

BREGMAN VANTAGE POINT TREES FOR EFFICIENT NEAREST NEIGHBOR QUERIES

Frank Nielsen

École Polytechnique / Sony CSL
Palaiseau, France / Tokyo, Japan

Paolo Piro and Michel Barlaud

University of Nice-Sophia Antipolis / CNRS
Sophia Antipolis, France

ABSTRACT

Nearest Neighbor (NN) retrieval is a crucial tool of many computer vision tasks. Since the brute-force naive search is too time consuming for most applications, several tailored data structures have been proposed to improve the efficiency of NN search. Among these, vantage point tree (*vp-tree*) was introduced for information retrieval in metric spaces. Vp-trees have recently shown very good performances for image patch retrieval with respect to the L_2 metric. In this paper we generalize the seminal vp-tree construction and search algorithms to the broader class of Bregman divergences. These distortion measures are preferred in many cases, as they also handle entropic distances (e.g., Kullback-Leibler divergence) besides quadratic distances. We also extend vp-tree to deal with symmetrized Bregman divergences, which are commonplace in applications of content-based multimedia retrieval. We evaluated performances of our *Bvp-tree* for exact and approximate NN search on two image feature datasets. Our results show good performances of Bvp-tree, specially for symmetrized Bregman NN queries.

Index Terms— Nearest neighbor queries, vantage-point trees, Bregman divergences.

1. INTRODUCTION

Nearest neighbor (NN) search is a common task in several multimedia and computer vision applications such as audio/image/video retrieval and object/scene recognition. Given a set $\mathcal{S} = \{p_1, \dots, p_n\}$ of n d -dimensional points and a query point q , the nearest neighbor $\text{NN}(q)$ is defined as the point of \mathcal{S} that is closest to q with respect to a dissimilarity measure D : $\text{NN}(q) = \arg \min_i D(q, p_i)$. If no prior preprocessing of data is provided, finding the NN has linear computational cost $O(dn)$, which is prohibitive in applications involving huge amounts of data. Hence, much effort has been devoted to design tailored data structures that speed-up NN search by exploiting space properties (e.g., the *triangle inequality* for metrics). A major category of such techniques includes tree-like space partitions, such as k D-trees, metric ball and vantage point trees. These methods improve over linear brute force search by *pruning* sub-trees whose exploration can be discarded. They are also expected to handle approximate NN queries more efficiently than the *randomized sampling*

method, which performs brute force search on a random sample of the dataset. Namely, vantage point tree (*vp-tree*) has been historically introduced by Yianilos [1] in 1993 as a data structure for partitioning a general metric space in a hierarchical way. A tree structure is built by recursively splitting each node covering a set into two siblings covering corresponding subsets. Node partitioning is based on a randomly chosen *vantage point*. For each point of the node, its distance to the vantage point is compared to a distance threshold. Points with distances smaller than the threshold are classified as “near” and assigned to, say, the left subtree, the remaining others are classified as “far” and assigned to the right subtree. The threshold is usually computed as the median of all distances to the vantage point, thus balancing both sub-trees (Figure 1).

Vp-trees have been rarely but successfully used in applications like image indexing [2] and music information retrieval [3]. In addition, they have recently shown very good performances for image patch retrieval wrt. the L_2 metric [4]. Such promising results motivated us to further investigate the use of vp-tree and tailor it to other distortion measures, which may be more appropriate than the Euclidian metric for multimedia features. A broad class of such measures is represented by Bregman divergences, which are not metrics in general and do not satisfy the triangle inequality. A Bregman divergence D_F on vector set $\mathcal{X} \subset \mathbb{R}^d$ is defined for a strictly *convex* and *differentiable* generator $F(x) : \mathcal{X} \subset \mathbb{R}^d \mapsto \mathbb{R}$ as $D_F(p||q) = F(p) - F(q) - (p - q)^T \nabla F(q)$, where ∇F denotes the gradient of F . This class of divergences (parameterized by a generator F) includes all quadratic distances (e.g., the Mahalanobis squared distances) and the asymmetric Kullback-Leibler (KL) divergence ($F(x) = \sum_{j=1}^d x_j \log x_j$, the negative Shannon entropy). Bregman divergences are essential distortion measures as they are provably the *canonical distances* that generalize the Euclidean flat geometry [5, 6].

