Probabilistic Cost Model for Nearest Neighbor Search in Image Retrieval

Kunho Kim[a], Mohammad K. Hasan[a], Jae-Pil Heo[a], Yu-Wing Tai[a], Sung-eui Yoon[a,b]

[a] *Dept. of Computer Science, KAIST*
[b] *Div. of Web Science and Technology, KAIST*

Abstract

We present a probabilistic cost model to analyze the performance of the kd-tree for nearest neighbor search in the context of content-based image retrieval. Our cost model measures the expected number of kd-tree nodes traversed during the search query. We show that our cost model has high correlations with both the observed number of traversed nodes and the runtime performance of search queries used in image retrieval. Furthermore, we prove that, if the query points follow the distribution of data used to construct the kd-trees, the median-based partitioning method as well as PCA-based partitioning technique can produce near-optimal kd-trees in terms of minimizing our cost model. The probabilistic cost model is validated through experiments in SIFT-based image retrieval.

Keywords: Image retrieval, nearest neighbor search, kd-tree

1. Introduction

The nearest neighbor search [1] is one of the most widely used proximity queries with various applications such as image retrieval [2], pattern recognition [3], etc. In this paper we focus on exact or approximate nearest neighbor search queries used for content-based image retrieval (CBIR). In CBIR, various image features such as SIFT [4] are used for finding similar images in image databases. In order to find the similar images reliably, multiple features (e.g., around 1 K features) are typically extracted from each query image. The problem of CBIR then becomes finding the nearest neighbor image such that its image features match closely with those from the query image.

Since the number of images in an image database can be very large, a kd-tree [5] or kd-tree forest [6] has been widely used to accelerate the performance of nearest neighbor search. In CBIR, however, the dimension of image features is very high. For example, the dimension of a SIFT feature vector for a single feature point in an image is 128. Thus, finding the exact nearest neighbor in the image database can be very slow. A common practice in CBIR is to find an approximate nearest neighbor instead of the exact nearest neighbor, by limiting the number of nodes traversed during the traversal of kd-trees [7].

In order to construct the kd-trees for CBIR, we need to partition the image feature points contained in a node into its two child nodes. This is performed with a hyperplane, which is defined by its normal and position

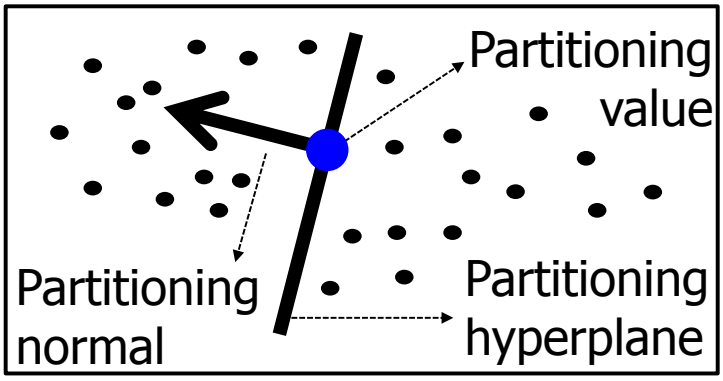

Figure 1: 2D example of a partitioning hyperplane.

(Fig. 1). The normal and position of the hyperplane can be called as the partitioning normal and the partitioning value respectively.

Optimizing kd-trees has been actively studied in recent years [8, 9], since constructing a high-quality kd-tree is critical for improving the performance of many CBIR techniques. These prior techniques improve the performance of nearest neighbor search by computing a partitioning normal that separates feature points well and reduces computation overheads of accessing nodes at the query time. In order to split the feature points in a node, the mean or the median point along the chosen partitioning normals are typically selected as the partitioning values. However, to the best of our knowledge, there has been no prior work that quantifies the quality of kd-trees in a cost model for CBIR and optimizes partitioning values and normals with the cost model.

Contributions: In order to quantify the quality of kd-trees and hence to evaluate the performance of the nearest neighbor search in CBIR,

we propose a probabilistic cost model (Sec. 4) that measures the expected number of nodes traversed during the process of the search query. We have also proved that the conventional wisdom of partitioning data points in each node of kd-trees in the median point along a partitioning normal that is close to the principal directions of the data points can produce near-optimal kd-trees, given our proposed cost model (Sec. 5). Our cost model and assumptions are validated through experiments in SIFT-based image retrieval, which show strong linear correlations (i.e. up to 0.9) between our cost model and both the observed number of nodes traversed and the running time spent for queries used in CBIR applications (Sec. 6). These results can be served as a theoretic basis for other optimization problems such as updating kd-trees, to achieve a high performance for dynamic data sets.

2. Related Work

In this section we review prior techniques on content-based image retrieval (CBIR) and optimizing kd-trees. We discuss examples most relevant to our works and refer reader to [2] for the survey of recent approaches in CBIR.

2.1. Image Retrieval

Given a query image, CBIR compares the image features of images from an image database with the features from the query image, and returns the images that are most similar to the query image. Such CBIR problem can be reduced to the well-known nearest neighbor problem. Data structures

such as kd-tree [5] are widely used to efficiently process the nearest neighbor problem, which performs the matching between query features and image features stored in the image database [2].

Unique in CBIR, the dimension of image features are usually very high. Using a kd-tree for accessing such high-dimensional data is well-known to be very slow, since the nearest neighbor computation algorithms have to traverse most nodes of a kd-tree. Hence, approximate nearest neighbor search algorithms [7] that use a priority queue and/or the best-bin-first search technique [4] have been commonly adopted in CBIR. Recently, Muja and Lowe [10] presented an automatic tuning algorithm for various kd-tree based approximate nearest neighbor search methods.

Visual words [11] (or bag-of-features), vocabulary tree [12], and locality sensitive hashing [13] are some of the popular techniques for large CBIR systems. A recent study by Philbin et al. [6] showed that performing nearest neighbor computation with kd-trees outperforms the technique using the vocabulary tree in terms of K-means clustering. It has also been shown that quality of searching results of locality sensitive hashing can be inferior to those of kd-trees [8]. While these previous works are related, we focus our analysis on evaluating kd-trees for high quality nearest neighbor computation. Our results will be useful to evaluate the performance of kd-trees.

