Nearest Neighbours Search Using the PM-Tree

Tomáš Skopal[1], Jaroslav Pokorný[1], and Václav Snášel[2]

[1] Charles University in Prague, FMP, Department of Software Engineering,
Malostranské nám. 25, 118 00 Prague, Czech Republic, EU
tomas@skopal.net, jaroslav.pokorny@mff.cuni.cz

[2] VŠB–Technical University of Ostrava, FECS, Dept. of Computer Science,
tř. 17. listopadu 15, 708 33 Ostrava, Czech Republic, EU
vaclav.snasel@vsb.cz

Abstract. We introduce a method of searching the k nearest neighbours (k-NN) using PM-tree. The PM-tree is a metric access method for similarity search in large multimedia databases. As an extension of M-tree, the structure of PM-tree exploits local dynamic pivots (like M-tree does it) as well as global static pivots (used by LAESA-like methods). While in M-tree a metric region is represented by a hyper-sphere, in PM-tree the "volume" of metric region is further reduced by a set of hyper-rings. As a consequence, the shape of PM-tree's metric region bounds the indexed objects more tightly which, in turn, improves the overall search efficiency. Besides the description of PM-tree, we propose an optimal k-NN search algorithm. Finally, the efficiency of k-NN search is experimentally evaluated on large synthetic as well as real-world datasets.

1 Introduction

The volume of multimedia databases rapidly increases and the need for efficient content-based search in large multimedia databases becomes stronger. In particular, there is a need for searching for the k most similar documents (called the k nearest neighbours – k-NN) to a given query document.

Since multimedia documents are modelled by objects (usually vectors) in a feature space $\mathbb{U}$, the multimedia database can be represented by a dataset $\mathbb{S} \subset \mathbb{U}$, where $n = |\mathbb{S}|$ is size of the dataset. The search in $\mathbb{S}$ is accomplished by an access method, which retrieves objects relevant to a given similarity query. The similarity measure is often modelled by a *metric*, i.e. a distance d satisfying properties of reflexivity, positivity, symmetry, and triangular inequality. Given a metric space $\mathcal{M} = (\mathbb{U}, d)$, the *metric access methods* (MAMs) [4] organize objects in $\mathbb{S}$ such that a structure in $\mathbb{S}$ is recognized (i.e. a kind of *metric index* is constructed) and exploited for efficient (i.e. quick) search in $\mathbb{S}$. To keep the search as efficient as possible, the MAMs should minimize the *computation costs* (CC) and the *I/O costs*. The computation costs represent the number of (computationally expensive) distance computations spent by the query evaluation. The I/O costs are related to the volume of data needed to be transfered from secondary memory (also referred to as the disk access costs).

L. Zhou, B.C. Ooi, and X. Meng (Eds.): DASFAA 2005, LNCS 3453, pp. 803–815, 2005.

In this paper we propose a method of k-NN searching using PM-tree, which is a metric access method for similarity search in large multimedia databases.

2 M-Tree

Among the MAMs developed so far, the M-tree [5, 7] (and its modifications) is still the only dynamic MAM suitable for efficient similarity search in large multimedia databases. Like other dynamic and paged trees, the M-tree is a balanced hierarchy of nodes. Given a metric d, the data objects $O_i \in \mathbb{S}$ are organized in a hierarchy of nested clusters, called *metric regions*. The leaf nodes contain *ground entries* of the indexed data objects, while the *routing entries* (stored in the inner nodes) describe the metric regions. A ground entry is denoted as:

$$grnd(O_i) = [O_i, oid(O_i), d(O_i, \mathrm{Par}(O_i))]$$

where $O_i \in \mathbb{S}$ is the data object, $oid(O_i)$ is identifier of the original DB object (stored externally), and $d(O_i, \mathrm{Par}(O_i))$ is precomputed distance between O_i and the data object of its parent routing entry. A routing entry is denoted as:

$$rout(O_i) = [O_i, ptr(T(O_i)), r_{O_i}, d(O_i, \mathrm{Par}(O_i))]$$

where $O_i \in \mathbb{S}$ is a *routing object* (local pivot), $ptr(T(O_i))$ is pointer to the covering subtree, and r_{O_i} is the covering radius. The routing entry determines a hyper-spherical metric region (O_i, r_{O_i}) in $\mathcal{M}$, for which routing object O_i is the center and r_{O_i} is the radius bounding the region. In Figure 1 see several data objects partitioned among (possibly overlapping) metric regions of M-tree.

