Clustered Pivot Tables for I/O-optimized Similarity Search

Juraj Moško
Charles University in Prague, Faculty of Mathematics and Physics, SIRET research group
mosko.juro@centrum.sk

Jakub Lokoč
Charles University in Prague, Faculty of Mathematics and Physics, SIRET research group
lokoc@ksi.mff.cuni.cz

Tomáš Skopal
Charles University in Prague, Faculty of Mathematics and Physics, SIRET research group
skopal@ksi.mff.cuni.cz

ABSTRACT

The pivot tables are a popular metric access method, primarily designed as a main-memory index structure. It has been many times proven that pivot tables are very efficient in terms of distance computations, hence, when assuming a computationally expensive distance function. However, for cheaper distance functions and/or huge datasets exceeding the capacity of the main memory, the classic pivot tables become inefficient. The situation is dramatically changing with the rise of solid state disks that decrease the seek times, so we can now efficiently access also small fragments of data stored in the secondary memory. In this paper, we propose a persistent variant of pivot tables, the clustered pivot tables, focusing on minimizing I/O cost when accessing small data blocks (a few kilobytes). The clustered pivot tables employs a preprocessing method utilizing the M-tree in the role of clustering technique and an original heuristic for I/O-optimized kNN query processing. In the experiments we empirically show that our proposed method significantly reduces the number of necessary I/O operations during query processing.

1. INTRODUCTION

The explosive growth of complex multimedia data including images, videos, and music challenges the effectiveness and efficiency of today's multimedia databases. In order to provide the users with access and insight into these inevitably increasing masses, multimedia databases have to manage data objects effectively and appropriately with respect to content-based retrieval. When searching multimedia databases in a content-based way, users issue similarity queries by selecting multimedia objects or by sketching the intended object contents. Given an example multimedia object or sketch, the multimedia database searches for the most related objects with respect to the query by measuring the similarity between the query and each database object by means of a distance function. As a result, the multimedia objects with the lowest distance to the query are returned to the user. In order to answer a similarity query as fast as possible, there were various indexing structures proposed. The most established class of such indexes is the class of metric access methods, formed by database structures and algorithms aiming at minimizing the distance computations and I/O operations needed to answer a query. The only restriction of metric access methods is that they require the distance function to be a metric.

SISAP '11, June 30-July 1, 2011, Lipari, Italy.

1.1 Metric access methods

A *metric space* $(\mathbb{U}, \delta)$ consists of a feature representation domain $\mathbb{U}$, and a distance function δ which has to satisfy the metric postulates: *identity*, *symmetry*, and *triangle inequality*. In this way, metric spaces allow domain experts to model their notion of content-based similarity by an appropriate feature representation and distance function serving as similarity measure. At the same time, this approach allows database experts to design index structures, so-called *metric access methods* (MAMs) or *metric indexes* [4, 20, 14], for efficient query processing of content-based similarity queries in a database $\mathbb{S} \subset \mathbb{U}$. These methods rely on the distance function δ only, i.e., they do not necessarily know the structure of the feature representation of the objects.

Metric access methods organize database objects $o_i \in \mathbb{S}$ by grouping them based on their distances, with the aim of minimizing not only traditional database costs like I/O but also the number of costly distance function evaluations. For this purpose, nearly all metric access methods apply some form of filtering based on cheap lower bounds. For the case of pivoting, these bounds are based on the fact that exact pivot–object distances are pre-computed.

We illustrate this fundamental principle in Figure 1 where we depict the query object $q \in \mathbb{U}$, some pivot element $p \in \mathbb{S}$, and a database object $o \in \mathbb{S}$ in some metric space. Note that pivot elements are used to group database objects and to improve the efficiency of the search process by pruning

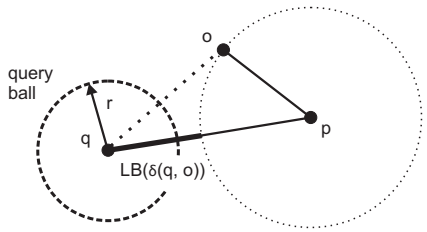

Figure 1: The lower-bounding principle.

whole parts of the index structure. Given a range query (q, r), we aim at estimating the distance $\delta(q, o)$ by making use of $\delta(q, p)$ and $\delta(o, p)$, with the latter already stored in the metric index. Due to the triangle inequality, we can safely filter object o without the need of (costly) computing $\delta(q, o)$ if the triangular lower bound

$$\delta_{\mathrm{T}}(q, o) = |\delta(q, p) - \delta(o, p)|\,, \tag{1}$$

also known as the *inverse triangle inequality*, is greater than the query radius r.

1.2 Paper contribution

In this paper we propose a new variant of the pivot tables method, the clustered pivot tables, optimized for efficient persistent metric indexing. The main contributions of this paper can be summarized as:

- We propose a new method of persistent metric indexing based on the classic pivot tables. The method employs the M-tree as a general metric clustering method.
- We have designed a new I/O friendly heuristic for kNN query processing.
- We have experimentally evaluated the proposed methods on four different datasets, where we demonstrated that the clustering approach has positive effect on I/O cost spent during query processing.