2.2. Optimizing kd-Trees

A few techniques have been proposed to improve the quality of kd-trees for the acceleration of tree access in CBIR. Silpa-anan et al. [8] proposed to create multiple kd-trees for an input data set and use them together as a concurrent search with a pooled priority queue. They have also demonstrated that a higher performance can be achieved by aligning the principal axis of data with the partitioning normal of a partitioning hyperplane used for data. Recently, Jia et al. [9] proposed a binary combination of axis-aligned axes for partitioning normals for kd-trees, instead of using arbitrary axes.

Once a partitioning normal is decided as proposed in above techniques, a mean value or a median value along the chosen partitioning normal is used to define a partitioning hyperplane. While this is a common practice, there have been no prior techniques that pay attention to optimizing the partitioning value along a chosen partitioning normal.

Optimized kd-trees for ray tracing: kd-trees have been widely used in other applications, especially ray tracing in the field of computer graphics. It has been widely known that the quality of kd-trees plays one of the key factors that govern the performance of ray tracing and, thus, optimizing kd-trees for the application has been actively studied. For kd-trees used in ray tracing, the Surface Area Heuristic (SAH) metric that measures the traversal cost during the kd-tree traversal is proposed [14]. Also, various kd-tree construction techniques [15, 16, 17] that optimize the metric have been presented. However, the usage of kd-trees for ray tracing is different from

that for image retrieval; the structures and ordering of polygons used for kd-trees in ray tracing are very different from the one in image retrieval. As a result, it is unclear how the metric proposed for ray tracing can be applied to image retrieval.

2.3. Cost Models for kd-Trees

The database and computational theory communities have been developing various cost models for proximity queries including nearest neighbor search accelerated by spatial data structures such as kd-trees or R-trees. Bentley [18] introduced the concept of kd-trees and intuitively suggested that the median-based partitioning scheme for the kd-tree construction leads to the optimal tree in terms of a path length computed from the root node to a leaf node. This claim was an intuitive generalization of the optimality of one dimensional binary trees [19] to higher dimensional trees.

Friedman et al. [5] proposed a cost model for processing nearest neighbor search. Their cost model estimates the number of leaf nodes accessed in the kd-tree, while processing nearest neighbor queries. Extensions from the cost model [5] have been proposed for other spatial data structures like R-tree [20] and non-rectangular regional nodes [21]. However, these cost models have an unrealistic assumption that the number of data points is assumed to be infinite [22]. Lee and Wong [23] gave an worst-case analysis of balanced kd-trees for region queries in terms of the total number of node visits as a cost. Their worst-case analysis indicates that the dominant factor in the cost is the

dimensionality of data points. Sproull [24] later pointed out that these cost models are valid, when the number of points is exponential increasing as a function of the dimensionality of data points, in order to dilute the boundary effect. Arya et al. [25] developed an improved cost model that considers the boundary effect, but has an unrealistic assumption about the number of points as Sproull [24] pointed out. Berchtold et al. [26] proposed a cost metric mainly for high dimensional data, while the prior model of Friedman et al. [5] overestimates the cost by orders of magnitude for uniformly distributed high-dimensional data. Unfortunately, this method relied on the expensive Monte Carlo integration for evaluating the cost model, because the cost model is too complex to be calculated directly. Furthermore, it requires a huge number of samples for the Monte Carlo integration to achieve a reasonably accurate approximation. As a result, this model has been applied to a small scale of kd-trees.

These various cost models have been designed mainly for range queries and investigated in theoretical contexts. As a result, it is unclear how well these cost models are applicable to our domain, nearest neighbor search for high-dimensional image descriptors used in CBIR. Moreover, cost models proposed in theoretical communities makes assumptions inappropriate for CBIR. Some of them include that data points are uniformly distributed.

3. Access Patterns on kd-Trees and Terminologies

In this section we describe how a kd-tree is accessed given a query image for image retrieval. We will also define the terminologies used for the rest of the paper. We assume that the kd-tree is constructed based on image features (e.g. SIFT), and the same type of image features are extracted from the query image for searching nearest neighbor images.

Given an image feature, q_i, of a query image, we start to traverse the kd-tree from the root node of the kd-tree. During the kd-tree traversal, we maintain two variables: 1) the current minimum distance, min_d, and 2) the current candidate for the nearest neighbor feature that has the current minimum distance to the query image feature q_i. The initial minimum distance is set as infinity before traversal. Since there are many methods to traverse a kd-tree, we analyze two of the most common methods, the depth-first traversal [1, p.517] and the best-bin-first search [27].

Depth-first traversal (DFT): The DFT is the simplest traversal method for the nearest neighbor search. In this scheme, once we visit an intermediate node, we compare the features in its left and its right child nodes to the given image feature q_i. We traverse the child node that is closer to the given image feature and store the other child node in a stack, called *traversal stack*. Once we visit a leaf node, we measure the distance between q_i and each image feature stored in the leaf node. We pick the image feature, q_k, stored in the leaf node that gives the shortest distance to q_i, and check whether the shortest distance is smaller than the current minimum distance

min_d. If so, we update both the current candidate for the nearest neighbor as q_k and the current minimum distance as the distance between q_i and q_k. Then we pop a node from the traversal stack and access the node. This operation is commonly known as *back-tracking*. The process is performed recursively until the traversal stack becomes empty.

Culling techniques: The DFT is simple, but has to access all the nodes of the kd-tree, leading to a slow performance. A simple remedy to this problem is to employ various culling techniques. The most common culling method is to utilize a conservative distance bound; let us call this culling method *conservative distance culling*. For example, we can compute a conservative distance bound between the query point and the bounding box of a node of a kd-tree. If the conservative bound is bigger than the current minimum distance min_d, we cull the traversal operations on the child nodes located in the sub-tree of the node.