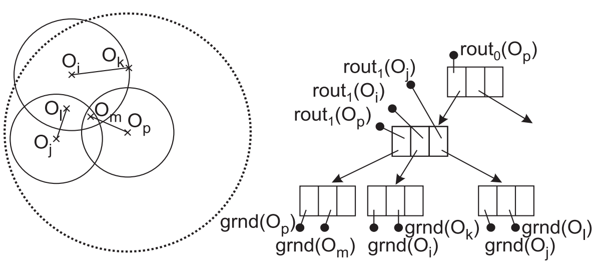

Fig. 1. Hierarchy of metric regions and the appropriate M-tree

2.1 Similarity Queries in M-Tree

The structure of M-tree was designed to support similarity queries (proximity queries actually). We distinguish two basic kinds of queries. The *range query* is specified as a hyper-spherical *query region* (Q, r_Q), defined by a query object Q and a covering query radius r_Q. The purpose of range query is to select all objects $O_i \in \mathbb{S}$ satisfying $d(Q, O_i) \leq r_Q$ (i.e. located inside the query region). The *k nearest neighbours query* (k-NN query) is specified by a query object Q and a number k. A k-NN query selects the first k nearest (most similar) objects to Q. Technically, the k-NN query can be formulated as a range query $(Q, d(Q, O_k))$, where O_k is the k-th nearest neighbour. During query processing, the M-tree hierarchy is traversed down. Given a routing entry $rout(O_i)$, the subtree $T(O_i)$ is processed only if the region defined by $rout(O_i)$ overlaps the query region.

Range Search. The range query algorithm [5, 7] has to follow all M-tree paths leading to data objects O_j inside the query region, i.e. satisfying $d(Q, O_j) \leq r_Q$. In fact, the range query algorithm recursively accesses nodes the metric regions of which (described by the parent routing entries $rout(O_i)$) overlap the query region, i.e. such that $d(O_i, Q) \leq r_{O_i} + r_Q$ is satisfied.

2.2 Nearest Neighbours Search

In fact, the k-NN query algorithm for M-tree is a more complicated range query algorithm. Since the query radius r_Q is not known in advance, it must be determined dynamically (during the query processing). For this purpose a *branch-and-bound* heuristic algorithm has been introduced [5], quite similar to that one for R-trees [8]. The k-NN query algorithm utilizes a priority queue PR of pending requests, and a k-elements array NN used to store the k-NN candidates and which, at the end of the processing, contains the result. At the beginning, the *dynamic radius* r_Q is set to ∞, while during query processing r_Q is consecutively reduced down to the "true" distance between Q and the k-th nearest neighbour.

PR Queue. The priority queue PR of pending requests $[ptr(T(O_i)), d_{min}(T(O_i))]$ is used to keep (pointers to) such subtrees $T(O_i)$, which (still) cannot be excluded from the search, due to overlap of their metric regions (O_i, r_{O_i}) with the *dynamic query region* (Q, r_Q). The priority order of each such request is given by $d_{min}(T(O_i))$, which is the smallest possible distance between an object stored in $T(O_i)$ and the query object Q. The smallest distance is denoted as the lower-bound distance between Q and the metric region (O_i, r_{O_i}):

$$d_{min}(T(O_i)) = max\{0, d(O_i, Q) - r_{O_i}\}$$

During k-NN query execution, requests from PR are being processed in the priority order, i.e. the request with smallest lower-bound distance goes first.

NN Array. The NN array contains k entries of form either $[oid(O_i), d(Q, O_i)]$ or $[-, d_{max}(T(O_i))]$. The array is sorted according to ascending distance values. Entry of form $[oid(O_i), d(Q, O_i)]$ on the j-th position in NN represents a candidate object O_i for the j-th nearest neighbour. In the second case (i.e. entry of form $[-, d_{max}(T(O_i))]$), the value $d_{max}(T(O_i))$ represents upper-bound distance between Q and objects in subtree $T(O_i)$ (in which some k-NN candidates could be stored). The upper-bound distance $d_{max}(T(O_i))$ is defined as:

$$d_{max}(T(O_i)) = d(O_i, Q) + r_{O_i}$$

Since NN is a sorted array containing the k nearest neighbours candidates (or at least upper-bound distances of the still relevant subtrees), the dynamic query radius r_Q can be determined as the current distance stored in the last entry NN[k]. During the query processing, only the closer candidates (or smaller upper-bound distances) are inserted into NN array, i.e. such candidates, which are currently located inside the dynamic query region (Q, r_Q).

After insertion into NN, the query radius r_Q is decreased (because NN[k] entry was replaced). The priority queue PR must contain only the (still) relevant subtrees, i.e. such subtrees the regions of which overlap the dynamic query region (Q, r_Q). Hence, after the dynamic radius r_Q is decreased, all irrelevant requests (for which $d_{min}(T(O_i)) > r_Q$) must be deleted from PR.

At the beginning of k-NN search, the NN candidates are unknown, thus all entries in the NN array are set to $[-, \infty]$. The query processing starts at the root level, so that $[ptr(root), \infty]$ is the first and only request in PR. For a more detailed description of the k-NN query algorithm we refer to [7, 10].

