PAPER *Special Section on Data Engineering*

Optimal Pivot Selection Method Based on the Partition and the Pruning Effect for Metric Space Indexes

Hisashi KURASAWA[†a)], **Daiji FUKAGAWA**[††b)], **Atsuhiro TAKASU**[†††c)], ***and*** **Jun ADACHI**[†††d)], ***Members***

SUMMARY This paper proposes a new method to reduce the cost of nearest neighbor searches in metric spaces. Many similarity search indexes recursively divide a region into subregions by using pivots, and construct a tree-structured index. Most of recently developed indexes focus on pruning objects and do not pay much attention to the tree balancing. As a result, indexes having imbalanced tree-structure may be constructed and the search cost is degraded. We propose a similarity search index called the Partitioning Capacity (PC) Tree. It selects the optimal pivot in terms of the PC that quantifies the balance of the regions partitioned by a pivot as well as the estimated effectiveness of the search pruning by the pivot. As a result, PCTree reduces the search cost for various data distributions. We experimentally compared PCTree with four indexes using synthetic data and five real datasets. The experimental results shows that the PCTree successfully reduces the search cost.

key words: *pivot selection, similarity search, metric space*

1. Introduction

Similar object search is a key function for speeding up various algorithms. For example, image completion algorithms fill in the missing regions in images by searching for similar image regions from a large photo dataset [12]. Applications using such algorithms require an efficient similar object search algorithm to reduce the query execution cost.

Similarity search indexes are used for pruning objects dissimilar to a query [26], and reduce the search cost, such as the distance computations and the disk accesses. Indexes that are based on a metric space can be applied to all types of data whose distances obey the metric space postulates such as the triangle inequality. Most indexing schemes use reference objects called *pivots*. They recursively divide a region into subregions by using pivots, and construct a tree structure index. That is, the methods of selecting pivots and dividing up the space by using these pivots determine the index structure and pruning performance. The existing pivot selection studies have mainly focused on increasing the number of pruned objects at a branch in the tree [6]. However, most previous works do not take into account the balance of the tree. Tree balancing techniques are used in database management systems (DBMSs), and it is well known that the tree height reduction contributes to reduce the average search cost [5]. Thus, even if the pivots of the methods are enough good for pruning objects at the branch, the effect of search cost reduction may be limited according to the data distribution of a database. Therefore, we developed a new pivot selection method for optimizing both the pruning and the balancing [18][*].

The main contributions of this paper are as follows.

- We propose a new information theoretic criterion called the *Partitioning Capacity (PC)* for pivot selection. The PC takes both the object pruning effect and index tree balance into account.
- We developed a metric space index called the *Partitioning Capacity Tree (PCTree)*. The PCTree provides the necessary functions for constructing an index tree based on PC and for efficiently retrieving similar objects using the index tree.
- We show the efficiency of PCTree empirically through several experiments where we compared the PCTree with several metric space indexes using synthetic multi-dimensional data and five real datasets.

2. Related Work

Let $M = (D, d)$ be a metric space defined for a domain of objects D and a distance function $d : D \times D \mapsto \mathbb{R}$. The Minkowski distance, Jaccard's coefficient, and the edit distance are examples of the distance function d. Our index deals with the space and these distances.

The early indexing schemes of similarity searches used balanced tree structures. Vantage Point Tree (VPT) divides a region into two subregions based on the distance from a pivot [24]. Its distance is set for equally partitioning the objects in the region, so that the two subregions contain the same number of objects. Thus, VPT is a completely balanced binary tree. However, the early indexes are weak at pruning objects while searching. They selects pivots that are far away from the other objects and other pivots by using simple statical features, such as the variance.

Some indexing schemes focus on pruning dissimilar

Manuscript received June 5, 2010.
Manuscript revised October 4, 2010.

[†]The author is with the Graduate School of Information Science and Technology, The University of Tokyo, Tokyo, 101–8430 Japan.
[††]The author is with the Faculty of Culture and Information Science, Doshisha University, Kyotanabe-shi, 610–0394 Japan.
[†††]The authors are with National Institute of Informatics, Tokyo, 101–8430 Japan.
a) E-mail: kurasawa@nii.ac.jp
b) E-mail: dfukagaw@mail.doshisha.ac.jp
c) E-mail: takasu@nii.ac.jp
d) E-mail: adachi@nii.ac.jp

DOI: 10.1587/transinf.E94.D.504

[*]This paper is the extended version of [18]. In this paper, we describe the indexing technique in details and add experiments using synthetic data to clarify the characteristics of the PCTree.

objects rather than balancing the tree. Some schemes use other partitioning techniques instead of Ball partitioning. Vantage Point Forest (VPF) [25] and D-Index [11] divide a region into three subregions according to the distance from the pivot. They aim to exclude the middle area in the region because it is difficult to judge whether objects in this area are similar or dissimilar to a query. Generalized Hyper-plane Tree (GHT) [23] divides the space by using a hyper-plane equidistant from the two pivots.

Other schemes improve the pivot selection. Many studies have attempted to select pivots that are distant from the other objects and other pivots [6], [24]. These methods used the variance [24], the mean [7], [11], the sum [10], and the ratio [22]. iDistance [13] selects pivots based on the clustering result and reduces the number of regions accessed during a search. OMNI-Family [15] chooses a set of pivots based on the minimum bounding region. MMMP [17] has a pivot selection based on the maximal margin, and it classifies dense regions. Moreover, some methods combine different pivot selection techniques [19], [27].

Still other schemes propose various index structures. Fixed Queries Array [8] is an array of distances between objects and pivots and it is used to save memory. List of Clusters (LC) [9] is a list of compact regions. It recursively divides the space into a compact region. iDistance consists of a list of pivots and a set of B^+-trees for storing objects. Thus, its index structure is partially balanced. Although the B^+-trees are balanced, the pivots are linearly accessed. Spatial Approximation Tree (SAT) [20] approximates a graph and uses pivots for approaching the query spatially. M-Tree [10] is a dynamic tree structure with object insertion and deletion. Slim-Tree [16] minimizes the overlap space managed by node in M-Tree.

We focus on the pivot selection method in this paper. The simple statistics-based approach [6], [24] aims at determining the various distances between the pivot and each object and reducing the number of useless pivots. However, they don't consider the partitioning boundary of the pivot and the selected pivot may not be good for pruning. The clustering-based approach [13], [17] aims at selecting a pivot and its partitioning boundary based on the clusters in the dataset. Although it can select better pivots for pruning than the simple statistics-based approach, it doesn't take into consideration the tree balancing. An imbalanced index tree increases the average search cost. Thus, we aim at reducing the search cost by taking both the object pruning and the tree balance into account. For this purpose, we extract even more information concerning the data distribution.

