Fast Kd-Trees for the Kullback–Leibler Divergence and Other Decomposable Bregman Divergences

Tuyen Pham
University of Florida, Gainesville, FL, USA

Hubert Wagner
University of Florida, Gainesville, Fl, USA

Abstract

The contributions of the paper span theoretical and implementational results. First, we prove that Kd-trees can be extended to $\mathbb{R}^d$ with the distance measured by an arbitrary Bregman divergence. Perhaps surprisingly, this shows that the triangle inequality is not necessary for correct pruning in Kd-trees. Second, we offer an efficient algorithm and C++ implementation for nearest neighbour search for decomposable Bregman divergences.

The implementation supports the Kullback–Leibler divergence (relative entropy) which is a popular distance between probability vectors and is commonly used in statistics and machine learning. This is a step toward broadening the usage of computational geometry algorithms.

Our benchmarks show that our implementation efficiently handles both exact and approximate nearest neighbour queries. Compared to a linear search, we achieve two orders of magnitude speedup for practical scenarios in dimension up to 100. Our solution is simpler and more efficient than competing methods.

2012 ACM Subject Classification Theory of computation → Computational geometry; Information systems → Data structures; Mathematics of computing → Combinatorial algorithms

Keywords and phrases Kd-tree, k-d tree, nearest neighbour search, Bregman divergence, decomposable Bregman divergence, KL divergence, relative entropy, cross entropy, Shannon's entropy

Digital Object Identifier 10.4230/LIPIcs.WADS.2025.45

Funding 2022 Google Research Scholar Award in Algorithms and Optimization.

1 Motivation

Nearest neighbour search is a fundamental method offered by computational geometry, with applications in a wide range of fields. Bentley's k-dimensional tree [11], Kd-tree for short, is among the simplest and most practical data structures for this task.

Like many other computational geometry techniques, Kd-trees were initially designed for Euclidean space and later extended to more general metric spaces. However, many modern geometric problems, particularly in data science, use distances that are not proper metrics. For example, it is common to represent data as probability vectors and use specialized dissimilarities to measure distances between them. While standard geometric algorithms do not work with non-metric distances, they can often be extended to such settings.

Indeed, it is interesting that many computational geometry algorithms – that are typically assumed to require a metric – can work with significantly weaker assumptions. Specifically, we may omit the requirement of symmetry or the triangle inequality. We will mention prominent examples in Section 2, and prove that the above statement extends to Kd-trees. In particular, correctness and efficiency guarantees can be retained.

Applications. One practical example of such a non-metric distance is the Kullback–Leibler (KL) divergence [28]. Originating in information theory [47], the KL divergence is a standard way of comparing discrete probability distributions (probability vectors). For example, it

is used as a loss function minimized in machine learning [34, 37, 50], such as in variational auto-encoders [26]. Approximate nearest neighbours queries are becoming an increasingly important component of modern machine learning. In particular, retrieval-augmented generation (RAG) aims to improve large language models by searching for existing documents to support generated answers. This is done by a nearest neighbour search within probability vectors. Typically, heuristic methods, such as the small world graphs [30], which lack performance guarantees, are used [33, 35, 51]. Recently, Indyk and Xu [24] warned that the methods used in such contexts can catastrophically fail. Regardless, there is a growing field of *vector databases* focusing on supporting nearest neighbour queries [23].

Problem statement. We revisit the topic of nearest neighbour search, focusing on algorithms that provide exact answers as well as approximate results with guarantees. In particular, we investigate non-metric geometries induced by Bregman divergences – of which the KL divergence is a prominent member.

Given a finite collection of points $X \subset \Omega \subset \mathbb{R}^d$ and a query point $q \in \Omega$, select k points from X with the smallest distance to q. Specifically, the distance will be measured using a Bregman divergence, which we discuss in Section 3. We will design and implement a modified Kd-tree for answering queries in this setting.

Contributions. We list the contributions of this paper:

1. Theoretical results on correctness and efficiency of Kd-tree queries in the setting of Bregman divergences.
2. The first implementation of Kd-trees for Bregman divergences. It is currently the fastest method for exact k-nearest neighbour queries in the Bregman setting and works for arbitrary decomposable divergences[1].
3. Benchmarks showing the method is usable in practical situations with real datasets.

2 Related work

Kd-Trees were introduced by Jon Bentley in 1975 [11]. Further improvements were made by him, Friedman and Finkel [22] and many others. Bregman divergences were introduced by Lev Bregman in 1967 [16].

Many computational geometry techniques have been extended to operate with Bregman divergences instead of a metric. In the context of nearest neighbour search, Cayton first extended ball-trees [19], implementing prototype software for the KL and Itakura–Saito divergences. Cayton also proved theoretical results towards extending Kd-trees [17], which we strengthen as well as provide algorithms and an efficient implementation. In turn, Nielsen, Piro, and Barlaud extended Vantage point trees [40, 41]. The same authors further improved Bregman ball-trees [18, 42]. These methods rely on a bisection search and in some cases a preprocessing transformation. We show that Kd-trees are able to perform exact nearest neighbour searches without relying on these computations.

Rectangle-trees (R-trees) and vector approximation files (VA-files) were extended by Zhang and collaborators [53]. These implementations inspired Song and collaborators to develop BrePartitions [48]. Ring-trees combined with a quad-tree decomposition have been proven to work sublinearly for finding approximate nearest neighbours by Abdullah, Moeller,

[1] We are working on incorporating our implementation into the popular scikit-learn [44] library, which will increase both the generality and efficiency of its existing Kd-trees implementation.

and Venkatasubramanian [2]. In 2013, Boytsov and Naidan developed their own Bregman VP-trees extension [15] for approximate nearest neighbours. Naidan later incorporated his VP-trees and Cayton's ball-trees into the Non-Metric Space Library (NMSLIB) [14]. This library also includes other approximate Bregman similarity searches including small world graphs [32]. The hierarchical navigable small world graph has been a popular choice for similarity searches in vector databases [33, 35, 51] and performs well in benchmarks for metrics [8]. However, its implementation in NMSLIB is currently experimental for Bregman divergences [31]. Recently Abdelkader, Arya, da Fonseca and Mount proposed an approach to proximity search in non-metric settings, which includes Bregman divergences [1]; as we understand it, this has not yet been implemented.

More broadly, Banerjee and collaborators extended k-means clustering [9] to arbitrary Bregman divergences – with the surprising twist that the existing algorithm works without changes. Coresets have also been extended to the Bregman setting by Ackermann and Blömer [3]. Nielsen, Boissonnat, and Nock developed Bregman Voronoi diagrams and Delaunay triangulations [13]. Edelsbrunner and Wagner extended topological data analysis methods to the Bregman case [21].

In the Euclidean case, robust software is available for all of these techniques. One popular package in the Euclidean case is the ANN library by Mount and Arya [5–7, 36]. Our current implementation is inspired by this library.

3 Background on Bregman divergences

We begin by setting up definitions for Bregman divergences [16], which we use as a measure of distance. These divergences are usually asymmetric and do not generally satisfy the triangle inequality – and as such do not define a proper metric. Despite this limitation, decomposable Bregman divergences will efficiently work with Kd-trees with minimal changes.

Each Bregman divergence is parametrized by a convex function with particular properties [10]. We set the stage by letting $\Omega \subseteq \mathbb{R}^d$ be an open convex set. Next, we define a **function of Legendre type** [46] as a function $F : \Omega \to \mathbb{R}$ that is:

- I differentiable and
- II strictly convex.
- III We additionally require that $\lim_{x \to \partial\Omega} \|\nabla F(x)\| = \infty$, provided $\partial\Omega$ is nonempty.

The third requirement is often omitted, but will prove important in Section 5.