In the following section we review the related work that we utilize and combine in this paper. In section 3 we detail the principles of the proposed method. After that, we report and discuss experimental results in section 4 and finally we conclude contributions of our method in section 5.

2. RELATED WORK

In this section we overview two MAMs that we use in our approach – the M-tree and the Pivot tables.

2.1 M-tree

The *M-tree* [6] is a dynamic index structure that provides good performance in secondary memory (i.e., in database environments). The M-tree is a hierarchical index, where some of the data objects are selected as centers (local pivots) of ball-shaped regions, while the remaining objects are partitioned among the regions in order to build up a balanced and compact hierarchy of data regions, see Figure 2. Each region (subtree) is indexed recursively in a B-tree-like (bottom-up) way of construction.

The inner nodes of M-tree store *routing entries* $rout_l(o_i) = [o_i, rad_{o_i}, \delta(o_i, Par(o_i)), ptr(T(o_i))]$, where $o_i \in \mathbb{S}$ is a data object representing the center of the respective ball region, rad_{o_i} is a *covering radius* of the ball, $\delta(o_i, Par(o_i))$ is the so-called *to-parent* distance (the distance from o_i to the object of the parent routing entry), and finally $ptr(T(o_i))$ is a pointer to the entry's subtree. The data is stored in the leaves of M-tree. Each leaf contains *ground entries* $grnd(o_i) = [o_i, \delta(o_i, Par(o_i))]$, where $o_i \in \mathbb{S}$ is an indexed database object and $\delta(o_i, Par(o_i))$ is, again, the to-parent distance.

Range and kNN queries are implemented by traversing the tree, starting from the root. Those nodes are accessed, whose parent regions (described by the routing entries) are overlapped by the query ball (q, rad). In case of a kNN query the radius rad is not known beforehand, so we have to additionally employ a process to dynamically decrease the radius during the search (initially set to ∞). The kNN algorithm performs a best-first traversal of the index, where regions are accessed in the order of increasing lower bound distance to q.

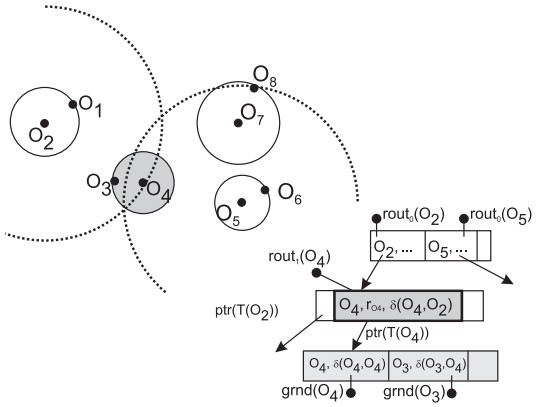

Figure 2: M-tree (hierarchical space decomposition and the tree structure).

2.2 M-tree construction

The M-tree index is created dynamically in the bottom-up fashion like the B-tree. The insertion of a new object starts in the root node and tries to find a suitable leaf node for the new object using a leaf selection strategy. If the found leaf node is overfull the node is split, moreover, the split may cause chain of other splits of the corresponding parent nodes. In general, during splitting the two new nodes are created and all the objects from the split node are divided within the two new nodes using a splitting policy. For more details about splitting policies see [6].

The static way of M-tree construction, motivated by speeding up the construction process, is to load more data at once (i.e., bulk loading). For more details about the idea of bulk loading see [5, 15] as this is out of scope of this paper.

When selecting a suitable leaf node for a dynamic object (re)insertion, we could choose from the following techniques.

2.2.1 Single-way leaf selection

The single-way leaf selection strategy considers a single path from the root node to a suitable leaf node. At each level of the tree the strategy deterministically selects one best subtree for the inserted object (the subtrees are represented by the routing items in the actually processed node). Such a subtree is preferred the covering radius of which would stay unchanged after insertion of the new object. Furthermore, if more such subtrees exist, the one with the closest center to the inserted object is chosen. For more details see [6].

2.2.2 Multi-way leaf selection

The motivation for the multi-way leaf selection strategy [18] is to take globally the single-way effort to minimize the enlargements of the M-tree regions. The multi-way algorithm considers nondeterministically all paths related to the inserted object instead of just one path as the single-way method. First, a point query with the inserted object as a

query object is issued and all possible candidate leaf nodes for the new object are returned. Then, the leaf node the parent routing object of which is the closest to the inserted object is chosen. In case when the point query does not return any leaf node the single-way leaf selection is used. Although this approach creates more tight and less overlapped M-tree regions, the indexing cost becomes an order of magnitude more expensive.

