IEICE TRANS. INF. & SYST., VOL.E100–D, NO.10 OCTOBER 2017

PAPER

Efficient Similarity Search with a Pivot-Based Complete Binary Tree

Yuki YAMAGISHI[†a)], ***Nonmember***, **Kazuo AOYAMA**[††], **Kazumi SAITO**[†], ***and*** **Tetsuo IKEDA**[†], ***Members***

SUMMARY This paper presents an efficient similarity search method utilizing as an index a complete binary tree (CBT) based on optimized pivots for a large-scale and high-dimensional data set. A similarity search method, in general, requires high-speed performance on both index construction off-line and similarity search itself online. To fulfill the requirement, we introduce novel techniques into an index construction and a similarity search algorithm in the proposed method for a range query. The index construction algorithm recursively employs the following two main functions, resulting in a CBT index. One is a pivot generation function that obtains one effective pivot at each node by efficiently maximizing a defined objective function. The other is a node bisection function that partitions a set of objects at a node into two almost equal-sized subsets based on the optimized pivot. The similarity search algorithm employs a three-stage process that narrows down candidate objects within a given range by pruning unnecessary branches and filtering objects in each stage. Experimental results on one million real image data set with high dimensionality demonstrate that the proposed method finds an exact solution for a range query at around one-quarter to half of the computational cost of one of the state-of-the-art methods, by using a CBT index constructed off-line at a reasonable computational cost.

key words: *algorithm, similarity search, index, tree, pivot*

1. Introduction

Efficient similarity search has attracted a lot of attentions as a fundamental tool in various scientific fields [1]. Developing similarity search methods has become a more important challenge along with the increasing volume of data which we deal with [2], [3]. From a perspective of resultant search accuracy, similarity search methods are classified into three main categories of exact, approximate, and heuristic search. Lately, to solve similarity search problems for large-scale data sets, approximate search methods that guarantee some resultant accuracy have been studied with considerable effort, which contain those based on a tree-type index [4] and locality-sensitive hashing (LSH) family [5]–[7]. In contrast, exact methods have received interest for a long time in the application domains where a data set has relatively low intrinsic dimensionality [3]. Their acceleration is also required for the accuracy evaluation of the approximate and the heuristic methods since an exact search result is necessary as the ground truth.

Manuscript received March 22, 2017.
Manuscript revised June 6, 2017.
Manuscript publicized July 4, 2017.

[†]The authors are with University of Shizuoka, Shizuoka-shi, 422–8526 Japan.
[††]The author is with NTT Communication Science Laboratories, Kyoto-fu, 619–0237 Japan.
a) E-mail: yamagissy@gmail.com

DOI: 10.1587/transinf.2017EDP7100

We consider the acceleration of the exact similarity search methods for a range query as a subject, which is the most basic type of queries. A popular acceleration technique for improving a naïve exact search algorithm such as a linear-scan (brute-force) algorithm is to exploit an index built off-line from a given data set. A search method with the index saves distance calculations between a given query object and data objects based on the *triangle inequality* in a metric space. Most of these employ distinguished objects or fixed reference points called *pivots*. The pivots are used in various types of indexes such as a simple pivot-object distance table [8] or tree data structures [9]–[13]. A tree-based method, which has been widely used, utilizes the pivots as centers for data clustering or split points for data partitioning, and as base points in the triangle inequality for filtering out data objects.

We focus on a search method with a tree-type index employing pivots. In a tree-index construction phase, a pivot selection and a partition method are important for a better data structure. Pivots for a general metric space are determined from data objects by simply choosing them at random [9], [13], selecting them according to a rule [10]–[12], [14], [15], and maximizing a defined objective function [11], [16]. Of these, the incremental selection in [11], [16] sequentially adopts an object as a pivot one by one. The others select a set of pivots at once in the initial stage and form Voronoi diagrams (or Dirichlet domains). In a search phase, those methods basically proceed in a depth-first way, pruning branches and filtering objects out of a given range measured from a query object with various techniques [2].

Suppose that a subject space is not a general metric space but a Euclidean space. That is, an object is represented by a feature vector. We often address a search problem in this situation, for instance, where a given data set consists of real images and documents. Then a better pivot can be generated at any point in a Euclidean space by optimization. In order to maximize the effects of such generated pivots, we challenge to rebuild a similarity search method.

In this paper, we present an efficient exact similarity search method for a large-scale and high-dimensional data set in a Euclidean space. The proposed method consists of an index construction and a similarity search algorithm. The index construction algorithm recursively employs two main functions of a pivot generation and a node bisection function at a node. The algorithm makes a complete binary tree (CBT) grow level by level until the level reaches

Table 1 Notation

Symbol	Description and Definitions
N	**Number** of data objects, $N=\|\mathcal{X}\|$
r	**Range** for range query
H	**Dimensionality** of feature vector of object
L	Maximum **level** of complete binary tree index
$\tilde{L}(r;\mathcal{X})$	CBT **level** estimated for early termination in search with r
U	**Number** of objects in unpruned nodes
Y_i	**Number** of unbounded objects at level i
i^*	**Level** at which Y_i is minimum
$\mathcal{Z}$	**Set** of objects; $\{q\}$, $\mathcal{X} \subset \mathcal{Z}$
$\mathcal{X}$	**Set** of given data objects, $x_h \in \mathcal{X}$, $h=1,2,\cdots,N$
$\mathcal{Q}$	**Set** of Query objects for search, $q \in \mathcal{Q}$
$\mathcal{P}$	**Set** of pivots, $p_j \in \mathcal{P}$
$\mathcal{T}$	Complete binary **tree** with level L
v_j	**Node** of CBT at level i, $2^{(i-1)} \le j < 2^i$
$\mathcal{X}_j$	**Set** of objects included in v_j, $N_j=\|\mathcal{X}_j\|$
$\mathcal{B}_j$	**Range** of distance from p_j to region within r from q, $\lceil \max\{0, d(q,p_j)-r\}, d(q,p_j)+r \rceil$
$\mathcal{R}_{2j}$ (or $\mathcal{R}_{(2j+1)}$)	**Range** of distances of objects in v_{2j} (or $v_{(2j+1)}$) measured from its parent pivot p_j
$\tilde{\mathcal{V}}_j$	**Set** of unpruned child nodes at node v_j
$\mathcal{V}_i$	**Set** of unpruned nodes at level i
$\mathcal{S}_i$	**Set** of unbounded objects at level i
$\mathcal{S}$	**Set** of unbounded objects at level i^*
$\mathcal{W}$	**Set** of unfiltered objects, where objects' exact distances to q are finally calculated
$\mathcal{X}(q;r)$	**Set** of objects within r from q: Search result
$d(x_h, p_j)$	**Function** that returns a distance between object x_h and pivot p_j
$D(x_h, q; \mathcal{P})$	**Function** that returns a lower-bound distance between object x_h and query q using pivots $\mathcal{P}$
$F(\mathcal{P};\mathcal{X})$	**Function** that returns sum of lower-bound distances of objects in $\mathcal{X}$ by $\mathcal{P}$
$PGM\ (\mathcal{X}_j)$	**Function** that returns optimized pivot p_j for $\mathcal{X}_j$ by maximizing $F(\{p_j\};\mathcal{X}_j)$
$NBS\ (v_j)$	**Function** that returns a pair of subsets of $\mathcal{X}_j$, that is, $\mathcal{X}_{2j}$ and $\mathcal{X}_{(2j+1)}$
$\delta_j(h)$	**Function** that returns the h-th smallest object-pivot distance in v_j
$CF_j(\delta)$	**Function** that returns the number of objects in v_j with the distance from pivot p_j not larger than δ
$C(q, r; \mathcal{X}, L)$	**Function** that returns the search cost for query q and range r under given $\mathcal{X}$ and L
$AC(L)$	**Function** that returns the average search cost for level L

a predefined maximum level. The pivot generation function obtains an effective pivot by efficiently maximizing an objective function with a novel technique, where the objective function is similar to that in [16] and has been already used in a Euclidean space as described in [17], [18]. The novel technique achieves the less computational complexity of $O(N \log N)$ for N, where N denotes the number of data objects. The node bisection function partitions objects in a node into two subsets with almost the same size, by using a *median* point based on a distance from the optimized pivot. The search algorithm exploits a three-stage process that narrows down candidate objects within a range measured from a query object by pruning unnecessary branches and filtering objects in the stages. Besides, we newly introduce an early termination scheme that determines an appropriate search level in the CBT index built off-line before the search. These algorithms with novel techniques allow the proposed method to efficiently perform exact similarity search.

