# Pivot Selection Method for Optimizing both Pruning and Balancing in Metric Space Indexes

Hisashi Kurasawa[1], Daiji Fukagawa[2], Atsuhiro Takasu[3], and Jun Adachi[3]

[1] The University of Tokyo, 2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo, Japan
[2] Doshisha University, 1-3 Tatara Miyakodani, Kyotanabe-shi, Kyoto, Japan
[3] National Institute of Informatics, 2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo, Japan

**Abstract.** We researched to try to find a way 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 structure index. A problem in the existing indexes is that they only focus on the pruning objects and do not take into consideration the tree balancing. The balance of the indexes depends on the data distribution and the indexes don't reduce the search cost for all data. We propose a similarity search index called the Partitioning Capacity Tree (PCTree). PCTree automatically optimizes the pivot selection based on both the balance of the regions partitioned by a pivot and the estimated effectiveness of the search pruning by the pivot. As a result, PCTree reduces the search cost for various data distributions. Our evaluations comparing it with four indexes on three real datasets showed that PCTree successfully reduces the search cost and is good at handling various data distributions.

**Keywords:** Similarity Search, Metric Space, Indexing.

## 1 Introduction

Finding similar objects in a large dataset is a fundamental process of many applications, such as image completion [7]. These applications can be speeded up by reducing the query execution cost of a similarity search.

Similarity search indexes are used for pruning objects dissimilar to a query, and reduce the search cost, such as the distance computations and the disk accesses. Most indexing schemes use *pivots*, which are reference objects. They recursively divide a region into subregions by using pivots, and construct a tree structure index. They prune some of the subregions by using the triangle inequality while searching. 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 [2]. However, most previous works do not take into account the balance of the tree. It is well known that a reduction in the tree height reduces the average search cost. Thus, even if the pivots of the methods are enough good for pruning objects at the branch, they may not reduce the average search cost for every data distribution. Therefore, we developed a new pivot selection for optimizing both the pruning and the balancing.

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 the results from several experiments where we compared the PCTree with several metric space indexes using three real datasets.

## 2 Related Work

Let $M = (D, d)$ be a metric space defined for a domain of objects $D$ and a distance function $d : D \times D \mapsto \mathbb{R}$. $M$ satisfies the postulates, such as the triangle inequality. Our index deals with the space and the distance.

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 [15]. 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. Their pivots are selected by using simple statical features, such as the variance. As a result, they require a huge search cost because they have to traverse many nodes in the tree.

The recent indexing schemes focus on pruning dissimilar objects rather than balancing the tree. Some schemes use other partitioning techniques instead of Ball partitioning. Vantage Point Forest (VPF) [16] and D-Index [6] divide a region into three sub-regions according 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) [14] 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 [15, 2]. These methods used the mean [6], the variance [15], and the sum [5]. iDistance [8] selects pivots based on the clustering result and reduces the number of regions accessed during a search. OMNI-Family [10] chooses a set of pivots based on the minimum bounding region. MMMP [12] has a pivot selection based on the maximal margin, and it classifies dense regions. Moreover, some methods combine different pivot selection techniques [17].

**Table 1.** Notation

| Notation | Meaning |
|---|---|
| $D$ | domain of objects |
| $d$ | distance function |
| $R$ | region |
| $S$ | object set |
| $S_R$ | object set in $R$ |
| $S_{R,\mathrm{sub}}$ | subset of $S_R$ |
| $m$ | sampling parameter for $S_{R,\mathrm{sub}}$ |
| $S_p$ | pivot candidate set |
| $s$ | sampling parameter for $S_p$ |
| $p$ | pivot |
| $r_p$ | partitioning distance of $p$ |
| $q$ | query |
| $r_{o,k,S}$ | distance between $o$ and $o$'s $k$-nearest neighbor in $S$ |
| PC | pivot capacity |
| $X_{i,p,r_p}$ | *partitioning distribution* |
| $Y_{i,p,r_p}$ | *pruning distribution* |
| $S_r$ | result set |
| $k$ | # results |
| $r_q$ | query range |