2.2.3 *Forced reinserting*

The method of forced reinserting [9, 17] concerns splitting of the tree nodes. Forced reinserting tries to postpone splitting by reinserting some of the objects from the overfull leaf node in a hope they will be inserted into other (non-overfull) leaf nodes. There are several selection strategies for reinserted objects, for more details see [9]. Since object reinsertion may lead to another split (leading again to reinserting), the recursion depth limiting the number of reinsertion attempts is required as the parameter. When the recursion depth is reached, all remaining objects are inserted in the standard way, where nodes are split if necessary. This approach also leads to more tight and less overlapping regions, but the indexing cost is not as increased as in the case of the multi-way leaf selection.

2.3 Pivot tables

One of the most efficient (yet simple) metric indexes is the *pivot table* [12], originally introduced as LAESA [11]. Basically, the structure of a pivot table is a simple matrix of distances $\delta(o_i, p_j)$ between the database objects $o_i \in \mathbb{S}$ and a pre-selected static set of m pivots $p_j \in \mathbb{P} \subset \mathbb{S}$. For querying, pivot tables allow us to perform cheap lower-bound filtering by computing the maximum lower bound (Eq. 1) to $\delta(q, o)$ using all the pivots.

From a more intuitive perspective, pivot tables index the database objects as m-dimensional vectors in a pivot space. When querying, the range query ball (q, r) (or kNN ball with the current radius) is mapped into the pivot space, such that its center is $(\delta(q, p_1), \delta(q, p_2), \ldots, \delta(q, p_m))$. An important property of the mapping is that δ in the original space is lower-bounded by L_∞ distance in the pivot space (i.e., it is a non-expansive mapping). The query ball in the pivot space (i.e., the L_∞-ball of radius r) can therefore be used to retrieve all the objects inside the query ball in the original space, possibly with some false positives that must be filtered out by δ in a refinement step. See Figure 3 for an illustration of the pivot-based mapping from the original space into the pivot space, and the respective query balls.

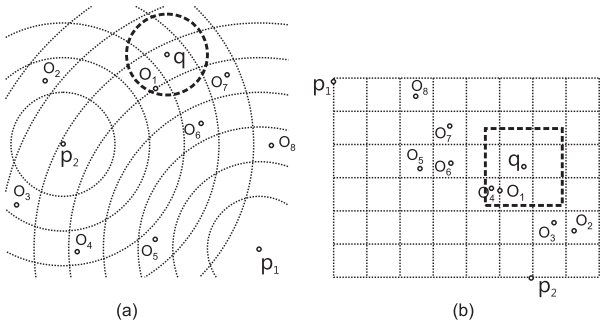

Figure 3: (a) Original space (b) Pivot space

The concept of LAESA was implemented many times under different conditions, we name, e.g., TLAESA [10] (pivot table indexed by GH-tree-like structure), Spaghettis [3] (pivot table indexed by multiple sorted arrays), OMNI family [19] (pivot table indexed by R-tree) and PM-tree [16] (hybrid approach combining M-tree and pivot tables).

3. CLUSTERED PIVOT TABLES

In this section, we introduce a new persistent variant of pivot tables, the *clustered pivot tables*, employing the M-tree as a preprocessing clustering algorithm. The structure of the clustered pivot tables consists of two separate parts – the first part, the *distance matrix* of precomputed distances between all the database objects and a static set of pivots, is located in the main memory, while the second part is a *persistent datafile* containing data objects (usually large) organized in the disk pages of fixed size. The goal of our method is to reduce the I/O cost during pivot table query processing, i.e., the number of necessarily read disk pages storing data objects. In the following subsections we propose static and dynamic variants of clustered pivot tables. We also discuss the kNN query processing, because when considering also I/O cost, the LAESA-based kNN query processing is not optimal. We do not consider caching of disk pages and pivot selection techniques [11, 2, 13], although they can further improve the performance of our method. We also assume that the whole distance matrix fits into the main memory (but not yet the data objects).

3.1 Motivation

In general, we have to assume that objects from the input dataset arrive in random order, while in the classic approach (denoted as the *non-clustered pivot tables*) the objects are just appended one after the other to the datafile (and the respective vectors to the distance matrix). When a query is being processed, the distance matrix is traversed to compute the lower-bound distances to all the database objects, while the pages of the non-filtered candidate objects must be fetched from the datafile. Because in the classic approach there is a high probability that each of the non-filtered candidates will occupy a different disk page (because similar objects are not clustered in datafile), the entire processing would exhibit a high I/O cost.

3.2 Clustering the datafile

To avoid the high I/O cost, we have focused on clustering the datafile, because there is a higher probability that lower-bounding could filter all the objects located in the clustered disk pages, which would lead to lower I/O cost. We decided to employ M-tree as the clustering method, because it provides many dynamic techniques organizing objects within compact clusters. We focus on the dynamic M-tree construction techniques, which influence the number and quality of the created clusters. In particular, we consider single-way and multi-way leaf selection strategies and the forced reinsertions. After construction of the M-tree we have all data within clustered index and from this point we consider two ways of creating either static or dynamic variant of the clustered pivot tables.