Given a function F of Legendre type, the **Bregman divergence generated** by F is a function $D_F : \Omega \times \Omega \to \mathbb{R}$. The value of the divergence between x and y is the difference between $F(x)$ and the best affine approximation of F at y also evaluated at x, or simply

$$D_F(x\|y) = F(x) - (F(y) + \langle \nabla F(y), x - y \rangle). \tag{1}$$

See Figure 1 for an illustration. We refer to $D_F(x\|y)$ as the divergence in the **direction from x to y**. Due to the lack of symmetry, we will be mindful about the direction in which we compute it.

All Bregman divergences fulfill the following:

▶ **Property 1** (Bregman Nonnegativity). *$D_F(x\|y) \geq 0$ for each $x, y \in \Omega$, with equality if and only if $x = y$.*

Despite failing to satisfy the requirements for a metric, various computational geometry algorithms extend to Bregman divergences. Also, despite the seemingly simple definition, the resulting divergences have interesting properties and interpretations.

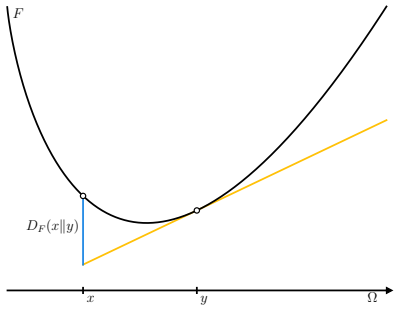

Figure 1 Visualization of a Bregman divergence construction for a one-dimensional domain.

Table 1 List of common decomposable Bregman divergences.

Domain	Legendre type function	Divergence	Name
$\mathbb{R}^d$	$\sum_{i=1}^d x_i^2$	$\sum_{i=1}^d (x_i - y_i)^2$	Squared Euclidean (SE)
$\mathbb{R}^d_+$	$-\sum_{i=1}^d x_i \log_2 \frac{1}{x_i}$	$\sum_{i=1}^d x_i \log_2 \frac{x_i}{y_i} + \frac{y_i - x_i}{\ln 2}$	Generalized Kullback–Leibler (GKL)
$\triangle^{d-1}$	$-\sum_{i=1}^d x_i \log_2 \frac{1}{x_i}$	$\sum_{i=1}^d x_i \log_2 \frac{x_i}{y_i}$	Kullback–Leibler (KL)
$\mathbb{R}^d_+$	$-\sum_{i=1}^d \log x_i$	$\sum_{i=1}^d \frac{x_i}{y_i} - \log \frac{x_i}{y_i} - 1$	Itakura–Saito (IS)
$\mathbb{R}^d_+$	$-\sum_{i=1}^d -\sqrt{x_i}$	$\sum_{i=1}^d \frac{\sqrt{y_i}}{2} + \frac{x_i}{2\sqrt{y_i}} - \sqrt{x_i}$	Bhattacharyya-Like
$\Omega_1 \cap \Omega_2$	$\lambda F_1 + (1-\lambda)F_2$, $\lambda \in [0,1]$	$\lambda D_{F_1} + (1-\lambda) D_{F_2}$	Interpolated divergence

Decomposable Bregman divergences. Our focus is on a sub-family of Bregman divergences called **decomposable Bregman divergence** [38, 52]. They are generated by a function $F = \sum_{i=1}^d f_i$, where each f_i is a univariate function of Legendre type. The function, F, generates a Bregman divergence of the form $D_F(x\|y) = \sum_{i=1}^d D_{f_i}(x_i\|y_i)$ for x_i, y_i (namely the components of x and y) lying in the domain of f_i [39]. Most divergences used in practice belong to this family.

We list common decomposable Bregman divergences in Table 1. The most commonly used decomposable Bregman divergence is the **squared Euclidean distance** (SE). Of particular interest is the **generalized Kullback–Leibler divergence** (GKL) defined over $\mathbb{R}^d_+$. It reduces to the standard KL divergence for points on the **open standard simplex**, $\triangle^{d-1} = \{x \in \mathbb{R}^d : \sum_{i=1}^d x_i = 1,\ x_i > 0\}$. Another example is the **Itakura–Saito** (IS) divergence [25], which is useful for working with speech and sound data [20]. There are many other decomposable Bregman divergences, some inspired by popular tools in statistics such as the Bhattacharyya distance. Additionally, given functions of Legendre type, F_1 and F_2, we can form a new divergence generated by the weighted sum $\lambda F_1 + (1-\lambda)F_2$ for $\lambda \in [0,1]$ [46]. This new Bregman divergence which can be viewed as an interpolation between the two Bregman divergences D_{F_1} and D_{F_2}.

One outlier is the squared Mahalanobis distance [29] which is popular in statistics. While not decomposable, nearest neighbour problems involving this divergence can be reduced to the squared Euclidean distance that is decomposable [27]. Another one is the KL divergence on the *closed simplex*. While it does not fall under our definition, it can be treated as a limit case of the KL on the open simplex.

Overall, as the prominent Bregman divergences are decomposable, restricting our attention to decomposable divergences over open domains will not limit the choice of divergences handled in practice.

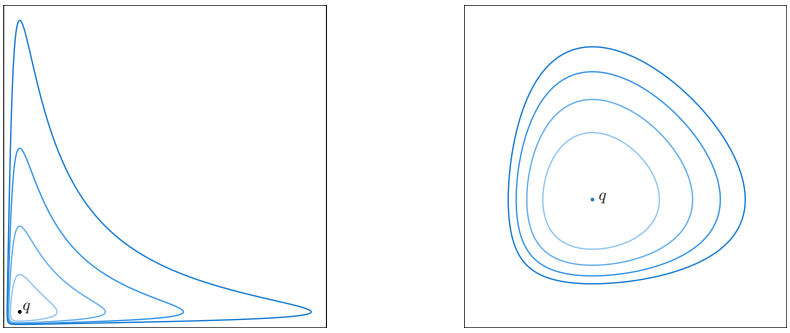

Figure 2 Left: primal Itakura–Saito balls. Right: primal generalized Kullback–Leibler balls.

Bregman balls. Due to the asymmetry, one can define two types of Bregman balls [43]. We start from the **primal Bregman ball** of radius $r \geq 0$ centered at q which is defined as

$$B_F(q; r) = \{y \in \Omega : D_F(q\|y) \leq r\}. \tag{2}$$

Namely, it is the collection of points with Bregman divergence measured *from* the center not exceeding r. See Figure 2 for an illustration.

The **dual Bregman ball** of radius $r \geq 0$ centered at q is defined as

$$B'_F(q; r) = \{y \in \Omega : D_F(y\|q) \leq r\}. \tag{3}$$

As seen in Figure 2, and observed in [13], primal Bregman balls can be non-convex (when viewed as a subset of Euclidean space). It is reasonable to question if all balls are necessarily connected. In Section 5 we will show that the balls are indeed connected, and emphasize why this property is crucial.

Bregman projections. While different in many aspects from metrics, Bregman divergences often exhibit familiar behaviours. We mention standard results related to projections, which we sharpen in the subsequent sections.

Given a Bregman divergence D_F, we consider the **Bregman projection** to a point q from a nonempty $C \subset \Omega$:

$$\mathrm{proj}_F(q, C) = \operatorname*{arginf}_{x \in C} D_F(x\|q).$$

When C is closed and convex, this projection exists and is unique. In this case, this point is declared to be the Bregman projection of q onto C [4]. In analogy with projection distance, we define the Bregman **projection divergence** as $D_F(\mathrm{proj}_F(q, C)\|q)$, the infimum of divergences from C to q. We state the following useful statement [13].

▶ **Lemma 1** (Bregman Projection [13]). *Given a nonempty closed convex set $C \subset \Omega$ and $q \in \Omega$, denote $q_C = \mathrm{proj}_F(q, C)$. For all $x \in C$:*

$$D_F(x\|q) \geq D_F(x\|q_C) + D_F(q_C\|q).$$

If C is an affine subspace, the above is an equality.