We demonstrate in experimental results on a data set of one million real images with high dimensionality in the CoPhIR test-collection [19] that the proposed method performs exact similarity search for a range query at around one-quarter to half of the computational costs of a state-of-the-art method called M-Index [13], by using a CBT index constructed in a reasonable computational cost.

The remainder of this paper is organized as follows. Section 2 briefly reviews related work. Section 3 provides background knowledge for understanding our proposed method. Section 4 explains the proposed method in detail. Section 5 shows the experimental settings and reports our results. The final section offers our conclusions. We also summarize the notation in Table 1 for easy reference.

2. Related Work

This section reviews two related topics: pivot generation and data partitioning algorithms for index construction, and search methods with a pivot-based tree index.

2.1 Pivot Generation and Data Partitioning Algorithms

An advantage of the index construction algorithm in our proposed method is a fast pivot optimization technique. The existing pivot selection and generation algorithms are stated as regards their objective functions and computational costs.

The objective functions have been designed depending on their usage and hypotheses [11], [20]. A hypothesis is that better pivots are far away from each other [8], [10]. A simple algorithm based on the hypothesis sequentially selects pivot $\hat{p}$ from a set of data objects $\mathcal{X}$ one by one with the same way as ϵ-net construction in [15], [21], as expressed by

$$\hat{p} = \arg\max_{x_h \in \mathcal{X}\setminus\mathcal{P}} \{\min_{p\in\mathcal{P}} \{d(x_h, p)\}\} \quad \text{and} \quad \mathcal{P} = \mathcal{P} \cup \{\hat{p}\}\,. \tag{1}$$

where $\mathcal{P}$ denotes a set of the selected pivots and $d(x_h, p)$ a distance between x_h and p. Another sequential algorithm selects pivot $\hat{p}$ in the similar manner to that in Eq. (1) as follows [8].

$$\hat{p} = \arg\max_{x_h \in \mathcal{X}\setminus\mathcal{P}} \{ \textstyle\sum_{p\in\mathcal{P}} d(x_h, p) \}\,. \tag{2}$$

The following algorithm is based on another hypothesis of a query object appears with the identical distribution to that of the data objects. The algorithm incrementally selects pivot $\hat{p}$ that maximizes the sum of the lower bounds on distances between data objects [16] as shown in Eqs. (3)

and (4). The distance lower bound is derived by the *triangle inequality* applied to the triangle with the pivot and two objects.

$$\hat{p} = \arg\max_{p \in \mathcal{X} \setminus \mathcal{P}} \{ \textstyle\sum_{x,y \in \mathcal{X} \setminus (\mathcal{P} \cup \{p\})} D(x, y; \mathcal{P} \cup \{p\}) \} , \quad (3)$$

$$D(x, y; \mathcal{P} \cup \{p\}) = \max_{p' \in \mathcal{P} \cup \{p\}} \left(| \, d(x, p') - d(y, p') \, | \right) , \quad (4)$$

where $\mathcal{P} = \emptyset$ at the initial state, and after $\hat{p}$ is obtained by Eq. (3), $\mathcal{P}$ is updated as $\mathcal{P} \leftarrow \mathcal{P} \cup \{\hat{p}\}$.

Since this objective function is valid for some applications, it has been utilized in a Euclidean space [17], [18], where a pivot is not selected from the data objects but generated as a point in the Euclidean space. Algorithms with objective function $F(\mathcal{P}; \mathcal{X})$ in Eq. (5) similar to that in Eq. (3) generate a set of pivots $\mathcal{P}$ with some iterative methods for a Euclidean distance (L_2 norm) [17] and for a Manhattan distance (L_1 norm) [18]. These algorithms are collectively referred to as *PGM* (Pivot Generation by Maximizing the objective function).

$$F(\mathcal{P}; \mathcal{X}) = \textstyle\sum_{i=1}^{N-1} \sum_{j=i+1}^{N} D(x_i, x_j; \mathcal{P}) , \quad (5)$$

$$D(x_i, x_j; \mathcal{P}) = \max_{p \in \mathcal{P}} \left(| \, d(x_i, p) - d(x_j, p) \, | \right) . \quad (6)$$

However, the foregoing algorithm requires a high computational cost of $O(N^2)$ for each iteration. This requirement makes it difficult to apply the algorithm to a large-scale data set. Thus to deal with a large-scale data set, we have to improve a pivot generation algorithm with respect to a computational cost.

Next, we briefly overview data partitioning algorithms for constructing a tree index. We, in particular, pick up four [9], [10], [13], [14] from many such algorithms and compare them in terms of a pivot-selection stage, a partitioning criterion, and the number of partitions.

A metric tree with ball partitioning (decomposition) [9] randomly chooses a pivot from objects at each node, and partitions a set of the other objects into two subsets with the criterion of whether an object is inside a median object based on a distance measured from the pivot or not. The metric tree recursively performs this procedure. A simple vantage point tree (vp-tree) [14] uses the same way as the metric tree except a pivot selection algorithm. Both the algorithms commonly build a complete binary tree by the recursive bisection at a median object.

By contrast, a GNAT (geometric near-neighbor access tree) [10] differs from the foregoing algorithms in a partitioning criterion and the number of partitions (or branches). This exploits $|\mathcal{P}|$ pivots and assigns each object to the closest pivot, i.e., builds a hierarchical Dirichlet domain. Like the GNAT, M-Index [13] assigns each object to a pivot closest to it, but differs in its pivot-selection stage, i.e., chooses a set of pivots at the initial stage at once and recursively assigns an object to one of its unassigned pivots. These algorithms may build an unbalanced tree with the different number of branches. Our index construction algorithm adopts the same strategy as the metric tree and the vp-tree.

2.2 Search Methods with a Pivot-Based Tree Index

Similarity search methods with a tree index constructed by using pivots reduce a search cost with various techniques [2] that avoid calculating costly exact distances between a given query object and data objects. Most methods exploit the *triangle inequality* that is applied to the triangle of pivot p, query object q, and data object x.

The techniques for a range query are categorized into two main types. One is to prune branches connected to child nodes in which all the objects are out of range r from query object q, i.e., to delete the child nodes and their descendants *at once* from candidate objects for exact distance calculations to the query. In a simple case of $d(p, x_{min}) < r$ where x_{min} denotes an object closest to pivot p in a node, a node deleted by using p satisfies $d(p, x_{max}) < d(p, q) - r$ where x_{max} denotes an object farthest from p. The other technique is to filter out an object *individually*, whose lower bound on a distance to q is larger than range r. Given a set of pivots $\mathcal{P}$, the filtering object x satisfies $D(q, x; \mathcal{P}) > r$.

Each of the four search methods described in Sect. 2.1 traverses its tree index with a depth-first manner from the root to a leaf node. In the traverse, the methods determine whether to visit child nodes from the current node or not to do, based on the above-mentioned node pruning condition. M-Index [13] performs to filter out an object individually according to the foregoing condition in addition to pruning unnecessary branches. Our search algorithm also employs the two main techniques for high search efficiency.

3. Preliminaries: Similarity Search

Let us consider an object space consisting of a set of objects denoted by $\mathcal{Z}$ and distance $d(x_i, x_j)$ that is defined for any pair of objects $x_i, x_j \in \mathcal{Z}$. Let $\mathcal{X}$ and q denote a set of given data objects and a query object, respectively. Then $\mathcal{X} = \{x_1, x_2, \cdots, x_h, \cdots, x_N\} \subset \mathcal{Z}$, and $q \in \mathcal{Z}$. A range query that we deal with is defined as a task of finding a set $\mathcal{X}(q; r)$ of all the objects x_h satisfying $d(x_h, q) \le r$, i.e., $\mathcal{X}(q; r) = \{x_h \in \mathcal{X} \mid d(q, x_h) \le r\}$ when $\mathcal{X}$, q, and a range distance r measured from q are given.

