn.*, 2013, pp. 468–476.
[35] L. Cayton, "Efficient bregman range search," in *Proc. 23rd Annu. Conf. Neural Inf. Process. Syst.*, 2009, pp. 243–251.
[36] M. R. Ackermann and J. Blömer, "Coresets and approximate clustering for bregman divergences," in *Proc. 20th Annu. ACM-SIAM Symp. Discrete Algorithms*, 2009, pp. 1088–1097.
[37] A. Abdullah, J. Moeller, and S. Venkatasubramanian, "Approximate bregman near neighbors in sublinear time: beyond the triangle inequality," *Int. J. Comput. Geometry Appl.*, vol. 23, no. 4–5, pp. 253–302, 2013.
[38] D. P. L. Ferreira, B. M. Rocha, and C. A. Z. Barcelos, "Nearest neighbor search on total bregman balls tree," in *Proc. Symp. Appl. Comput.*, 2017, pp. 120–124.
[39] C. Faloutsos and K. Lin, "Fastmap: A fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 1995, pp. 163–174.
[40] X. Wang, J. T. Wang, K. Lin, D. E. Shasha, B. A. Shapiro, and K. Zhang, "An index structure for data mining and clustering," *Knowl. Inf. Syst.*, vol. 2, no. 2, pp. 161–184, 2000.
[41] V. Athitsos, J. Alon, S. Sclaroff, and G. Kollios, "Boostmap: A method for efficient approximate similarity rankings," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, 2004, pp. 268–275.
[42] V. Athitsos, M. Hadjieleftheriou, G. Kollios, and S. Sclaroff, "Query-sensitive embeddings," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2005, pp. 706–717.
[43] T. Skopal, "Unified framework for fast exact and approximate search in dissimilarity spaces," *ACM Trans. Database Syst.*, vol. 32, no. 4, 2007, Art. no. 29.
[44] T. Skopal and J. Lokoc, "Nm-tree: Flexible approximate similarity search in metric and non-metric spaces," in *Proc. 19th Int. Conf. Database Expert Syst. Appl.*, 2008, pp. 312–325.
[45] D. Liu and K. A. Hua, "Transfer non-metric measures into metric for similarity search," in *Proc. 17th Int. Conf. Multimedia*, 2009, pp. 693–696.
[46] L. Chen and X. Lian, "Efficient similarity search in nonmetric spaces with local constant embedding," *IEEE Trans. Knowl. Data Eng.*, vol. 20, no. 3, pp. 321–336, Mar. 2008.
[47] P. Jangyodsuk, P. Papapetrou, and V. Athitsos, "Optimizing hashing functions for similarity indexing in arbitrary metric and non-metric spaces," in *Proc. SIAM Int. Conf. Data Mining*, 2015, pp. 828–836.
[48] K. Goh, B. Li, and E. Y. Chang, "Dyndex: A dynamic and non-metric space indexer," in *Proc. 10th ACM Int. Conf. Multimedia*, 2002, pp. 466–475.
[49] Y. A. Malkov and D. A. Yashunin, "Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs," *CoRR*, vol. abs/1603.09320, 2016.

Yang Song received the BE degree in automation from Northeastern University, China, in 2016. He is currently working toward the PhD degree in computer application technology at Northeastern University, China. His research interests include query processing and query optimization.

Yu Gu received the PhD degree in computer software and theory from Northeastern University, China, in 2010. He is currently a professor at Northeastern University, China. His current research interests include big data processing, spatial data management, and graph data management. He is a senior member of China Computer Federation (CCF).

Rui Zhang is a Professor at the School of Computing and Information Systems of the University of Melbourne. His research interests include big data and machine learning, particularly in spatial and temporal data analytics, database indexing, chatbots and recommender systems. He has won several awards including the Future fellowship by the Australian Research Council in 2012, Chris Wallace Award for Outstanding Research by the Computing Research and Education Association of Australasia (CORE) in 2015, and Google faculty Research Award in 2017.

Ge Yu (Member, IEEE) received the PhD degree in computer science from the Kyushu University of Japan, in 1996. He is currently a professor with the Northeastern University of China. His research interests include distributed and parallel database, OLAP and data warehousing, data integration, graph data management, etc. He has published more than 200 papers in refereed journals and conferences. He is a fellow of CCF and a member of the IEEE Computer Society and ACM.