However, this definition and theorem only apply when D_F is computed to q from q_C. As $D_F(x\|y)$ may not be convex in the second coordinate, a projection, $\arg\min_{x \in C} D_F(q\|x)$, is not generally defined in Bregman divergence literature. However, by restricting the setup to axis-aligned boxes, we obtain a similar result working in both directions. We provide results for divergences computed *from* a query, but results and proofs for the reverse direction are analogous.

4 Kd-trees

We briefly overview a version of the Kd-tree data structure introduced by Bentley [11], focusing on nearest neighbour queries. It is a binary tree which encodes recursive partitioning of $\mathbb{R}^d$ (in practice: a sufficiently large box contained in it) into axis-aligned boxes. We consider a variant in which each node corresponds to an axis-aligned box and the data points are stored only in the leaves. We highlight the changes required to extend the standard method to the Bregman setting.

Construction. Kd-trees partition the space by cutting it with axis-aligned hyperplanes, often called splitting planes or cutting planes. The details of the construction, i.e. the order and locations of the splits, can have a significant impact on efficiency of Kd-trees [11,12,22,49]. However, the construction does not depend on the choice of a metric, or divergence. We therefore only mention that the standard splitting methods work in the Bregman case. It is worth emphasizing that once the tree is constructed, each query can be made efficiently using any *decomposable* divergence.

Nearest neighbour queries in the Bregman case. Consider a Kd-tree constructed from a finite set of **data points**, $X \subset \Omega \subset \mathbb{R}^d$. To simplify exposition, we focus on finding the single, exact nearest neighbour of the query point q among the points in X. More precisely, we consider the primal Bregman nearest neighbour, $\operatorname{argmin}_{x \in X} D_F(q\|x)$. The proposed algorithm is applicable for any Bregman divergence and is particularly efficient in the decomposable case.

We recall that the query can be performed using a simple recursive procedure. It traverses the tree trying to prune as many subtrees as possible, while guaranteeing that all viable candidates are considered. We overview the algorithm in the Bregman case now, and present an implementation in Section 7.

- The base case: divergences from q to the points stored in a leaf are compared with the divergence, r_{nn}, to the current best candidate which is updated if needed.
- We first visit child nodes based on the relative location of the query point and the splitting plane at this node. As such, this step does not depend on the choice of divergence.
- Moving back to the root, we visit the remaining subtrees only if they cannot be safely pruned. This is decided by what we call the **pruning test**. Conceptually, we rephrase it as an **intersection test** between the axis-aligned box corresponding to the remaining node and a primal Bregman ball. Specifically, we mean the ball centered at q of radius r_{nn}. Clearly, if the box and the ball are disjoint, all data points in the box are in the complement of the ball, and are therefore too far away to contribute.

Details. We mention that finding the dual Bregman nearest neighbour is completely analogous. Finding the k nearest neighbours, is another easy modification involving a priority queue. In this case r is set to the divergence to the current k-th nearest neighbour, or infinity. To allow approximate queries, the radius of the ball is decreased to $\frac{r}{1+\varepsilon}$. Our implementation supports all these options. We skip the details for brevity.

Pruning test in practice. In practice, to determine if a given box can be safely pruned, we will perform a Bregman projection of the query point onto the boundary of the box. Before we describe the implementation, we must prove that – despite the lack of symmetry and triangle inequality – correct and efficient pruning is possible. To this end, we focus on problems related to intersecting a Bregman ball $B \subset \Omega \subset \mathbb{R}^d$ with an axis-aligned box $A \subset \mathbb{R}^d$.

We divide our argument into two parts. Part I is presented in Section 5 and is more topological: we show that intersecting B with the boundary of A is an equivalent test. This argument works for arbitrary Bregman divergences, not only decomposable ones. Part II is presented in Section 6 and is more geometric: we replace the intersection test with a projection and show it can be computed in a simple efficient way. This part is specific to decomposable divergences.

5 Proof of pruning correctness

We consider a Bregman ball $B \subset \Omega \subset \mathbb{R}^d$ and an axis-aligned box $A \subset \mathbb{R}^d$. The intersection test checks if the intersection $A \cap B$ is nonempty. We first prove a crucial result which relies on the Legendre Transform.

Legendre transform. The Legendre transform is a tool from convex geometry [46]. In the context of Bregman divergences, it is used to map a Bregman generator over a domain into another generator over a possibly different domain [13]. In particular, it transforms primal balls in one domain into dual balls in the other domain, and vice versa. We will see one basic application of this tool.

More technically, given a function of Legendre type, $F : \Omega \to \mathbb{R}$, there exists the **Legendre transform** which maps F to a conjugate $F^* : \Omega^* \to \mathbb{R}$, where $\Omega^* = \{\nabla F(x) : x \in \Omega\}$ is the conjugate domain. Under this transformation, F^* is also a function of Legendre type [46] and we can define the Bregman divergence associated to F^*. We now use this result to prove the connectedness of Bregman balls.

▶ **Lemma 2** (Connectedness). *Primal and dual Bregman balls are connected.*

Proof. The dual balls are trivially convex [13], hence connected.

The primal balls are more interesting, so we show an explicit proof. First, recall that F is strictly convex and differentiable, implying it is continuously differentiable [46]. Therefore, the Legendre transform of F induces a homeomorphism $h : \Omega \to \Omega^*$. In particular, h maps dual balls in Ω^* to primal balls in Ω. Since connectedness is a topological property, any primal ball in Ω is connected as the homeomorphic preimage of a connected dual ball in Ω^*. ◀

This particular proof is useful for clarifying the importance of the three assumptions in the definition of the Legendre-type function. (I) and (II) gives *continuous* differentiability, and consequently the crucial homeomorphism. As for (III), let us show how things can go wrong without it. Specifically, if we allowed arbitrary convex restrictions of the domain. Consider Ω' as a restriction of Ω to the preimage under h of a non-convex primal ball in Ω^*. Since Ω' is convex, everything appears to work. However, the restricted h now maps Ω' to a non-convex conjugate domain, where the Legendre transform is not well defined. The above proof would therefore fail if we restricted the domain in this way and we could not rule out the existence of non-connected balls. Requiring condition (III) prevents us from making this mistake. Rockefellar [46] mentions that this is a very common mistake in general – it is also present in the Bregman divergence literature.

We now use connectedness for the following lemma.

▶ **Lemma 3** (Boundary Intersection). *Let $\Omega \subset \mathbb{R}^d$ be the domain for a Bregman divergence, D_F, $A \subset \mathbb{R}^d$ be an axis-aligned box of positive finite volume with boundary ∂A and $q \in \Omega \setminus A$ be the center of a Bregman ball B_F of finite radius r. If $B \cap \partial A = \emptyset$, then $A \cap \Omega$ lies in $\Omega \setminus B$.*

Proof. As A has finite volume, it is a codimension-1 topological sphere, and thus ∂A divides $\mathbb{R}^d$ into the inside and the outside, by the Jordan–Brouwer separation theorem. Because $B \cap \partial A = \emptyset$, $q \notin A$, and B is connected, we have that B is necessarily on the outside of ∂A and so $B \cap A = \emptyset$. Therefore, $A \cap \Omega$ indeed lies in the complement of B in Ω. ◀

Thus $B \cap \partial A = \emptyset$ implies that the divergence from q to each potential *data* point in A exceeds the radius of B, namely r. This means that the intersection test with the boundary of A is sufficient to safely prune points in the Kd-tree query. We omit the analogous case for dual balls. We mention that the finite volume assumption is just a technicality as in practice Kd-trees partition a box of finite volume.

