Subspace Similarity Search Using the Ideas of Ranking and Top-k Retrieval

Thomas Bernecker, Tobias Emrich, Franz Graf, Hans-Peter Kriegel,
Peer Kröger, Matthias Renz, Erich Schubert, Arthur Zimek

Institut für Informatik, Ludwig-Maximilians Universität München
http://www.dbs.ifi.lmu.de
{bernecker,emrich,graf,kriegel,kroeger,renz,schube,zimek}@dbs.ifi.lmu.de

***Abstract*— There are abundant scenarios for applications of similarity search in databases where the similarity of objects is defined for a subset of attributes, i.e., in a subspace, only. While much research has been done in efficient support of single column similarity queries or of similarity queries in the full space, scarcely any support of similarity search in subspaces has been provided so far. The three existing approaches are variations of the sequential scan. Here, we propose the first index-based solution to subspace similarity search in arbitrary subspaces which is based on the concepts of nearest neighbor ranking and top-k retrieval.**

I. Introduction

While much effort has been spent on studying possibilities to facilitate efficient similarity search in high dimensional data, scarcely ever the question arose how to support similarity search when the similarity of objects is based on a subset of attributes only. Aside from fundamentally studying the behavior of data structures in such settings, this is a practically highly relevant question. It could be interesting for any user to search, e.g., in a database of images represented by color-, shape-, and texture-descriptions, for objects similar to a certain image where the similarity is related to the shape of the motifs only but not to their color or even the color of the background. An online-store could like to propose similar objects to a customer where similarity can be based on different subsets of features. While in such scenarios, meaningful subspaces can be suggested beforehand [12], [10], in other scenarios, possibly any subspace could be interesting. For example, for different queries, different regions of interest in a picture may be relevant. Since there are 2^D possible subspaces of a D-dimensional data set, it is practically impossible to provide data structures for each of these possible subspaces in order to facilitate efficient similarity search. Another application where efficient support of subspace similarity queries is required are many subspace clustering algorithms [14] that rely on searching for clusters in a potentially large number of subspaces (starting with all one-dimensional subspaces, many combinations of one-dimensional subspaces to two-dimensional subspaces and so on). If efficient support of subspace range queries or subspace nearest neighbor queries were available, virtually all subspace cluster approaches could be accelerated considerably. Note that this problem is essentially different from the feature selection problem [8], [14].

In this paper, we formally define the problem of subspace similarity search in Section II. We discuss related work and the algorithmic sources of inspiration to our new solution in Section III. We propose an index-based solution using the ideas of ranking and top-k retrieval in Section IV. An experimental evaluation of this new method is presented in Section V. Section VI concludes the paper.

II. Subspace Similarity Search

A common restriction for the small number of approaches tackling subspace similarity search (see Section III) is that L_p-norms are assumed as distance measures. Hence we will also rely on this restriction in the problem definition. In the following, we assume that $\mathcal{DB}$ is a database of N objects in a D-dimensional space $\mathbb{R}^D$ and the distance between points in $\mathcal{DB}$ is measured by a distance function $dist : \mathbb{R}^D \times \mathbb{R}^D \rightarrow \mathbb{R}_0^+$ which is one of the L_p-norms ($p \in [1, \infty)$). In order to perform subspace similarity search, a d-dimensional query subspace will be represented by a D-dimensional bit vector S of weights, where d weights are 1 and the remaining $D - d$ weights are 0. Formally:

Definition 1: Subspace

A subspace S *of the* D*-dimensional data space is represented by a vector* $S = (S_1, \ldots, S_D) \in \{0,1\}^D$*, where* $S_i = 1$*, if the* i*th attribute is an element of the subspace, and* $S_i = 0$*, otherwise. The number* d *of 1 entries in* S*, i.e.,* $d = \sum_{i=1}^{D} S_i$ *is called the* dimensionality *of* S*.*

For example, in a 3D data space, the subspace representing the projection on the first and third axis is represented by $S = (1, 0, 1)$. The dimensionality of this subspace is 2.