Note: The k-NN query algorithm is optimal in I/O costs, since it only accesses nodes, the metric regions of which overlap the query region $(Q, d(Q,\text{NN}[k].d_{max}))$. In other words, the I/O costs of a k-NN query (Q, k) and I/O costs of the equivalent range query $(Q, d(Q,\text{NN}[k].d_{max}))$ are equal.

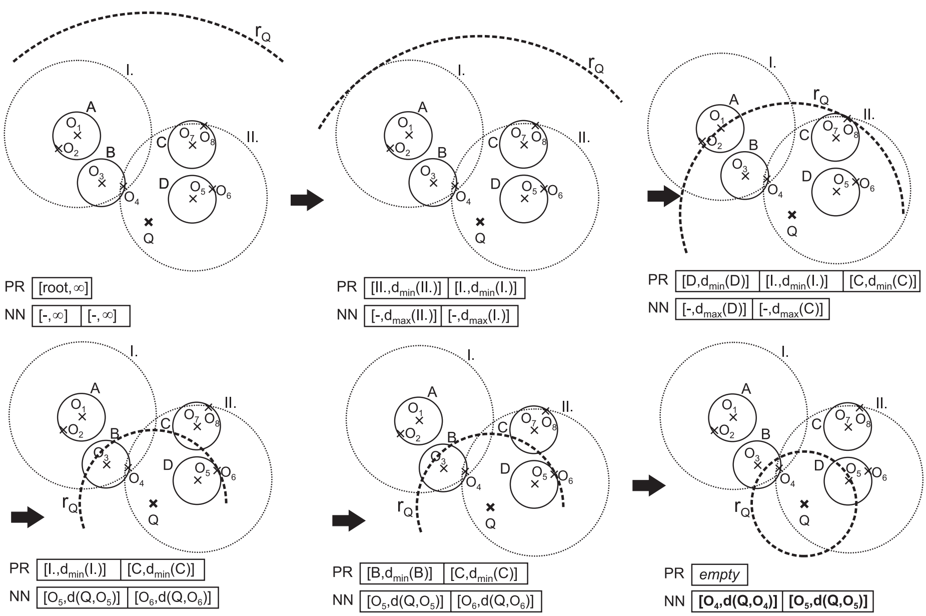

Fig. 2. An example of 2-NN search in M-tree

Example 1

In Figure 2 see an example of 2-NN query processing. Each of the depicted phases shows the content of PR queue and NN array, right before processing a request from PR. Due to the decreasing query radius r_Q, the dynamic query region (Q, r_Q) (represented by bold-dashed line) is reduced down to $(Q, d(Q, O_5))$. Note the algorithm accesses 5 nodes (processing of single request in PR involves a single node access), while the equivalent range query takes also 5 node accesses.

3 PM-Tree

Each metric region in M-tree is described by a bounding hyper-sphere. However, the shape of hyper-sphere is far from optimal, since it does not bound the data objects tightly together and the region "volume" is too large. Relatively to the hyper-sphere volume, there are only "few" objects spread inside the hyper-sphere – a huge proportion of dead space [1] is covered. Consequently, for hyper-spherical regions the probability of overlap with query region grows, thus query processing becomes less efficient. This observation was the major motivation for introduction of the *Pivoting M-tree* (PM-tree) [12, 10], an extension of M-tree.

3.1 Structure of PM-Tree

Some metric access methods (e.g. AESA, LAESA [4, 6]) exploit *global static pivots*, i.e. objects to which all objects of the dataset $\mathbb{S}$ (all parts of the index structure respectively) are related. The global pivots actually represent "anchors" or "viewpoints", due to which better filtering of irrelevant data objects is possible.

In PM-tree, the original M-tree hierarchy of hyper-spherical regions (driven by local pivots) is combined with so-called *hyper-ring regions*, centered in global pivots. Since PM-tree is a generalization of M-tree, we just describe the new facts instead of a comprehensive definition. First of all, a set of p global pivots $P_t \in \mathbb{S}$ must be chosen. This set is fixed for all the lifetime of a particular PM-tree index. A routing entry in PM-tree inner node is defined as:

$$rout_{PM}(O_i) = [O_i, ptr(T(O_i)), r_{O_i}, d(O_i, \mathrm{Par}(O_i)), \mathrm{HR}]$$

The new HR attribute is an array of p_{hr} intervals ($p_{hr} \leq p$), where the t-th interval HR[t] is the smallest interval covering distances between the pivot P_t and each of the objects stored in leaves of $T(O_i)$, i.e. $\mathrm{HR}[t] = \langle \mathrm{HR}[t].\mathrm{min}, \mathrm{HR}[t].\mathrm{max} \rangle$, $\mathrm{HR}[t].\mathrm{min} = min\{d(O_j, P_t)\}$, $\mathrm{HR}[t].\mathrm{max} = max\{d(O_j, P_t)\}$, $\forall O_j \in T(O_i)$. The interval HR[t] together with pivot P_t define a hyper-ring region (P_t,HR[t]); a hyper-spherical region (P_t,HR[t].max) reduced by a "hole" (P_t,HR[t].min).