3.3 Static Clustered Pivot Tables

The first proposal (depicted in Figure 4) is the classic pivot table index which just reorganizes input dataset using the

M-tree – data objects from M-tree leaves (whole clusters) are serialized into the datafile. The datafile is paged, where the page size is defined as a parameter before the construction. Let us denote that the mapping between M-tree leaf node and datafile block is not one to one, because M-tree leaf nodes have not 100% utilization. Because of uneven M-tree leaf utilization, the created clusters contain different number of objects and because the datafile block has a fixed size (the same as a full M-tree leaf), one block within the datafile could contain data from more clusters (M-tree leaves) than one. Although this variant considers static creation of the datafile, dynamic insertions to the datafile and distance matrix are possible – the clusters just become less compact. Moreover, the M-tree could be maintained up-to-date during the dynamic insertions, so that a possible future reindexing (new datafile creation) could be cheaper.

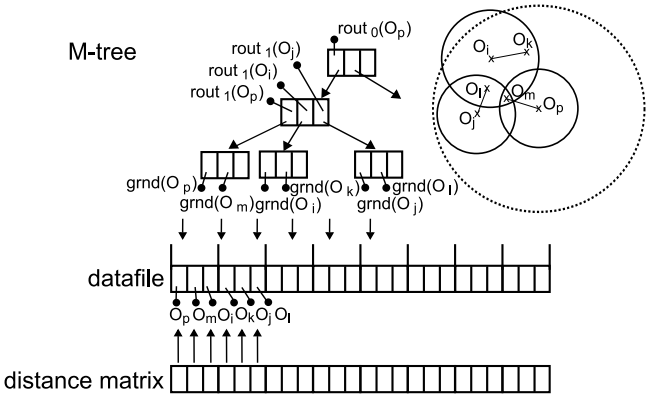

Figure 4: Clustered PT – data in sequential file

3.4 Dynamic Clustered Pivot Tables

In the second proposal (depicted in Figure 5), we do not copy data objects from the M-tree to a separate datafile, but we consider the set of M-tree leaves as the datafile itself. The M-tree is here not only a clustering algorithm but also a persistent dynamic part of the index. Thus, the dynamic clustered pivot tables consist from the distance matrix connected to the M-tree, where the vectors in the distance matrix are grouped according to the corresponding leaf nodes. Let us denote that for the dynamic variant (data in the M-tree), the number of disk pages is greater than for the static index type (separated datafile) due to lower utilization of the M-tree leaf pages. On the other hand, we expect that more compact clusters will lead to more efficient filtering of the corresponding disk pages.

3.5 Querying

The main idea behind efficient query processing in the clustered pivot tables is to minimize the volume of data retrieved from the disk (we consider solid state disks with fast seek time, see the discussion in section 4.4). As the proposed variants of clustered pivot tables group similar objects within the datafile (or M-tree leaves), many of the data pages could be filtered out because the clusters they represent do not overlap with the query region.

The range query processing is nearly the same as for the original pivot tables. The only difference is that for non-filtered objects we have to load the corresponding pages from

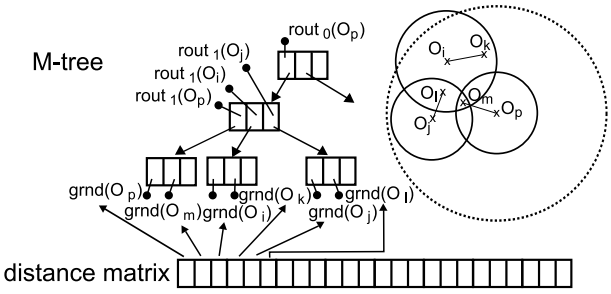

Figure 5: Clustered PT – data within M-tree leaves

the disk. However, for kNN query we have to consider completely different heuristic for query processing if we want to minimize also the I/O cost. The original algorithm is optimal in terms of minimal distance computations spent, but it can lead to high I/O cost because the same page can be accessed many times during the query processing. Although, the I/O cost for kNN query can be reduced by a disk page cache, the overall cost can be higher than for serial processing. Thus, to prevent multiple accesses to disk pages, we do not use reordering of objects according to their lower bound distance to the query object (as the original LAESA algorithm does).

Unfortunately, the I/O cost benefits caused by storing similar objects together turns into a drawback when the query radius is computed dynamically (in kNN search). The kNN query algorithm simply processes objects from 'left to right' in the order the objects are stored within the leaves of the M-tree (or pages of datafile). This means all objects from a cluster are processed before another cluster is entered. In case the early processed clusters are far from the query, the dynamic query radius can decrease to the final value very slowly. This may result in higher number of retrieved disk pages and also to more distance computations. Therefore, we keep data in first few disk pages unsorted and non-clustered, while the rest of the dataset is reordered by the M-tree. When the kNN query tries to determine the dynamic query radius, it first processes objects from the non-clustered part (randomly distributed through the whole space) so that the radius is decreased quickly. A similar method of initial query radius approximation based on non-indexed data was proposed in [8].