6 Proof of pruning efficiency

In this section we show that the pruning test can be performed efficiently in the case of decomposable Bregman divergences. Specifically, we aim to update the projection in $O(1)$ running time, independently of the dimension of the ambient space. To this aim, we rephrase the pruning test in terms of a Bregman projection onto the *boundary* of the box. We call this the **boundary projection test**.

▶ **Lemma 4.** *Let D'_F be a decomposable Bregman divergence defined on $\Omega' \times \Omega'$. Then D'_F is a restriction of a divergence defined on an axis-aligned box.*

Proof. Let D'_F be a decomposable Bregman divergence. Then $F = \sum_{i=1}^d f_i$, where each f_i is a univariate function of Legendre type. Thus, each f_i has a convex domain in $\mathbb{R}$, ω_i, which must be an interval. Thus, D_F is a Bregman divergence defined on an axis-aligned box. Then, as both are generated by the same function of Legendre type, $D_F|_{\Omega' \times \Omega'} = D'_F$ and thus D'_F is a restriction of D_F. ◀

This lemma further emphasizes the importance of Legendre-type function assumption (III), as a restriction on a Bregman divergence's domain is induced by a domain restriction on the parametrization function F. For example, the GKL divergence on $\mathbb{R}^d_+$ may be restricted to the KL divergence on $\triangle^{d-1}$ but not to $(0,1)^d$.

For a given query and set of data points the nearest neighbours are identical under both D_F and a restricted D'_F, allowing use of either divergence. The assumption on the domain $\Omega = \prod_{i=1}^d \omega_i$ ensures that Lemma 5 and Corollary 1 apply to Kd-trees. This is important because Kd-trees decompose $\mathbb{R}^d$ and not the chosen domain of a Bregman divergence. Additionally, in the unrestricted domain, Lemma 6 enables efficient query processing and ensures that our underlying algorithm remains robust under any domain restriction.

From boxes to hyperplanes. We first consider a simplified problem, namely a Bregman projection onto a single axis-aligned hyperplane.

▶ **Lemma 5** (Axis-Aligned Projection)**.** *Let $F = \sum_{i=1}^d f_i$ be a decomposable function of Legendre type defined on an axis-aligned box, Ω.*

Let $P \subset \mathbb{R}^d$ be an axis-aligned hyperplane such that $P \cap \Omega \neq \emptyset$. Let q_P be the Bregman projection of a point $q \in \Omega$ onto P with respect to D_F, $\arg\min_{x \in P} D_F(x\|q)$. Then q_P coincides with the orthogonal projection of q onto P. (The same is true for $\arg\min_{x \in P} D_F(q\|x)$.)

Proof. Let P be an axis-aligned hyperplane orthogonal to the j-th standard basis vector. Specifically, each point $p \in P$ has its j-th coordinate fixed.

Because: (1) q is fixed; (2) p_j is fixed for each $p \in P$; and (3) D_F is decomposable, we can write $D_F(q\|p) = \sum_{i=1}^{d} D_{f_i}(q_i\|p_i) = D_{f_j}(q_j\|p_j) + \sum_{i\neq j} D_{f_i}(p_i\|q_i)$. To minimize $D_F(q\|p)$, we minimize $\sum_{i\neq j} D_{f_i}(q_i\|p_i)$, and since each D_{f_i} is a Bregman divergence, Bregman Nonnegativity (Property 1) applies. So each $D_{f_i}(q_i\|p_i) \geq 0$, with equality if and only if $p_i = q_i$. Consequently, $\arg\inf_{p\in P\cap\Omega} D_F(q\|p) = (q_1, q_2, \ldots, p_j, \ldots, q_d)$. Therefore, the Bregman projection of q onto P is precisely the orthogonal projection. ◀

Generally our Bregman projection *from* q, $\arg\min_{x\in P} D_F(q\|x)$, would *not* be considered a Bregman projection, and Lemma 1 would not apply to it. In our case, the two projections coincide, so we will refer to the resulting point as the Bregman projection onto the axis-aligned hyperplane. With this we get the following corollary.

▶ **Corollary 1** (Box Projection Divergence). *Let $F = \sum_{i=1}^{d} f_i$ be a decomposable function of Legendre type defined on an axis-aligned box. The Bregman projection divergence of $q \in \Omega$ onto A, with respect to the Bregman divergence generated by F, can be computed as*

$$\sum_{i=1}^{d} D_{f_i}(q_i\|p_i) = D_F(q\|p),$$

where p is the (squared) Euclidean projection of q onto A.

Back to the pruning test. To decide if the input points in the current box can be pruned, we compare two values. One is the divergence to the current best candidate; the second one is the projection divergence of q onto an axis-aligned box A. This also works for divergence computed in the reverse direction.

In the end, the situation is very simple. This simplicity allows us to compute the projection divergence in time $O(d)$ – exactly as in the (squared) Euclidean case.

Efficient projection. We focus on *maintaining* the projection divergence during the course of the query, rather than computing it every time. It turns out a single update can be done in constant time, independent of the dimension.

▶ **Lemma 6** (Updating Projection Divergence in Constant Time). *Let $F = \sum_{i=1}^{d} f_i$ be a decomposable function of Legendre type, where each f_i has domain $\omega_i \subseteq \mathbb{R}$. Then the projection divergence can be updated in constant time.*

Proof. Let $q \in \Omega$ and $B = \prod_{i=1}^{d}[a_i, b_i]$ be a box corresponding to a splitting node of our Kd-tree. By corollary 1, the Bregman projection of q onto B is on the boundary of B. Denote this Bregman projection $x = \arg\inf_{p\in B} D_F(q\|p)$. As x lies on the boundary, x_i is either a_i, q_i, or b_i.

For the box C corresponding to a child node, we change only one wall of B by the construction of the Kd-tree. Without loss of generality, $C = [c, b_1] \times \prod_{i=2}^{d}[a_i, b_i]$, with $a_1 < c < b_1$. For $y = \arg\inf_{p\in C} D_F(q\|p)$ we similarly have $y_i = x_i$ for $i = 2, \ldots, d$.

Since D_F is decomposable, $D_{f_i}(q_i\|x_i) = D_{f_i}(q_i\|y_i)$ for $i = 2, \ldots, d$. Thus $D_F(q\|\rho) = D_F(q\|\omega) - D_{f_1}(q_1\|\omega_1) + D_{f_1}(q_1\|\rho_1)$. This is illustrated in Figure 3.

Thus, as we move from a splitting node to its child, updating the projection divergence is independent of the dimension, d. ◀

Moving from a Kd-tree node to its child, the corresponding box shrinks along a single dimension. The projection divergence can therefore be updated using at most two divergence computations (one negative, one positive) along the same dimension. The update is $O(1)$ and independent of the embedding dimension. See Figure 3.

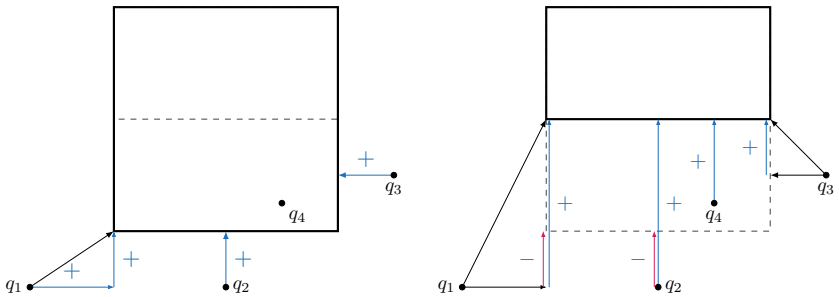

Figure 3 Left: Calculation of projection divergences of each point q_i onto the box decomposed as the sum of divergence computations along individual dimension. **Right:** Efficient update of projection divergence.