Since each hyper-ring region (P_t, HR[t]) defines a metric region bounding *all* the objects stored in $T(O_i)$, the intersection of all the hyper-rings and the hyper-sphere forms a metric region bounding *all* the objects in $T(O_i)$ as well. Due to the intersection with hyper-sphere, the PM-tree metric region is always smaller than the original hyper-spherical region. The probability of overlap between PM-tree region and query region is smaller, thus the search becomes more efficient (see Figure 3). A ground entry in PM-tree leaf is defined as:

$$grnd_{PM}(O_i) = [O_i, oid(O_i), d(O_i, \mathrm{Par}(O_i)), \mathrm{PD}]$$

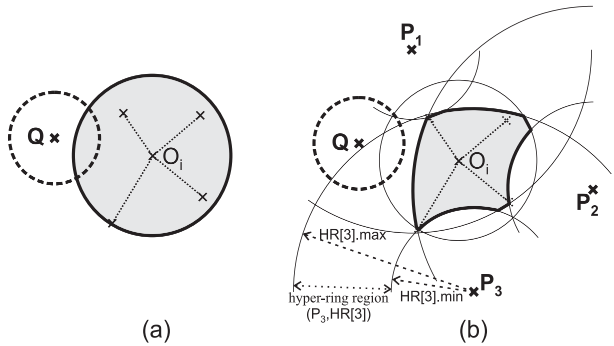

Fig. 3. (a) Region of M-tree. (b) Region of PM-tree (sphere reduced by 3 hyper-rings)

The new PD attribute stands for an array of p_{pd} pivot distances ($p_{pd} \leq p$) where the t-th distance $\mathrm{PD}[t] = d(O_i, P_t)$. The distances $\mathrm{PD}[t]$ between data objects and the global pivots are used for simple sequential filtering in leaves, as it is accomplished in LAESA-like methods. For details concerning PM-tree construction as well as representation and storage of the hyper-ring intervals (HR and PD arrays) we refer to [12, 10].

3.2 Choosing the Global Pivots

Problems about choosing the global pivots have been intensively studied for a long time [9, 3, 2]. In general, we can say that pivots should be far from each other (close pivots give almost the same information) and outside data clusters. Distant pivots cause increased variance in distance distribution [4] (the dataset is "viewed" from different "sides"), which is reflected in better filtering properties.

We use a cheap but effective method of pivots choice, described as follows. First, m groups of p objects are randomly sampled from the dataset $\mathbb{S}$, each group representing a candidate set of pivots. Second, such group of pivots is chosen, for which the sum of distances between objects is maximal.

3.3 Similarity Queries in PM-Tree

The distances $d(Q, P_t), \forall t \leq max(p_{hr}, p_{pd})$ have to be computed before the query processing itself is started. The query is processed by accessing nodes, the regions of which are overlapped by the query region (similarly as M-tree is queried, see Section 2.1). A PM-tree node is accessed if the query region overlaps *all* the hyper-rings stored in the parent routing entry. Hence, prior to the standard hyper-sphere overlap check (used by M-tree), the overlap of hyper-rings $\mathrm{HR}[t]$ against the query region is tested as follows (no additional distance is computed):

$$\bigwedge_{t=1}^{p_{hr}} d(Q, P_t) - r_Q \leq \mathrm{HR}[t].\mathrm{max} \ \wedge \ d(Q, P_t) + r_Q \geq \mathrm{HR}[t].\mathrm{min} \tag{1}$$

If the above condition is false, the subtree $T(O_i)$ is not relevant to the query, and can be excluded from further processing. At the leaf level, an irrelevant ground entry is determined such that the following condition is not satisfied:

$$\bigwedge_{t=1}^{p_{pd}} |d(Q, P_t) - \mathrm{PD}[t]| \leq r_Q \qquad (2)$$

In Figure 3 see that M-tree region cannot be filtered out, but PM-tree region can be excluded from the search, since the hyper-ring HR[2] is not overlapped.

4 Nearest Neighbours Search in PM-Tree

The hyper-ring overlap condition (1) can be integrated into the original M-tree's range query as well as into k-NN query algorithms. In case of range query the adjustment is straightforward – the hyper-ring overlap condition is combined with the original hyper-sphere overlap condition (we refer to [12]).

The M-tree's k-NN algorithm can be modified for the PM-tree, we only need to respect the changed region shape. As in the range query algorithm, the check for overlap between the query region and a PM-tree region is combined with the hyper-ring overlap condition (1). Furthermore, to obtain an *optimal* k-NN algorithm, there must be adjusted the lower-bound distance d_{min} (used by PR queue) and the upper-bound distance d_{max} (used by NN array), as follows.