4. EXPERIMENTAL RESULTS

We have performed experiments with four methods of M-tree construction in the preprocessing phase and two approaches of organization of the clustered pivot tables index – the dynamic and static. According to the results of the first set of experiments, we chose one method for building M-tree and performed several tests comparing our proposal to the non-clustered pivot tables (pivot tables without M-tree preprocessing). The implementation of the non-clustered pivot tables method is the same as the proposed static method which has pages of fixed size and data stored in the separate datafile. As we have mentioned, our proposal is aimed at the minimal number of I/O operations, so I/O cost is the only value we present in the graphs. The kNN heuristic employing lower-bound ordering (original LAESA algorithm) was two times faster in distance computations than our kNN

algorithm (one-pass scan with the initial query radius approximation), however, around five times worse in I/O cost.

4.1 The testbed

We tested our method on two real-world datasets (Cophir, ColorHistogram) and two synthetic datasets (PolygonSet, Cloud). For both real-world and synthetic datasets we have selected data with lower and higher intrinsic dimensionality.

We used a subset of the *CoPhIR* database [1] consisting of 1,000,000 feature vectors representing 64-dimensional MPEG7 color structure descriptor. As a distance function the Euclidean (L_2) distance was used. The second real-world dataset, the ColorHistogram, was a subset of the *Corel* database [7] and consisted of 68,040 32-dimensional vectors representing color histograms of the images. We have considered Euclidean (L_2) distance also for this dataset.

The PolygonSet was a synthetic database of 250,000 2D polygons, each polygon consisted of 5-15 vertices. The second synthetic database, the Cloud, consisted of 110,000 clouds of 60 6-dimensional points, where all these points were from a unitary 6D cube. The first point (the center of cloud) was generated at random and the rest of the cloud points were generated around the center under normal distribution. The synthetic databases consisting of non-vectorial data were tested as an alternative to normally distributed vector datasets. Both of the synthetic databases used the Hausdorff distance with L_2 ground distance.

If not mentioned otherwise, in the experiments we used all data from the ColorHistogram and the Cloud, while concerning Cophir and PolygonSet we used random subset consisting of 150,000 data objects. For each test, 100 different query objects were considered and the result cost was an average. These query objects were distributed in space according to dataset distribution, but they were not presented in the respective dataset.

Label	Description
PT(n, c)	PT indexing non-clustered data
CPT-SF(n, c, *M-tree p.*)	PT indexing clustered data stored in sequential datafile (static)
CPT-MT(n, c, *M-tree p.*)	PT indexing clustered data stored in M-tree leaves (dynamic)
	n is the number of pivots
	c is the data page capacity
M-tree p.	can be (MW|SW|MW RI|SW RI)
	MW means Multi-way leaf selection
	SW means Single-way leaf selection
	RI means forced reinserting
range query(k)	range query with approximately k objects in result
sorted by LB	kNN query (LAESA) sorts objects before filtering by their lower bound of real distance

Table 1. Labels used in the figures.

4.2 Experiment settings

At first we have considered four variants of M-tree construction. Single-way and multi-way leaf selection was tested both with or without forced reinserting. The parameters of forced reinserting were set as follows – recursion depth was set to 10 and the maximal number of reinserted objects was set to 5. For the M-tree splitting policies the mM_RAD method was employed and the minimal utilization of the M-tree nodes was set to 20%.

Since we compare our proposal to the classic pivot tables, we tested all methods with the same size of the disk page. Size of the disk page for each dataset was adapted to keep approximately 32 database objects, that is, 4kB for the ColorHistogram and the PolygonSet, 8kB for the Cophir and 44kB for the Cloud. In tests with varying page size the values varied from 16kB to 64kB. In all tests we used randomly selected set of pivots, where the number of pivots varied from 10 to 90. The number of results of range and kNN queries varied from 1 to 256 objects. The number of non-clustered objects at the beginning of indexed data (used for kNN query radius approximation) was set to 2% of the dataset volume for each test. For better understanding the graphs we have formed a set of labels for the tested methods and their parameters, see Table 1.

4.3 The results

4.3.1 The M-tree variants

In the first set of experiments (see Figure 6 and Figure 7) we tested querying performance for different methods of M-tree construction. The multi-way leaf selection has beaten the single-way approach in all tests. For dynamic clustered pivot tables the CPT-MT(30,32,SW RI) behaves like CPT-MT(30,32,MW), where CPT-MT(30,32,MW RI) is the best and CPT-MT(30,32,SW) the worst strategy, according to the query costs. As we can see, the construction cost (see Table 2) of M-tree built using the multi-way leaf selection is an order of magnitude higher than using the single-way leaf selection. For this reason, we have performed the following tests using the single-way leaf selection method with forced reinserting only.

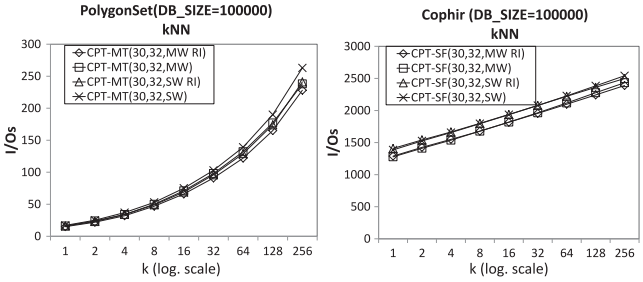

Figure 6: kNN query, size of query result: (a) data in M-tree leaves (b) data in sequential file.