Best-bin-first search (BBFS): The BBFS has been known to show a higher performance than the DFT. The BBFS attempts to access nodes that are likely to have nearest neighbor points earlier than other nodes. To traverse kd-trees, the BBFS uses a priority queue instead of a stack. For each intermediate node, n_i, the BBFS computes a conservative distance bound between the query's image feature q_i and two child nodes of n_i. We traverse the child node that gives the smaller distance bound and store the other node into the priority queue with its conservative distance bound. Once we reach a leaf node, we perform the back-tracking operation, which dequeues

a node from the priority queue that gives the smallest distance bound and recursively traverse the node.

Approximate search: Approximate search is a common method to further speed up the kd-tree traversal by trading off the quality of search results. The approximate search is applicable for both DFS and BBFS kd-tree traversal methods. Typically, approximate search is achieved by limiting the number of traversed nodes during the kd-tree traversal, or by interrupting the search process based upon a real time clock. It has been well-known that if we limit the number of traversed nodes more, the quality of search results deteriorates.

Terminologies: We define $T(n)$ of a node n to denote the expected number of traversed nodes under the sub-tree of the node n, when the node n is accessed. We use n_l and n_r to denote the left and the right child nodes of a node n respectively. We also define $P_{[n_l|n]}$ to denote a conditional probability that the left node n_l is traversed, given the node n is accessed; $P_{[n_r|n]}$ is defined in a similar manner with the right node n_r. We use $n_{pdf}(i)$ to denote the probability distribution function for values of image features that are projected onto the chosen partitioning normal given a node n, where i is in the range of the minimum, m, and the maximum values, M, among the projected values.

Assumptions for mathematical derivations: In order to simplify our derivations for the sake of the clarity, we make a few assumptions: 1) each leaf node of a kd-tree contains only a single image feature, 2) when we

partition image features of a node n with a hyperplane that has a partitioning value p and a chosen partitioning normal, we assign image features whose values are equal to or less than the partitioning value p into the left node of the node n, and others into the right node, and 3) we treat the values of image features as continuous. Note that all of these assumptions are introduced to simplify our derivations, and all of our theoretic results can be shown to be valid even with kd-trees that do not satisfy such assumptions, after minor modifications to our derivations.

In the following sections we explain how each one of these assumptions are used for deriving our cost model and its theoretical results. Note that our cost model measures the expected number of nodes traversed during the kd-tree traversal. We would also like to point out that our cost model is still useful for approximate queries that is performed with a fixed number of traversed nodes. For example, if our cost value for a kd-tree, T_1, is less than that of another tree, T_2, it also means that given a fixed number of nodes traversed during the kd-tree traversal, T_1 can lead to more accurate results as compared with T_2. In this case, we show that our cost model has correlations with a quality measure on results of approximate queries. More specifically, we use a *distance error ratio* as the quality measure. This distance error ratio measures how much the distance between a nearest neighbor computed within the fixed number of traversed nodes and the query point is over the ground-truth shortest distance of the nearest neighbor to the query.

4. Probabilistic Cost Model

We define our probabilistic cost model that quantifies the quality of the kd-tree by measuring the expected number of nodes traversed during the nearest neighbor search. Once we access a node, we can access its left or its right child nodes, irrespective of different traversal methods on a kd-tree for nearest neighbor search queries. Therefore, we define our probabilistic cost model for a node n that measures the expected number of nodes traversed under the sub-tree rooted at the node n in a recursive manner, as follows:

$$T(n) = 1 + P_{[n_l|n]}T(n_l) + P_{[n_r|n]}T(n_r), \tag{1}$$

where 1 is added after the node n is traversed. Note that we can easily extend our cost model to include costs incurred by computing distances between the query image feature and features contained in leaf nodes. This is because such cost linearly depends on the number of image features contained in each leaf node. However, in order to simplify our discussions, we intentionally ignore such costs.

The problem is now reduced to how to accurately define the two conditional probabilities, $P_{[n_l|n]}$ and $P_{[n_r|n]}$, such that they can reflect the actual performance of kd-tree traversal. There are two main factors for computing these probabilities: 1) the distribution of image features of potential query images, and 2) the local geometric configurations of image features around the node n and its neighboring nodes.

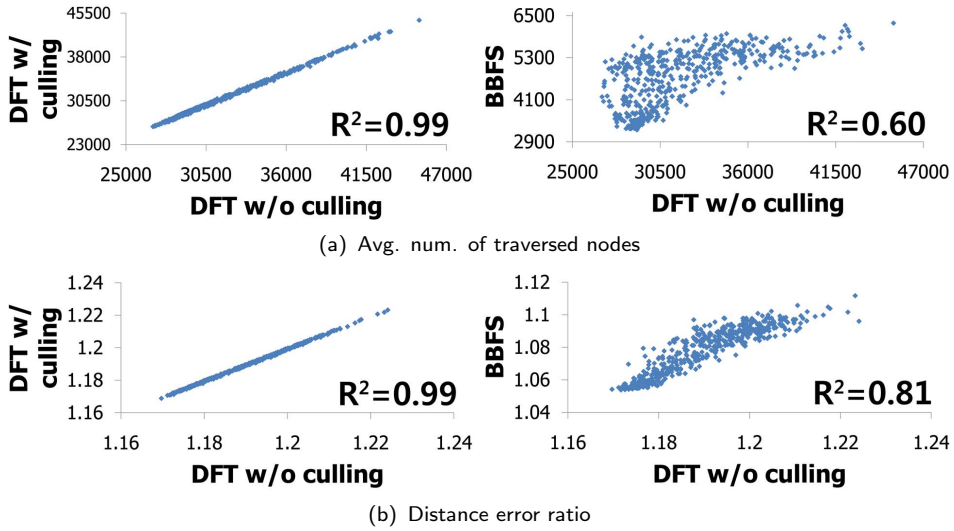

Figure 2: The left figures show correlations between the depth-first traversal (DFT) w/ and w/o culling, in terms of both the number of nodes traversed to find the exact nearest neighbor given query image features and distance error ratios given a fixed number of traversed nodes. The right figures show the correlations between the DFT and best-bin-first search (BBFS) in the same settings used for the left figures.