Figure 1 shows a diagram of a complete binary tree (CBT). We represent a depth (or height) of the CBT as a *level*, and set the root level and the leaf level (maximum level) at 1 and L, respectively. Then a node at level i ($1 \le i \le$

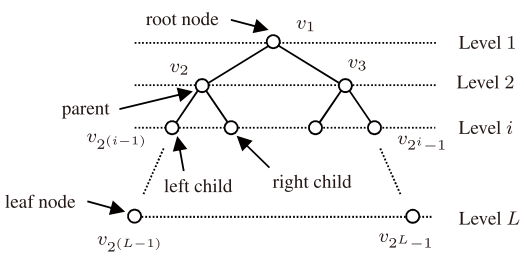


Fig. 1 Diagram of complete binary tree index

L) is expressed by v_j, where $2^{(i-1)} \le j < 2^i$. That is, there are $2^{(i-1)}$ nodes at level i. If node v_j is a parent node, nodes v_{2j} and v_{2j+1} are a left and a right child node, respectively. Each node may contain information related to a subset of data objects $\mathcal{X}_j$, one pivot p_j, and object-pivot distances (or related information). Using these background, we explain our proposed method in the following section.

4. Proposed Algorithm

The proposed method consists of an index construction and a similarity search algorithm. Given a set of objects represented by feature vectors in a metric space and a predefined maximum level of a complete binary tree (CBT), the index algorithm builds the CBT as a search index *off-line* by generating pivots. When a query object and a range for search are given, i.e., a range query *online*, the similarity search algorithm efficiently finds a set of objects within the range from the query object. Below, we describe the details of these algorithms.

4.1 Index Construction Algorithm

We first show the overview of our index construction algorithm in **Algorithm 1**, which produces CBT $\mathcal{T}$ for given object set $\mathcal{X}$ and maximum level parameter L. Our proposed algorithm incrementally makes the CBT grow from the root level to the leaf level. In an initial state, object set $\mathcal{X}_1$ in the root v_1 is set up by $\mathcal{X}$ containing the whole objects, and pivot set $\mathcal{P}$ initialized by $\emptyset$. The following two steps are executed at each node v_j at level i. In the first step, pivot p_j is generated by the pivot generation function $PGM(\mathcal{X}_j)$ shown at line 6, and is added into pivot set $\mathcal{P}$. In the second step, $\mathcal{X}_j$ in node v_j is partitioned into two subsets with almost the same size, $\mathcal{X}_{2j}$ and $\mathcal{X}_{(2j+1)}$, by the node bisection function $NBS(v_j)$ shown at line 9. After the bisection of v_j at level i, these steps are applied to nodes at the next level $i+1$ while $i \le L$. We detail the *PGM* function and the *NBS* function in Sects. 4.1.1 and 4.1.2, respectively.

Algorithm 1 Index construction algorithm

```
1: Input: X, L
2: Output: T
3: X_1 ← X, P ← ∅, i ← 1                          ▷ Initialization
4: while i ≤ L do
5:     for all j such that 2^(i-1) ≤ j < 2^i do
6:         p_j ← PGM(X_j),  P ← P ∪ {p_j}
7:         v_j ← (X_j, p_j, {d(x_h, p_j) | x_h ∈ X_j})
8:         if i < L then
9:             {X_2j, X_(2j+1)} ← NBS(v_j)
10:        else
11:            NBS(v_j)
12:        end if
13:    end for
14:    i ← i + 1
15: end while
16: return T
```

4.1.1 Pivot Generation

We show our pivot generation function $PGM(\mathcal{X}_j)$ in **Algorithm 2**, which produces pivot p_j using a set of objects $\mathcal{X}_j$ in node v_j. The *PGM* function chooses object $x_h \in \mathcal{X}_j$ as the initial pivot p_j uniformly at random. Until convergence, the steps at lines 5–9 are iterated. In the step at line 7, we use a technique that efficiently computes objective function $F(\{p_j\}; \mathcal{X}_j)$ for *one* pivot p_j, which is the same function as that in Eq. (5), by using the sorted distances at line 6. The computational cost of the objective function is generally $O(N_j^2)$, where $N_j = |\mathcal{X}_j|$. The new technique, however, reduces the cost to that for sorting N_j, i.e., $O(N_j \log N_j)$. Now, we assume $h \in \{1, \cdots, N_j\}$ without loss of generality. Suppose that $d(x_{j(1)}, p_j) \le d(x_{j(2)}, p_j) \le \cdots \le d(x_{j(h)}, p_j) \le \cdots \le d(x_{j(N_j)}, p_j)$, where $\{j(1), \cdots, j(N_j)\} = \{1, \cdots, N_j\}$. Then, without using any absolute operation, we can obtain the objective function value as follows:

$$
\begin{aligned}
F(\{p_j\}; \mathcal{X}_j) &= \sum_{h=1}^{N_j-1} \sum_{h'=h+1}^{N_j} D(x_{h'}, x_h; \{p_j\}) \\
&= \sum_{h=1}^{N_j-1} \sum_{h'=h+1}^{N_j} \left| d(x_{h'}, p_j) - d(x_h, p_j) \right| \\
&= \sum_{h=1}^{N_j-1} \sum_{h'=h+1}^{N_j} \left(d(x_{j(h')}, p_j) - d(x_{j(h)}, p_j) \right) \\
&= \sum_{h=1}^{N_j} (2h - 1 - N_j) \cdot d(x_{j(h)}, p_j) \,. \qquad (7)
\end{aligned}
$$

From Eq. (7), given the sorted distances, the computational complexity of calculating the objective function itself is linear to N_j. In the steps at lines 8–9, the formula for obtaining p'_j depends on a given metric space [17], [18]. In the case of the L_1 distance employed in our experiments, we can calculate the distance of $d(x_{j(h)}, p_j)$ in Eq. (7) as the sum of distances with respect to dimensions, $\sum_{\eta=1}^{H} |x_{j(h),\eta} - p_{j,\eta}|$, where $x_{h,\eta}$ and $p_{j,\eta}$ denote the η-th elements of the H-dimensional feature vectors of x_h and p_j. The objective function is rewritten as

$$
F(\{p_j\}; \mathcal{X}_j) = \sum_{\eta=1}^{H} \sum_{h=1}^{N_j} (2h - 1 - N_j) \, |x_{j(h),\eta} - p_{j,\eta}| \,. \qquad (8)
$$

We can maximize $F(\{p_j\}; \mathcal{X}_j)$ in Eq. (8) by maximizing a piecewise linear function $f_\eta(p_{j,\eta}) = \sum_{h=1}^{N_j} (2h-1-N_j) \, |x_{j(h),\eta} - p_{j,\eta}|$ with respect to $p_{j,\eta}$. The piecewise linear function $f_\eta(p_{j,\eta})$ with at most N_j change points can be maximized by a direct search method with the computational cost of $O(N_j)$ as shown in [18]. By performing the direct search on each dimension, we can find p_j that maximizes $F(\{p_j\}; \mathcal{X}_j)$, i.e., p'_j. Thus, we update p_j to p'_j with the computational cost of $O(N_j \cdot H)$.

Next, we show the computational complexity of the pivot generation algorithm in *PGM* function. Consider that

Algorithm 2 Pivot generation function: $PGM(\mathcal{X}_j)$

1: **Input:** $\mathcal{X}_j$
2: **Output:** p_j
3: $p_j \leftarrow x_h$ ▷ Randomly chosen $x_h \in \mathcal{X}_j$
4: **while** not converge **do**
5: Calculate distance $d(x_h, p_j)$ for each $x_h \in \mathcal{X}_j$
6: $\{d(x_{j(h)}, p_j)\} \leftarrow$Sort $\{d(x_h, p_j)\}$ (ascending order)
7: Compute objective function $F(\{p_j\}; \mathcal{X}_j)$
8: Update p_j to p'_j so as to maximize $F(\{p_j\}; \mathcal{X}_j)$
9: $p_j \leftarrow p'_j$
10: **end while**
11: **return** p_j