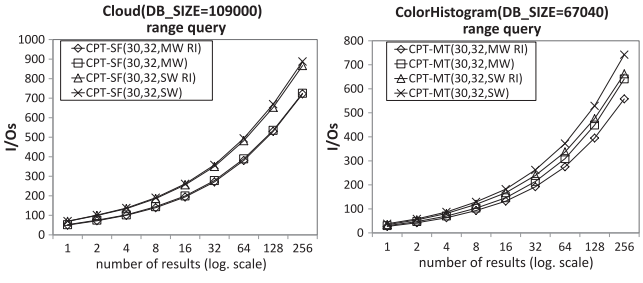

Figure 7: Range query, size of query result: (a) data in sequential file (b) data in M-tree leaves.

Dataset	SW	SW RI	MW	MW RI
ColorHist.	4,115,903	6,249,718	49,557,479	86,196,298
PolygonSet	6,164,869	9,051,290	17,572,966	29,340,106
Cloud	7,829,020	11,313,893	103,194,717	164,596,502
Cophir	6,937,617	10,163,521	191,559,936	304,081,707

Table 2. Construction cost for Figure 6 and Figure 7.

4.3.2 *Number of pivots and disk page size*

In the second set of tests we have examined the behavior of the methods for varying number of pivots (see Figure 8). For increasing number of pivots both dynamic and static methods become very similar to each other. The optimal number of pivots is somewhere around 30 pivots, for lower values the number of I/O operations heavily increases and for higher values the effect is negligible. We can also observe that for range queries with small number of results (i.e., small query radius) and a small number of pivots the improvement on the real and synthetic datasets is significant.

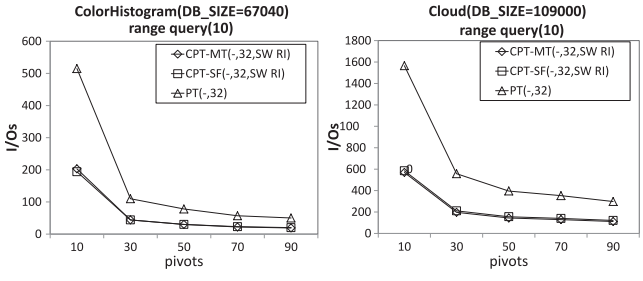

Figure 8: Number of pivots: (a) real-world dataset (b) synthetic dataset.

Furthermore, we have inspected the varying disk page size. The higher number of objects in the disk page can positively impact the clustering (more clusters are filtered out) by adding more similar objects in one place. However, it can also negatively lead to regions that cover greater unnecessary space and then the effect of clustering is not so significant. From this test (see Figure 9) we can observe the positive effect of increasing number of objects in the disk page in our synthetic databases, being of lower intrinsic dimensionality. However, in the CoPhIR datasets the negative effect has taken place due to high intrinsic dimensionality of this real-world dataset.

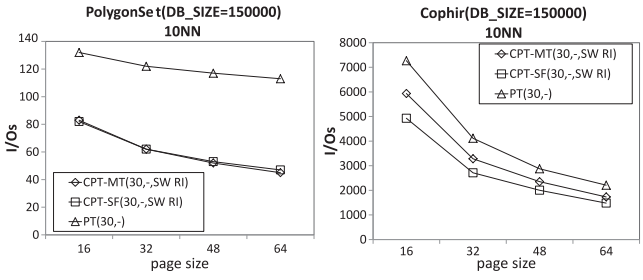

Figure 9: Disk page size: (a) synthetic dataset (b) real dataset.

4.3.3 *Database size and query selectivity*

In the third set of tests we have examined the standard LAESA approach for kNN search, which means to order the indexed objects according to their estimated lower-bound distances and in this order to process all the database objects. We did not consider this approach in our heuristics because with sorting according to the lower bound the I/O cost heavily increases for higher query radius. To illustrate this effect, in Figure 10 see the kNN results for the proposed heuristics including the lower-bound sorting. We can observe total elimination of the positive clustering effect, since clustered and non-clustered pivot tables behave the same.

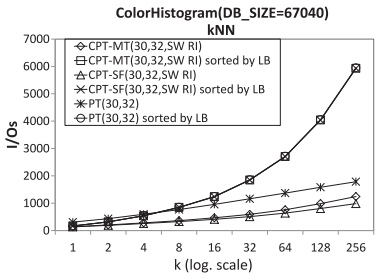

Figure 10: Size of query result – comparison with LAESA-like kNN query.

In the following experiment, we have examined the impact of the growing database. In this test we have also compared two variants of M-tree preprocessing. As we can see in Figure 11, for range query the impact of clustering to eliminate unwanted regions is increasing with growing database. Unfortunately, kNN queries do not follow this effect. From the left graph in Figure 11 we can see the Multi-way leaf selection method is only slightly better than the Single-way leaf selection method with forced reinserting.