As Corollary 1 follows from Lemma 5 and Lemma 6 solely depends on the decomposable structure, the results apply for Bregman divergences computed in either direction.

7 Implementation

Our implementation is based on the ANN (Approximate Nearest Neighbour) C++ library by Mount and Arya [36]. It is an optimized library for Kd-trees.

Bregman query implementation. Algorithm 1 shows a C++ implementation of the Bregman query algorithm using this optimization (modulo unimportant technicalities). The code is structured after the implementation in the ANN library. We show only the part of the code for splitting nodes; handling leaf nodes is straightforward.

We assume that splitting nodes are instances of class `kd_tree_splitting_node`. Leaf nodes store input points and queries are handled with a linear search algorithm. Variable `eps` is used for approximate queries. Finally, `D_f` is assumed to compute the decomposable Bregman divergence *along a selected coordinate* – for all practical decomposable divergences this takes time $O(1)$ by utilizing Lemma 6 at lines 18 and 20. Line 18 adds the new projection divergence, `new_proj_div`, and the old projection divergence is removed in line 20. We remark that many implementations, including KDTree from the popular sklearn library [44], use a slower $O(d)$ approach, adding unnecessary work in higher dimensions.

The variable `knn_priority_queue` is used to maintain the k-nearest neighbours. To perform the dual query, just swap the parameters in the function used to compute `D_f`. The extra argument in `D_f` is just a technicality which allows one to use different 1-dimensional divergence depending on the currently considered dimension.

Expected computational complexity of a query. While we perform a single visit of an internal node in an optimal (constant) time, the expected complexity of the query remains an open problem. In particular, proving the $O(\log n)$ bound for uniformly distributed data is significantly harder in our setting. First, it is not clear what it means to uniformly distribute points with respect to a given divergence. Second, standard proofs rely on volume arguments [11] – but in our case the Euclidean volume of a Bregman ball depends on its location.

Algorithm 1 Bregman Kd-tree query implementation.

```
struct kd_tree_splitting_node : kd_tree_node {
  kd_tree_node *child_lower, *child_higher;
  int cut_dim;
  float cut_val, upper_bound, lower_bound;
  virtual void search(...);
};

using div_t = std::function<float(const float, const float, const int)>;

float D_GKL(const float x_i, const float y_i, const int dim) { // example
  return x_i*log(x_i) - x_i*log(y_i) - x_i + y_i;
}

void virtual kd_tree_splitting_node::search(const point& q,
                float box_proj_div, div_t D_f, float eps=0.0) {
  if (q[cut_dim] < cut_val) { // q lower than the cutting plane
    child_lower->search(q, box_proj_div, D_f, eps); // more promising child
    float new_box_proj_div = box_proj_div+D_f(q[cut_dim], cut_val, cut_dim);
    if (lower_bound > q[cut_dim])
      new_box_proj_div -= D_f(q[cut_dim], lower_bound, cut_dim);
    if (box_div*(1+eps) < knn_priority_queue->max_divergence())
      child_higher->search(q, new_box_proj_div, D_f, eps); // recursive call
  }
  else {/* analogous for q higher than the cutting plane... */ }
}
```

Although the constant time computations only rely on splitting planes being axis aligned, the impact of how splitting planes are chosen in the Bregman case similarly relies on the properties of Bregman balls. While the depth of the trees is not influenced by the choice of metric or divergence, the interactions of Bregman balls and axis-aligned boxes can be highly nonuniform.

These issues necessitate new, significantly more sophisticated, proof techniques. We leave this as an open problem, and show experimentally that the method performs well in practical situations.

8 Experiments

The main practical motivation of our work is to apply computational geometric algorithms to the point clouds produced by machine learning models. In particular, we wish to compare two collections of probabilistic (soft) predictions using the KL divergence. Efficient nearest neighbour queries are useful in this setting, however using the Euclidean tools in this setting can lead to severe discrepancies, seen in Section 8.1. Finally, the dimension is often not overly high (often 10-100) which gives hope that Kd-trees can be efficient.

We will benchmark Bregman Kd-trees for exact and approximate nearest neighbour queries and compare with other methods. Additional results are reported in Section A.

We stress that efficiency is just one aspect determining the practicality of a method. Bregman Kd-trees have several unique advantages which make them practical. For example, once a Bregman Kd-tree is constructed, each query can be performed using a different decomposable Bregman divergence.

Data sets. We use synthetic data, standard datasets, and probabilistic predictions coming from a machine learning model. For all data, we use 50,000 points and 10,000 query points.

In the machine learning setup, we consider popular image datasets CIFAR10 and CIFAR100. Each contains 50,000 training and 10,000 test images, with 10 and 100 different labels respectively. We train two neural networks, M_1 and M_2, on a classification task on CIFAR100. They achieve 80.22%, and 71.74% test accuracy respectively. From each model, we produce two sets of probabilistic predictions: $(\text{trn}_i, \text{tst}_i)$, for $i \in \{1, 2\}$. By $Q \to D$ we mean we query dataset D with queries Q. Since the network is trained to minimize the total KL divergence, these predictions lie on the $\triangle^{99} \subset \mathbb{R}^{100}$ equipped with KL divergence.

We also consider a model trained to 95.2% test accuracy on CIFAR10 and extract its probabilistic predictions on training and test points. These predictions are contained in $\mathbb{R}^{10}$. We also use the standard Corel Image Features data contained in $\triangle^{99}$.

Compiler and hardware. Software was compiled with Clang 14.0.3. The experiments were done on a single core of a 3.5 GHz ARM64-based CPU with 4MB L2 cache using 32GB RAM. We observed similar speed ups on an x86-64 CPU.

8.1 Nearest neighbour comparisons

One may assume that a Euclidean ball and Bregman ball with the same center and radius have large intersections and thus a query would return the same nearest neighbours in both cases. We experimentally show that this is not the case, which necessitates the usage of data structures specialized for the Bregman case.

We first compare the sets of nearest neighbours obtained by using Euclidean distance and the KL divergence. For $\text{tst}_1 \to \text{trn}_1$, we find the 10 nearest for each query in tst_1 with respect to the KL divergence and Euclidean distance. For each query, we compare the two sets of neighbours while disregarding order. Of the 10,000 queries, 9,962 had different sets of nearest neighbours with 134 having no common nearest neighbours. This is expected as the geometry of KL balls can vary depending on the location of the center whereas the Euclidean balls grow more uniformly. A lower dimensional example with three nearest neighbours computed can be seen in Figure 4. When the sets are ordered, the average of number of neighbours with matching indices is 0.8422 of 10, with only two queries having the same nearest neighbours in the same order. In conclusion, the reported nearest neighbours rarely coincide in the same order and often have different neighbours completely. Thus we cannot simply use fast Euclidean algorithms to analyze machine learning models trained using Bregman divergences.

8.2 Baselines

We stress that there are no robust, general libraries for the exact and approximate Bregman nearest neighbour computations. We compare our package to Cayton's experimental implementation of Bregman ball-trees [19] (**BBT**) for exact and approximate nearest neighbours. Two other available (experimental) implementations [40] are not usable, due to severe compilation issues (the code is non-portable) and limited documentation.

We additionally compare our package to the fastest implementation in NMSLIB [14] for Bregman divergences. We stress that these methods are recall-approximate: they may return the correct nearest neighbour, but offer no guarantees (unlike our method). These methods are therefore not in direct competition with our method, but it is interesting to observe the trade-offs between efficiency and guarantees.