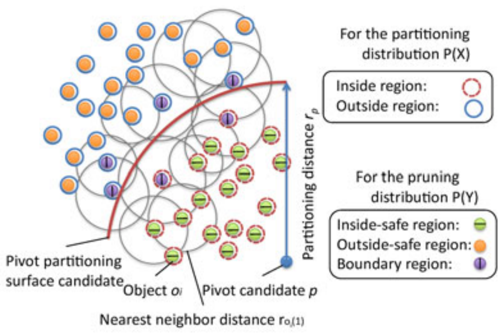

For the partitioning distribution P(X)
Inside region:
Outside region:
For the pruning distribution P(Y)
Inside-safe region:
Outside-safe region:
Boundary region:
Partitioning distance $r_p$
Pivot partitioning surface candidate
Object $o_i$
Pivot candidate $p$
Nearest neighbor distance $r_{o,(1)}$

**Fig. 1.** Labels for measuring the PC

Still other schemes propose various index structures. Fixed Queries Array [3] is an array of distances between objects and pivots and it is used to save memory. List of Clusters (LC) [4] 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 balancing. Although the $B^+$-trees are balanced, the pivots are linearly accessed. Spatial Approximation Tree (SAT) [13] approximates a graph and uses pivots for approaching the query spatially. M-Tree [5] is a dynamic tree structure with object insertion and deletion. Slim-Tree [11] minimizes the overlap space managed by node in M-Tree.

As mentioned above, Almost all the recent methods do not take into consideration the tree balancing as VPT proposed. Our index thus aims to reduce the search cost by taking into account both the object pruning and tree balance.

## 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. Its indexing method selects pivots that optimize pruning objects and balance the index tree. Table 1 summarizes the symbols used in this article.

### 3.1 Partitioning Capacity

PC is a measure of the balance of the regions partitioned by a pivot and the estimated effectiveness of the pruning by the pivot. 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:

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

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,

$$X = \begin{cases} X_{1,p,r_p} & \text{if } q \in R_1(p, r_p) \\ X_{2,p,r_p} & \text{if } q \in R_2(p, r_p) \ . \end{cases} \tag{1}$$

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

$$P(X_{1,p,r_p}) = \frac{|S_{R_1(p,r_p)}|}{|S_R|}, \quad P(X_{1,p,r_p}) = \frac{|S_{R_2(p,r_p)}|}{|S_R|} \ , \tag{2}$$

where $S_R$ denote the set 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 $k$th nearest neighbor in $S$. Let us consider the following three regions:

$$R'_1(p, r_p, k) = \{o \in R \mid d(o, p) + r_{o,k,S} \leq r_p\} \ , \tag{3}$$

$$R'_2(p, r_p, k) = \{o \in R \mid d(o, p) - r_{o,k,S} > r_p\} \ , \tag{4}$$

$$R'_3(p, r_p, k) = R - R'_1(p, r_p, k) - R'_2(p, r_p, k) \ . \tag{5}$$

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,

$$Y = \begin{cases} Y_{1,p,r_p} & \text{if } q \in R'_1(p, r_p, k) \ , \\ Y_{2,p,r_p} & \text{if } q \in R'_2(p, r_p, k) \ , \\ Y_{3,p,r_p} & \text{if } q \in R'_3(p, r_p, k) \ . \end{cases} \tag{6}$$

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

$$P(Y_{1,p,r_p,k}) = \frac{|S_{R'_1(p,r_p)}|}{|S_R|} \ , \tag{7}$$

$$P(Y_{2,p,r_p,k}) = \frac{|S_{R'_2(p,r_p)}|)}{|S_R|} \ , \tag{8}$$

$$P(Y_{3,p,r_p,k}) = \frac{|S_{R'_3(p,r_p)}|}{|S_R|} \ . \tag{9}$$

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

$$\begin{aligned} & PC_k(p) \\ & \equiv \max_{r_p} I(X;Y) \\ & = \max_{r_p} \left( \sum_i \sum_j \left( P(X_{i,p,r_p}, Y_{j,p,r_p,k}) \cdot \log \left( \frac{P(X_{i,p,r_p}, Y_{j,p,r_p,k})}{P(X_{i,p,r_p}) P(Y_{j,p,r_p,k})} \right) \right) \right) \end{aligned} \tag{10}$$

where $I(\cdot;\cdot)$ is the mutual information. We choose the object $p$ (resp. distance $r_p$) that maximizes Eq. (10) as a pivot (resp. partitioning distance).