In this paper, we propose the Bregman vantage point tree (*Bvp-tree*) as a generalization of the metric vp-tree to Bregman divergences. Our main theoretical contribution is to replace the triangle inequality by an analogous criterion based on the intersection of Bregman balls, which allows one for checking pruning conditions wrt. Bregman divergences. Finally, we show experimental results on two large image feature datasets for both exact and approximate NN queries.

2. BREGMAN VANTAGE-POINT TREES

We first describe the *Bvp-tree* data structure based on the seminal description of Yianilos' *vp-tree* [1] (Section 2.1). Then we describe our algorithm for checking the pruning conditions when visiting the tree (Section 2.2). Since Bregman divergences are, in general, asymmetric distortion measures, we consider both sided and symmetrized NN queries. Without loss of generality, we consider only right-sided NN queries, defined as: $\text{NN}_F^r(q) = \arg \min_i D_F(p_i||q)$, since left-sided NN queries can be handled similarly by using the duality property: $D_F(p||q) = D_{F^*}(\nabla F(q)||\nabla F(p))$ (F^* is the Legendre conjugate of F . See [6] for details.) On the other hand, symmetrized NN queries are defined as: $\text{NN}_F(q) = \arg \min_i (D_F(p_i||q) + D_F(q||p_i))/2$.

2.1. Outline of Bregman vantage point tree (Bvp-tree)

A Bvp-tree is built by applying recursively the same partitioning procedure as described in [1], except for replacing distances by Bregman divergences. First, the tree root is created, which represents the whole dataset \mathcal{S} . Then a vantage point v is randomly drawn from the dataset and the distance $D_F(x||v)$ is computed for each point $x \in \mathcal{S}$. A distance threshold τ defines a partition $\mathcal{S} = \mathcal{S}_l \cup \mathcal{S}_r$, where \mathcal{S}_l is covered by the Bregman ball $B(v, \tau) = \{x \mid D_F(x||v) < \tau\}$ and \mathcal{S}_r is contained in the complement subset $\mathcal{S} \setminus B(v, \tau) = \{x \mid D_F(x||v) \geq \tau\}$. In our implementation, we define τ as the median of the distances $\{D_F(x||v) \mid x\}$. This hierarchical decomposition of \mathcal{S} is applied recursively on \mathcal{S}_l and \mathcal{S}_r until a stopping criterion is eventually met. This criterion is typically based either on setting: (1) the maximum number of leaf points *bs* (*bucket size*), or (2) the maximum leaf radius r_0 . When such a criterion is met, two leaves are created to store the corresponding source points. (All internal nodes store only Bregman balls $B(v, \tau)$.) Figure 1 shows an example of the space partition induced by a Bvp-tree.

In order to retrieve the NN for a given query q , we perform a branch-and-bound traversal of the tree. The tree is visited by traversing it from the root to the leaves following a given branching order (depth first search). Namely, at any internal node, we choose to branch first on the sub-tree whose corresponding ball is *closer* to the query q . Temporarily ignored siblings are added to a *priority queue* for successive exploration. (The smaller the distance of a sibling node to the query point, the largest its priority.) The first visited leaf yields the very first NN candidate point p' , thus giving an upper bound $D_F(p'||q)$ to the NN distance. Indeed, the true NN cannot lie out the boundary of the Bregman ball $B(q, D_F(p'||q))$ (query ball). Then, in exact search, all formerly ignored subtrees are explored according to their priority order. Each subtree needs to be visited or not depending on whether $D_F(p'||q) > \min_{x \in B(c, R)} D_F(x||q)$, where p' is the current NN candidate. This test is performed by check-

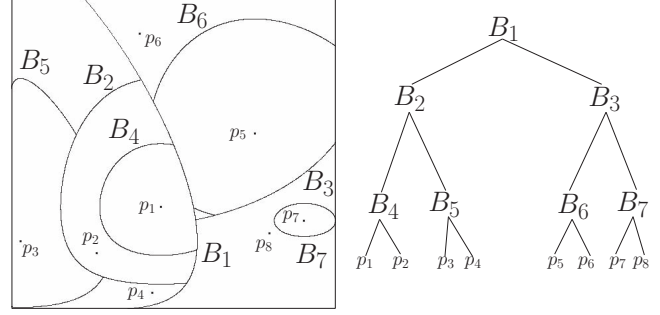


Fig. 1. Schematic description of a Bvp-tree construction on a set of 8 points (wrt. SKL aka. Jensen-Shannon divergence).

ing whether the ball $B(c, R)$ intersects the current query ball $B(p', D_F(p'||q))$ or not. If the two balls do not intersect, then the node can be pruned. Otherwise, it must be explored. Every time a new NN candidate is found, the upper bound is updated, thus reducing the size of the search subspace. Pruning subspaces is a major advantage of such data structures, as the more leaves are pruned out, the more significant is the computational speed-up over brute-force search. In the following section we describe how this test is efficiently implemented.