A distance measure for a subspace S can then be figured as weighted L_p-norm where the weights can either be 1 (if this particular attribute is relevant to the query) or 0 (if this particular attribute is irrelevant to the query), formally:

Definition 2: Subspace Distance

The distance in a subspace S *between two points* $x, y \in \mathcal{DB}$ *is given by:*

$$dist_S(x, y) = \sqrt[p]{\sum_{i=1}^{d} S_i \, |x_i - y_i|^p},$$

where x_i, y_i, and S_i denote the values of the ith component of the vectors x, y, and S, respectively.

Accordingly, a subspace ε-range query can be formalized as:

Definition 3: Subspace ε-Range Query

Given a query object q and a d-dimensional ($d \leq D$) query subspace represented by a corresponding vector S of weights, a subspace ε-range query retrieves the set $RQ(\varepsilon, S, q)$ that contains all objects from $\mathcal{DB}$ for which the following condition holds:

$$\forall o \in RQ(\varepsilon, S, q) : dist_S(o, q) \leq \varepsilon.$$

The related problem of subspace k nearest neighbor (k-NN) queries can be formally defined as follows.

Definition 4: Subspace k-NN Query

Given a query object q and a d-dimensional ($d \leq D$) query subspace represented by a corresponding vector S of weights, a subspace k-NN query retrieves the set $NN(k, S, q)$ that contains k objects from $\mathcal{DB}$ for which the following condition holds:

$$\forall o \in NN(k, S, q), \forall o' \in \mathcal{DB} \setminus NN(k, S, q) : dist_S(o, q) \leq dist_S(o', q).$$

Some of the rare existing approaches for subspace similarity search focus on ε-range queries. This is a considerable lack because k-NN queries are more user friendly and more flexible. Choosing the number k of results that should be returned by a query is usually much more intuitive than selecting some query radius ε. This is even more evident when searching subspaces of different dimensionality because in this case, also the value of ε needs to be adjusted to the subspace dimensionality in order to produce meaningful results. This is a non-trivial task even for expert users since recall and precision of an ε-sphere becomes highly sensitive to even small changes of ε depending on the dimensionality of the data space.

In addition, many applications like data mining algorithms that further process the results of subspace similarity queries require to control the cardinality of such query results [14].

III. Related Work

Established index structures (such as [7], [2], [3], [11]) are designed and optimized for the complete data space where all attributes contribute to partitioning, clustering etc. For these data structures, the space of queries facilitated by the index structure must be fixed prior to the construction of the index structure.

While the results of research on such index structures designed for one single query space are abundant [17], so far there are only three methods addressing the problem of *subspace similarity search*, implicitly or explicitly. All three are variations of the sequential scan. We review these methods below in Section III-A

The solution to subspace similarity search we are proposing in this paper is based on the ad hoc combination of one-dimensional index structures. The combination technique is algorithmically inspired by top-k queries on a number of different rankings of objects according to different criteria. We review these techniques below in Section III-B.

A. Adaptations of the Sequential Scan for Subspace Similarity Search

BOND [4] is essentially also a search strategy for the full-dimensional space enhancing the sequential scan. The basic idea is to use a column store, sort the columns according to their potential impact on distances and prune later columns if their impact becomes too small to change the query result. By the design of this method, subspace queries can be *implicitly* facilitated with the same architecture. However, BOND requires certain properties of a data set which restricts the application considerably. In particular, since one application scenario of BOND are histogram data, in the basic approach the length of each data vector is assumed to be normalized to 1. Relaxing this condition, an extension of the basic approach assumes the unit hypercube $[0, 1]^D$ as the data space. In this extension, still the length of each vector is required for pruning columns with low impact. This length is precomputed for the full-dimensional data space and materialized in a separate table. For any subspace query, the potential impact of columns not contributing to the subspace can be set to 0 in advance. However, the pruning power based on the precomputed length of the vector will deteriorate for subspace queries with lower subspace dimensionality.