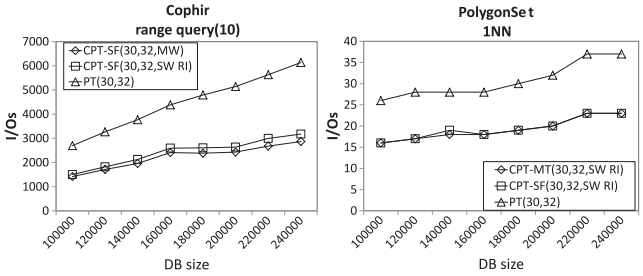

Figure 11: Database size: (a) range query (b) kNN query.

In the next test we have examined increasing size of query result (query selectivity). That is, either directly k in case of kNN queries, or range queries with query radius set to cover k objects, i.e., kNN processed as a range query. We can see (Figure 12) that the effect of clustering is positively increasing (w.r.t. PT) with the increasing query selectivity. Concerning kNN queries, the positive effect of clustering (w.r.t. PT) is significant just for lower values of k. We can observe from the next test that there exists an optimal query selectivity also for range queries.

In another test (see Figure 13) we have observed how the proposed methods cope with large dataset – we used

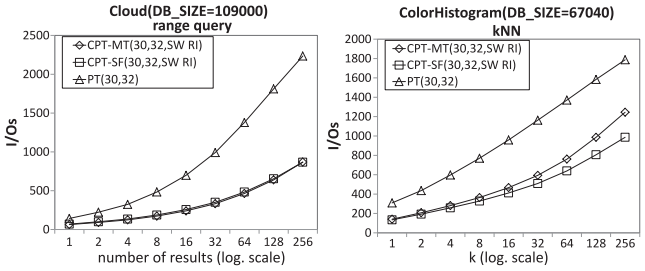

Figure 12: Size of query result: (a) range query (b) kNN query.

1,000,000 vectors from CoPhIR database. We can observe that our method shows better performance for range queries than for kNN queries. With the increasing size of query result, the clusters of the dynamic clustered pivot tables lose their positive effect on decreasing I/O cost faster than for the static clustered pivot tables. When performing kNN query, some clusters that are within dynamic query radius and not within the final query radius have to be also fetched from the datafile/M-tree. Due to these "false positive" clusters, the number of I/O operations for kNN queries is higher than for range queries.

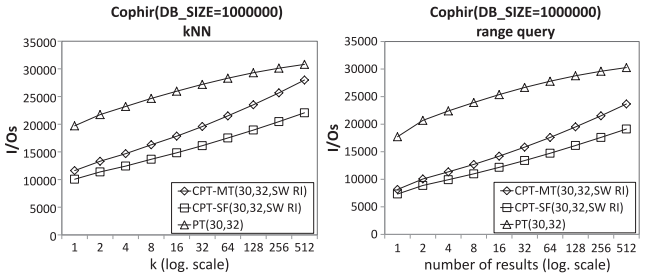

Figure 13: Size of query result on large dataset: (a) range query (b) kNN query.

4.3.4 *Comparison to M-tree*

In the last test we have compared our proposal to the M-tree. We have compared the number of performed I/O operations while querying. As we can see in Figure 14 the proposed method has reached better results for range queries than M-tree within all tested parameters. However, for kNN queries the proposed method was better only for lower values of k.

4.4 Discussion

The experiments proved that the proposed clustered pivot tables can significantly improve the I/O-efficiency of query processing if we use the M-tree in the role of a clustering method. In all tests the clustered pivot tables beat the classic non-clustered version of pivot tables for all tested parameters. However, there are also disadvantages of the new method, which can be summarized as:

- We have to construct and manage two MAMs. Moreover, M-tree indexing is more expensive to manage.
- If we use storage devices with slow seek time, it is more efficient to read one large block from the secondary memory instead of random access to multiple small disk pages. Nevertheless, we explicitly assume zero seek-time storage devices, such as the solid state drives (SSD).
- If the distance function is really expensive, the I/O cost is negligible and the LAESA-based kNN query processing is then more efficient.
- If the distance space suffers from high intrinsic dimensionality, any compact clustering is impossible and the preprocessing by M-tree becomes just an overhead.

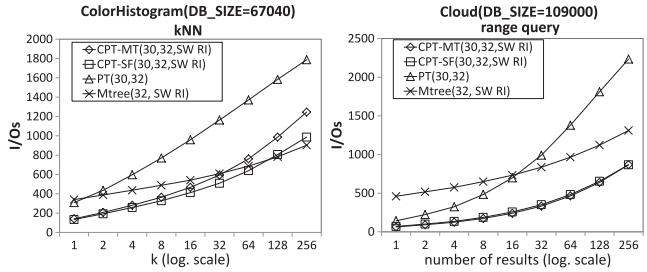

Figure 14: Comparison to M-tree: (a) kNN query (b) range query.