The requests $[ptr(T(O_i)), d_{min}(T(O_i))]$ in PR represent the relevant subtrees $T(O_i)$ to be examined, i.e. such subtrees, the parent metric regions of which overlap the dynamic query region (Q, r_Q). Taking the hyper-rings HR$[t]$ of a PM-tree region into account, the lower-bound distance is possibly increased, as:

$$d_{min}(T(O_i)) = max\{0, d(O_i, Q) - r_{O_i}, d^{low}_{HRmax}, d^{low}_{HRmin}\}$$

$$d^{low}_{HRmax} = max \bigcup_{t=1}^{p_{hr}} \{d(P_t, Q) - \mathrm{HR}[t].\mathrm{max}\} \quad d^{low}_{HRmin} = max \bigcup_{t=1}^{p_{hr}} \{\mathrm{HR}[t].\mathrm{min} - d(P_t, Q)\}$$

where $max\{d^{low}_{HRmax}, d^{low}_{HRmin}\}$ determines the lower-bound distance between the query object Q and objects located in the farthest hyper-ring. Comparing to M-tree's k-NN algorithm, the lower-bound distance $d_{min}(T(O_i))$ for a PM-tree region can be additionally increased, since the farthest hyper-ring contains all the objects stored in $T(O_i)$.

The entries $[oid(O_i), d(Q, O_i)]$ or $[-, d_{max}(T(O_i))]$ in NN represent the current k candidates for nearest neighbours (or at least the still relevant subtrees). Taking the hyper-rings HR$[t]$ into account, the upper-bound distance $d_{max}(T(O_i))$ is possibly decreased, as:

$$d_{max}(T(O_i)) = min\{d(O_i, Q) + r_{O_i}, d^{up}_{HR}\} \quad d^{up}_{HR} = min \bigcup_{t=1}^{p_{hr}} \{d(P_t, Q) + \mathrm{HR}[t].\mathrm{max}\}$$

where d_{HR}^{up} determines the upper-bound distance between the query object Q and objects located in the nearest hyper-ring.

In summary, the modification of M-tree's k-NN algorithm for the PM-tree differs in the overlap condition, which has to be additionally combined with the hyper-ring overlap check (1) and (2), respectively. Another difference is in the construction of $d_{max}(T(O_i))$ and $d_{min}(T(O_i))$ bounds.

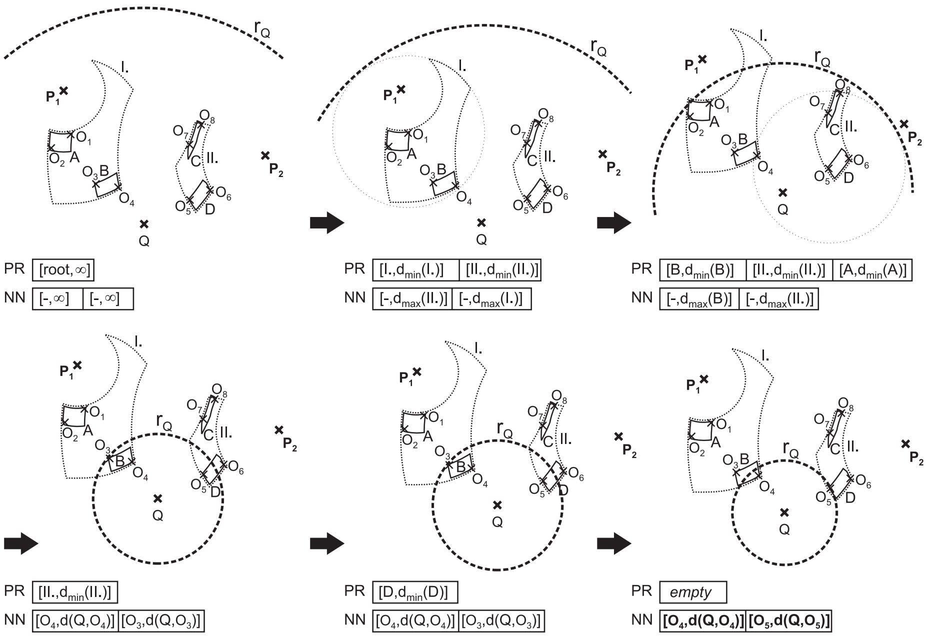

Fig. 4. An example of 2-NN search in PM-tree

Example 2

In Figure 4 see an example of 2-NN query processing. The PM-tree hierarchy is the same as the M-tree hierarchy presented in Example 1, but the query processing runs a bit differently. Although in this particular example both the M-tree's and the PM-tree's k-NN query algorithms access 4 nodes, searching the PM-tree saves one insertion into the PR queue.