3. Partitioning Capacity Tree

This section proposes PCTree, a similarity search index for a metric space. The index is designed for nearest neighbor searches and aims at reducing the search cost for various data distributions. It selects pivots that have effect on pruning objects as well as balancing the index tree. It classifies the space with two criteria for measuring a pivot (Fig. 1).

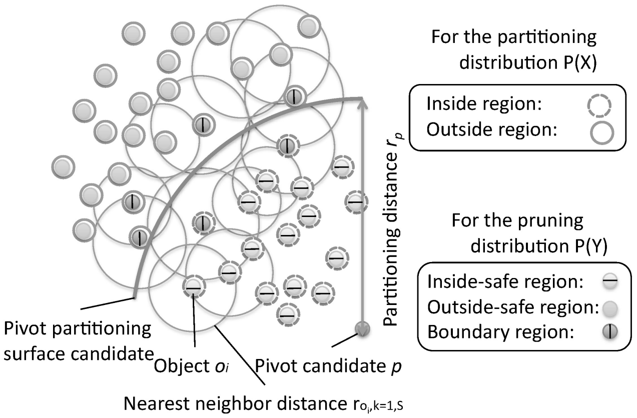

For the partitioning distribution P(X)
Inside region:
Outside region:
For the pruning distribution P(Y)
Inside-safe region:
Outside-safe region:
Boundary region:
Partitioning distance r_p
Pivot partitioning surface candidate
Object o_i
Pivot candidate p
Nearest neighbor distance $r_{o_i,k=1,S}$

Fig. 1 Labels for measuring the PC.

3.1 Partitioning Capacity

We first introduce a measure of a pivot called PC. It represents the expected index performance for a given query distribution. We assume that both the queries and objects in the database are drawn from the same probability distribution.

For a metric space $M = (D, d)$ and a region $R \subseteq D$ in the space, suppose that a pivot p and its partitioning distance r_p divide R into two subregions:

$$R_1(p, r_p) = \{o \in R \mid d(o, p) \le r_p\},$$
$$R_2(p, r_p) = \{o \in R \mid d(o, p) > r_p\} . \quad (1)$$

We respectively refer to $R_1(p, r_p)$ and $R_2(p, r_p)$ as an *inside region* and an *outside region* with respect to pivot p and distance r_p.

For a query q, let X be the random variable that represents the region in which q is included, that is,

$$X = X_{i,p,r_p} \quad \text{if } q \in R_i(p, r_p) \quad (i = 1, 2) . \quad (2)$$

We call $P(X)$ a *partitioning distribution*. Since we assume that a query is drawn from the same distribution as the database, we estimate the partitioning distribution as

$$P(X_{i,p,r_p}) = \frac{|S_{R_i(p,r_p)}|}{|S_R|} \quad (i = 1, 2) , \quad (3)$$

where S_R denote the sets of database objects that are in the region R.

Similarly, we define the *pruning distribution* of the query. For a set S of objects in the database and an object o in the region R, let $r_{o,k,S}$ denote the distance between o and o's kth nearest neighbor in S. Let us consider the following three regions:

$$R'_1(p, r_p, k) = \{o \in R \mid d(o, p) + r_{o,k,S} \le r_p\} , \quad (4)$$
$$R'_2(p, r_p, k) = \{o \in R \mid d(o, p) - r_{o,k,S} > r_p\} , \quad (5)$$
$$R'_3(p, r_p, k) = R - R'_1(p, r_p, k) - R'_2(p, r_p, k) . \quad (6)$$

Intuitively, $R'_1(p, r_p, k)$ is the set of the objects whose k-nearest neighbor objects are within $R_1(p, r_p)$. This means that, if a query q belongs to the region $R'_1(p, r_p, k)$, we can prune $R_2(p, r_p)$ when executing the k-nearest neighbor search. Similarly, $R'_2(p, r_p, k)$ is the set of the objects whose k-nearest neighbor objects are within $R_2(p, r_p)$. We respectively refer to $R'_1(p, r_p, k)$, $R'_2(p, r_p, k)$, and $R'_3(p, r_p, k)$ as an *inside-safe region*, an *outside-safe region*, and a *boundary region* w.r.t the pivot p, the distance r_p, and the number k of nearest neighbors. Let Y be a random variable that represents the region in which the k-nearest neighbor ranges of q is included, that is,

$$Y = Y_{i,p,r_p,k} \quad \text{if } q \in R'_i(p, r_p, k) \quad (i = 1, 2, 3)\,. \tag{7}$$

We call $P(Y)$ a *pruning distribution*. We estimate the distribution as

$$P(Y_{i,p,r_p,k}) = \frac{|S_{R'_i(p,r_p,k)}|}{|S_R|} \quad (i = 1, 2, 3)\,. \tag{8}$$

By using the two random variables X and Y, the PC for a pivot p is defined as

$$\begin{aligned} &PC_k(p) \\ &\equiv \max_{r_p} I(X; Y) \\ &= \max_{r_p} \left(\sum_i \sum_j \left(P(X_{i,p,r_p}, Y_{j,p,r_p,k}) \right.\right. \\ &\quad \left.\left. \cdot \log \left(\frac{P(X_{i,p,r_p}, Y_{j,p,r_p,k})}{P(X_{i,p,r_p}) P(Y_{j,p,r_p,k})} \right) \right) \right), \end{aligned} \tag{9}$$

where $I(\cdot;\cdot)$ is the mutual information. We choose the object p (resp. distance r_p) that maximizes Eq. (9) as a pivot (resp. partitioning distance).

The mutual information satisfies

$$I(X; Y) = H(X) - H(X \mid Y) \leq H(X)\,, \tag{10}$$

$$I(X; Y) = H(Y) - H(Y \mid X) \leq H(Y)\,, \tag{11}$$

where $H(\cdot)$ is the entropy. The PC represents the mutual information of the random variables for the partitioning and pruning distributions. We define the PC usage by using Eq. (10). $H(X)$ represents the entropy of random variable X for the partitioning distribution. It represents the balance of the partitioned regions. When we set the partitioning with a larger entropy, the index tree is balanced and is closer to a complete binary tree. A balancing tree is good for reducing the average search cost. $H(X \mid Y)$ represents the entropy of the probability X under the condition Y, where Y is the random variable for the pruning distribution. The pruning distribution classifies a region into three subregions on the basis of the pruning. For a query in the inside-safe region (resp. the outside-safe region), the pivot can prune the outside region (resp. the inside region). For a query in the boundary region, the pivot cannot prune objects. From the definition, the inside-safe region (resp. the outside-safe region) is included in the inside region (resp. the outside region). The boundary region is part of the inside and the outside region. Thus, when the boundary region gets smaller, $H(X \mid Y)$ decreases. We use $H(X|Y)$ for estimating the effectiveness of the pruning. We define the effectiveness of both the balancing and pruning by using entropy and we avoid considering their weights. The criterion for the PC is inspired by the formula of the channel capacity of a binary erasure channel in the information and coding theory [14].