Effects of culling and traversal methods: Many different culling techniques including the conservative distance culling can be used. These culling techniques as well as traversal methods can affect conditional probabilities of nodes. We found that it is very hard to consider effects caused by these culling and traversal methods, since these effects depend on the local geometric configurations of data stored on a node and its neighboring nodes. We, however, make an interesting observation: it is highly likely that employed culling or traversal methods do not change the relative qualities of different kd-trees. In other words, if a kd-tree, T_1 has a fewer number of traversed nodes than another tree, T_2, then T_1 is likely to give a fewer number of traversed nodes than T_2 even with culling techniques.

To verify this observation, we measure correlations between the DFT with and without culling in terms of the number of traversed nodes that have been performed to find the nearest neighbor node given a query. To measure correlations, we construct 500 different kd-trees with image features from the Caltech 101 image benchmark that consists of around 10 K images; The details about how these 500 different kd-trees are constructed will be given in Sec. 6. We perform 10 K different queries used for SIFT-based image retrieval for each kd-tree. These queries are chosen from images that are in the same benchmark, but are not used for the tree construction.

Correlation results are plotted in Fig. 2 that contains results for both exact and approximate search queries. For Fig. 2-(a) representing results from the exact search queries, the x-axis is the number of nodes traversed by DFT without culling, and the y-axis is the number of nodes traversed by DFT with culling or BBFS. The left of Fig. 2-(a) shows correlation, 0.99, between the DFT with and without using the conservative culling. We have also measured correlations between the DFT and BBFS, and we found that the correlation is also very high (i.e. 0.6) as shown in the right of Fig. 2-(a). A simple observation from this graph is that the number of nodes visited with and without culling are linearly correlated. For a query image, if the number of nodes visited is K in DFT without culling, then the number of nodes visited in DFT with culling is αK, where α is a number less than 1.

We have also checked the correlations in approximate nearest neighbor search queries. In this case, we measure correlations in terms of distance

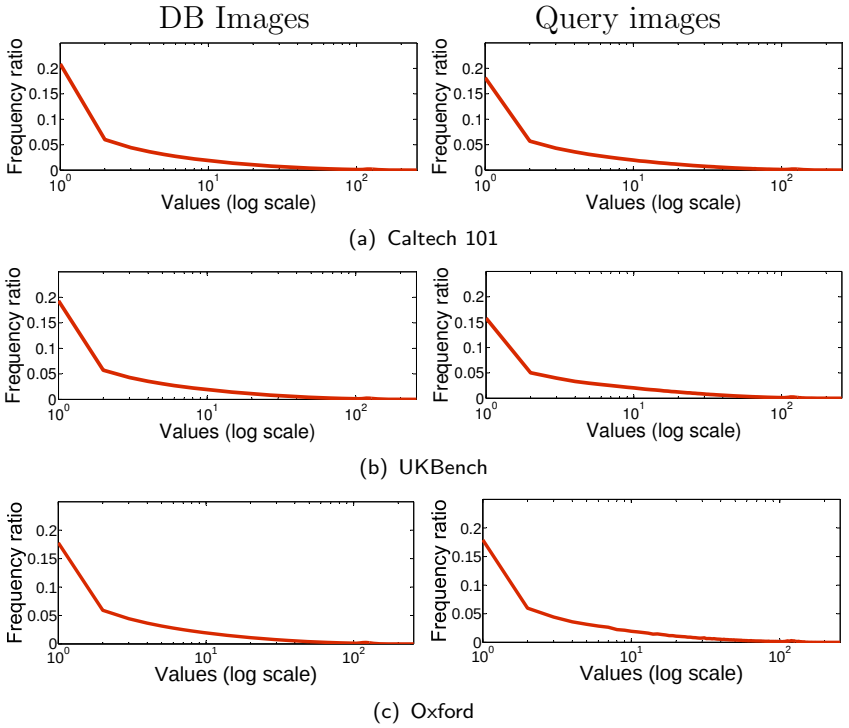

Figure 3: This figure shows distributions of values of SIFT image features; the top and bottom curves are computed from images of Caltech 101, UKBench, and Oxford benchmarks respectively. We use randomly chosen 500 images from Caltech 101 and UKBench (that create 100 K features), and 5000 images from the Oxford benchmark (that creates 1 M features) for building kd-trees (left figures under "DB Images") and another 500 images (that create another 10 K features) for runtime query images (right figures under "Query images").

error ratios. Again, we observe a high correlation, 0.99, and 0.81, in Fig. 2-(b) for the DFT with and without culling, and for the DFT and the BBFS respectively. Since there is such high correlation relationship, both in the exact nearest neighbor search and the approximate nearest neighbor search, we do not consider the effects caused by culling and traversal methods within our cost model. Nonetheless, we believe that our model is valid for kd-tree traversal method with culling or BBFS.

Distribution of potential query images: Fig. 3 shows the distributions of SIFT image features in our tested image benchmark datasets, Caltech 101, UKBench, and Oxford [6]. We simply project values of all the dimensions of these image features into one dimensional space to draw the distributions. We draw distributions of these values of image features drawn from different image sets of the same benchmark and drawn from different images of two different benchmarks. Even though they are computed from different images across different image benchmarks, these distributions have a nearly similar tendency. This is because inside natural images, most regions are smooth regions or regions with small gradients [28] and our plotting on the image features agrees with such natural image statistics. Hence, we can expect the distribution of potential query images follows the same distribution as the distribution of image features that were used to construct the kd-tree from an image database.