2.2. Pruning condition

We present a novel algorithm for checking whether the intersection between two Bregman balls is void or not. This test enables us to prune nodes that cannot contain a closer neighbor than the actual NN candidate. Indeed, given a query point q and a NN candidate p' , the Bregman ball $B(q, r')$ of radius $r' = D_F(p'||q)$ centered at q defines the only space region where a closer NN may be found. As a result, any subtree rooted at a node that does not intersect $B(q, r')$ can be pruned without impacting the retrieval precision.

Consider two non-concentric Bregman balls $B_1(p, r_p)$ and $B_2(q, r_q)$. The *radical hyperplane* H_{12} is the locus of points that have equal power with respect to these two balls. It is defined as: $H_{12} : B_1(x) - B_2(x) = 0$, where $B_1(x) : D_F(x||p) - r_p = 0$ and $B_2(x) : D_F(x||q) - r_q = 0$ are the power equations of the two balls. Plugging the definition of $D_F(x||\cdot)$ yields the following equation of the radical hyperplane: $H_{12} : F(q) - F(p) + r_2 - r_1 + \langle x, \nabla F(q) - \nabla F(p) \rangle + \langle p, \nabla F(p) \rangle - \langle q, \nabla F(q) \rangle = 0$, where $\langle x, y \rangle$ denote the vector inner product $x^T y$. If the balls intersect, then the radical hyperplane necessarily contains points of intersection (Figure 2). In order to test this condition, we propose to check the point of intersection between the radical hyperplane and the geodesic Γ_{pq} linking p to q , which is defined as $\Gamma_{pq} = \{\text{LERP}(\lambda, p, q) \mid \lambda \in \mathbb{R}\}$, with $\text{LERP}(\lambda, p, q) = \nabla F^{-1}((1 - \lambda)\nabla F(p) + \lambda\nabla F(q))$. If neither ball contains this point, then their intersection is empty. We implement this test as a bisection geodesic walk algorithm similar to that used in [7] for computing symmetrized

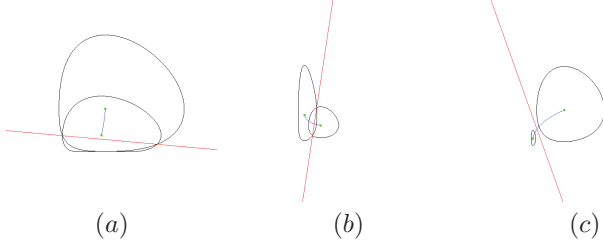


Fig. 2. Examples radical hyperplanes for two (a,b) intersecting, and (c) non-intersecting Bregman balls.

Bregman centroids. Note that we first check whether a ball contains the center of the other, as this would make more iterations useless. Then, we use bounds to stop the bisection algorithm as soon as possible, without need for precisely computing the common intersection point (Algorithm 1).

Algorithm 1 Ball Intersection($B_q(q, R_q), B_v(v, R_v), \lambda_l, \lambda_r$)

Input: Bregman Balls $B_q(q, R_q)$ (query point ball), $B_v(v, R_v)$ (vantage point ball), $\lambda_l, \lambda_r \in (0, 1)$;
 SET $x_{\lambda_l} = \nabla F^{-1}((1 - \lambda_l)\nabla F(q) + \lambda_l\nabla F(v))$;
 SET $x_{\lambda_r} = \nabla F^{-1}((1 - \lambda_r)\nabla F(q) + \lambda_r\nabla F(v))$;
if $D_F(x_{\lambda_r}||q) < R_q$ OR $D_F(x_{\lambda_l}||v) < R_v$ **then**
 return YES;
end if
 SET $\lambda = \frac{\lambda_l + \lambda_r}{2}$;
 SET $x_\lambda = \nabla F^{-1}((1 - \lambda)\nabla F(q) + \lambda\nabla F(v))$;
if $D_F(x_\lambda||q) > R_q$ AND $D_F(x_\lambda||v) > R_v$ **then**
 return NO;
else if $D_F(x_\lambda||q) < R_q$ AND $D_F(x_\lambda||v) < R_v$ **then**
 return YES;
else if $D_F(x_\lambda||q) < R_q$ AND $D_F(x_\lambda||v) > R_v$ **then**
 SET $\lambda_l = \lambda$;
 return Ball Intersection($B_q(q, R_q), B_v(v, R_v), \lambda_l, \lambda_r$);
else if $D_F(x_\lambda||q) > R_q$ AND $D_F(x_\lambda||v) < R_v$ **then**
 SET $\lambda_r = \lambda$;
 return Ball Intersection($B_q(q, R_q), B_v(v, R_v), \lambda_l, \lambda_r$);
end if