3.2 PCTree Construction

PCTree has a tree consisting of *internal nodes* and *leaves*. An internal node is associated with one pivot and its partitioning distance whereas a leaf node is associated with one pivot and the objects. This section shows an outline of the index construction process. Details of some of the steps are described in the following subsections.

In the indexing phase, PCTree recursively divides a region into its inside and outside subregions by using pivots. For a metric space $M = (D, d)$ and a set S of objects, the PCTree constructs an index by

1. selecting a pivot candidate set S_p from S (Sect. 3.4 for details), and
2. dividing D recursively into inside and outside regions.

We first select the most effective pivot candidates S_p because a database usually contains large amount of objects and calculating PC for all these objects is computationally infeasible. Note that the pivot candidate selection is done once before constructing an index.

For a region R and set of objects S, we recursively executes the following procedure $\mathcal{P}(R)$:

1. if the stopping condition described in Sect. 3.3 is satisfied,
    a. choose a pivot p randomly from S_R,
    b. make a sorted list L of objects in S_R according to the distance from p, and
    c. return a leaf node associated to p and L
2. otherwise
    a. choose predefined number m of samples T from S_R randomly,
    b. select the pivot $p \in S_p$ and its partitioning distance r_p which maximize the PC given by Eq. (9) for the random samples T,
    c. call the region partitioning function with the inside and the outside regions w.r.t. p and r_p, and obtain the respective node $v_1 := \mathcal{P}(R_1(p, r_p))$ and $v_2 := \mathcal{P}(R_2(p, r_p))$, and
    d. return an internal node v associated with the pivot p, partitioning distance r_p and child nodes v_1, v_2.

We use random samples T instead of R_S in step 2-(a) to reduce the computational cost of the pivot selection.

3.3 Condition for Stopping the Partition

Let us consider the minimum PC for stopping partitioning. If we do not use a similarity search index, the search requires

$O(N)$ distance computations by using the linear scan method where N is the number of objects in the database. Therefore we should not use PCTree if its estimated number of distance calculations is higher than the naive sequential access method. Thus, we define the minimum PC for filtering inappropriate pivot candidates that has less search effect than the naive method. For measuring the PC of the pivots in the linear scan method, we regard the method as that in which all the objects in the dataset are pivots and each pivot prunes itself while searching. Therefore we set the minimum PC as

$$\mathrm{MinimumPC}(S) = -\frac{1}{|S|} \cdot \log \frac{1}{|S|} - \frac{|S|-1}{|S|} \cdot \log \frac{|S|-1}{|S|} , \quad (12)$$

where S is the object set. When the PCs of the pivot candidates in a region are less than the minimum PC, the partitioning finishes and the region is set as a leaf node in the tree. For example, when $|S|$ is 100, $P(X_{1,p,r_p})$ and $P(X_{2,p,r_p})$ are 0.5, $P(Y_{1,p,r_p,k})$ and $P(Y_{3,p,r_p,k})$ are 0.04, and $P(Y_{3,p,r_p,k})$ is 0.92, the PC is about 0.080 and the minimum PC is about 0.081, so that the partitioning stops.

3.4 Sampling the Pivot Candidates

The indexing algorithm of PCTree is a kind of greedy algorithm. It incurs a heavy construction cost for finding the optimal pivot and its partitioning distance. We reduce the cost by sampling the pivot candidates.

We select pivot candidates such that they are separated from each other by a certain distance d_p. The reason why we apply it to the sampling of the pivot candidates is as follows. For pivot candidates p_1 and p_2, the distance from p_2 to an object o satisfies

$$|d(p_1, o) - d(p_2, o)| \leq d(p_1, p_2) . \quad (13)$$

From the inequality, we can expect that the larger $d(p_1, p_2)$ is, the larger the difference between $d(p_1, o)$ and $d(p_2, o)$ will be. Thus, we choose objects with the following two policies:

- the pivot candidates are separate from each other in the space, and
- the pivot candidates are not selected from the partitioned region but from all the objects in the dataset.

Therefore, we choose a pivot candidate set such that the distance between any pivot pair in the set is more than $d_{\text{ave}} \cdot s$, where d_{ave} is the approximate average distance between objects and s is a parameter. The approximate average distance is calculated by using randomly sampled objects. Pivot candidates are sampled from the data set by performing the following steps:

1. Let S_p, the pivot candidate set, be an empty set. Insert all the objects in the data set into the object set S.
2. Randomly select an object p from the object set S and add p to S_p.
3. While S is not empty:
   - Remove all objects from S whose distances to p are less than $d_{\text{ave}} \cdot s$.
   - Select the object p' that is nearest to p, add p' to S_p, and set p' to p.

Let N (resp. n) denote the number of objects in the dataset (resp. pivot candidates), the cost is at most $O(N \cdot n)$.

3.5 Indexing Cost

We estimate the indexing cost by using the number of distance computations between objects. The indexing cost consists of the pivot selection cost and the PC calculation cost. The pivot selection needs to be done once for an index. On the other hand, the PC calculation needs to be done for every partition.

As shown in Sect. 3.4, the pivot candidate selection requires $O(N \cdot n)$ distance computations.

The required computations for a partition at a node are as follows. For an internal node v, let S_v denote the set of objects managed by the subtree rooted at v, and T_v denote the randomly sampled m objects from S_v. As shown in Sect. 3.1, two distributions are needed for measuring the PC. The distance from each object in T_v to its kth nearest neighbor object in S_v and the distances from each pivot candidate in S_p to the objects in T_v are needed for measuring the distributions at a node. The former distance requires at most $O(|S_v| \cdot |T_v|)$ computations. The latter requires $O(|T_v| \cdot |S_p|)$ computations. The total cost for the partitions $\mathrm{Cost}_{\text{partition}}$ is

$$\mathrm{Cost}_{\text{partition}} = \textstyle\sum_v \left(|S_v| + |S_p|\right) \cdot |T_v| . \quad (14)$$

The indexing cost depends on the data distribution and the sampling parameters. The maximum number of nodes is $O(N)$. Thus, the indexing cost is at most $O(N^2)$.

3.6 k-Nearest Neighbor Search

For the k-Nearest Neighbor searching, the PCTree receives a query q and the number k of results. The search procedure is almost the same as for VPT [24] and uses the following steps:

1. create an empty set as a result set S_r and set the query range r_q to ∞.
2. traverse from the root node in the PCTree and recursively accesses the nodes in the depth-first way:
   a. if the node is a leaf,
      i. find the objects whose distances to q are within r_q in the node while using the object-pivot distance constraint [26], and add them to S_r.
      ii. update r_q to be the k-nearest neighbor radius of q in S_r.
   b. otherwise
      i. read the pivot p and its partitioning distance r_p associated to the node.