The first approach addressing the problem of subspace similarity search *explicitly* is presented in [13]. There, the authors propose an adaptation of the VA-file [19] to the problem of subspace similarity search. The basic idea of this approach is to split the original VA-file, into D *partial* VA-files, where D is the data dimensionality, i.e. we get one file for each dimension containing the approximation of the original full-dimensional VA-file in that dimension. Based on the information of the partial VA-files, upper and lower bounds of the true distance between data objects and the query are derived. Subspace similarity queries are processed by scanning only the relevant files in the order of relevance, i.e. the files are ranked by the selectivity of the query in the corresponding dimension. As long as there are still candidates that cannot be pruned or reported using the upper and lower distance bounds, the next ranked file is read to improve the distance approximations or (if all partial VA-files have been scanned) the exact information of the candidates accessed to refine the exact distance.

Another approach to the problem is proposed in [15], although only ε-similarity range queries are supported. The idea is to derive lower and upper bounds for distances based on the average minimal and maximal impact of a possible range of d dimensions, $d \in [d_{\min}, d_{\max}]$. The bounds are computed in a preprocessing step for a couple of pivot points. To optimize the selection of pivot points, also a distribution of possible values for ε is required. The lower and upper bounds w.r.t. all pivot points are annotated to each database object. Essentially, this approach allows to sequentially scan the database reading

only the information on lower and upper bounds and to refine the retrieved candidates in a postprocessing step.

B. Top-k Queries

The subspace similarity problem addressed in this paper is closely related to the top-k query problem. Let us assume that we have a set of objects that are ranked according to m different score functions (e.g. different rankings for m different attributes). The objective of a top-k query is to retrieve the k objects with the highest combined (e.g. average) score. In our scenario, if we assume the objects are ranked for each dimension according to the distance to the query object, respectively, we can apply top-k methods to solve subspace k-NN queries with the rankings of the given subspace dimensions.

For the top-k query problem, there basically exist two modes of access to the data given by the m rankings, the sequential access (SA) and the random access (RA) [6]. While the SA mode accesses the data in a sorted way by proceeding through one of the m rankings sequentially from the top, the RA mode has random access to the rank of a given object w.r.t. a given ranking. A naive solution for the given problem is to sequentially access the data (in SA mode) according to a specific ranking list, request the scores of the other attributes in RA mode and compute the combined score, respectively. Fagin introduced in [5] a more efficient solution with the assumption that the combined score is built by a fixed monotone aggregation function. The basic idea is to access the objects sequentially from different ranking lists in parallel until at least k objects have been seen in each of the m ranking lists. A similar access procedure is used in the threshold algorithm TA proposed in [6] and quite similar approaches [16], [9]. The common idea of these algorithms is to determine a threshold from the scores of the last objects seen in all ranking lists which is used as a stopping criterion for further data accesses. In addition to the TA algorithm which is based on both sequential and random access, a variant called *no random access* (NRA) is proposed in [6] that applies only sorted access.

IV. Index-Based Subspace Similarity Search

In the following we propose an index structure for subspace similarity search adapting the technique of the top-k algorithm proposed in [6].

A. Data Structures

Our key idea is to vertically decompose the data contained in $\mathcal{DB}$ and organize each dimension separately in an index $\mathcal{I}_i$ $(1 \leq i \leq D)$, using the feature value of the dimension as spatial key and the id of the corresponding object as value. For this purpose a B+-Tree seems adequate as it is specialized for indexing one-dimensional data. In this problem, however, it is even more appropriate to use a (one-dimensional) R*-Tree as it heuristically tries to minimize the extension of nodes, which was shown to be important for spatial queries. The used R*-Tree has the following two modifications:

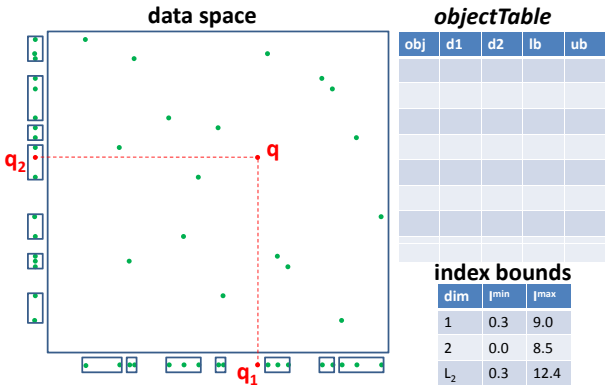


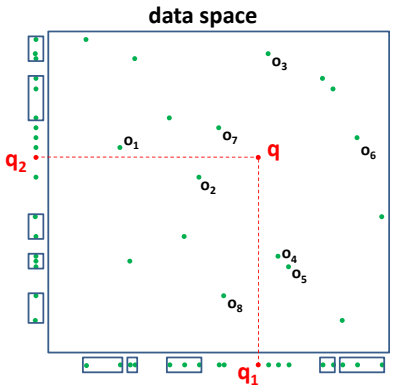
Fig. 1. Initial situation of indexes and objectTable

- Each leaf node has a link to its left and right neighbor. This relation is well defined since the tree only organizes a single dimension on which a canonical order is defined.
- Each leaf node stores the values of the facing boundaries of its two neighbors.

The second data structure needed is a hash table for storing (possibly incomplete) object information with the object ID as key. This table is referred to as *objectTable*. It is used to store for each object the distance to the query object in each dimension. If this information is not known, the corresponding field remains empty. In Figure 1, an example for a two-dimensional subspace query is shown. In the example, the leaf nodes (pages) of the two relevant index structures organizing the objects in the dimensions of the subspace are illustrated at the borders of the data space. Initially, the *objectTable* is empty. Along the fields for distance values for each dimension in the *objectTable*, the values for lower and upper bounds can be computed, using the current information of the index bounds for the one-dimensional indexes $\mathcal{I}_1$ and $\mathcal{I}_2$. The computation of these bounds is detailed in the following.

B. Query Processing

When a subspace query (q, S) arrives, only those indexes $\mathcal{I}_i$ are considered where $S_i = 1$. On these one-dimensional indexes, we perform incremental nearest neighbor queries (where q_i is the query for $\mathcal{I}_i$). In our setting, a call of *getNext()* on the index $\mathcal{I}_i$ returns the leaf node closest to the query q_i in dimension i, whose contained objects have not yet been reported. The challenge is to combine the results of the single dimensions to a result on the whole subspace. This is done by the *objectTable* which is empty at the beginning of the query process. For each object o which was reported by an index $\mathcal{I}_i$, an entry in the *objectTable* is created. If it already exists, the corresponding entry is updated (i.e., the dimension i of the object is set to o_i). If an object o has not yet been seen in index $\mathcal{I}_j$, its value in dimension j in the object table is undefined. The distance between an object $o \in \mathcal{DB}$ and q in



objectTable

obj	d1	d2	lb	ub
o_1		0.5	2.94	9.01
o_2		1.0	3.06	9.06
o_3	0.4		1.94	8.51
o_4	1.0		2.15	8.56
o_5	1.5		2.42	8.63
o_6		1.0	3.07	9.06
o_7	1.9	1.5	2.42	2.42
o_8	1.7		2.55	8.67

index bounds

dim	l^{min}	l^{max}
1	2.9	9.0
2	1.9	8.5
L_2	3.47	12.4

Fig. 2. Situation after 4 getNext()-calls

the subspace S $dist_S(q,o)$ can be bounded by:

$$ub_S(q,o) = \sqrt[p]{\sum_{i=1}^{d} S_i \begin{cases} |o_i - q_i|^p & * \\ \max(|\mathcal{I}_i^{min} - q_i|, |\mathcal{I}_i^{max} - q_i|)^p & ** \end{cases}}$$