Now, we can define our final cost model. As discussed above, we do not consider the effects of culling and traversal methods such as BBFS, since they do not drastically change the relative qualities of kd-trees. Also, we assume that the distribution of image features of query images follows that of image features stored in the kd-tree. Then the conditional probability $P_{[n_l|n]}$ for the left node given a node n can be computed as $\int_m^p n_{pdf}(i)$, where p is the chosen partitioning value used for defining a partitioning hyperplane. Similarly, $P_{[n_r|n]}$ is defined as $\int_{p+}^M n_{pdf}(i)$. We use the range of $(p, M]$ for $P_{[n_r|n]}$, since we assumed that image features that have the partitioning value p is assigned

to the left node. As a result, our final cost model is defined as follows:

$$T(n) = 1 + \int_{m}^{p} n_{pdf}(i)T(n_l) + \int_{p+}^{M} n_{pdf}(i)T(n_r). \quad (2)$$

Our cost model will be validated with SIFT-based image retrieval in Sec. 6.

5. Near-Optimal Partitioning Value

In this section we prove that a near-optimal partitioning value exists given a chosen partitioning normal, to define a partitioning hyperplane for the high-quality kd-tree construction. Also, we show that the median-based partitioning can achieve a near-optimal result given our probabilistic cost model.

5.1. Lower Bound of Our Cost Model

We first show a lower bound of our cost model.

Lemma 5.1. *The following equation has the minimum value of* -1*, when* $x = \frac{1}{2}$*:*

$$y = x \log x + (1 - x) \log (1 - x), \quad (3)$$

where the logarithm function uses base 2 *and* $0 < x < 1$*.*

Proof: We rewrite y as the following:

$$y = x\frac{\ln x}{\ln 2} + (1 - x)\frac{\ln (1 - x)}{\ln 2}.$$

We calculate its first derivative, $\frac{dy}{dx}$:

$$\begin{aligned}\frac{dy}{dx} &= \frac{1}{\ln 2}(\ln x + 1) - \frac{1}{\ln 2}[1 + \ln(1-x)] \\ &= \frac{1}{\ln 2}[\ln x - \ln(1-x)].\end{aligned}$$

The first derivative becomes 0, when $x = \frac{1}{2}$. In order to confirm whether y has the global minima or maxima with $x = \frac{1}{2}$, we compute the second derivative, $\frac{d^2y}{dx^2}$:

$$\frac{d^2y}{dx^2} = \frac{1}{\ln 2}[\frac{1}{x} + \frac{1}{1-x}].$$

Since $0 < x < 1$, values of the second derivative are always positive. As a result, y has the minima, -1, when $x = \frac{1}{2}$. □

Theorem 5.1 (Lower Bound). *Given a kd-tree with k image features, the cost for this kd-tree according to our probabilistic cost model (Eq. 2) is at least* $1 + \log k$.

Proof:

Suppose that a node n has k image features under the sub-tree rooted at n. We define $f(n)$ to be the probability of accessing the left child node n_l after accessing node n; that is $f(n) = P_{[n_l|n]}$. Equivalently, $1 - f(n) = P_{[n_r|n]}$ is the probability of accessing the right child node n_r. Then, the left child

node n_l has $kf(n)$ image features [1] under its sub-tree, while the right child node n_r has $k(1-f(n))$ image features. For any intermediate node, the range of x is in $(0, 1)$, since some probability exists to access its child nodes.

Our cost model shown in Eq. 2 can be rewritten as follows:

$$T(n) = 1 + f(n)T(n_l) + (1 - f(n))T(n_r). \tag{4}$$

We first prove that $T(n) \geq 1 + \log k$ by induction. The base case arises when a node n is a leaf. In this case, $T(n) = 1$, since the number of traversed node is 1 for the leaf node. Since we assumed that each leaf node, n_{leaf} has a single image feature, $T(n_{leaf}) \geq 1 + \log(1)$. [2]

Let us consider cases when a node n is intermediate and thus $k > 1$. We start from the assumption for two child nodes of the node n: $T(n_l) \geq 1 + \log(kf(n))$ and $T(n_r) \geq 1 + \log(k(1 - f(n)))$. Then, $T(n)$ can be rewritten

[1] For the sake of clarity, we allow a continuous number of image features in our derivation. One can prove our theorem in a discrete manner by taking similar steps shown in the paper.

[2] If a leaf node has multiple image features, we can extend our derivation by showing $T(n) \geq \log k + c$, where c is a constant.

as follows:

$$
\begin{aligned}
T(n) &= 1 + f(n)T(n_l) + (1 - f(n))T(n_r) \\
&\geq 1 + f(n)(1 + \log(kf(n))) + (1 - f(n))(1 + \log(k(1 - f(n)))) \\
&= 2 + f(n)\log k + f(n)\log f(n) + \\
&\quad (1 - f(n))\log k + (1 - f(n))\log(1 - f(n)) \\
&= 2 + \log k + \\
&\quad f(n)\log f(n) + (1 - f(n))\log(1 - f(n)).
\end{aligned}
$$

According to Lemma 5.1, the term of $f(n)\log f(n)+(1-f(n))\log(1-f(n))$ has the minimum value of -1, when $f(n) = 1 - f(n) = \frac{1}{2}$. As a result, even when the node n is intermediate, we can show that $T(n) \geq 1 + \log k$. And this proves the theorem. □

As identified while we prove Theorem 5.1, the lower bound of our cost model is minimized when for every node, the conditional probability for accessing the left child node after accessing this node is exactly half. This does not directly indicate that our cost model is minimized at such case. Nonetheless, such case can be one of promising candidates for constructing optimal kd-trees given our cost model.

5.2. Median-based Partitioning

The derived lower bound of our probabilistic cost model for a kd-tree is minimized when for every node, the conditional probability of accessing its

left node is exactly equal to that of the right node. In practice it may be impossible to partition image features of a node into left and right nodes with the equal conditional probabilities, since SIFT image features in each dimension have discrete values and there may be multiple features that have the same value along a chosen partitioning normal. The median point among image features of a node, however, can be a fairly good candidate to make nearly equal conditional probabilities for the left and right child nodes.

In the same vein, partitioning normals that are close to be the principal eigenvectors computed with Principal Component Analysis (PCA) can serve as an excellent choice, especially when used together with median-based partitioning, since they can lead to nearly equal conditional probabilities for child nodes. These partitioning normals have been demonstrated to work quite well in practice [8, 9].