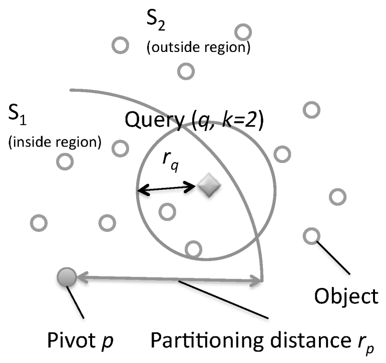


Fig. 2 k-Nearest neighbor search.

ii. if the inequality $d(p, q) \le r_p$ is satisfied,

A. access the child node for the inside region and update r_q.

B. after this, if the inequality $d(p, q) + r_q > r_p$ is satisfied, access the other child node and update r_q. Figure 2 shows a case where the inequalities $d(p, q) \le r_p$ and $d(p, q) + r_q > r_p$ are satisfied.

iii. if the inequality $d(p, q) \ge r_p$ is satisfied,

A. access the child node for the outside region and update r_q.

B. if the inequality $d(p, q) - r_q \le r_p$ is satisfied, access the other child node and update r_q.

3. answer the k closest objects.

When the balancing and the pruning of the index work best, the search cost is $O(\log N)$.

4. Performance Evaluation

4.1 Outline of Experiments

We implemented PCTree on the Metric Space Library [3]. The library is written in C. It provides several indexing algorithms, metric spaces, and datasets. We compared PCTree with GHT [23], MVP [6], [24], LC [9], and SAT [20], which are also in the library. As in the related work [20], the performance of indexes was evaluated by the number of the distance computations divided by the total number of objects in the datasets. We conducted the experiment on a Linux PC equipped with an Intel(R) Quad Core Xeon(TM) X5492 3.40 GHz CPU and that had 64 GB of memory. The library and our codes were compiled with GCC 4.2. All the indexes fit in the memory, as in the related study [20].

We used two types of datasets to evaluate our scheme: vectors and strings. The vector datasets consisted of synthetic vectors generated by us and four real vector datasets. The string dataset contained English words. We used the Euclid distance for the vector datasets and Levenshtein distance for the string dataset.

Synthetic vectors are generated in 2, 4, 8, 16, 32, and 64-dimensional feature spaces according to the uniform and Gaussian mixture distributions. Hereafter, we refer to a dataset consisting of vectors in n-dimensional space generated according to a uniform (resp. Gaussian mixture) distribution as *nd-uniform vectors* (resp. *nd-clustered vectors*). As for the nd-clustered vectors, we used 100 component Gaussian distributions in the following experiments if not explicitly mentioned. The mean vector of the component Gaussian distribution was randomly selected whereas the covariance matrixes were a diagonal matrix whose diagonal components were set to 0.02. The number of objects for each cluster was randomly chosen. The objects in the cluster were based on the Gaussian distribution. By referring to the previous study [9], we chose the size of the data to be 100,000. We randomly chose 1,000 queries from the same distribution as the dataset. When the chosen query happened to be the same point in the dataset, we discarded it. We use the average number of distance calculations over these 1,000 queries as the search cost in the following discussion.

NASA is a set of feature vectors made by NASA [3]. It consists of 40,150 vectors in a 20-dimensional feature space.

Color histograms are the color histograms of 112,544 images represented by vectors in 112-dimensional space [3].

Corel Image Features contains color histogram vectors generated from the Corel image collection [4]. It consists of 68,040 vectors in a 32-dimensional space.

English dictionary consists of 69,069 English words in the form of strings and that was generated by [3].

flickr is a set of 2,000,000 images that we downloaded from flickr. We extracted 960-dimensional features from each image by using a GIST image descriptor [21] provided by [2]. This dataset is available from [1].

For each real dataset, we randomly selected 1,000 objects for the queries and conducted the k-nearest neighbor queries on the remaining objects. All the real datasets are available from the Web.

4.2 Synthetic Data

4.2.1 Parameter Tuning

The PCTree requires three parameters during indexing. One parameter k is for calculating the PC in Eqs. (4), (5) and (6), and the other parameters s and m are for sampling as shown in Sect. 3.2 and Sect. 3.4. We evaluated how the sampling parameters affect the search cost using the 8d-uniform, 8d-clustered, and 16d-clustered vectors. We also evaluated how the sampling method affects it by using 2d, 4d, 8d, 16d, 32d, and 64d-clustered vectors.

The results are shown in Fig. 3. The vertical axes show the cost for the 1-nearest neighbor search represented by the

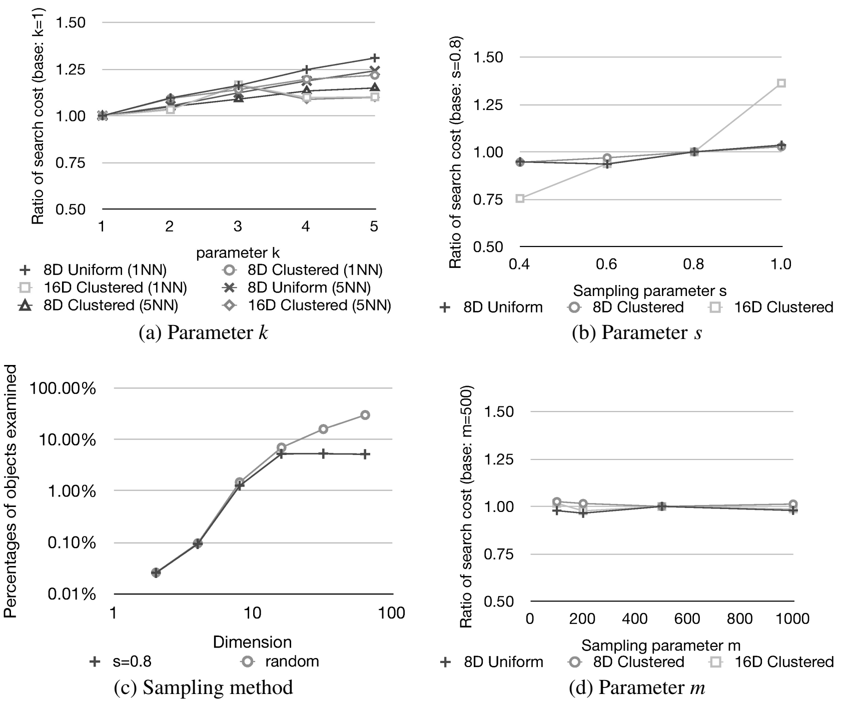


Fig. 3 Search cost w.r.t. sampling parameters and sampling methods.

relative distance computation. Figure 3 (a) includes the cost for the 5-nearest neighbors as well. The horizontal axes are the parameters. We set the search cost with *k* to be 1, *s* to be 0.8, and *m* to be 500 as the standard values, and plot the ratio of each search cost to the standard search cost.