3. EXPERIMENTS

We carried out experiments for both KL and SKL (Symmetrized KL aka. Jensen-Shannon divergence) queries performed on Bvp-trees. We evaluated construction and search performances for two datasets of image features, which have been already used by Cayton [8] to evaluate his Bb-tree data structure. Each dataset contains two parts: (1) A reference point database, which we used for building the tree structure at pre-processing time, and (2) a query point set, which we used for performing on-line NN queries on Bvp-trees. The first dataset (SIFT) contains high-dimensional histograms of SIFT descriptors ($d = 1111$), which are among the most widely used descriptors in several applications of computer vision. The SIFT dataset contains 10,000 reference points

and 2,300 query points. The second dataset (Corel) is a collection of color histograms of Corel images, which have been extensively used to benchmark several methods of image retrieval and categorization. It contains 60,000 reference points and 6,616 query points.

In this section, we present results of our Bvp-tree implementation for both asymmetric (KL) and symmetrized (SKL) Bregman queries. First, we analyze the main properties of the tree structure when varying the partitioning termination criterion, as well as the construction cost (Section 3.1). Then we discuss the most significant results of both *exact* and *approximate* NN retrieval (Section 3.2).

3.1. Bvp-tree construction

Low construction cost is a major advantage of Bvp-tree over other data structures like Bb-trees, which usually require running Bregman k -means clustering [9]. Indeed, when using random vantage points, only one divergence has to be computed for each point in a Bregman ball. Hence, the overall construction time is $O(n\delta)$, n being the dataset size and δ the average depth of the tree. (Note that the computational cost of Bb-tree construction is $O(rn\delta)$, with $r \geq 2$ increasing with the number of k -means iterations.) As mentioned in Section 2.1, we used either the *bucket size* (bs) or the *leaf radius* (r_0) criterion for stopping the node branching. The first criterion enables us to build a perfectly balanced vantage point tree, i.e., all leaves have equal depth and store the same number of data points. On the contrary, the second criterion enables to partition even large Bregman balls that contain very sparse data, thus giving smaller number of leaves. Table 1 summarizes the typical characteristics of Bvp-trees for the Corel dataset, as well as the construction cost, for different values of the construction parameters. The construction cost is expressed as the total number of divergences computed when building the tree. The top half-table shows characteristics of trees built with the bucket size criterion. In this case, trees are perfectly balanced, as the average tree depth equals the maximum one. The bottom half-table refers to trees built with the leaf radius criterion, which give unbalanced trees, as proven by the difference between maximum and average tree depth. Also note that trees having roughly equal average depth, but constructed with different criteria, have significantly different overall sizes (i.e., number of leaves).

3.2. Bvp-tree search

We tested our Bvp-tree algorithm for both *exact* and *approximate* NN retrieval. In addition, we evaluated performances of both sided and symmetrized Bregman NN queries on the two datasets. Table 2 displays results of exact search for different settings of the tree construction. Performances are expressed in terms of the computational cost ratio between brute-force and Bvp-tree search. These results show a significant *order of magnitude* speed-up over the brute-force method for the

dataset	bs	depth	depth _{avg}	nLeaves	cost
Corel	50	11	11	2048	$6.6 \cdot 10^9$
Corel	100	10	10	1024	$6.0 \cdot 10^5$
Corel	200	9	9	512	$5.4 \cdot 10^5$
dataset	r ₀	depth	depth _{avg}	nLeaves	cost
Corel	$1.0 \cdot 10^0$	13	10.50	3790	$6.3 \cdot 10^5$
Corel	$1.5 \cdot 10^0$	13	10.09	3586	$6.1 \cdot 10^5$
Corel	$2.0 \cdot 10^0$	13	6.71	718	$4.0 \cdot 10^5$

Table 1. Bvp-tree construction results for different stopping criteria: bucket size (bs) or leaf radius (r_0).

div	dataset	bs=50	bs=100	bs=200	Bb-tree
KL	Corel	2.12	2.33	2.04	2.4
KL	SIFT	0.90	0.95	0.97	0.9
SKL	Corel	3.24	3.13	2.79	-
SKL	SIFT	0.96	1.01	1.05	-

Table 2. Performances of exact Bvp-tree search, measured as computational speed-up over brute-force search. The last column displays results reported by Cayton [8] for Bb-trees.