where $\mathcal{I}_i^{min}$ and $\mathcal{I}_i^{max}$ are the lower and upper bound of the data contained in $\mathcal{DB}$ in dimension i, which can be obtained directly from the index $\mathcal{I}_i$ as this corresponds to the boundaries of the root node. It holds that $ub_S(q,o) \geq dist_S(q,o)$. For the calculation of $ub_S(q,o)$ we distinguish two cases: If object o has been found in index $\mathcal{I}_i$ (*), the exact value in this dimension can be used. Otherwise (**), the bounds of the data space have to be used in order to approximate the value in this dimension. Using Equation 1 and the information contained in the *objectTable*, an upper bound for the distance $dist_S(q,o)$ can be obtained for each object $o \in \mathcal{DB}$. Therefore it is also possible to calculate an upper bound for the distance of the kth-nearest neighbor to the query object, which can be used as pruning distance. The upper bound is recorded in *objectTable*, and updated if necessary.

Analogously, a lower bound of the distance to the query for each object in the *objectTable* can be obtained:

$$lb_S(q,o) = \sqrt[p]{\sum_{i=1}^{d} S_i \begin{cases} |o_i - q_i|^p & * \\ |\mathcal{I}_i^{next} - q_i|^p & ** \end{cases}} \quad (2)$$

where $\mathcal{I}_i^{next}$ is the position of the query facing boundary of the page obtained by the next call of *getNext()* on $\mathcal{I}_i$. Again, we distinguish the cases where o_i has been reported (*) or where it is undefined at the moment (**). This lower bound is important for the refinement step of the query algorithm and it is recorded in *objectTable*, and updated if necessary.

The pseudo code for a subspace k-nearest-neighbor query on the dimension merge index is given in Algorithm 1. Initially, the upper bound of the kth-nearest neighbor distance (*maxKnnDist*) is set to infinity. As long as there exists an object which could have a lower distance than the current upper bound of the kth-nearest neighbor distance and which is not in the *objectTable*, we have to continue the filter step and thus insert more points in the *objectTable*. The minimum

Algorithm 1 kNN-Query on Dimension Merge Index

Require: q, S, k, $\mathcal{I}$

1: maxKnnDist := ∞
2: **while** maxKnnDist $\geq$ $minObjectDist_S(q,\mathcal{I})$ **do**
3: i = chooseIndex($\mathcal{I}$)
4: leafNode = $\mathcal{I}_i$.getNextNode(q_i)
5: *objectTable*.insert(leafNode.elements)
6: maxKnnDist = *objectTable*.getMaxKnnDist(k)
7: **end while**
8: *objectTable*.refine()

distance of an object which is not in the *objectTable* is given by:

$$minObjectDist_S(q,\mathcal{I}) = \sqrt[p]{\sum_{i=1}^{d} S_i mindist(\mathcal{I}_i^{next}, q)} \quad (3)$$

At the point where *minObjectDist* is bigger than the *maxKnnDist* (as seen in Figure 2 for k = 1), the algorithm enters the refinement step. Now, no object which is not in the *objectTable*, can be part of the result, therefore only objects contained in the *objectTable* at this time have to be considered. In order to keep the number of resolved objects (corresponding to the number of expensive page accesses) low, we use the technique for refinement optimal multi step processing, proposed in [18].

C. Discussion

Algorithm 1 can easily be adapted to ε-range queries. Only the *maxKnnDist* has to be set to ε, and need not be updated (i.e., line 6 is to be omitted). The most important part of the algorithm considering the performance is the *chooseIndex* method. In order for a fast termination of the filter-step it is necessary to

- find and minimize the upper bound of *maxKnnDist* and
- increase the minimum distance a page can have

as fast as possible.

We propose three heuristics for choosing the appropriate index in each step.

The first heuristic (*Round-Robin*) sequentially chooses the index in a round robin manner and can be seen as a simple baseline. The problem with this heuristic is, that it does not take the data distribution into account. Thus it does not meet with the two requirements for a fast processing of the filter step (see above).