The PC represents the mutual dependence of the random variables for the partitioning and the pruning distributions. We regard the value of the PC as the reduction in uncertainty when knowing either the partition or the pruning effect. That is, it is the amount of information shared by the partition and the pruning effect. The PC is less than the entropy of the partitioning distribution and that of the pruning distribution because of the mutual information theory. Partitions are effective for the search pruning result for a large PC value. Balanced partitions also tend to increase the PC value. The PC is almost the same as the channel capacity of a binary erasure channel in the information and coding theory [9].

## 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.

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. We select pivot candidates such that they are separated from each other by a certain distance $d_p$. We set $d_p$ to be $d_{\text{ave}} \cdot s$, where $d_{\text{ave}}$ is the approximate average distance between objects and $s$ is a parameter. The approximate average distance is calculated by using randomly sampled objects. Let $N$ (resp. $n$) denote the number of objects in the dataset (resp. pivot candidates), the cost is at most $O(N \cdot n)$. Note that the pivot candidate selection is done once before constructing an index.

Then, PCTree recursively divides a region into subregions by using pivots as well as VPT [15]. Each pivot is selected from the candidates $S_p$ according to the criterion PC (Sec. 3.1 for details) for the predefined number $m$ of samples from the region. 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. we set the minimum PC as

$$\text{MinimumPC}(S) = -\tfrac{1}{|S|} \cdot \log \tfrac{1}{|S|} - \tfrac{|S|-1}{|S|} \cdot \log \tfrac{|S|-1}{|S|} \ , \qquad (11)$$

where $S$ is the object set. The minimum PC represents the effect of 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. The indexing cost is $O(N^2)$ at most.

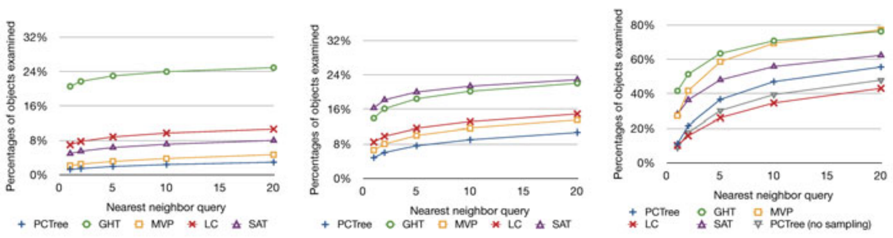
The PCTree requires three parameters during indexing. One parameter $k$ is for calculating the PC in Eqs. (3), (4) and (5), and the other parameters $s$ and $m$ are for sampling objects. From preliminary experiments, we decided to set $k$, $s$, and $m$ to 1, 0.8, and 500, respectively, in the index performance evaluations.

## 3.3 k-Nearest Neighbor Search

For the k-Nearest Neighbor searching, the PCTree receives a query $q$ and the number of results $k$. The search procedure is almost the same as for VPT [15]. First, it creates an empty set as a result set $S_r$ and sets the query range $r_q$ to be $\infty$. It starts the search from the root node in the PCTree and recursively accesses the nodes. If the node is a leaf, it finds the objects whose distance to $q$ are within $r_q$ in the node while using the object-pivot distance constraint, and adds them to $S_r$. Then, it updates $r_q$ to be the $k$-nearest neighbor radius of $q$ in $S_r$. If the node is an internal node, it reads the pivot $p$ and its partitioning distance $r_p$ associated to the node. Then, if the inequality $d(p,q) \leq r_p$ is satisfied, it accesses the child node for the inside region and updates $r_q$. After this, if the inequality $d(p,q) + r_q > r_p$ is satisfied, it accesses the other child node and updates $r_q$. Similarly, if the inequality $d(p,q) \geq r_p$ is satisfied, it accesses the child node for the outside region and updates $r_q$. After this, if the inequality $d(p,q) - r_q \leq r_p$ is satisfied, it accesses the other child node and updates $r_q$. Finally, it answers the $k$ closest objects. The search cost is $O(\log N)$ at best.