In this section we show that median-based partitioning produces near-optimal kd-trees in terms of minimizing our cost model.

Consider a kd-tree constructed using median-based partitioning. According to our cost model, the cost of a node n of the kd-tree containing k image features can be written as follows:

$$T(n) = 1 + \frac{\lceil k/2 \rceil}{k} T(n_l) + \frac{\lfloor k/2 \rfloor}{k} T(n_r),$$

where $\lceil k/2 \rceil$ and $\lfloor k/2 \rfloor$ image features are assigned to the left and right nodes respectively.

Given a kd-tree constructed by median based partitioning, one can observe that the cost of the tree only depends on the number of image features. Let $S(k)$ denote the cost of a kd-tree that is computed by median-based partitioning and has set of k image features. $S(k)$ can be recursively defined as follows:

$$S(k) = \begin{cases} 1 & \text{if } k = 1, \\ 1 + \frac{l}{2l} S(l) + \frac{l}{2l} S(l) & \text{if } k = 2l \text{ for some } l \geq 1, \\ 1 + \frac{l+1}{2l+1} S(l+1) + \frac{l}{2l+1} S(l) & \text{if } k = 2l+1 \text{ for some } l \geq 1. \end{cases}$$

When k is a power of two, we can easily show that $S(k)$ realizes the lower bound, which is $1+\log k$, in the next lemma (Lemma 5.2). As a result, in this case kd-trees computed by median-based partitioning are optimal in terms of our cost model.

Lemma 5.2. *If $k = 2^t$ for an integer $t \geq 0$, then $S(k) = 1 + \log k = 1 + t$.*

Proof: We use induction to prove this lemma. In the basic step, $t = 0$ and $k = 1$. Then $S(k) = 1$ by its definition. Now suppose that $k = 2^t$ for some $t > 0$. Assuming that $S(2^{t-1}) = 1 + t - 1$, we get that $S(k) = S(2^t) = 1 + \frac{1}{2} S(2^{t-1}) + \frac{1}{2} S(2^{t-1}) = 1 + S(2^{t-1}) = 1 + 1 + t - 1 = 1 + t$.

□

In practice, however, k may not be a power of two. We now introduce a series of lemmas to prove that in general cases, costs, $S(k)$, of kd-trees with k image features constructed by median-based partitioning is near-optimal.

Suppose that two kd-trees are constructed by using median-based parti-

tioning, but one kd-tree is constructed with k image features, while another one with $k+1$ features. We then have the following lemma.

Lemma 5.3. *$S(k+1) > S(k)$ for $k \geq 1$.*

Proof: We prove this lemma by induction. It is easy to see that $S(1) = 1$, $S(2) = 1 + \frac{1}{2}S(1) + \frac{1}{2}S(1) = 2$, and $S(3) = 1 + \frac{2}{3}S(2) + \frac{1}{3}S(1) = 8/3$. Therefore, when $k = 1$ or $k = 2$, we have $S(k+1) > S(k)$. We now show that $S(k+1) > S(k)$, when $k = 2l+1$, an odd number greater than 1. We assume that $S(t+1) > S(t)$ for $1 \leq t \leq 2l$. Then,

$$\begin{aligned} S(k+1) &= S((2l+1)+1) = S(2l+2) \\ &= 1 + \frac{l+1}{2l+2}S(l+1) + \frac{l+1}{2l+2}S(l+1) = 1 + S(l+1) \\ &= 1 + \frac{l+1}{2l+1}S(l+1) + \frac{l}{2l+1}S(l+1) \\ &> 1 + \frac{l+1}{2l+1}S(l+1) + \frac{l}{2l+1}S(l) \\ &= S(2l+1) = S(k) \end{aligned}$$

Let us consider the case of $k = 2l$, an even number greater than 2. We can assume that $S(t+1) > S(t)$ for $1 \leq t \leq 2l-1$.

$$
\begin{aligned}
S(k+1) &= S(2l+1) \\
&= 1 + \frac{l+1}{2l+1} S(l+1) + \frac{l}{2l+1} S(l) \\
&> 1 + \frac{l+1}{2l+1} S(l) + \frac{l}{2l+1} S(l) \\
&= 1 + S(l) = 1 + \frac{l}{2l} S(l) + \frac{l}{2l} S(l) = S(2l) = S(k)
\end{aligned}
$$

This proves the lemma. □

Corollary 5.1. *$S(k+t) \geq S(k)$ for any integer $t \geq 0$.*

Let a function $\varphi : \mathbb{Z}^+ \to \mathbb{Z}^+$ to denote $\varphi(k) = 2^{\lceil \log k \rceil}$; $\varphi(k)$ is the smallest positive integer greater than or equal to k such that $\varphi(k)$ is a power of two. If k itself is a power of two then $\varphi(k) = k$. This function's useful properties are given below:

$$\varphi(k) \geq k \tag{5}$$

$$\log(\varphi(k)) \leq 1 + \log k \tag{6}$$

$$S(\varphi(k)) = 1 + \log(\varphi(k)) \tag{7}$$

Equation 5 is obvious from its definition. Equation 6 comes from the fact that $\log(\varphi(k)) = \log(2^{\lceil \log k \rceil}) = \lceil \log k \rceil \leq 1 + \log k$. Finally, Equation 7 follows from Lemma 5.2 and the fact that $\varphi(k)$ is a power of two. We now show a tight upper bound of $S(K)$ based on the function φ.

Lemma 5.4. $S(k) \leq 2 + \log k$

Proof:

$$\begin{aligned} S(k) &\leq S(\varphi(k)) && [\text{ by Corollary 5.1 and (5) }] \\ &= 1 + \log(\varphi(k)) && [\text{by Equation 7}] \\ &\leq 2 + \log k && [\text{by Equation 6}] \end{aligned}$$

□

For a kd-tree constructed by median-based partitioning, Lemma 5.4 shows that its cost is at most $2 + \log k$, where k is the number of image features. Theorem 5.1 shows that the cost of any kd-tree with k image features is lower bounded by $1 + \log k$. As a result, the following theorem naturally holds.