The second heuristic, called *GlobalMinDist*-Heuristic, aims at the first point: it always chooses the index $\mathcal{I}_i$ which has the closest page to the query q_i considering dimension i. As will be shown in the experimental evaluation, this heuristic yields a much better performance of the query processing. However the *GlobalMinDist*-Heuristic will perform very bad in a subspace where one dimension has a much larger scale than the other dimensions. In this setting the *GlobalMinDist*-Heuristic will prefer resolving pages from the indexes organizing the dimensions with a small extent, as in these dimensions the *mindist*

from the query will be very low compared to the dimension with the high extend. Thus the second requirement is not met, and a lot of pages get resolved without much information gain.

To overcome this drawback, we propose a third heuristic which we will refer to as *MinScore*-Heuristic. For each Index $\mathcal{I}_i$, we compute the following score

$$score(\mathcal{I}_i) = \frac{|q_i - \mathcal{I}_i^{next}|}{\mathcal{I}_i^{max} - \mathcal{I}_i^{min}} \quad (4)$$

and choose the index minimizing $score(\mathcal{I}_i)$. This normalization prevents the algorithm from preferring dimensions with small extent.

V. Evaluation

For our evaluation we used the following two synthetic data sets and one real world data set:

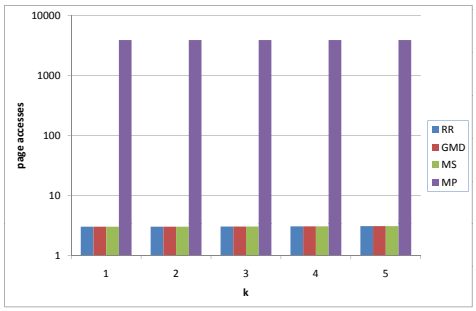
- UNIFORM: A synthetic data set with 20 dimensions and 100,000 points that are uniformly distributed.
- CLUSTERED: A synthetic data set with 20 dimensions and 100,000 points that are distributed in 1000 multivariate Gaussian clusters. The means of the Gaussians are uniformly distributed in the data space.
- FOREST: Forest Cover Type data set, obtained from the *UCI Machine Learning Repository* [1]: 581,012 data points with 10 real-valued attributes, each representing 30×30 square meters of a forest region. We used the first 100,000 points of the data set for our evaluation.

We evaluated the three proposed heuristics Round-Robin (RR), Global-MinDist (GMD) and MinScore (MS) against the multi-pivot based structure (MP [15]) for subspace similarity, which is the latest variant of related approaches and, thus, can be considered state of the art. As MP is only capable of answering ε-range queries and it is hard to set a meaningful ε in an ad-hoc subspace (cf. Section II), we used the k-nearest neighbor distance obtained by a previous k-NN-query for this approach. We show results of experiments with varying k and varying subspace size on the three data sets. We report the number of page accesses, which is the most objective value for I/O bound operations.

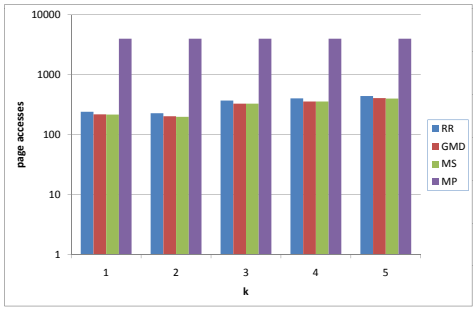
On the UNIFORM data set (cf. Figure 3), we can observe the index-based subspace similarity search beating MP. The different heuristics, however, are not really discriminant. This is the result to be expected on uniformly distributed data equally scaled in each attribute, where no heuristic yields substantial benefits.

The CLUSTERED data set (cf. Figure 4), is easier to index for both data structures than the UNIFORM data set. The query performance increases drastically. However the difference between the three heuristics and MP remains very similar.