Algorithm 3 Node bisection function: $NBS(v_j)$

1: **Input:** $v_j = (\mathcal{X}_j,\ p_j,\ \{d(x_{j(h)}, p_j) \mid x_{j(h)} \in \mathcal{X}_j\})$
2: **Output:** $\{\mathcal{X}_{2j}, \mathcal{X}_{(2j+1)}, \mathcal{R}_{2j}, \mathcal{R}_{(2j+1)}\}$ or void
3: [∗] Create look-up table of $CF_j(\delta)$ ▷ $\delta \in [\delta_j(1), \delta_j(N_j)]$
4: **if** $j < 2^{(L-1)}$ **then**
5: $\mathcal{X}_{2j} \leftarrow \{x_{j(1)}, x_{j(2)}, \cdots, x_{j(\lceil N_j/2 \rceil)}\}$
6: $\mathcal{X}_{(2j+1)} \leftarrow \{x_{j(\lceil N_j/2 \rceil+1)}, \cdots, x_{j(N_j)}\}$
7: [∗] $\mathcal{R}_{2j} \leftarrow [\delta_j(1), \delta_j(\lceil N_j/2 \rceil)]$
8: [∗] $\mathcal{R}_{(2j+1)} \leftarrow [\delta_j(\lceil N_j/2 \rceil + 1), \delta_j(N_j)]$
9: **return** $\{\mathcal{X}_{2j}, \mathcal{X}_{(2j+1)}, \mathcal{R}_{2j}, \mathcal{R}_{(2j+1)}\}$
10: **end if**

a given metric space is an H-dimensional vector space with the L_1 distance. The complexity at line 5 in **Algorithm 2** is $O(N_j \cdot H)$, and that at line 6 is $O(N_j \log N_j)$. As described before, the complexity from line 7 to line 8 is $O(N_j \cdot H)$ for each iteration. The computational complexity of lines 5–9 for one iteration is $O(N_j \cdot (H + \log N_j))$. In our experiments, the number of the iterations was *quite small* compared with N_j and H, in fact, that was less than 100 in most trials. For a convergence criterion in our experiments, we employed that $F(\{p'_j\}; \mathcal{X}_j)/F(\{p_j\}; \mathcal{X}_j) < (1 + \epsilon)$, where p'_j denotes the updated pivot, and we used $\epsilon = 10^{-8}$. For this reason, we eliminate it from the complexity although the number of iterations until the pivot generation algorithm converges may be included in the complexity. Therefore, we obtain the computational complexity of the pivot generation algorithm for p_j in v_j as follows:

$$O\left(N_j \cdot (H + \log N_j)\right).$$

4.1.2 Node Bisection

The *NBS* function in **Algorithm 3** performs the bisection based on a median point in terms of the sorted distances of objects $x_{j(h)} \in \mathcal{X}_j$ from pivot p_j in v_j, where h of the median point is $\lceil N_j/2 \rceil$. Consequently, the *NBS* function generates two subsets with almost the same size, $\mathcal{X}_{2j}$ and $\mathcal{X}_{(2j+1)}$, contained in two child nodes, v_{2j} and $v_{(2j+1)}$, respectively. This scheme is similar to the ball partitioning [9] and the vantage point tree (vp-tree) [14].

In this function, we prepare for the similarity search *online*, as shown at lines with symbol [∗] in **Algorithm 3**. To this end, for arbitrary value δ, we introduce a cumulative frequency function $CF_j(\delta)$ that returns the number of objects whose distances from pivot p_j are less than or equal to some value δ, i.e.,

$$CF_j(\delta) = |\{x_{j(h)} \in \mathcal{X}_j \mid d(x_{j(h)}, p_j) \le \delta\}|. \quad (9)$$

Note that the CF function is a monotonically increasing function whose value is only the integer in $[0, N_j]$ for any δ, i.e., by setting $\delta_j(h) = d(x_{j(h)}, p_j)$, we obtain $CF_j(\delta) = 0$ if $\delta < \delta_j(1)$ and $CF_j(\delta) = N_j$ if $\delta > \delta_j(N_j)$.

In our experiments, we treated a metric space where an object and a pivot represented by integer vectors and a distance is measured by L_1 norm, and then all the object-pivot distances are originally integer. In this case, $\delta_j(h)$ for the CF function is basically an integer. Then we implement the CF function as a look-up table from the *domain* of $[\,\delta_j(1), \delta_j(N_j)\,]$ to the *range* of $1, 2, \cdots, N_j$. We can also apply this scheme to a general metric space without loss of generality, by using a hash function that transforms all the object-pivot distances to integers so that their orders are not reversed. For instance, a simple transformation is implemented by multiplying the distances by a large integer and truncating them at the decimal point. Hereinafter, we assume that the distances are transformed to integers.

The look-up table of the CF function allows us to immediately identify objects in node v_j within a given range of distances. Suppose that general range $\mathcal{R} = [\,\delta_1, \delta_2\,]$, i.e., the *domain* of CF function, is given. Then we can extract a set of objects in node v_j within $\mathcal{R}$ as

$$\mathcal{X}_j(\mathcal{R}) = \left\{x_{j((CF_j(\delta_1-1))+1)}, \cdots, x_{j(CF_j(\delta_2))}\right\}, \quad (10)$$

and the number of the objects as

$$|\mathcal{X}_j(\mathcal{R})| = CF_j(\delta_2) - CF_j(\delta_1 - 1). \quad (11)$$

For our purpose of node pruning, we define the following distance ranges at nodes v_{2j} and $v_{(2j+1)}$ measured from the pivot p_j in their parent node v_j.

$$\mathcal{R}_{2j} = [\delta_j(1), \delta_j(\lceil N_j/2 \rceil)], \quad (12)$$

$$\mathcal{R}_{(2j+1)} = [\delta_j(\lceil N_j/2 \rceil + 1), \delta_j(N_j)]. \quad (13)$$

The computational complexity of the *NBS* algorithm at node v_j coincides with that for generating the look-up table of the CF function at line 3 in **Algorithm 3**. Suppose that this procedure is implemented as follows.

1. Initialize CF function values of all integers δ.
   $CF_j(\delta) \leftarrow 0$ for $\delta_j(1) - 1 \le \delta \le \delta_j(N_j)$.
2. Initialize CF function values of integers $\delta_j(h)$ corresponding to $d(x_{j(h)}, p_j)$.
   $CF_j(\delta_j(h)) \leftarrow CF_j(\delta_j(h)) + 1$ for $1 \le h \le N_j$.
3. Set CF function values of all the integers δ one by one in ascending order of δ as follows.
   $CF_j(\delta) \leftarrow CF_j(\delta) + CF_j(\delta - 1)$
   from $\delta = \delta_j(1)$ to $\delta = \delta_j(N_j)$.

Then the complexity at node v_j is $O(N_j + M_j)$, where $M_j = \delta_j(N_j) - \delta_j(1)$ and that at level i is approximated by $O(N +$

Algorithm 4 Similarity search algorithm

Input: $\mathcal{T}, \mathcal{X}, q, r, L$ ▷ $L \leftarrow \tilde{L}(r; \mathcal{X})$
Output: $\mathcal{X}(q; r) \subset \mathcal{X}$
$\mathcal{V}_1(q; r) \leftarrow \{v_1\}, \mathcal{V}_i(q; r) \leftarrow \emptyset$ for $1 < i \leq L$
$Y_i = 0$ for $1 \leq i \leq L$
$i \leftarrow 1$
while $i \leq L$ **do** ▷ *First process*
  for all $v_j \in \mathcal{V}_i$ **do**
    Calculate $d(q, p_j)$ and $\mathcal{B}_j = [b_{(j,1)}, b_{(j,2)}]$
    Update $Y_i \leftarrow Y_i + (CF_j(b_{(j,2)}) - CF_j(b_{(j,1)}))$
    if $i < L$ **then**
      $\tilde{\mathcal{V}}_j \leftarrow \{v_{2j+a} \mid \mathcal{B}_j \cap \mathcal{R}_{(2j+a)} \neq \emptyset, a \in \{0, 1\}\}$
      Update $\mathcal{V}_{(i+1)} \leftarrow \mathcal{V}_{(i+1)} \cup \tilde{\mathcal{V}}_j$
    end if
  end for
  $i \leftarrow i + 1$
end while
$i^* = \arg\min_{1 \leq i \leq L} Y_i$ ▷ *Second process*
$\mathcal{S} = \bigcup_{v_j \in \mathcal{V}_{i^*}} \mathcal{X}_j(\mathcal{B}_j)$
$\mathcal{W} \leftarrow \{x_h \in \mathcal{S} \mid D(x_h, q; \mathcal{P}(x_h)) \leq r\}$ ▷ *Third process*
return $\mathcal{X}(q; r) \leftarrow \{x_h \in \mathcal{W} \mid d(x_h, q) \leq r\}$

M), where $M = \delta_1(N) - \delta_1(1)$. In our settings $M \ll N$, the complexity is represented by $O(N)$. In fact, M was much smaller than N in our experiments.