Corel dataset. On the contrary, the SIFT dataset revealed to be more challenging because of the very high dimensionality of data. Interestingly, SKL queries show better performances than asymmetric KL queries. A comparison with results reported by Cayton [8] for the KL divergence shows that Bvp-tree does not improve over Bb-tree (last column in Table 2). However, experiments of SKL queries gave better results, although a direct comparison with Bb-tree is not possible, as this latter does not handle symmetrized queries.

Besides investigating exact search, we also tested our Bvp-tree for approximate NN retrieval. This is more interesting for practical applications, where the search task is generally relaxed to retrieve a “good” NN, i.e., a point that is close enough to the true NN. This approximation allows for significant speed-ups when searching the tree. In order to find an approximate NN, we performed the iterative Bvp-tree search procedure up to a maximum prescribed number of visited leaves. In each experiment, we fixed a value of this parameter, ranging from *near-exact* search to the exploration of a *single node*, then we evaluated the *Number Closer* (i.e., the number of closer points to the approximated NN) and the *Speed-up* (i.e., the ratio between the number of divergence computations of brute-force and tree search). Figure 3 displays the most significant results of such experiments (log-log plot).

Acknowledgments

We gratefully acknowledge financial support from DIGITEO GAS 2008-16D, ANR GAIA 07-BLAN-0328-01 and ANR ICOS-HD.

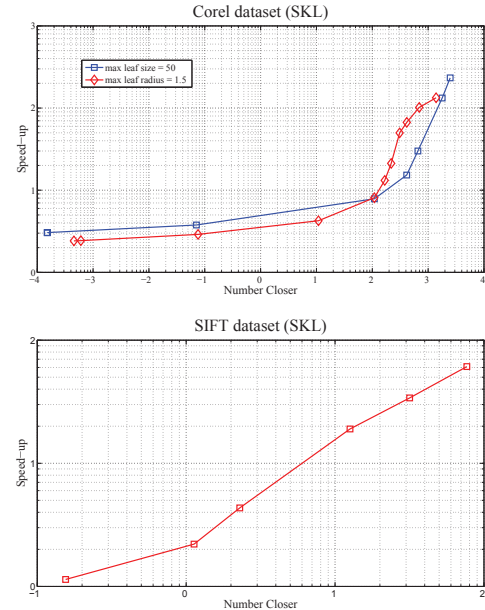


Fig. 3. Results of approximate NN retrieval (log-log plot).

4. REFERENCES

- [1] P.-N. Yianilos, “Data structures and algorithms for nearest neighbor search in general metric spaces,” in *SODA*, 1993, pp. 311–321.
- [2] H. Shao, T. Svoboda, V. Ferrari, T. Tuytelaars, and L.-J. Van Gool, “Fast indexing for image retrieval based on local appearance with re-ranking,” in *ICIP*, 2003, pp. 737–740.
- [3] M. Skalak, J. Han, and B. Pardo, “Speeding melody search with vantage point trees,” in *ISMIR*, 2008.
- [4] N. Kumar, L. Zhang, and S.-K. Nayar, “What is a good nearest neighbors algorithm for finding similar patches in images?,” in *ECCV (2)*, 2008, pp. 364–378.
- [5] S.-I. Amari and N. Nagaoka, *Methods of Information Geometry*, Oxford University Press, 2000.
- [6] F. Nielsen, J.-D. Boissonnat, and R. Nock, “On Bregman voronoi diagrams,” in *SODA*, 2007, pp. 746–755, SIAM.
- [7] F. Nielsen and R. Nock, “Bregman sided and symmetrized centroids,” in *ICPR*, 2008, IEEE CS Press.
- [8] L. Cayton, “Fast nearest neighbor retrieval for Bregman divergences,” in *ICML*, 2008, pp. 112–119.
- [9] A. Banerjee, S. Merugu, I.-S. Dhillon, and J. Ghosh, “Clustering with Bregman divergences,” *Journal of Machine Learning Research*, vol. 6, pp. 1705–1749, 2005.