This study could also encourage to adapt other MAMs for indexing large datasets by separating the storage of data objects from the index information itself. We have demonstrated that separation of data objects from the index structure can be useful and that we can use just the precomputed distance matrix for avoiding I/O operations. Let us also denote, that the static version beats proposed dynamic version, so utilizing underfull disk pages makes no sense. However, when the dynamic updates become frequent, the M-tree could be successfully used also as a page index. Several M-tree constructing techniques were considered, while although the Multi-way leaf selection is a better choice for querying, we can utilize this method only in the case when high construction costs are not a problem. The forced reinserting used by M-tree is thus better choice, because it is much cheaper technique which also improves query processing. In the future we would like to compare M-tree, classic pivot tables and the clustered pivot tables for really huge datasets, using SSD devices.

5. CONCLUSIONS

In this paper we have proposed the clustered pivot tables method, which employs the M-tree leaf regions to serialize the database objects into the data part of the index. The usage of clustered data structure within pivot tables fulfilled our goal to decrease the number of I/O operations spent during query processing. In the future research we plan to adapt also other metric access method by using separated storage of the data objects and the index information. This strategy together with modern storage devices with zero seek-time access (such as solid state drives) could lead to a new class of metric access methods optimal in I/O cost.

Acknowledgments

This research has been supported by Czech Science Foundation (GAČR) projects 201/09/0683 and 202/11/0968.

6. REFERENCES

[1] 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*, abs/0905.4627v2, 2009.

[2] B. Bustos, O. Pedreira, and N. Brisaboa. A dynamic pivot selection technique for similarity search. *Similarity Search and Applications, International Workshop on*, 0:105–112, 2008.

[3] E. Chávez, J. L. Marroquín, and R. Baeza-Yates. Spaghettis: An array based algorithm for similarity queries in metric spaces. In *Proceedings of the String Processing and Information Retrieval Symposium & International Workshop on Groupware*, SPIRE '99, pages 38–46, Washington, DC, USA, 1999. IEEE Computer Society.

[4] E. Chávez, G. Navarro, R. Baeza-Yates, and J. L. Marroquín. Searching in metric spaces. *ACM Computing Surveys*, 33(3):273–321, 2001.

[5] P. Ciaccia and M. Patella. Bulk loading the m-tree. In *In Proceedings of the 9th Australasian Database Conference (ADC'98*, pages 15–26, 1998.

[6] P. Ciaccia, M. Patella, and P. Zezula. M-tree: An Efficient Access Method for Similarity Search in Metric Spaces. In *VLDB'97*, pages 426–435, 1997.

[7] S. Hettich and S. D. Bay. The UCI KDD archive. http://kdd.ics.uci.edu, 1999.

[8] L. Jin, N. Koudas, and C. Li. Nnh: Improving performance of nearest-neighbor searches using histograms. In *EDBT*, pages 385–402, 2004.

[9] J. Lokoc and T. Skopal. On reinsertions in m-tree. In *Proceedings of the First International Workshop on Similarity Search and Applications (sisap 2008)*, pages 121–128, Washington, DC, USA, 2008. IEEE Computer Society.

[10] L. Micó, J. Oncina, and R. C. Carrasco. A Fast Branch & Bound Nnearest Neighbour Classifier in Metric Spaces. *Pattern Recogn. Lett.*, 17(7):731–739, 1996.

[11] M. L. Mico, 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 Recogn. Lett.*, 15(1):9–17, 1994.

[12] G. Navarro. Analyzing metric space indexes: What for? In *IEEE SISAP 2009*, pages 3–10, 2009.

[13] O. Pedreira and N. R. Brisaboa. Spatial selection of sparse pivots for similarity search in metric spaces. In *Proceedings of the 33rd conference on Current Trends in Theory and Practice of Computer Science*, SOFSEM '07, pages 434–445, Berlin, Heidelberg, 2007. Springer-Verlag.

[14] H. Samet. *Foundations of Multidimensional and Metric Data Structures*. Morgan Kaufmann, 2006.

[15] A. P. Sexton and R. Swinbank. Bulk loading the m-tree to enhance query performance. In H. Williams and L. MacKinnon, editors, *Key Technologies for Data Management*, volume 3112 of *Lecture Notes in Computer Science*, pages 190–202. Springer Berlin / Heidelberg, 2004.

[16] T. Skopal. Pivoting M-tree: A Metric Access Method for Efficient Similarity Search. In *Proceedings of the 4th annual workshop DATESO, Desná, Czech Republic, ISBN 80-248-0457-3, also available at CEUR, Volume 98, ISSN 1613-0073*, `http://www.ceur-ws.org/Vol-98`, pages 21–31, 2004.

[17] T. Skopal and J. Lokoč. New dynamic construction techniques for m-tree. *J. Discrete Algorithms*, 7(1):62–77, 2009.

[18] T. Skopal, J. Pokorný, M. Krátký, and V. Snášel. Revisiting m-tree building principles. In *Advances in Databases and Information Systems*, volume 2798 of *Lecture Notes in Computer Science*, pages 148–162. Springer Berlin / Heidelberg, 2003.

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

[20] P. Zezula, G. Amato, V. Dohnal, and M. Batko. *Similarity Search: The Metric Space Approach*. Springer, 2005.