We discuss the total computational complexity of our index construction algorithm. The main parts are the *PGM* function and the *NBS* function. The *PGM* function requires $O(N_j \cdot (H + \log N_j))$ for nodes v_j, $(2^{(i-1)} \leq j < 2^i)$ at level i, $(1 \leq i \leq L)$. Let us focus on the dominant factor such as $N_j \cdot H$ and $N_j \log N_j$. Level i contains $2^{(i-1)}$ nodes and each node has $\mathcal{X}_j$ with around $N/2^{(i-1)}$ size. Then $\sum_{i=1}^{L} \sum_{j=2^{(i-1)}}^{2^i-1} (N_j \cdot H) = L \cdot N \cdot H$. Similarly, $\sum_{i=1}^{L} \sum_{j=2^{(i-1)}}^{2^i-1} (N_j \log N_j) \sim L \cdot N \log N$. In contrast, the *NBS* function at each level requires $O(N)$ under a natural situation. Then the computational complexity of the *NBS* algorithm is $O(L \cdot N)$. Therefore, we obtain the total computational complexity of the index construction algorithm as follows:

$$O(L \cdot N \cdot (H + \log N)) . \tag{14}$$

Thus, our index construction algorithm has the reasonably good scalable properties that the computational complexity to N is quasi-linear while those to L and H are linear.

4.2 Similarity Search Algorithm

We propose the similarity search algorithm in **Algorithm 4**, which efficiently finds a subset of given objects, $\mathcal{X}(q; r) \subset \mathcal{X}$, within the range r measured from query object q, using the CBT $\mathcal{T}$ constructed from $\mathcal{X}$. To increase the computational efficiency, we newly introduce to the algorithm both a *three-stage process* and an *early termination scheme*. The three-stage process consists of branch pruning, level selection, and object filtering, and narrows down candidate objects within the range.

We first define symbols and terms for given query q and range r. Let $\mathcal{B}_j = [b_{(j,1)}, b_{(j,2)}]$ denote the distance range within r from q, which is measured as a distance from p_j in v_j, that is,

$$\mathcal{B}_j = [\, \max\{0,\, d(q, p_j) - r\},\, d(q, p_j) + r\,] \tag{15}$$

$$b_{(j,1)} = \max\{0,\, d(q, p_j) - r\} \tag{16}$$

$$b_{(j,2)} = d(q, p_j) + r\,. \tag{17}$$

Let $\tilde{\mathcal{V}}_j$ denote a set of *unpruned nodes*, which is a subset of child nodes v_{2j} and v_{2j+1} in parent nodes v_j, and each of which satisfies

$$\tilde{\mathcal{V}}_j = \{v_{2j+a} \mid \mathcal{B}_j \cap \mathcal{R}_{(2j+a)} \neq \emptyset, a \in \{0, 1\}\}. \tag{18}$$

Let $\mathcal{V}_{(i+1)}$ denote a set of the *unpruned nodes* at level $(i+1)$, which is expressed as,

$$\mathcal{V}_{(i+1)} = \bigcup_{2^{(i-1)} \leq j < 2^i} \tilde{\mathcal{V}}_j. \tag{19}$$

Then, at the maximum level L, we can obtain the number of the objects U in the set of *unpruned nodes* $\mathcal{V}_L$ as

$$U = \sum_{v_j \in \mathcal{V}_L} |\mathcal{X}_j| \,. \tag{20}$$

Next, we define an *unbounded object* as an object in unpruned node v_j, which satisfies the unbounded condition, i.e., $x_h \in \mathcal{X}_j(\mathcal{B}_j)$. Then, we can obtain the set of unbounded objects at level i, denoted as $\mathcal{S}_i$, as follows:

$$\mathcal{S}_i = \bigcup_{v_j \in \mathcal{V}_i} \mathcal{X}_j(\mathcal{B}_j) \,, \tag{21}$$

where $\mathcal{X}_j(\mathcal{B}_j)$ is defined in the same way as Eq. (10). Recall that the number of objects in $\mathcal{S}_i$, denoted by $Y_i = |\mathcal{S}_i|$, can be directly calculated from the look-up table of the CF function as follows.

$$Y_i = \sum_{v_j \in \mathcal{V}_i} |\mathcal{X}_j(\mathcal{B}_j)| = \sum_{v_j \in \mathcal{V}_i} (CF_j(b_{(j,2)}) - CF_j(b_{(j,1)})) \tag{22}$$

Let $\mathcal{S}$ denote a set of *unbounded objects* with the minimum number of elements among $\{\mathcal{S}_1, \cdots, \mathcal{S}_L\}$, i.e.,

$$\mathcal{S} \leftarrow \mathcal{S}_{i^*}, \quad \text{where } i^* = \arg\min_{1 \leq i \leq L} Y_i. \tag{23}$$

Finally, we define an *unfiltered object* x_h as an unbounded object whose lower bound on distance to q is not larger than r. Then a set of *unfiltered objects* $\mathcal{W}$ is expressed by

$$\mathcal{W} = \{x_h \in \mathcal{S} \mid D(x_h, q; \mathcal{P}(x_h)) \leq r\}, \tag{24}$$

where the distance lower bound of *unfiltered object* x_h is calculated by using pivot set $\mathcal{P}(x_h) = \{p_j \mid x_h \in \mathcal{X}_j$ through all the levels of $1 \leq i \leq L\}$, and $D(x_h, q; \mathcal{P}(x_h))$ in Eq. (24) is defined in Eq. (6). Each pivot in $\mathcal{P}(x_h)$ corresponds to that in a node including x_h at each level from the root to the leaf level, and then $|\mathcal{P}(x_h)| = L$. Note that the set of unbounded objects is expressed by

$$\mathcal{S} = \bigcup_{v_j \in \mathcal{V}_{i^*}} \{x_h \in \mathcal{X}_j \mid D(x_h, q; \{p_j\}) \leq r\}. \tag{25}$$

Namely, we can gurantee that if an object is included in a

search result, i.e., within r from q, it is an unbounded object. Here, we can regard Eq. (24) as an improved formula of Eq. (25) by changing from $\{p_j\}$ to $\mathcal{P}(x_h)$.

Based on the foregoing definitions, we explain the proposed algorithm, shedding light on the three-stage process followed by the early termination scheme. This scheme determines a termination level of CBT $\mathcal{T}$ depending on r before search *off-line* and sets the level $\tilde{L}(r;\mathcal{X})$ shown in **Algorithm 4**. For simplicity, we first use the maximum level L and last replace it with $\tilde{L}(r;\mathcal{X})$ as shown at line 1.

For the initialization, the unpruned node set $\mathcal{V}_i$ and the number of unbounded objects Y_i at all the levels are set as shown at lines 3 and 4.

1. The *first* process in the *three-stage process* prunes branches in $\mathcal{T}$, i.e., to omit nodes connected the branches and their descendants. This process is performed from the root level $i=1$ to the leaf level $i=L$ one by one as shown at lines 6–16. For each unpruned node v_j at level i, $\mathcal{B}_j = [b_{(j,1)}, b_{(j,2)}]$ in Eq. (15) is obtained by calculating $d(q, p_j)$. Then Y_i is updated by addition of $(CF_j(b_{(j,2)}) - CF_j(b_{(j,1)}))$ as in Eq. (22). After that, the algorithm prunes branches connected to nodes at the next level $i+1$ based on the condition in Eq. (18), and extracts the remaining nodes as the unpruned nodes $\mathcal{V}_{(i+1)}$ as in Eq. (19).
2. The *second* process selects level i^* at which the number of the unbounded objects Y_i is minimum as shown at line 17, and then generates the minimum set of the unbounded objects by distance range constraints at the level i^* as $\mathcal{S}$ at line 18.
3. The *third* process filters each of the unbounded objects generated by the second process individually at line 19. The distance of each unbounded object $x_h \in \mathcal{S}$ to pivot p_j has already been calculated at each level. Then the lower bound on a distance of x_h to q based on $\mathcal{P}(x_h)$ is obtained as $D(x_h, q; \mathcal{P}(x_h))$. The algorithm filters object x_h that satisfies $D(x_h, q; \mathcal{P}(x_h)) > r$, and keeps the others as the unfiltered objects in $\mathcal{W}$ at line 19. Finally, the algorithm identifies a set of objects within r from q, $\mathcal{X}(q; r)$, by calculating an exact distance between q and the unfiltered object in $\mathcal{W}$.