As we can see in Fig. 3 (a), a smaller *k* is better for all the vectors. Figure 3 (b) shows the search cost w.r.t the parameter *s* that controls the candidate pivot size (see Sect. 3.4). Intuitively, a smaller *s* means more candidates are selected, and consequently, a lower query cost is achieved in the query processing. However, this requires more computations for the index construction. As we can see in the figure, *s* has the largest influence on the search cost. This indicates that more pivot candidates are better for the PC-Tree. Figure 3 (c) shows the search cost w.r.t. the sampling method. We compared our method (see Sect. 3.4) with random sampling. We can see that our sampling technique is more important for the larger dimensional vectors. Figure 3 (d) shows the search cost w.r.t the parameter *m* that determines the sample size when estimating the partitioning capacity described in Sect. 3.2. As we can see in the figure, *m* only slightly affects the search cost and 500 samples seems to be sufficient.

From these experiments, we decided to set *k*, *s*, and *m* to 1, 0.8, and 500, respectively, in the index performance evaluations.

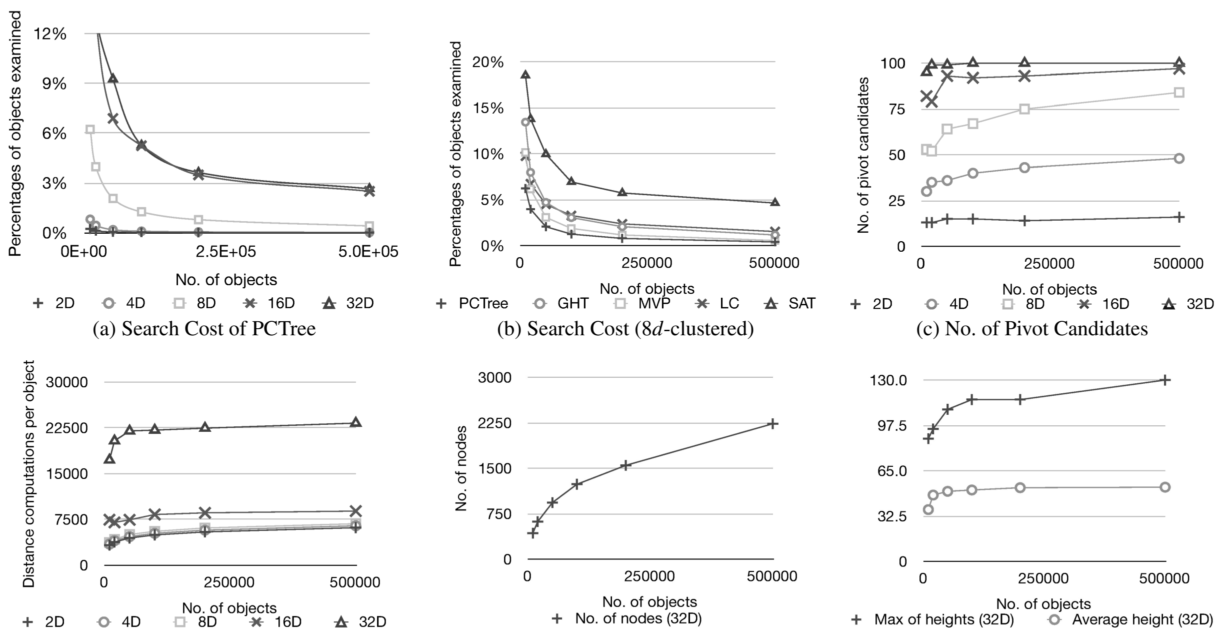
4.2.2 Dataset Size

We evaluated the index structure and the performance with respect to the number of objects by using 2*d*, 4*d*, 8*d*, 16*d*, and 32*d*-clustered vectors.

Figure 4 (a) shows the search cost of the PCTree. Figure 4 (b) compares the search cost with the 8*d*-clustered vectors. As in the related study [20], the search cost was evaluated as the percentage of objects examined. The vertical axes represent the cost for the 1-nearest neighbor queries. The horizontal axes are the number of objects.

From the results, we can see that the percentage of objects examined decreases as the dataset increases. The sublinearity of the search cost with respect to the number of objects is clear. Furthermore, we can see that the PCTree outperforms the other methods for a large amount of vectors. Figure 4 (c) shows how the dataset size affects the pivot candidate selection. The vertical axis represents the number of pivot candidates. The horizontal axis is the number of objects. We can see that the number of pivot candidates does not vary with the dataset size. The dimension has more influence on it.

Figures 4 (d), (e), and (f) show the index construction cost, the number of nodes, and the index height, respectively. The construction cost was evaluated by using the number of distance computations per object during indexing. Since we did not cache calculated distances, the distances between two objects may be calculated more than once. As we can see in the figure of the index construction cost, the number of required distance calculations is almost sublinear to the number of objects. From the figures of the index structure, we can see that both the number of nodes and index height are almost sublinear to the number of objects. We interpret this as the index tree is imbalanced for clustered vectors and the index does not increase linearly with respect to the number of objects. The index cost of the PCTree is at most $O(N^2)$ (see Sect. 3.5), and the actual cost



(a) Search Cost of PCTree (b) Search Cost (8d-clustered) (c) No. of Pivot Candidates

(d) Indexing Cost (e) No. of nodes (f) Index Tree Height

Fig. 4 Index performance w.r.t. number of objects.

is smaller than that. However, as we said, the coefficient is large and the indexing takes a long time. One reason is that all the PC calculations use the same pivot candidate set and the same sampling parameters in the PCTree construction. We have to improve the sampling and the indexing algorithm and reduce the indexing cost.

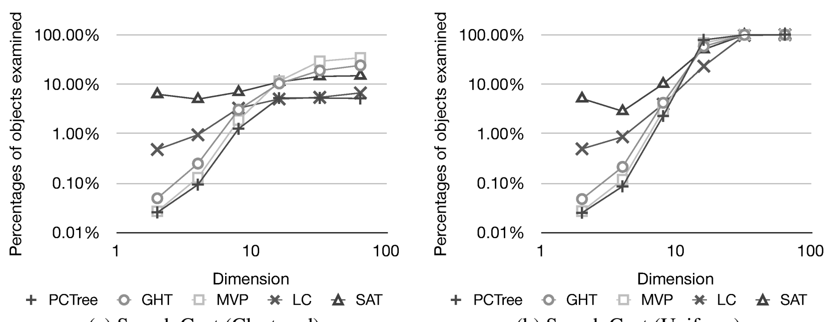
4.2.3 Dimension

We evaluated the index structure and the performance with respect to the number of dimensions. We used 2d, 4d, 8d, 16d, 32d, and 64d-clustered vectors as well as uniform vectors in each dimensional space in this experiment.