Theorem 5.2 (Near-Optimality). *Given our cost model (Eq. 2), let C to be a cost of a kd-tree with k image features constructed by median-based partition. Then $C \leq 1 + OPT$, where OPT is the minimum cost of any kd-tree with the same set of those image features.*

This theorem indicates that median-based partitioning produces near-optimal kd-trees given our cost model.

6. Experimental Validations

We randomly choose 500 images from the Caltech 101 image benchmark and extract around 100 K SIFT features from those images for constructing the kd-trees. 500 different kd-trees are constructed by randomly choosing the partitioning values given the partitioning normals. For each kd-tree, we

evaluate our cost model with the tree by measuring the cost value associated with the root node of the kd-tree. We have also measured the number of traversed nodes to find the nearest neighbor features given 10 K query SIFT features. These 10 K SIFT features are extracted from different 500 query images for our SIFT-based image retrieval. These query images are randomly chosen from the same image benchmark, but are not used for the tree construction.

In this configuration, we found that our cost model shows a positive correlation, 0.526, against the average number of nodes traversed with the DFT. This correlation score is lower than the other correlation score presented in this paper. This is mainly because when we perform the exact search queries, the traversal methods tend to traverse most of nodes of kd-trees irrespective of the qualities of kd-trees. Because of this behavior of the exact nearest neighbor query, approximate search queries are typically used in practice. We have also measured the correlation by applying the same setting, but this time we use BBFS and culling with the UKBench benchmark. The correlation score goes up to 0.79.

To further analyze the correlation of our cost model against the quality of kd-tree, we measure the time spent to traversing kd-trees. Our cost model shows a high correlations, 0.74, with the average time spent on performing exact search queries that run by using the BBFS in the UK Bench image benchmark and the Caltech 101 image benchmark.

We check the correlation of our cost model with approximate nearest

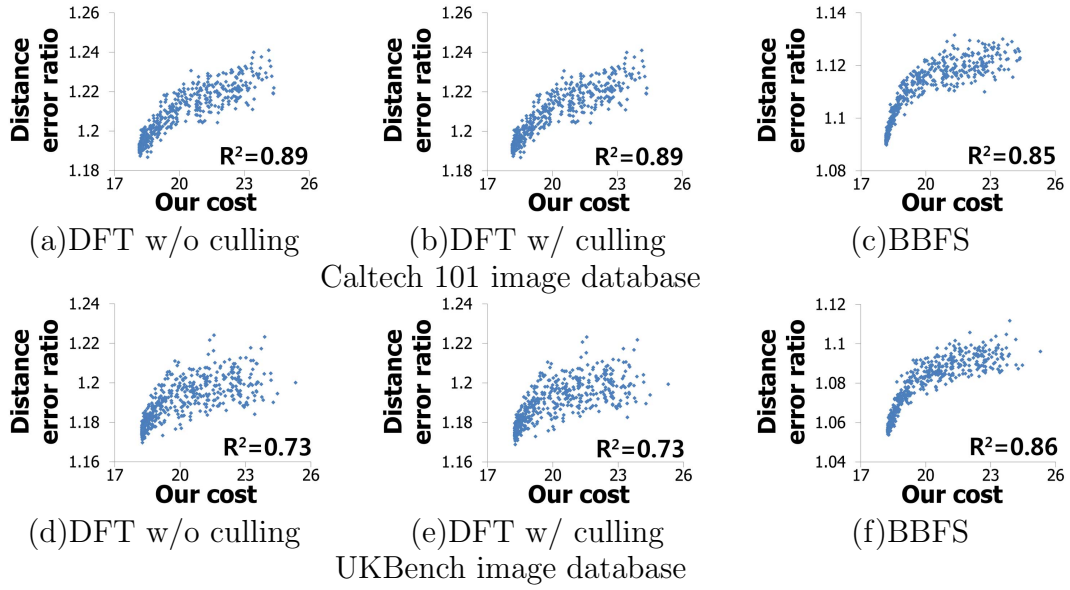

Figure 4: This figure shows correlations of our cost model against the distance error ratios achieved by (a)(d) the DFT w/o culling, (b)(e) DFT with culling, and (c)(f) BBFS in the Caltech 101 and UKBench image databases.

neighbor search queries. In this case, we measure the correlation between values of our cost model and distance error ratios achieved when we limit the number of traversed nodes to 1 K nodes. Fig. 4 shows the correlations of our cost model against the distance error ratios achieved by different traversal methods for the Caltech 101 and UKBench image benchmarks. Our cost model shows high correlations, 0.89, 0.89, and 0.85, over DFT w/o culling, DFT w/ culling, and BBFS respectively in the Caltech 101 benchmark. We observe that our cost model shows similar, high correlations, 0.73, 0.73, and 0.86, against DFT w/o culling, DFT w/ culling, and BBFS respectively in the UKBench image benchmark.

We have also found that median partitioning consistently gives the lowest

cost in term of both our cost model and the distance error ratios, as conjectured in Sec. 5.2. More specifically speaking, a kd-tree constructed by a partitioning value among the median, m, and $m \pm 1$ given a node shows the best result, compared to those 500 different kd-trees that are constructed by randomly choosing the partitioning values.

7. Conclusion

We have presented a probabilistic cost model that measures the expected number of traversed nodes during the kd-tree traversal. Our cost model has demonstrated to have high correlations with the observed numbers of traversed nodes as well as the time spent on image search under different culling, traversal methods including exact and approximate queries. Furthermore, our cost model can explain why the commonly adopted partitioning methods such as median-based and PCA-based techniques work well and showed that they can achieve a near-optimal quality for the kd-tree construction.