Next, let us evaluate the computational complexity of the similarity search algorithm. In the first process, the complexity depends on the calculations of a distance between q and pivot p_j at unpruned node v_j at level i, which is shown at line 8. The rest processes of the update processes of Y_i and $\mathcal{V}_{(i+1)}$ at lines 9 and 12 are efficiently performed by the CF function in Eq. (22). At each level, the copmutational complexity is $O(H \cdot |\mathcal{V}_i|)$ corresponding to that of the distance calculations, where H denotes the dimensionality of a given vector space. Through all the levels, it is $O(H \cdot |\mathcal{V}|)$, where $\mathcal{V} = \bigcup_{1 \le i \le L} \mathcal{V}_i$.

The second process determines the level i^* of the minimum Y_i with $O(L)$. Then it generates $\mathcal{S}$ by applying the CF function to each unpruned node v_j at level i^*. Given a distance from pivot p_j at v_j, the look-up table of the CF function directly returns the index value $j(h)$ of the corresponding object $x_{j(h)}$, i.e., the rank order. For this reason, the $\mathcal{S}$ generation is performed with $O(|\mathcal{S}|)$.

The third process calculates the absolute value of each difference of $d(x_h, p_j)$ and $d(q, p_j)$ that have been stored, and compares the resultant $D(x_h, q; \mathcal{P}(x_h))$ with r at line 19. Then the complexity is $O(L \cdot |\mathcal{S}|)$, where $L = |\mathcal{P}(x_h)|$ for any $x_h \in \mathcal{S}$. The last process at line 20 returns the final search solution by calculating exact distances of all unfiltered objects $x_h \in \mathcal{W}$ to q with $O(H \cdot |\mathcal{W}|)$. Therefore, the total computational complexity of the similarity search algorithm is expressed by

$$O\left(H \cdot |\mathcal{V}| + L \cdot |\mathcal{S}| + H \cdot |\mathcal{W}|\right) , \tag{26}$$

where the first, the second, and the third term come from the first, the third, and the last process, respectively.

We explain the early termination of the search algorithm. In general, by increasing the level L of the CBT index, it is expected to reduce the number of unfiltered objects $|\mathcal{W}|$. In contrast, the number of unpruned nodes $|\mathcal{V}|$, i.e., the number of pivots in the unpruned nodes, is likely to increase. Then, depending on a given range r, we want to balance the trade-off by determining an appropriate search termination level over the CBT so as to minimize the expected search cost. To this end, based on Eq. (26), we define the following total cost function for query q and range r, under a given set of objects $\mathcal{X}$ and level L:

$$C(q, r; \mathcal{X}, L) = |\mathcal{V}| + \frac{L}{H} \cdot |\mathcal{S}| + |\mathcal{W}| . \tag{27}$$

Note that $C(q, r; \mathcal{X}, L)$ can be intuitively interpreted as a computational cost in terms of the required number of *distance calculations* for query q and range r.

For a given set of objects $\mathcal{X}$ and range r, we propose to estimate the average of the cost defined in Eq. (27), based on a leave-one-out cross validation technique. Namely, for the CBT index with the different maximum level $L \in \{1, \cdots, L_{max}\}$, we can determine the appropriate search termination level $\tilde{L}(r;\mathcal{X})$ for the range r so that $C(x_h, r; \mathcal{X} \setminus \{x_h\}, L)$ is minimum, i.e.,

$$\tilde{L}(r;\mathcal{X}) = \underset{1 \le L \le L_{max}}{\arg\min} \frac{1}{|\mathcal{X}|} \sum_{x_h \in \mathcal{X}} C(x_h, r; \mathcal{X} \setminus \{x_h\}, L). \tag{28}$$

As shown later in our experiments, the appropriate termination levels of CBT indexes worked well, which were computed within a few hours for a large-scale data set consisting of 1×10^6 objects.

5. Experiments

We show the experimental data sets followed by our results on the performance as regards both the index construction and similarity search algorithms. In our experiments, all algorithms for index construction and similarity search were executed on a computer system equipped with two Xeon X5690 3.47GHz CPUs and a 198GB main memory with a single thread within the memory capacity.

5.1 Data sets

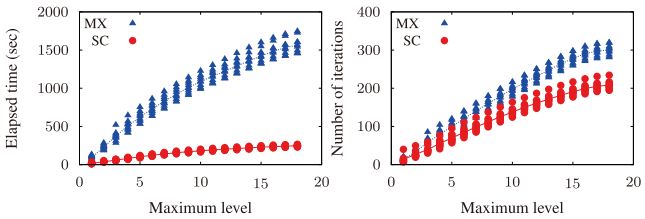
We employed the CoPhIR (Content-based Photo Image Retrieval) test-collection [19], [22]. The CoPhIR contains the following five MPEG-7 global descriptors extracted from each of more than 100 million images: scalable color (SC for short), color structure (CS), color layout (CL), edge histogram (EH), and homogeneous texture (HT). Each of the descriptors is represented as a feature vector, i.e., a point in a Euclidean space and a distinct distance measure for each descriptor is defined by the MPEG-7 group [23], [24]. The numbers of the elements (dimensionality) of descriptors, SC, CS, CL, EH, and HT, are originally 64, 64, 12, 80, and 62, respectively. Thus the dimensionality of each metric space by the descriptor except CL is high.

Of these, we, in particular, utilized the four descriptors of SC, CS, EH and, HT whose distances are measured by the weighted L_1 distance [23], [24]. In terms of EH, instead of the prepared 80-dimensional vector, we exploited a 150-dimensional vector that includes elements related to global and semi-global edge histograms as described in [25]. In addition to the four descriptors, we utilized a new descriptor created by concatenating weighted feature vectors of these descriptors (mixed descriptor, MX for short), just as the descriptor by aggregating distances between the feature vectors is utilized in [22]. Our descriptor's dimensionality reaches 340, which is the sum of the number of elements of the concatenated feature vector. Note that if the distance is defined as the L_1 distance of feature vectors in each descriptor, then the sum of the distances of a pair of the corresponding feature vectors in each descriptor is equal to a distance of a pair of feature vectors that are newly made by concatinating the corresponding feature vectors. For each descriptor, we prepared data set $\mathcal{X}$ randomly chosen from the original data set. The number of the feature vectors (or objects) in the data set $\mathcal{X}$ was $1{\times}10^6$. As the test set Q for the similarity search performance evaluation, we chose 1×10^3 query objects randomly in the same manner as the data sets so that the chosen query objects were not included in the data sets.

5.2 Index Construction Performance

We evaluated the index construction performance of our proposed method in **Algorithm 1** by varying the maximum level L from $L=1$ to 18, in terms of elapsed times and the number of iterations for generating pivots, each of which corresponds to lines 4–10 in **Algorithm 2**. More precisely, we obtained the number of PGM iterations at level L by calculating the average number of PGM iterations for nodes v_j ($2^{i-1} \le j < 2^i$) at level i and adding these average values from $i=1$ to L. We evaluated the performances through the ten trials of the pivot generation by changing initial pivots at line 3 in **Algorithm 2**.

Figures 2 (a) and (b) show the elapsed time and the cumulative average number of PGM iterations required by the proposed method, respectively, where we plotted the results of the ten trials by the markers, and their averages as the lines. Although we show only the SC and MX results, we observed the similar tendencies as those in Figs. 2 (a) and (b), in the other settings such as the other descriptors and the smaller data sets used in our preliminary experiments. Figure 2 (a) shows that even when the maximum level was set to $L=18$, the average index construction times for SC and MX were less than 5 and 30 minutes, respectively. Note that the ratio of these construction times for SC and MX reasonably coincides with that of their dimensionalities of 64 and 340. Moreover, we can see that the construction times are almost *sublinear* with respect to the level parameter L and relatively stable regardless of the initial pivot settings. From Fig. 2 (b), we can see that the average number of PGM iterations at each node were around 10 to 20. For instance, Fig. 2 (b) shows that the cumulative average numbers of iterations at level 10 are 137.8 and 197.2 for SC and MX, respectively. Then the corresponding average numbers for a node at each level are 13.78 and 19.72. The numbers for MX were larger than those for SC. This is attributed to the larger dimensionality of MX. As with the elapsed time, the average number of PGM iterations were almost *sublinear* with respect to the level parameter L. These experimental results indicate that the proposed index construction algorithm is applicable to a large-scale data set due to the level scalability.