Note: Like the M-tree's k-NN query algorithm, also the PM-tree's k-NN query algorithm is optimal in I/O costs, since it only accesses those PM-tree nodes, the metric regions of which overlap the query region $(Q, d(Q,\text{NN}[k].d_{max}))$. This is guaranteed (besides usage of the hyper-ring overlap check) by correct modification of lower/upper distance bounds stored in PR queue and NN array.

5 Experimental Results

In order to evaluate the performance of k-NN search, we present some experiments made on large synthetic as well as real-world vector datasets. The query objects were selected randomly from each respective dataset, while each particular test consisted of 1000 queries (the results were averaged). Euclidean (L_2) metric was used in all tests. The I/O costs were measured as the number of logic disk page retrievals. The experiments were aimed to compare PM-tree with M-tree – a comparison with other MAMs was out of scope of this paper.

Abbreviations in Figures. Each label of form "`PM-tree(x,y)`" stands for a PM-tree index where p_{hr} = `x` and p_{pd} = `y`. A label "<*index*> `+ SlimDown`" denotes an index subsequently post-processed by the slim-down algorithm [11, 10].

5.1 Synthetic Datasets

For the first set of experiments, a collection of 8 synthetic vector datasets of increasing dimensionality (from $D = 4$ to $D = 60$) was generated. Each dataset (embedded inside unitary hyper-cube) consisted of 100,000 D-dimensional tuples

Table 1. PM-tree index statistics (synthetic datasets)

Construction methods:	`SingleWay + MinMax (+ SlimDown)`		
Dimensionalities:	4,8,16,20,30,40,50,60	Inner node capacities:	10 – 28
Index file sizes:	4.5 MB – 55 MB	Leaf node capacities:	16 – 36
Pivot file sizes:	2 KB – 17 KB	Avg. node utilization:	66%
Node (disk page) sizes:	1 KB ($D = 4, 8$), 2 KB ($D = 16, 20$), 4 KB ($D \geq 30$)		