Limitations and future work: There are many avenues for future research directions. First, we would like to see how well our theoretical and experimental results extend to web-scale image databases that consist of billions of images. In Sec. 4, we showed that culling and kd-tree traversal methods do not change the relative qualities of different kd-trees much and thus we do not consider them in our probabilistic cost model. However, it may be possible to consider a particular traversal algorithm (e.g., using multiple randomized kd-trees [10]) with culling and back-tracking properties in a cost

model, and use it to construct a kd-tree that has a higher quality than the one constructed by the proposed probabilistic model. Moreover, we would like to extend our current cost model to consider the dimensionality of data points for more accurate estimation of costs. Also, we would like to apply our cost model and its optimal partitioning theorem to incrementally update the kd-trees for dynamic data sets in image retrieval. Prior approaches for dynamic image datasets are based on heuristic techniques that decide when and where to reconstruct kd-trees. However, through our cost model and optimal partitioning value theorem, we can approach this problem in a more rigorous manner. We wish that our cost model can serve as a theoretical basis that leads to more rigorous techniques for various problems related to nearest neighbor search queries including recent hashing techniques [29].

Acknowledgements

This work was supported in part by MEST/NRF (2011-0030822), MCST/KOCCA/CT/R&D 2011, MKE/KEIT [KI001810035261], MKE/MCST/IITA [2008-F-033-02], BK, DAPA/ADD (UD110006MD),MEST/NRF/WCU (R31-2010-000-30007-0), KMCC, and MSRA.

References

[1] H. Samet, Foundations of MultiDimensional and Metric Data Structures, Morgan Kaufmann, 2006.

[2] R. Datta, D. Joshi, J. Li, J. Z. Wang, Image retrieval: Ideas, influences, and trends of the new age, ACM Computing Survey 40 (2) (2008) 1–60.

[3] K. Q. Weinberger, J. Blitzer, L. K. Saul, Distance metric learning for large margin nearest neighbor classification, in: NIPS, 2006.

[4] D. Lowe, Distinctive image features from scale-invariant keypoints, IJCV 60 (2) (2004) 91–110.

[5] J. H. Friedman, J. L. Bentley, R. A. Finkel, An algorithm for finding best matches in logarithmic expected time, ACM TOMS 3 (3) (1977) 209–226.

[6] J. Philbin, O. Chum, M. Isard, J. Sivic, A. Zisserman, Object retrieval with large vocabularies and fast spatial matching, in: IEEE Conference on Computer Vision and Pattern Recognition, 2007, pp. 1–8.

[7] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, A. Wu, An optimal algorithm for approximate nearest neighbor searching, in: Symp. on Discrete Alg., 1994, pp. 573–582.

[8] C. Silpa-Anan, R. Hartley, S. Machines, A. Canberra, Optimised kd-trees for fast image descriptor matching, in: IEEE Conference on Computer Vision and Pattern Recognition, 2008, pp. 1–8.

[9] Y. Jia, J. Wang, G. Zeng, H. Zha, X.-S. Hua, Optimizing kd-trees for scalable visual descriptor indexing, in: CVPR, 2010.

[10] M. Muja, D. G. Lowe, Fast approximate nearest neighbors with automatic algorithm configuration, in: VISSAPP, 2009, pp. 331–340.

[11] J. Sivic, A. Zisserman, Video google: A text retrieval approach to object matching in videos, in: ICCV, 2003.

[12] D. Nistér, H. Stewénius, Scalable recognition with a vocabulary tree, in: CVPR, 2006.

[13] M. Datar, N. Immorlica, P. Indyk, V. S. Mirrokni, Locality-sensitive hashing scheme based on p-stable distributions, in: SoCG, 2004.

[14] D. MacDonald, B. K.S., Heuristics for ray tracing using space subdivision, Visual Computing 6 (3) (1990) 153–166.

[15] I. Wald, V. Havran, On building fast kd-trees for ray tracing, and on doing that in O(N log N), in: Proceedings of the 2006 IEEE Symposium on Interactive Ray Tracing, 2006, pp. 61–69.

[16] W. Hunt, W. Mark, G. Stoll, Fast kd-tree construction with an adaptive error-bounded heuristic, in: IEEE Symposium on Interactive Ray Tracing 2006, 2006, pp. 81–88.

[17] S. Yoon, S. Curtis, D. Manocha, Ray tracing dynamic scenes using selective restructuring, Eurographics Symp. on Rendering (2007) 73–84.

[18] J. L. Bentley, Multidimensional binary search trees used for associative searching, Communications of the ACM 18 (1975) 509–517.

[19] D. E. Knuth, The Art of Computer Programming, Vol. 1, Addison-Wesley, 1969.

[20] C. Faloutsos, T. Sellis, N. Roussopoulos, Analysis of object oriented spatial access methods, in: ACM SIGMOD, 1987.

[21] J. G. Cleary, Analysis of an algorithm for finding nearest neighbors in euclidean space, ACM Transactions on Mathematical Software (TOMS) 5 (1979) 183–192.

[22] C. Böhm, A cost model for query processing in high dimensional data spaces, ACM Transaction on Database Systems 25 (2000) 129–178.

[23] D. T. Lee, C. K. Wong, Worst-case analysis for region and partial region searches in multidimensional binary search trees and balanced quad trees, Acta Informatica 9 (1) (1977) 23–29.

[24] J. F. Sproull, Refinements to nearest-neighbor searching in k-dimensional trees, Algorithmica 6 (1991) 579–589.

[25] S. Arya, D. M. Mount, O. Narayan, Accounting for boundary effects in nearest neighbor searching, in: Symposium on Computational Geometry (SoCG), 1995.

[26] S. Berchtold, C. Böhm, D. A. Keim, H.-P. Kriegel, A cost model for nearest neighbor search in high-dimensional data space, in: Proceedings of the sixteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems, 1997.

[27] D. G. Lowe, Object recognition from local scale-invariant features, in: the International Conference on Computer Vision, 1999.

[28] A. Torralba, A. Oliva, Statistics of natural image categories, Network: Computation in Neural Systems 14 (3) (2003) 391–412.

[29] J.-P. Heo, Y. Lee, J. He, S.-F. Chang, S.-E. Yoon, Spherical hashing, in: CVPR, 2012, to appear.