(a) Elapsed time (b) Number of iterations

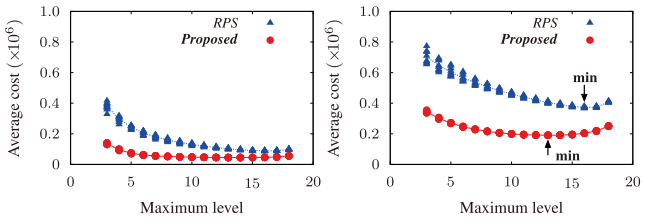
Fig. 2 Index construction performance for different descriptors of SC and MX measured by (a) elapsed time and (b) cumulative average number of PGM iterations.

5.3 Similarity Search Performance

We first show the settings for the performance evaluation, and next analyze the proposed method from a viewpoint of search performance, finally compare it with M-Index [13]. Note that M-Index is one of the most advanced similarity search method using pivots, which shows the comparable performance with state-of-the-arts similarity search methods described in Sect. 2.

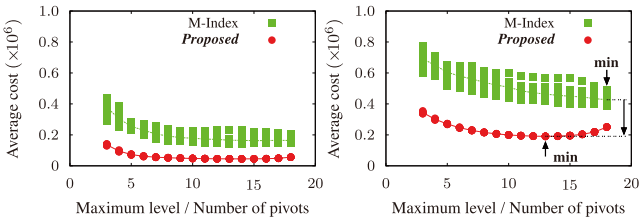
5.3.1 Settings

For range query, we set the range distance r at the average distance to the 100-th nearest objects from each of the 1×10^3 query objects $q \in Q$. The average distances of r for the descriptors, CS, HT, EH, SC, and MX, were 1000, 500,



(a) SC descriptor (b) MX descriptor

Fig. 3 Search performance depending on pivots that were generated by *PGM* in the proposed method and selected by *RPS* for (a) SC and (b) MX descriptor.



(a) SC descriptor (b) MX descriptor

Fig. 4 Search performance comparison of the proposed method with M-Index for (a) SC and (b) MX descriptor.

300, 200, and 13000 in our preliminary experiments, respectively. We applied the search methods evaluated for range query of r to the data set $\mathcal{X}$ and the test query set $\mathcal{Q}$ described in Sect. 5.1. We measured their search performances by the average of the cost function defined in the same way as Eq. (27), with respect to level L, i.e.,

$$AC(L) = \frac{1}{|\mathcal{Q}|} \sum_{q \in \mathcal{Q}} C(q, L; \mathcal{X}, r). \tag{29}$$

For a fair comparison, we did not employ elapsed time as the evaluation measure so that we assessed the impact of only algorithms without any coding techniques. However, for a reference, we report that the average execution times of similarity search for one query were 0.050, 0.146, 0.040, 0.037, and 0.221 seconds for CS, EH, HT, SC and MX in this order.

5.3.2 Analysis

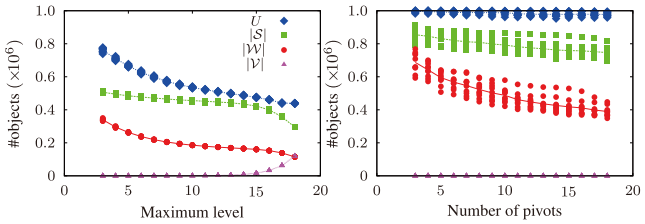
We analyzed the proposed method itself in terms of the search cost. We, in particular, focused on the effect of the index generated by using pivots on the search cost. Recall that the *PGM* function in the index construction algorithm generates a pivot by maximizing the objective function in Eq. (7). Instead of the *PGM* function, we employed for the compared index construction algorithm a random pivot selection function (*RPS* for short) that chooses a pivot at random from a data set $\mathcal{X}_j$.

Figures 3 (a) and (b) show similarity search performances of the *PGM* and the *RPS* algorithm for SC and MX descriptors. The average costs obtained through the ten trials with varied initial pivots for the *PGM* and the selected pivots for the *RPS* are depicted with marked points and their average values of the algorithms with lines along the maximum level of the CBT index from $L = 3$ to 18. The average cost of each algorithm for both descriptors reached the minimum at a level. For instance, the minimum values of the proposed method and *RPS* were 0.045 at level 14 and 0.089 at level 16 in Fig. 3 (a), and 0.19 at level 13 and 0.37 at level 16 in Fig. 3 (b), respectively. The proposed method with the *PGM* function stably achieved almost *half* the search cost of the method with the *RPS*. Besides, we also observed the similar tendencies as those in Figs. 3 (a) and (b), in the other settings such as the other descriptors, the smaller data sets, and the range settings. Thus, the proposed method worked well in search, compared with the method with the *RPS*.

5.3.3 Comparison

We compared the proposed method with M-Index [13] regarding the average cost in Eq. (29) and its components appearing in Eq. (27). As for M-Index, we employed the fixed 3-level M-Index structure without eliminating small clusters because such small clusters can contribute to improving the pruning performance although they sometimes increase actual response times. We randomly chose the pivots for M-Index from the objects in the data set $\mathcal{X}$ as suggested in [13] and changed the numbers of selected pivots from 3 to 18. In similarity search, M-Index utilizes four strategies of *double-pivot distance constraint*, *range-pivot distance constraint*, *object-pivot distance constraint*, and *pivot filtering* in this order. Note that both the double-pivot and range-pivot distance constraints correspond to the first process (branch pruning or nodes-omitting) in the proposed algorithm because some sets (or clusters) of objects are collectively pruned only by performing query-pivot distance calculation, which can be evaluated as the number U of the objects in unpruned nodes in Eq. (20). The object-pivot distance constraint and the pivot filtering correspond to the second process collecting objects within the range and the third process individually filtering objects in **Algorithm 4**, which can be evaluated as the number $|\mathcal{S}|$ of unbounded objects defined in Eq. (25), and as the number $|\curlyvee\mathcal{W}|$ of unfiltered objects defined in Eq. (24), respectively. Thus, we calculated the three numbers of objects, U, $|\mathcal{S}|$ and $|\curlyvee\mathcal{W}|$, obtained by applying M-Index, similarly to the proposed method. Since the cost function value is calculated in Eq. (29) by using the foregoing three values, U, $|\mathcal{S}|$ and $|\curlyvee\mathcal{W}|$, we adopted this cost function for a favorable comparison measure. Besides, the measure is not affected by some implementation issues.

Figures 4 (a) and (b) demonstrate that the proposed method outperformed M-Index in terms of the average costs for SC and MX descriptors, respectively. Note that the horizontal axis represents the maximum level for the proposed method and the number of pivots for M-Index. This is because the number of pivots that the proposed method uses to filter object x_h is equal to the maximum level, i.e.,



(a) Proposed method (b) M-Index

Fig. 5 Average numbers of remaining objects within the range r for MX descriptor just after each process or strategy in (a) proposed method and (b) M-Index. The points marked with triangles and the corresponding line of $|\mathcal{V}|$ denote the number of the applied pivots that is different from others.

$|\mathcal{P}(x_h)| = L$. The proposed method at all the levels kept the smaller average costs than M-Index at all the numbers of pivots for both the descriptors. Furthermore, the proposed method achieved almost a quarter of and half the minimum value of M-Index for SC and MX descriptors, respectively. We can see that the proposed method performed the search with the stable average costs independent of the initial settings while M-Index did with the large variations of the costs depending on the chosen pivots. Compared with M-Index, the proposed method provided the stable and lower computational costs. Moreover, we should report that the levels determined by the early termination scheme were 14, 11, 12, 12, and 13 for CS, EH, HT, SC and MX in this order. This indicates that the early termination scheme works reasonably well.