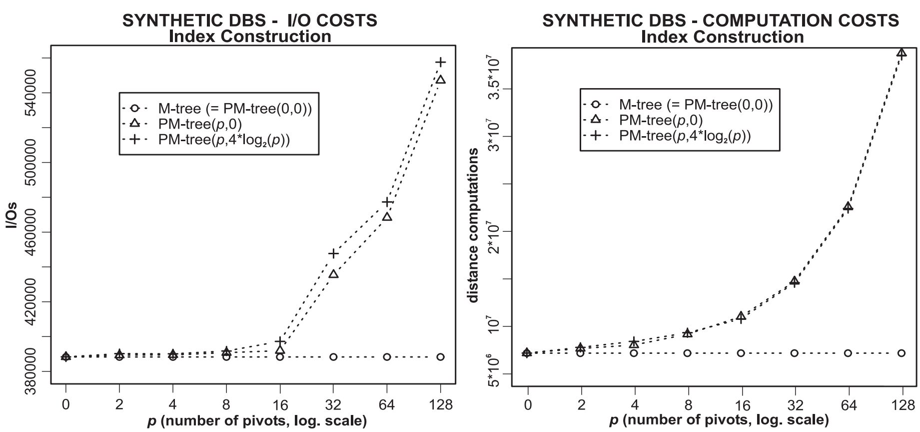

Fig. 5. Number of pivots: (a) I/O costs. (b) Computation costs

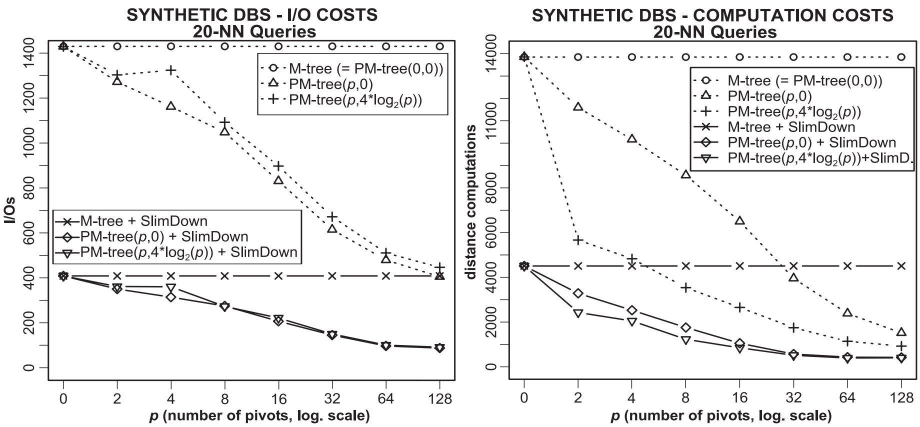

Fig. 6. Number of pivots: (a) I/O costs. (b) Computation costs

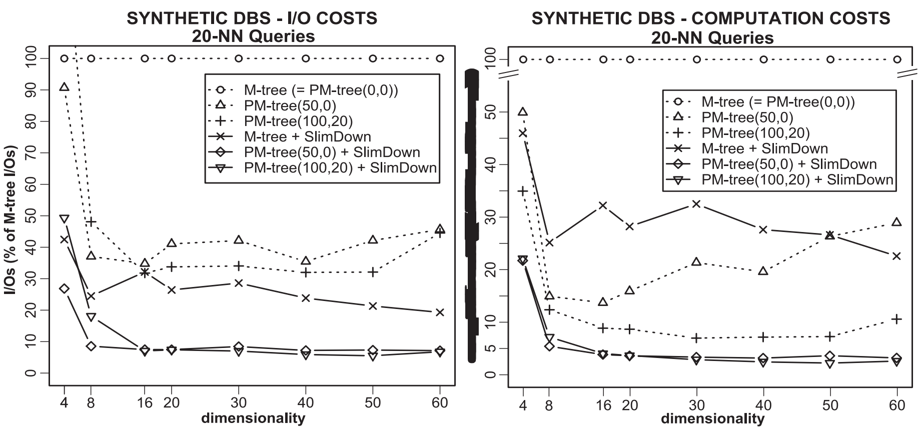

Fig. 7. Dimensionality: (a) I/O costs. (b) Computation costs

distributed uniformly among 1000 L_2-spherical uniformly distributed clusters. The diameter of each cluster was $\frac{d^+}{10}$ (where $d^+ = \sqrt{D}$). These datasets were indexed by PM-tree (for various p_{hr} and p_{pd}) as well as by M-tree. Some statistics about the created indices are shown in Table 1 (for details see [11]). Prior to k-NN experiments, in Figure 5 we present index construction costs (for 30-dimensional indices), according to the increasing number of pivots. The increasing I/O costs depend on the hyper-ring storage overhead (the storage ratio of PD or HR arrays to the data vectors becomes higher), while the increasing computation costs depend on the object-to-pivot distance computations performed before each object insertion.

In Figure 6 the 20-NN search costs (for 30-dimensional indices) according to the number of pivots are presented. The I/O costs rapidly decrease with the increasing number of pivots. Moreover, the PM-tree is superior even after postprocessing by the slim-down algorithm. The decreasing trend of computation costs is even quicker than of I/O costs, see Figure 6b.

The influence of increasing dimensionality D is depicted in Figure 7. Since the disk pages for different (P)M-tree indices were not of the same size, the I/O costs as well as the computation costs are related (in percent) to the I/O costs (CC resp.) of M-tree indices. For $8 \leq D \leq 40$ the I/O costs stay approximately fixed, for $D > 40$ they slightly increase. In case of $D = 4$, the higher PM-tree I/O costs are caused by higher hyper-ring storage overhead.

5.2 Image Database

For the second set of experiments, a collection of approx. 10,000 web-crawled images [13] was used. Each image was converted into 256-level gray scale and a frequency histogram was extracted. As indexed objects the histograms (256-dimensional vectors) were used. The index statistics are presented in Table 2.

Table 2. PM-tree index statistics (image database)

Construction methods:	`SingleWay + MinMax (+ SlimDown)`		
Dimensionality:	256	Inner node capacities:	10 – 31
Index file sizes:	16 MB – 20 MB	Leaf node capacities:	29 – 31
Pivot file sizes:	4 KB – 1 MB	Avg. node utilization:	67%
Node (disk page) size:	32 KB		

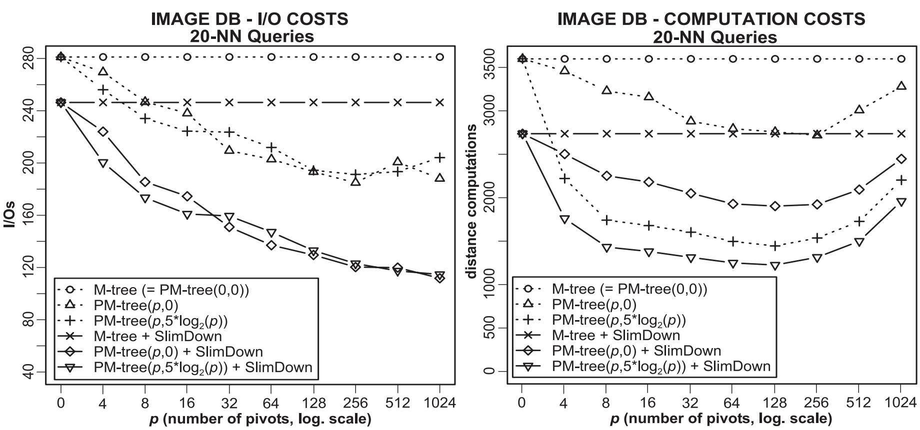

Fig. 8. Number of pivots: (a) I/O costs. (b) Computation costs

In Figure 8a the I/O search costs for increasing number of pivots are presented. The computation costs (see Figure 8b) for $p \leq 64$ decrease. However, for $p > 64$ the overall computation costs grow, since the number of necessarily computed query-to-pivot distances (i.e. p distance computations for each query) is proportionally too large. Nevertheless, this observation is dependent on the