## 4 Performance Evaluation

We implemented PCTree on the Metric Space Library [1]. It provides several indexing algorithms, metric spaces, and datasets. We compared PCTree with GHT [14], MVP [15, 2], LC [4], and SAT [13], which are also in the library. As in the related work [13], the indexes were evaluated in terms of the distance computations. We measured the percentage of objects examined by the total number of distance calculations during the search. In the evaluation, 100 % of objects examined represents the cost of the linear scan algorithm. We conducted the experiment on a Linux PC equipped with an Intel(R) Quad Core Xeon(TM) X5492 3.40GHz CPU and that had 64GB of memory.



(a) Nasa (b) Color histograms (c) English dictionary

**Fig. 2.** Index Performance on Real Datasets

We used the following datasets.

**Nasa** is a set of feature vectors made by NASA [1]. 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 [1].

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

We compared the PCTree with other methods for the real datasets. Figure 2 shows the index performances for the real dataset. The vertical axis represents the number of distance computations for the $k$-nearest neighbor queries where $k$ ranges from 1 to 20. The horizontal axis is $k$.

The PCTree outperforms the other methods for the two vector datasets whereas it outperforms the others except for LC on the English dictionary dataset. Compared with the other methods, the PCTree is better than the other methods on a wide range of data distributions. For example, MVP is good for the NASA and Color histograms, but is weak for the English dictionary. On the other hand, LC is good for the English dictionary, but is weak for the NASA and Color histograms. We think that the PCTree automatically optimizes the index structure according the data distribution.

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 Figure 2 (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] Metric spaces library, `http://www.sisap.org/metric_space_library.html`

[2] Bozkaya, T., Ozsoyoglu, Z.M.: Indexing large metric spaces for similarity search queries. ACM Trans. on Database Systems 24(3), 361–404 (1999)

[3] Chevez, E., Marroguin, J.L., Navarro, G.: Fixed queries array: A fast and economical data structure for proximity searching. Multimedia Tools Applications 14(2), 113–135 (2001)

[4] Chevez, E., Navarro, G.: A compact space decomposition for effective metric indexing. Pattern Recognition Letters 24(9), 1363–1376 (2005)

[5] Ciaccia, P., Patella, M., Zezula, P.: M-tree: An efficient access method for similarity search in metric spaces. In: VLDB (1997)

[6] Dohnal, V., Gennaro, C., Savino, P., Zezula, P.: D-index: Distance searching index for metric data sets. Multimedia Tools and Applications 21(1), 9–33 (2003)

[7] Hays, J., Efros, A.A.: Scene completion using millions of photographs. In: SIGGRAPH (2007)

[8] Jagadish, H.V., Ooi, B.C., Tran, K.L., Yu, C., Zhang, R.: idistance: An adaptive b+-tree based indexing method for nearest neighbor earch. ACM Trans. on Database Systems 30(2), 364–397 (2003)

[9] Jones, G.A., Jones, J.M.: Information and Coding Theory. Springer, Heidelberg (2000)

[10] Traina Jr., C., Santos Filho, R.F., Traina, A.J., Vieira, M.R., Faloutsos, C.: The omni-family of all-purpose access methods: a simple and effective way to make similarity search more efficient. The VLDB Journal 16(4), 483–505 (2007)

[11] Traina Jr., C., Traina, A.J.M., Seeger, B., Faloutsos, C.: Slim-trees: High performance metric trees minimizing overlap between nodes. In: Zaniolo, C., Grust, T., Scholl, M.H., Lockemann, P.C. (eds.) EDBT 2000. LNCS, vol. 1777, p. 51. Springer, Heidelberg (2000)

[12] Kurasawa, H., Fukagawa, D., Takasu, A., Adachi, J.: Maximal metric margin partitioning for similarity search indexes. In: CIKM (2009)

[13] Navarro, G.: Searching in metric spaces by spatial approximation. The VLDB Journal 11(1), 28–46 (2002)

[14] Uhlmann, J.K.: Satisfying general proximity/similarity queries with metric trees. Information Processing Letters 40(4), 175–179 (1991)

[15] Yianilos, P.N.: Data structures and algorithms for nearest neighbor search in general metric spaces. In: SODA (1993)

[16] Yianilos, P.N.: Excluded middle vantage point forests for nearest neighbor search. In: ALENEX (1999)

[17] Zhuang, Y., Zhuang, Y., Li, Q., Chen, L., Yu, Y.: Indexing high-dimensional data in dual distance spaces: a symmetrical encoding approach. In: EDBT (2008)