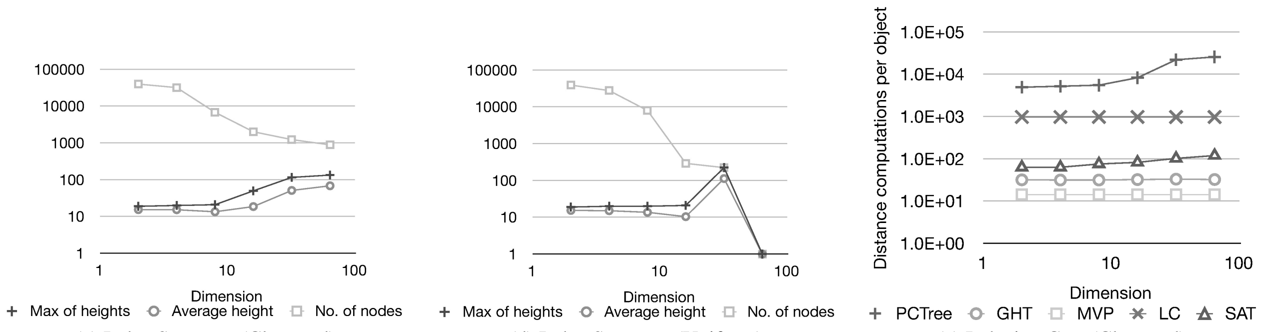
Figures 5 (a) and (b) show the search costs of the clustered and uniform vectors, respectively. The vertical axes represent the number of distance computations for the 1-nearest neighbor queries whereas the horizontal axes are the number of dimensions. We can see that PCTree, GHT, and MVP achieve good performance for lower dimensional vectors regardless of the data distribution. For higher dimensional clustered vectors, PCTree keeps outperforming the compared methods, but the performance of GHT and MVP is degraded. For higher dimensional uniform vectors, the costs of all three methods are large. In contrast, that of LC is large for lower dimensional vectors and is small for higher dimensional vectors. The PCTree is superior to the other methods for various dimensional vectors, except for the uniform vectors in a high dimensional space whereby the performance of the PCTree is almost the same as those of the other methods.

Figure 5 (c) (resp. Fig. 5 (d)) show the number of nodes and height of the index for clustered (resp. uniform) vectors. The horizontal axes are the numbers of dimensions. The difference between the maximum tree height and average tree height of the lower dimensional vectors is smaller. That is, the index trees of the lower dimensional vectors are well-balanced binary trees. On the other hand, the tree is likely to be imbalanced for high dimensional vectors. The small number of nodes relative to the tree height also shows the imbalance of the tree for high dimensional vectors. We interpret this as the PCTree changes the weights of the pruning and balancing according to the data distribution. For 64d-uniform vectors, no partition was done by the PCTree because the PC in the first partition was under the minimum PC and the PCTree judged there was no effective partition for the dataset. Compared with the other methods, the PCTree for the low dimensional vectors is similar to MVP as MVP is a completely balanced tree. The PCTree for the high dimensional vectors is close to LC as LC is a list and is the most imbalanced index. GHT doesn't use the partitioning distance and its balance depends on only its pivots. Thus, GHT cannot adjust the index balance and is fundamentally different from the PCTree. SAT is also different from the PCTree because the fan out of SAT tends to increase for the high dimensional vectors and consequently its height decreases.

Figure 5 (e) shows the index construction cost of the clustered vectors. It compared with other indexes. The vertical axis is the number of distance computations per object during indexing. For all the dimensional vectors, we can see that the PCTree require the largest indexing cost in the indexes. The indexing cost of MVP is the lowest. As shown in Sect. 3.5, the calculation of the PC causes a lot of distance computations. The cost of the PCTree is larger for a higher dimensional vectors. This is because the number of pivot candidates is larger for a higher dimensional vectors as we

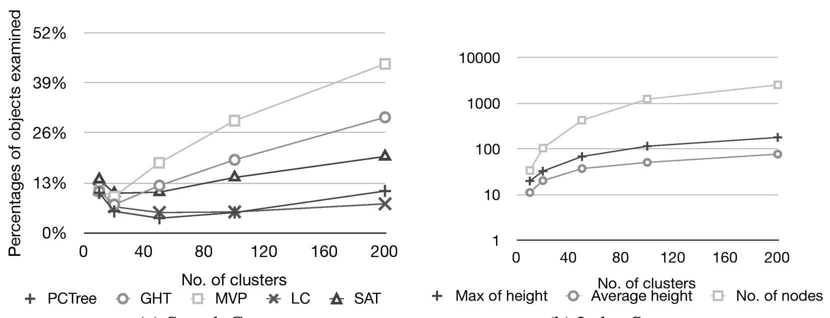


(a) Search Cost (Clustered) (b) Search Cost (Uniform)



(c) Index Structure (Clustered) (d) Index Structure (Uniform) (e) Indexing Cost (Clustered)

Fig. 5 Index performance w.r.t. number of dimensions.



(a) Search Cost (b) Index Structure

Fig. 6 Index performance w.r.t. the number of clusters.

see at Fig. 4 (c).

4.2.4 Number of Clusters

We evaluated the index structure and the performance with respect to the number of clusters. We used $32d$-clustered vectors generated by the mixture distributions with 10 to 200 Gaussian models whose variance was 0.02.

Figure 6 (a) shows the search cost. The vertical axis represents the number of distance computations for the 1-nearest neighbor queries. The horizontal axis is the number of clusters. The PCTree achieved the optimal performance for 50 clusters. This means that the margins between clusters would be large enough until the number of clusters reached 50, and the search cost is smaller for the vectors with more clusters. The PCTree could divide the space for the vectors with less than 50 clusters, and it could correctly access the small number of regions that were relevant to the query. However, the PCTree could not find good partitions for pruning for the vectors with more than 50 clusters because many clusters overlapped each other. It seems that the distribution of the vectors with 200 clusters is similar to that of the uniform vectors. Note that the PCTree outperforms the other methods for a wide range of cluster numbers except for 200. Even in this case, it still outperformed all the other methods except for LC. Figure 6 (b) shows the index structure. We can see the index tree height and the number of nodes increase with respect to the number of clusters. This means that the PCTree is likely to find more partitions that satisfy the partition condition defined by the minimum PC for data consisting of more clusters.