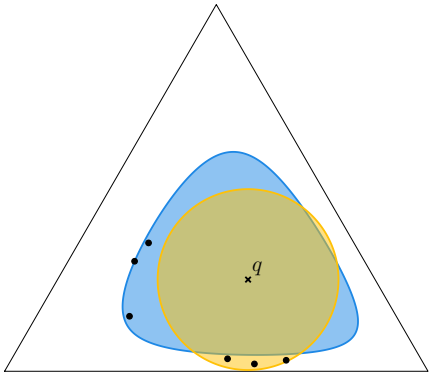

Figure 4 Three nearest neighbours of a query q with respect to the KL divergence and the Euclidean distance on $\triangle^2$. The blue area is the KL ball and the yellow is the Euclidean ball whose radius is determined by the respective third nearest neighbour.

Table 2 Build time comparison between Kd-trees and Bregman ball-trees for different divergences and data sets.

		Bregman Ball Trees [18]		
	Kd-tree	SE	KL	IS
trn_1	0.43s	5.75s	10.05s	7.71s
Corel 64	0.08s	1.30s	6.27s	6.08s
CIFAR10	0.04s	0.20s	1.30s	0.88s

8.3 Exact queries

In Table 2, we measure the construction time of Kd-trees and ball trees. Ball trees are constructed with Cayton's Bregman Ball trees. We note that the same Kd-tree works for any decomposable divergence for either direction, while a ball tree construction depends on the given divergence and direction.

Table 3 shows the speed up in finding nearest neighbours using our method compared to linear search and BBT. We use the KL, IS, BL, and an interpolated divergence. For exact queries with the KL divergence on the 100-dimensional CIFAR data sets we observe $\approx 100\times$ speed compared to the linear search. We compare our speeds to Cayton's Bregman ball trees, achieving minimum $3\times$ speed up with KL divergence and up to $20\times$ speed up for the IS divergence. As BBT has not implemented BL or interpolated divergences, these times are not available.

8.4 Approximate Bregman queries

Given ϵ, an **approximate nearest neighbour query** must return each nearest neighbour x' such that $D_F(q\|x') \leq (1+\epsilon)D_F(q\|x)$, where x is the true nearest neighbour.

To evaluate our method for approximate queries, we compare it with an implementation of Bregman Ball trees (**BB-trees**) by Cayton [18,19]. It is specialized for KL, with *experimental support* for IS. Unlike our method, extending it to other divergences is nontrivial.

Table 3 Runtimes of Kd-trees compared to Bregman ball-trees and linear search. Speed ups of Kd-trees compared to linear and BBT are listed. For example Kd-trees are 92.12× faster than a linear search on $\text{tst}_1 \rightarrow \text{trn}_1$. The interpolated (Int) divergence is 0.9KL + 0.1SE.

		$\text{tst}_1 \rightarrow \text{trn}_1$		$\text{tst}_2 \rightarrow \text{trn}_1$		Corel64		CIFAR10	
Kd-tree	KL	3.06s		3.66s		18.26s		0.30s	
	IS	24.95s		26.65s		67.95s		1.13s	
	BL	4.72s		5.78s		27.05s		0.49s	
	Int	5.55s		6.44s		21.94s		0.50s	
Linear	KL	281.90s	92.12×	286.63s	78.31×	177.77s	9.74×	30.53s	101.77×
	IS	277.87s	11.14×	274.58s	10.30×	173.79s	2.05×	30.20s	23.01×
	BL	88.59s	18.77×	87.84s	15.20×	55.46s	2.05×	8.89s	18.14×
	Int	309.21s	55.71×	311.63s	48.39×	196.99s	8.98×	34.20s	68.40×
BBT	KL	9.62s	3.14×	14.33s	3.92×	99.10s	5.43×	0.91s	3.03×
	IS	507.45s	20.34×	614.97s	23.08×	397.97s	5.86×	3.53s	3.53×
	BL	N/A		N/A		N/A		N/A	
	Int	N/A		N/A		N/A		N/A	

We compare the query times for the Kd-tree search to the Bregman ball tree search for a range of ϵ values in Figure 5. Our method is between 3-5 times faster for KL queries, and between 5-15 times faster for IS queries.

Our Kd-tree method works with arbitrary decomposable divergences (computed in either direction): one can either use a predefined divergence, or implement a custom one. This only requires implementing a single function in the user's code that computes the divergence – no changes to the Kd-tree library are required. This allows the method to work out of the box in various contexts.

The above is in contrast with Bregman Ball trees: they are more general but require tailoring to different divergences [19]. Also, there is a big difference between the simple squared Euclidean case and the Bregman case. Finding a projection onto a (dual) Bregman ball generally requires performing a 1-dimensional convex optimization. In practice, this is done using a binary search, with a full divergence computation at each step, making each step $\Omega(d)$. In our case this entire projection is $O(1)$. As evidenced by BB-tree's relatively lower performance for the IS divergence, extending BB-trees to other divergences poses an algorithmic challenge.

8.5 An unfair comparison with fast heuristics

The Non-Metric Space Library [14] by Naidan has algorithms adapted for working with Bregman divergences and other non-metric distances. Benchmarks for methods have been published specifically for the KL and IS divergences [45]. In particular, the small-world graph (SWG) [30] search is considered a state-of-the-art method. For brevity (and because these heuristic methods do not directly compete with methods for exact and approximate queries), we limit the comparison to this method.

Unlike Kd-trees, the SWG method does not offer guarantees on the number of correct nearest neighbours. While they tend to behave well in practice, Indyk and Xu showed [24] that such methods can fail catastrophically. Generally, recall tends to drop in higher dimensions. In particular, in dimension 100, 15.71% results for the KL divergence contained some incorrect nearest neighbours; in 4.54% of cases, *all* of the reported neighbours were incorrect.

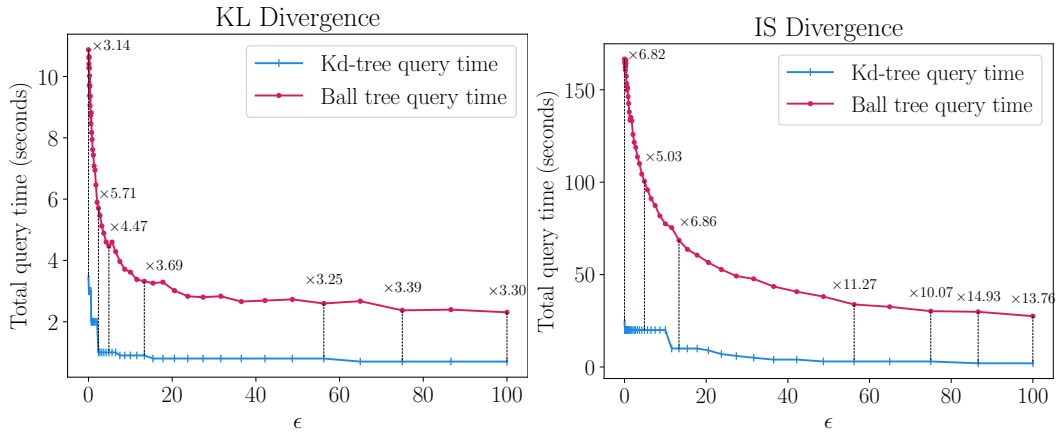

Figure 5 Total query time compared for $(1+\epsilon)$-approximate nearest neighbours for $\text{tst}_1 \rightarrow \text{trn}_2$ (lower is better). Left is KL and right is IS divergence. Starting from $\epsilon = 0.1$. Vertical bars mark the speed up of Kd-trees over ball-trees for a given ϵ.

Table 4 Comparing Kd-tree and SWG method for 10 nearest neighbours. For SWG, error frequency is number of times < 10 correct nearest neighbours are returned. Min recall is the minimum number of correct nearest neighbours. The interpolated (Int) divergence is 0.9KL+0.1Euc.