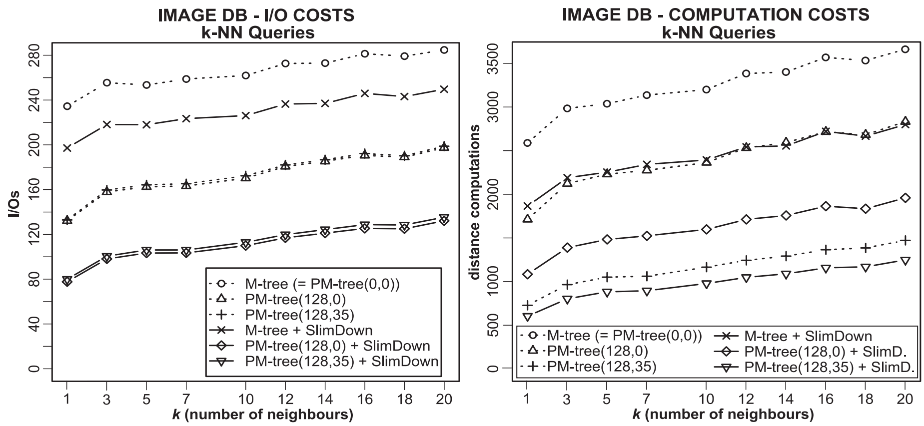

Fig. 9. Number of neighbours: (a) I/O costs. (b) Computation costs

database size – obviously, for million of images the proportion of p query-to-pivot distance computations would be smaller, when compared with the overall computation costs. Finally, the costs according to the increasing number of nearest neighbours are presented in Figure 9.

6 Conclusions

We have proposed an optimal k-NN search algorithm for the PM-tree. Experimental results on synthetic and real-world datasets have shown that searching in PM-tree is significantly more efficient, when compared with the M-tree.

Acknowledgements. This research has been partially supported by grant 201/05/P036 of the Czech Science Foundation (GAČR) and the National programme of research (Information society project 1ET100300419).

References

1. C. Böhm, S. Berchtold, and D. Keim. Searching in High-Dimensional Spaces – Index Structures for Improving the Performance of Multimedia Databases. *ACM Computing Surveys*, 33(3):322–373, 2001.
2. B. Bustos, G. Navarro, and E. Chávez. Pivot selection techniques for proximity searching in metric spaces. *Pattern Recognition Letters*, 24(14):2357–2366, 2003.
3. E. Chávez. Optimal discretization for pivot based algorithms. Manuscript. `ftp://garota.fismat.umich.mx/pub/users/elchavez/minimax.ps.gz`, 1999.
4. E. Chávez, G. Navarro, R. Baeza-Yates, and J. Marroquín. Searching in Metric Spaces. *ACM Computing Surveys*, 33(3):273–321, 2001.
5. P. Ciaccia, M. Patella, and P. Zezula. M-tree: An Efficient Access Method for Similarity Search in Metric Spaces. In *Proceedings of the 23rd Athens Intern. Conf. on VLDB*, pages 426–435. Morgan Kaufmann, 1997.

6. 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*, 15(1):9–17, 1994.
7. M. Patella. *Similarity Search in Multimedia Databases.* PhD thesis, University of Bologna, 1999.
8. N. Roussopoulos, S. Kelley, and F. Vincent. Nearest neighbor queries. In *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data, San Jose, CA*, pages 71–79, 1995.
9. M. Shapiro. The choice of reference points in best-match file searching. *Commun. ACM*, 20(5):339–343, 1977.
10. T. Skopal. *Metric Indexing in Information Retrieval.* PhD thesis, Technical University of Ostrava, `http://urtax.ms.mff.cuni.cz/~skopal/phd/thesis.pdf`, 2004.
11. T. Skopal, J. Pokorný, M. Krátký, and V. Snášel. Revisiting M-tree Building Principles. In *Proceedings of the 7th East-European Conference on Advances in Databases and Information Systems (ADBIS), Dresden, Germany, LNCS 2798, Springer-Verlag*, pages 148–162, 2003.
12. T. Skopal, J. Pokorný, and V. Snášel. PM-tree: Pivoting Metric Tree for Similarity Search in Multimedia Databases. In *Local proceedings of the 8th East-European Conference on Advances in Databases and Information Systems (ADBIS), Budapest, Hungary*, pages 99–114, 2004.
13. WBIIS project: Wavelet-based Image Indexing and Searching, Stanford University, `http://wang.ist.psu.edu/`.