4.3 Real Data

We compared the PCTree with other methods for five real datasets. Figure 8 shows the index performances for the real dataset. The vertical axis represents the number of distance computations for the k-nearest neighbor queries where

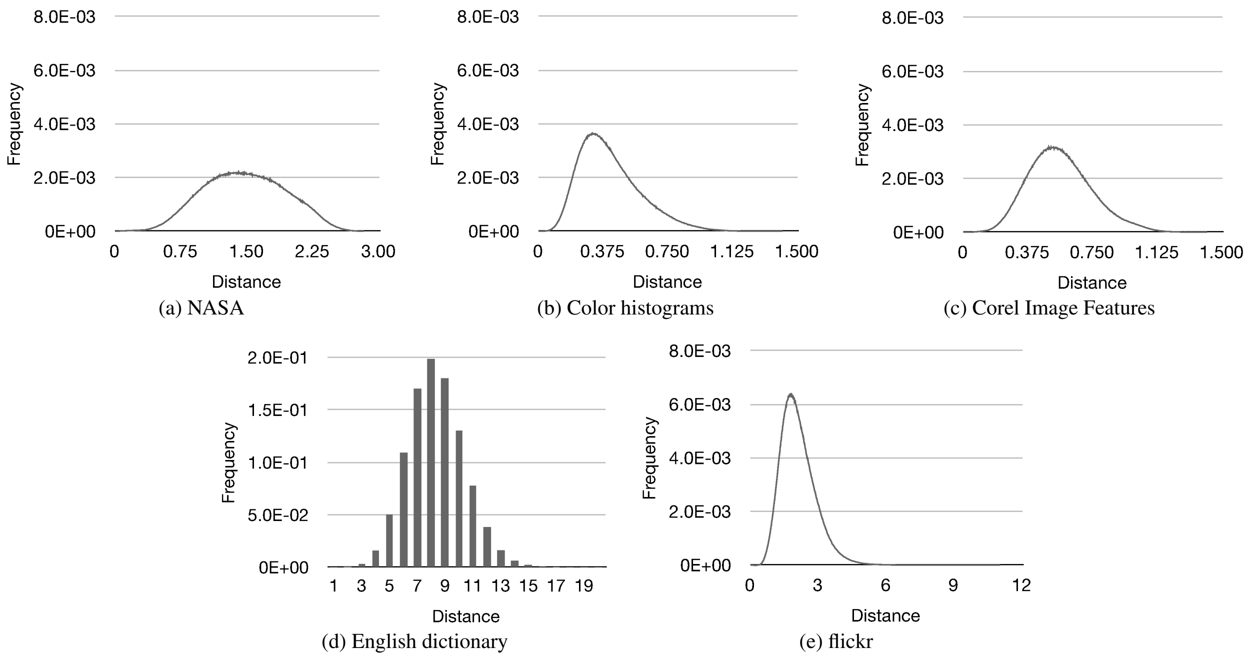


Fig. 7 Distance densities.

Table 1 Real dataset.

	Nasa	Color histograms	Corel Image Features	English dictionary	flickr
Distance	Euclid	Euclid	Euclid	Levenshtein	Euclid
Dimension	20	112	32	-	960
Average	1.48	0.415	0.564	8.35	2.12
Variance	0.211	0.0310	0.0332	4.10	0.591
Skewness	0.0447	0.828	0.444	0.271	0.972
Kurtosis	2.39	3.57	3.08	3.17	5.02

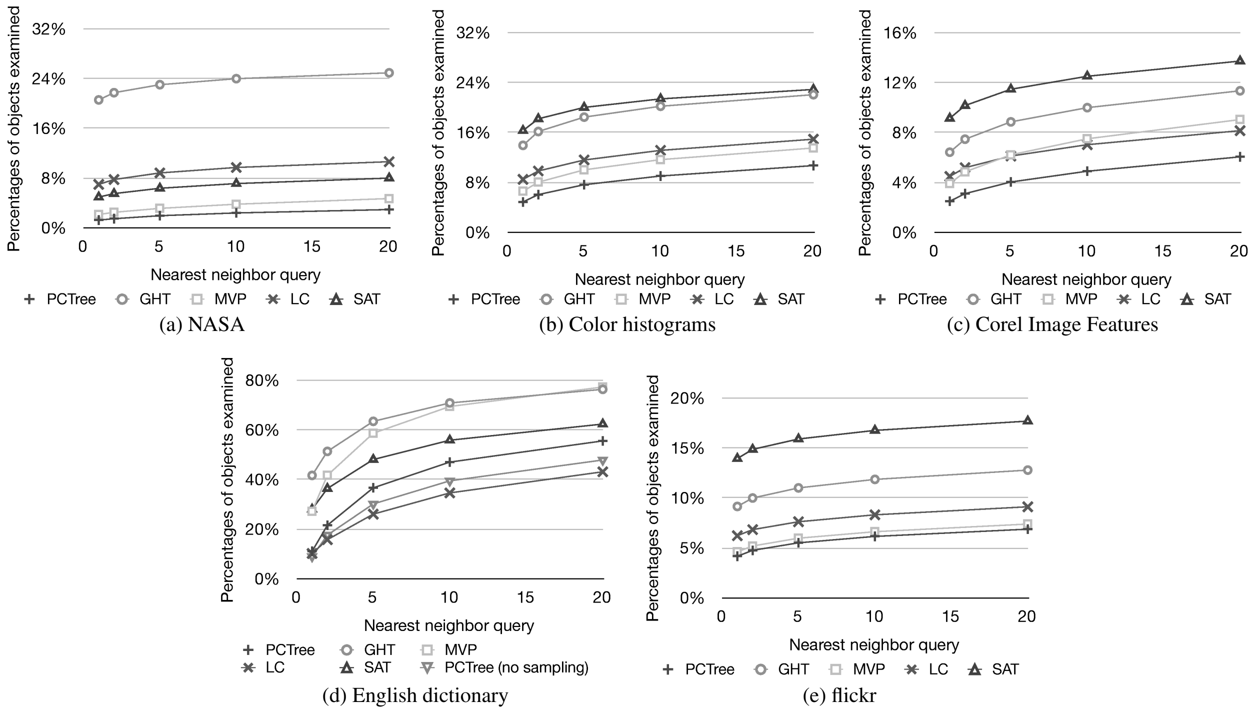


Fig. 8 Index performance on real datasets.

k ranges from 1 to 20. The horizontal axis is k.

Figure 7 shows the distance density of each dataset and Table 1 lists the properties of the distances between the objects in each dataset. We can see the following points in the figure and table.

- Only the English dictionary uses the Levenshtein distance whereas the others use the Euclid distance.
- NASA is in the lowest dimensional feature space and flickr is in the highest dimensional space.
- The skewness of the Color histograms and flickr is high.
- The kurtosis of flickr is the highest.
- NASA has a wide range of distances and its skewness is the lowest.

The PCTree in Fig. 8 outperforms the other methods for the four vector datasets whereas it outperforms the others except for LC for the English dictionary dataset. Compared with the other methods, the PCTree is better than the other methods for a wide range of data distributions. For example, MVP is good for NASA and flickr, but is weak for the English dictionary. On the other hand, LC is good for the English dictionary, but is weak for NASA. GHT and SAT are not good for any of the datasets. The flickr results show that all the indexes find similar objects by examining less than 20% of the objects in the dataset. From the flickr results and Sect. 4.2.3, we interpret that the metric space indexes can handle a high dimensional dataset if some of its dimensions are meaningless. For concluding the relationship between the distribution and the index performance, we need more experiments and have to more deeply analyze them and this will be our future work.