		Kd-tree Build time	Kd-tree query time	SWG build time	SWG query time	Avg recall	Error freq	Min recall	Min recall freq
KL	$\text{tst}_2 \rightarrow \text{trn}_1$	0.43s	5.76s	4.84s	1.32s	0.928	1571	0	454
	Corel64	0.08s	29.60s	6.99s	1.70s	0.998	152	7	5
	CIFAR10	0.04s	0.45s	2.90s	0.59s	0.998	76	1	1
IS	$\text{tst}_2 \rightarrow \text{trn}_1$	0.43s	29.29s	8.84s	2.31s	0.883	3292	0	387
	Corel64	0.08s	76.03s	17.13s	4.05s	0.900	4608	0	1
	CIFAR10	0.04s	2.05s	2.90s	0.85s	0.991	287	0	17
BL	All data sets			N/A	N/A	N/A	N/A	N/A	N/A
Int	All data sets			N/A	N/A	N/A	N/A	N/A	N/A

In any case, the benchmarks reported in Table 4 reveal an interesting trade-off. Compared to our implementation, the SWG offers faster query time (typically one order of magnitude faster), at the cost of slower build time (typically two orders of magnitude slower). We reiterate that these methods do not provide performance guarantees – while our method does. Overall, SWG is useful for performing numerous imprecise searches, while Kd-trees are useful for fewer searches or when guarantees are required.

9 Summary

We proved several results on Bregman divergences, demonstrating that the geometries they induce are well-behaved. In particular, we show that the lack of symmetry and triangle inequality does not preclude them from being used as a measurement for Kd-trees. This is perhaps unexpected, since the triangle inequality is typically used to prove the correctness of

Kd-trees. Furthermore, we show that certain additional properties of decomposable Bregman divergences enable an efficient query algorithm. These theoretical results provide the basis for an efficient implementation, whose properties are outlined below.

- Computational complexity: a crucial operation is optimized to work in $O(1)$ time. In comparison, several popular Euclidean Kd-trees implementations use a naive $O(d)$ algorithm.
- Speed: it is up to 100× faster than linear search and between 3 and 20× faster than competing methods on practical data in dimension 100.
- Simplicity: the algorithm is simple which makes it more likely to be adopted in practice.
- Ease of use: works for any decomposable Bregman divergence out of the box (competing approaches requires custom, nontrivial implementation for each divergence).
- Flexibility: handles exact and guaranteed ϵ-approximate Bregman queries with divergence computed in either direction.

From an applied perspective, one can now perform efficient queries for practical data measured with the KL divergence, in particular on medium-dimensional data coming from machine learning. This opens up new ways of using computational geometry algorithms within machine learning.

On the theoretical side, this work opens up new questions. Of primary importance is the expected computational complexity of a Kd-tree query. This problem is significantly more involved than in the Euclidean case, and will require developing novel proof techniques and deepening our understanding of the geometries induced by Bregman divergences.

References

1 Ahmed Abdelkader, Sunil Arya, Guilherme D da Fonseca, and David M Mount. Approximate nearest neighbor searching with non-euclidean and weighted distances. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 355–372. SIAM, 2019.

2 Amirali Abdullah, John Moeller, and Suresh Venkatasubramanian. Approximate bregman near neighbors in sublinear time: beyond the triangle inequality. In *Proceedings of the Twenty-Eighth Annual Symposium on Computational Geometry*, SoCG '12, pages 31–40, New York, NY, USA, 2012. Association for Computing Machinery. doi:10.1145/2261250.2261255.

3 Marcel R. Ackermann and Johannes Blömer. Coresets and approximate clustering for bregman divergences. In *Proceedings of the 2009 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1088–1097, 2009. doi:10.1137/1.9781611973068.118.

4 Martin Adamčík. The information geometry of bregman divergences and some applications in multi-expert reasoning. *Entropy*, 16(12):6338–6381, 2014. doi:10.3390/e16126338.

5 Sunil Arya and David M. Mount. Algorithms for fast vector quantization. In *[Proceedings] DCC '93: Data Compression Conference*, pages 381–390, 1993. doi:10.1109/DCC.1993.253111.

6 Sunil Arya, David M. Mount, and Onuttom Narayan. Accounting for Boundary Effects in Nearest Neighbor Searching. In *Proceedings of the Eleventh Annual Symposium on Computational Geometry*, SCG '95, pages 336–344, New York, NY, USA, 1995. Association for Computing Machinery. doi:10.1145/220279.220315.

7 Sunil Arya, David M. Mount, Nathan S. Netanyahu, Ruth Silverman, and Angela Y. Wu. An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *J. ACM*, 45(6):891–923, November 1998. doi:10.1145/293347.293348.

8 M. Aumüller, E. Bernhardsson, and A. Faithfull. ANN-Benchmarks: A benchmarking tool for approximate nearest neighbor algorithms. *Information Systems*, 87:101–374, 2020.

9 Arindam Banerjee, Srujana Merugu, Inderjit S. Dhillon, and Joydeep Ghosh. Clustering with Bregman divergences. *Journal of Machine Learning Research*, 6(58):1705–1749, 2005. URL: https://jmlr.org/papers/v6/banerjee05b.html.

10 Heinz H Bauschke, Jonathan M Borwein, et al. Legendre functions and the method of random bregman projections. *Journal of convex analysis*, 4(1):27–67, 1997.

11 Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18:509–517, 1975. doi:10.1145/361002.361007.

12 Jon Louis Bentley. K-d trees for semidynamic point sets. In *Proceedings of the Sixth Annual Symposium on Computational Geometry*, SCG '90, pages 187–197, New York, NY, USA, 1990. Association for Computing Machinery. doi:10.1145/98524.98564.

13 Jean-Daniel Boissonnat, Frank Nielsen, and Richard Nock. Bregman Voronoi diagrams. *Discrete and Computational Geometry*, 44:281–307, 2010. doi:10.1007/S00454-010-9256-1.

14 Leonid Boytsov and Bilegsaikhan Naidan. Engineering efficient and effective non-metric space library. In Nieves R. Brisaboa, Oscar Pedreira, and Pavel Zezula, editors, *Similarity Search and Applications - 6th International Conference*, volume 8199 of *Lecture Notes in Computer Science*, pages 280–293. Springer, 2013. doi:10.1007/978-3-642-41062-8_28.

15 Leonid Boytsov and Bilegsaikhan Naidan. Learning to prune in metric and non-metric spaces. In C.J. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 26. Curran Associates, Inc., 2013.

16 Lev M. 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.

17 L. Cayton. *Bregman Proximity Search.* Doctoral dissertation, UC San Diego, 2009.

18 Lawrence Cayton. Bbtrees. URL: https://github.com/lcayton/bbtree.

19 Lawrence Cayton. Fast nearest neighbor retrieval for bregman divergences. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, pages 112–119, New York, NY, USA, 2008. Association for Computing Machinery. doi:10.1145/1390156.1390171.

20 Minh N Do and Martin Vetterli. Wavelet-based texture retrieval using generalized Gaussian density and Kullback-Leibler distance. *IEEE Transactions on Image Processing*, 11(2):146–158, 2002. doi:10.1109/83.982822.

21 H. Edelsbrunner and H. Wagner. Topological data analysis with bregman divergences. In *"Proc. 33rd Ann. Symp. Comput. Geom., 2017", 1–16*, pages 1–16, 2017.

22 Jerome H. Friedman, Jon Louis Bentley, and Raphael Ari Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Trans. Math. Softw.*, 3(3):209–226, September 1977. doi:10.1145/355744.355745.

23 Yikun Han, Chunjiang Liu, and Pengfei Wang. A comprehensive survey on vector database: Storage and retrieval technique, challenge. *arXiv preprint arXiv:2310.11703*, 2023. doi:10.48550/arXiv.2310.11703.

24 Piotr Indyk and Haike Xu. Worst-case performance of popular approximate nearest neighbor search implementations: guarantees and limitations. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, Red Hook, NY, USA, 2024. Curran Associates Inc.