Figures 5 (a) and (b) compare the average numbers of remaining objects by the proposed method and M-Index for the MX descriptor, in terms of U, $|\mathcal{S}|$ and $|\mathcal{W}|$, together with the average number of query-pivot distance calculations, i.e., the number of unpruned nodes $|\mathcal{V}|$. Note that the average numbers of $|\mathcal{V}|$, $|\mathcal{S}|$, and $|\mathcal{W}|$ except U in Fig. 5 (a) are related to those of *Proposed* in Fig. 4 (b) according to Eq. (27). From these figures, we can see that all of the average numbers of remaining objects by the proposed method were much smaller than those by M-Index. In contrast, the required number of query-pivot distance calculations $|\mathcal{V}|$ of our method became larger than that of M-Index especially larger L although this cost was a fraction of the total cost shown in Fig. 4 (b). This observation indicates that the cost of our method reaches the minimum at not so high level. In order to terminate the search algorithm at an appropriate level, we can utilize the early termination scheme stated in Sect. 4.2.

Thus, the experimental results suggest that the proposed method achieves high search performance for a large-scale data set, due to the index construction algorithm based on the *PGM* function, and the similarity search algorithm with the three-stage process and the early termination condition.

6. Conclusion

We proposed a fast exact similarity search method with a complete binary tree (CBT) for large-scale data sets. The proposed method consists of two main algorithms. One is an index construction algorithm that recursively builds the CBT by using a pivot optimized with our skillful technique at each node. The other is a similarity search algorithm with our three-stage process that efficiently prunes unnecessary branches and filters objects, and an early termination scheme determining an appropriate level of the CBT for search, resulting in the reduction of a computational cost for search.

The experimental results on one million real images with high dimensionality demonstrated that the proposed method significantly reduced the computational cost to around 28–45% of M-Index, one of the state-of-the-art methods. Our method also constructed the CBT index in reasonable elapsed time and a computational cost. Thus we have shown that the proposed method is suitable for large-scale and high-dimensional data sets. Furthermore, our method can be easily expanded so that it is applicable to a shared-memory parallel computing system for higher efficiency.

Acknowledgments

This work was partly supported by JSPS Grant-in-Aid for Scientific Research (C) (No. 26330138).

References

[1] E. Chávez, G. Navarro, R. Baeza-Yates, and J.L. Marroquín, "Searching in metric spaces," ACM Computing Surveys, vol.33, no.3, pp.273–321, Sept. 2001.
[2] P. Zezula, G. Amato, V. Dohnal, and M. Batko, Similarity search: The metric space approach, Springer, New York, NY, 2006.
[3] H. Samet, Foundations of multidimensional and metric data structures, Morgan Kaufmann Publishers, San Francisco, CA, 2006.
[4] S. Arya, D.M. Mount, N.S. Netanyahu, R. Silverman, and A.Y. Wu, "An optimal algorithm for approximate nearest neighbor searching fixed dimensions," Journal of the ACM, vol.45, no.6, pp.891–923, Nov. 1998.
[5] P. Indyk, Handbook of Discrete and Computational Geometry (second edition), chapter 39: Nearest neighbors in high-dimensional spaces, Chapman and Hall/CRC, 2004.
[6] M.S. Charikar, "Similarity estimation techniques from rounding algorithms," Proc. Annual ACM Symp. Theory of Computing, New York, NY, pp.380–388, ACM, 2002.
[7] A. Andoni, M. Datar, N. Immorlica, P. Indyk, and V. Mirrokni, Nearest-neighbor methods in learning and vision: theory and practice, chapter 3: Locality-sensitive hashing using stable distributions, The MIT Press, Cambridge, Massachusetts, 2006.
[8] M.L. Micó, J. Oncina, and E. Vidal, "A new version of the nearest-neighbour approximating and eliminating search algorithm (aesa) with linear preprocessing time and memory requirements," Pattern Recognition Letters, vol.15, no.1, pp.9–17, Jan. 1994.
[9] J.K. Uhlmann, "Satisfying general proximity/similarity queries with metric trees," Information Processing Letters, vol.40, no.4, pp.175–179, Nov. 1991.
[10] S. Brin, "Near neighbor search in large metric spaces," Proc. Int. Conf. Very Large Data Bases, pp.574–584, Morgan Kaufmann, Sept. 1995.
[11] E. Chávez and G. Navarro, "A compact space decomposition for effective metric indexing," Pattern Recognition Letters, vol.26, no.9,

pp.1363–1376, July 2005.
[12] H.V. Jagadish, B.C. Ooi, K.L. Tan, C. Yu, and R. Zhang, "iDistance: An adaptive b^+-tree based indexing method for nearest neighbor search," ACM Trans. Database Systems, vol.30, no.2, pp.364–397, June 2005.
[13] D. Novak and M. Batko, "Metric index: An efficient and scalable solution for similarity search," Proc. Int. Conf. Similarity Search and Applications, pp.65–73, IEEE, 2009.
[14] P.N. Yianilos, "Data structures and algorithms for nearest neighbor search in general metric spaces," Proc. Annual ACM-SIAM Sympo. Discrete Algorithms, pp.311–321, ACM, Jan. 1993.
[15] K.L. Clarkson, "Building triangulations using ϵ-nets," Proc. Annual ACM Sympo. Theory of Computing, pp.326–335, ACM, 2006.
[16] B. Bustos, G. Navarro, and E. Chávez, "Pivot selection techniques for proximity searching in metric spaces," Pattern Recognition Letters, vol.24, no.14, pp.2357–2366, Oct. 2003.
[17] M. Kimura, K. Saito, and N. Ueda, "Pivot learning for efficient similarity search," Proc. Int. Conf. Knowledge-Based and Intelligent Information & Enginnering Systems, pp.227–234, Springer-Verlag, 2007.
[18] E. Kobayashi, T. Fushimi, K. Saito, and T. Ikeda, "Similarity search by generating pivots based on Manhattan distance," Proc. Pacific Rim Int. Conf. Artificial Intelligence, vol.8862, pp.435–446, Springer International Publishing, 2014.
[19] P. Bolettieri, A. Esuli, F. Falchi, C. Lucchese, R. Perego, T. Piccioli, and F. Rabitti, "CoPhIR: a test collection for content-based image retrieval," CoRR, vol.abs/0905.4627v2, 2009.
[20] G. Amato, A. Esuli, and F. Falchi, "A comparison of pivot selection techniques for permutation-based indexing," Information Systems, vol.52, pp.176–188, Aug.–Sept. 2015.
[21] T.F. Gonzalez, "Clustering to minimize the maximum intercluster distance," Theoretical Computer Science, vol.38, pp.293–306, 1985.
[22] M. Batko, P. Kohoutkova, and D. Novak, "CoPhIR image collection under the microscope," Proc. Int. Conf. Similarity Search and Applications, pp.47–54, IEEE, 2009.
[23] MPEG-7, "Multimedia content description interface – Part 3: Visual," ISO/IEC 15938-3:2002, 2002.
[24] B.S. Manjunath, P. Salembier, and T. Sikora, Introduction to MPEG-7: Multimedia content description interface, John wiley & Sons Ltd, England, 2003.
[25] D.K. Park, Y.S. Jeon, and C.S. Won, "Efficient use of local edge histogram descriptor," Proc. ACM Workshops on Multimedia, New York, NY, pp.51–54, ACM, 2000.

Yuki Yamagishi received the Ph.D. degree in arts and sciences from University of Shizuoka in 2017. In 2017, he joined the University of Shizuoka. He is currently a postdoctoral researcher at the School of Management and Information. His current research interests are machine learning and data mining.

Kazuo Aoyama received the B.E. degree in applied physics from Waseda University in 1986 and the M.E. degree from Tokyo Institute of Technology in 1988. In 1988, he joined NTT Corporation. His research interests include device modeling, LSI design methodology, computer architecture, data structure and algorithms, and machine learning.

Kazumi Saito received the B.S. degree in mathematics from Keio University in 1985 and the Ph.D. degree in engineering from University of Tokyo in 1998. In 1985, he joined the NTT Electrical Communication Laboratories. In 2007, he joined the University of Shizuoka. He is currently a professor at the School of Management and Information. His current research interests are machine learning and statistical analysis of complex networks.

Tetsuo Ikeda received the Master of Computer Science from University of Tokyo in 1981, and the Doctor of Engineering from University of Tokyo in 2001. In 1981, he joined the NTT Electrical Communication Laboratories. In 2002, he joined the Iwate Prefectural University. In 2007, he joined the University of Shizuoka. He is currently a professor at the School of Management and Information. His current research interests are data engineering, information retrieval, and GIS.