We guessed that the PCTree would need more samples for calculating the PC for English dictionary. Therefore, we plotted the search costs for the PCTrees and without sampling in Fig. 8 (d). The figure shows that the search cost without sampling is close to that of the LC. It is difficult to determine the appropriate parameters without more knowledge about the data distribution. The parameter tuning of the PCTree remains a future topic of study.

5. Conclusion

We presented a similarity search index named PCTree. PCTree is based on maximizing both the pruning and balance. We defined the Partitioning Capacity (PC) for selecting a pivot and its partitioning in PCTree. By using the PC, PCTree automatically optimizes the index structure according to the data distribution and reduces the search cost when using the PC.

We are currently attempting to improve the sampling scheme for pivot candidates. Having more pivot candidates can help to reduce the search cost. However, as the number of candidates increases, the indexing cost also increases. We have to reduce the indexing cost of PCTree before we can use it in practical situations.

References

[1] flickr dataset, http://www.adl.nii.ac.jp/file/flickr/flickrGIST2001000.ascii.gz

[2] Lear's gist implementation, http://lear.inrialpes.fr/software

[3] Metric spaces library, http://www.sisap.org/metric_space_library.html

[4] Uci kdd archive, http://kdd.ics.uci.edu/

[5] G.M. Adel'son-Vel'skii and E.M. Landis, "An algorithm for the organization of information," Soviet Mathematics Doklady, vol.3, pp.1259–1262, 1962.

[6] T. Bozkaya and Z.M. Ozsoyoglu, "Indexing large metric spaces for similarity search queries," ACM Trans. Database Syst., vol.24, no.3, pp.361–404, 1999.

[7] B. Bustos, G. Navarro, and E. Chevez, "Pivot selection techniques for proximity searching in metric spaces," Pattern Recognit. Lett., vol.24, no.14, pp.2357–2366, 2003.

[8] E. Chevez, J.L. Marroguin, and G. Navarro, "Fixed queries array: A fast and economical data structure for proximity searching," Multimedia Tools Applications, vol.14, no.2, pp.113–135, 2001.

[9] E. Chevez and G. Navarro, "A compact space decomposition for effective metric indexing," Pattern Recognit. Lett., vol.24, no.9, pp.1363–1376, 2005.

[10] P. Ciaccia, M. Patella, and P. Zezula, "M-tree: An efficient access method for similarity search in metric spaces," VLDB, 1997.

[11] V. Dohnal, C. Gennaro, P. Savino, and P. Zezula, "D-index: Distance searching index for metric data sets," Multimedia Tools and Applications, vol.21, no.1, pp.9–33, 2003.

[12] J. Hays and A.A. Efros, "Scene completion using millions of photographs," SIGGRAPH, 2007.

[13] H.V. Jagadish, B.C. Ooi, K.L. Tran, C. Yu, and R. Zhang, "idistance: An adaptive b+-tree based indexing method for nearest neighbor search," ACM Trans. Database Syst., vol.30, no.2, pp.364–397, 2003.

[14] G.A. Jones and J.M. Jones, Information and Coding Theory, Springer-Verlag, 2000.

[15] C.T. Jr, R.F. Filho, A.J. Traina, M.R. Vieira, and C. Faloutsos, "The omni-family of all-purpose access methods: A simple and effective way to make similarity search more efficient," The VLDB Journal, vol.16, no.4, pp.483–505, 2007.

[16] C.T., Jr, A. Traina, B. Seeger, and C. Faloutsos, "Slim-trees: High performance metric trees minimizing overlap between nodes," EDBT, 2000.

[17] H. Kurasawa, D. Fukagawa, A. Takasu, and J. Adachi, "Maximal metric margin partitioning for similarity search indexes," CIKM, 2009.

[18] H. Kurasawa, D. Fukagawa, A. Takasu, and J. Adachi, "Pivot selection method for optimizing both pruning and balancing in metric space indexes," DEXA, 2010.

[19] M. Marin, G.V. Costa, and R. Uribe, "Hybrid index for metric space databases," ICCS, 2008.

[20] G. Navarro, "Searching in metric spaces by spatial approximation," The VLDB Journal, vol.11, no.1, pp.28–46, 2002.

[21] A. Oliva and A. Torralba, "Modeling the shape of the scene: A holistic representation of the spatial envelope," Int. J. Comput. Vis., vol.42, no.3, pp.145–175, 2001.

[22] O. Pedreira and N.R. Brisaboa, "Spatial selection of sparse pivots for similarity search in metric spaces," SOFSEM, 2007.

[23] J.K. Uhlmann, "Satisfying general proximity/similarity queries with metric trees," Inf. Process. Lett., vol.40, no.4, pp.175–179, 1991.

[24] P.N. Yianilos, "Data structures and algorithms for nearest neighbor search in general metric spaces," SODA, 1993.

[25] P.N. Yianilos, "Excluded middle vantage point forests for nearest neighbor search," ALENEX, 1999.

[26] P. Zezula, G. Amato, V. Dohnal, and M. Batko, Similarity Search: The Metric Space Approach (Advances in Database Systems),

Springer-Verlag, 2005.

[27] Y. Zhuang, Y. Zhuang, Q. Li, L. Chen, and Y. Yu, "Indexing high-dimensional data in dual distance spaces: A symmetrical encoding approach," EDBT, 2008.

Hisashi Kurasawa received the B.E. and M.E. in Information Science and Technology from the University of Tokyo, Tokyo, Japan in 2006 and 2008. He is currently a Ph.D. candidate in the Graduate School of Information Science and Technology at the University of Tokyo. His research interests are distributed information retrieval systems and context-aware computing. He is a student member of IPSJ and DBSJ.

Daiji Fukagawa received the B.E. degree in Computer Science in 2001, M.S. degree in Informatics in 2003, and Ph.D. degree in Informatics in 2006, all from Kyoto University. Currently, he is an Assistant Professor of Doshisha University, Japan. His research interests include the theory of combinatorial optimization for trees and graphs. He is a member of IPSJ, DBSJ, and JSAI.

Atsuhiro Takasu received B.E., M.E. and Dr. Eng. from the University of Tokyo in 1984, 1986 and 1989, respectively. He is a professor at the National Institute of Informatics, Japan. His research interests are database systems and machine learning. He is a member of ACM, IEEE, IPSJ, DBSJ and JSAI.

Jun Adachi is a Professor in the Digital Content and Media Sciences Research Division, National Institute of Informatics (NII), Japan. He is also the Director of the Cyber Science Infrastructure Development Department of NII. His professional career has largely been spent in research and development of scholarly information systems, such as NACSIS-CAT and NII-ELS. He is also an adjunct professor of the Graduate School of Information Science and Technology (Department of Information and Communication Engineering), University of Tokyo. His research interests are information retrieval, text mining, digital library systems, and distributed information systems. Adachi received his B.E., M.E. and Doctor of Engineering in Electrical Engineering from the University of Tokyo in 1976, 1978, and 1981, respectively. He is a member of IPSJ, IEEE, and ACM.