25 Fumitada Itakura. Analysis synthesis telephony based on the maximum likelihood method. *Reports of the 6^{th} Int. Cong. Acoust.*, 1968.

26 Diederik P Kingma and Max Welling. Auto-encoding variational bayes, 2022. arXiv:1312.6114.

27 A. C. Koivunen and A. B. Kostinski. The feasibility of data whitening to improve performance of weather radar. *Journal of Applied Meteorology*, 38(6):741–749, 1999. doi:10.1175/1520-0450(1999)038<0741:TFODWT>2.0.CO;2.

28 Solomon Kullback and Richard A. Leibler. On information and sufficiency. *The Annals of Mathematical Statistics*, 22(1):79–86, 1951.

29 Prasanta Chandra Mahalanobis. On the generalised distance in statistics. *Proceedings of the National Institute of Sciences of India*, 2(1):49–55, 1936. Retrieved 2016-09-27.

30 Yu A. Malkov and D. A. Yashunin. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE Trans. Pattern Anal. Mach. Intell.*, 42(4):824–836, April 2020. doi:10.1109/TPAMI.2018.2889473.

31 Yu A Malkov and Dmitry A Yashunin. Hnswlib - fast approximate nearest neighbor search. https://github.com/nmslib/hnswlib, 2023.

32 Yury Malkov, Alexander Ponomarenko, Andrey Logvinov, and Vladimir Krylov. Approximate nearest neighbor algorithm based on navigable small world graphs. *Information Systems*, 45:61–68, 2014. doi:10.1016/J.IS.2013.10.006.

33 MaridDB. Mariadb vector. URL: https://mariadb.org/projects/mariadb-vector/.

34 Leland McInnes, John Healy, Nathaniel Saul, and Lukas Großberger. UMAP: Uniform manifold approximation and projection. *Journal of Open Source Software*, 3(29):861, 2018. doi:10.21105/JOSS.00861.

35 MongoDB. Atlas Vector Search. URL: https://www.mongodb.com/products/platform/atlas-vector-search.

36 David Mount. *The ANN Programming Manual*.

37 Kevin P. Murphy. *Machine learning: A Probabilistic Perspective*. MIT press, 2012.

38 Frank Nielsen and Richard Nock. The dual voronoi diagrams with respect to representational bregman divergences. In *2009 Sixth International Symposium on Voronoi Diagrams*, pages 71–78, 2009. doi:10.1109/ISVD.2009.15.

39 Frank Nielsen and Richard Nock. The Bregman chord divergence. In *4th International Conference on Geometric Science of Information (GSI)*, pages 299–308. Springer, 2019. doi:10.1007/978-3-030-26980-7_31.

40 Frank Nielsen, Paolo Piro, and Michel Barlaud. Vptrees and bbtreepp. URL: https://github.com/FrankNielsen/FrankNielsen.github.io/tree/master/BregmanProximity.

41 Frank Nielsen, Paolo Piro, and Michel Barlaud. Bregman vantage point trees for efficient nearest neighbor queries. In *Proceedings - 2009 IEEE International Conference on Multimedia and Expo, (ICME)*, pages 878–881, 2009. doi:10.1109/ICME.2009.5202635.

42 Frank Nielsen, Paolo Piro, and Michel Barlaud. Tailored Bregman ball trees for effective nearest neighbors. In *Proceedings of the 25th European Workshop on Computational Geometry (EuroCG)*, pages 29–32, 2009.

43 R. Nock and F. Nielsen. Fitting the Smallest Enclosing Bregman Ball. In *Machine Learning: ECML 2005*, pages 649–656, October 2005.

44 F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011. doi:10.5555/1953048.2078195.

45 Alexander Ponomarenko, Nikita Avrelin, Bilegsaikhan Naidan, and Leonid Boytsov. Comparative analysis of data structures for approximate nearest neighbor search. In *Proceedings of the Third International Conference on Data Analytics*, January 2014.

46 Ralph T. Rockafellar. *Convex Analysis*. Princeton University Press, 1970.

47 Claude Elwood Shannon. A mathematical theory of communication. *The Bell system technical journal*, 27(3):379–423, 1948. doi:10.1002/J.1538-7305.1948.TB01338.X.

48 Yang Song, Yu Gu, Rui Zhang, and Ge Yu. BrePartition: Optimized High-Dimensional kNN Search with Bregman Distances, 2020. arXiv:2006.00227.

49 Robert F. Sproull. Refinements to nearest-neighbor searching in k-dimensional trees. *Algorithmica*, 6:579–589, 1991. doi:10.1007/BF01759061.

50 Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9(11), 2008.

51 Ruben Winastwan. Vector Indexing in Milvus. URL: https://zilliz.com/learn/how-to-pick-a-vector-index-in-milvus-visual-guide.

52 Jun Zhang. Divergence Function, Duality, and Convex Analysis. *Neural Computation*, 16(1):159–195, January 2004. doi:10.1162/08997660460734047.

53 Zhenjie Zhang, Beng Chin Ooi, Srinivasan Parthasarathy, and Anthony K. H. Tung. Similarity Search on Bregman Divergence: Towards Non-Metric Indexing. *Proc. VLDB Endow.*, 2(1):13–24, 2009. doi:10.14778/1687627.1687630.

A Additional tests

A.1 Higher dimensional experiments

Baselines in higher dimensions. For 50,000 data points and 10,000 query points sampled sampled from the simplex, we can compare query times between Kd-trees and Cayton's BBT. In Table 5, we compare query times in higher dimensions. Although Kd-trees are often said to be slower in higher dimensions, we see a speed up in our method even at 500 dimensions.

Table 5 Kd-tree and BBT 10 nearest KL–neighbours search times for increasing dimensions.

Dimension	100	150	200	250	500	1000
Kd-tree query	208.39s	321.54s	469.21s	588.67s	1,229.72	2,689.70s
BBT query	247.80s	402.93s	517.07s	625.38s	1,237.84s	2,410.28s
Speed-up	$\approx 1.19\times$	$\approx 1.25\times$	$\approx 1.10\times$	$\approx 1.06\times$	$\approx 1.00\times$	$\approx 0.90\times$

SW-graph comparison. In comparison to Kd-trees, SWG has slow build times. We compare the SWG build time on 500,000 points in $\triangle^{999} \subset \mathbb{R}^{1000}$, with parameters reduced for speed while maintaining >0.9 average recall for 10 queries. SWG build time was 1464.07s, while Kd-tree build time and query time were 19.44s and 25.76s respectively. The build time for SWG is $>30\times$ longer than the sum of build and query time for Kd-trees.

A.2 Other exact query experiments

For these additional tests, $\square^{100}$ is a uniform sample of the unit cube with the same data sizes as above. In Table 6, we record total query time of other possible pairs of prediction data and $\square^{100}$.

Table 6 Additional total query time comparisons.

		$\text{tst}_1 \to \text{trn}_2$	$\square^{100}$	$\text{trn}_2 \to \text{trn}_1$	$\text{trn}_1 \to \text{trn}_2$
KL	Kd-Tree	4.46	17.22	12.73	264.84s
	Linear Search	285.16	1435.50	1431.93	407.80s
	Speed up	$63.94\times$	$83.36\times$	$112.49\times$	$1.54\times$
IS	Kd-Tree	26.76	121.49	180.55	170.58s
	Linear Search	277.77	1377.00	1365.40	397.75s
	Speed up	$10.38\times$	$11.33\times$	$7.56\times$	$2.33\times$
SE	Kd-Tree	0.41	1.89	1.87	89.95s
	Linear Search	23.32	116.44	116.466	24.36s
	Speed up	$56.88\times$	$61.61\times$	$62.28\times$	$0.27\times$