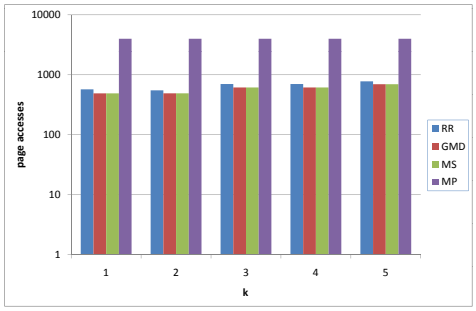
The dimensions of the FOREST data set are very unequal scaled. Therefore the mentioned problems of the GMD-Heuristic become visible (cf. Fig. 5). Here the MS-Heuristic shows the best performance. An interesting observation can be seen in Figure 5(c), where the number of page accesses can even decrease with increasing k. This is due to the fact, that a lower k leads to less refinements in the filter step. Thus, there is less information in the *objectTable* (as with a higher value of k), which can lead to a lower number of refinements in the refinement step. This effect is very interesting and possibly a starting point for further studies of new heuristics.



(a) $d = 1$.



(b) $d = 2$.

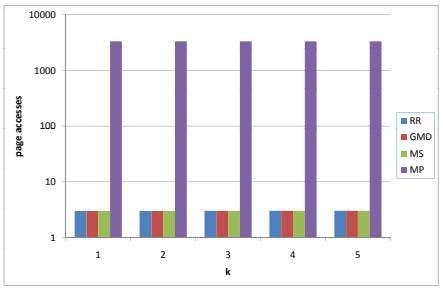


(c) $d = 3$.

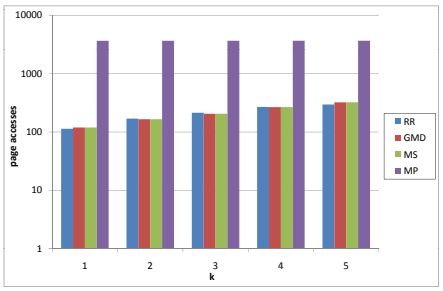
Fig. 3. Page accesses on the UNIFORM data set, varying subspace dimensionalities.

VI. Conclusions

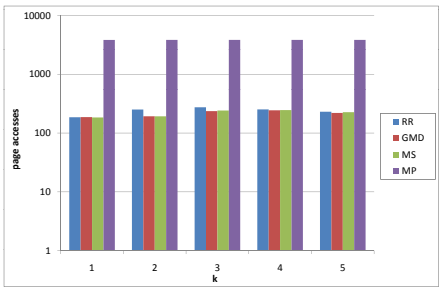
In this paper, we proposed and studied a new, index-based solution for supporting k-nearest neighbor queries in arbitrary subspaces of the original feature space, based on the concepts of nearest neighbor ranking and top-k retrieval. In an experimental evaluation on several synthetic and real data sets, this index-based solution is demonstrated to perform superior than the most recent competitor. As future work, we plan to study further heuristics based on our results and to perform a broad evaluation to study the impact of different data characteristics on all existing approaches to subspace similarity search.



(a) $d = 1$.

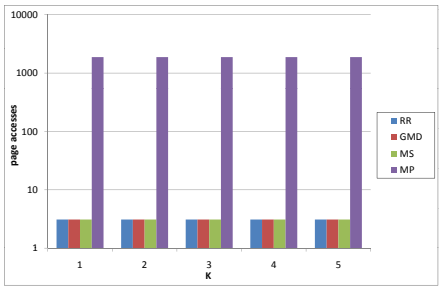


(b) $d = 2$.

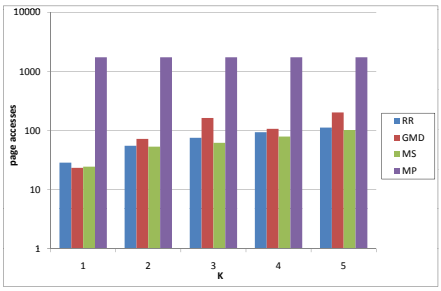


(c) $d = 3$.

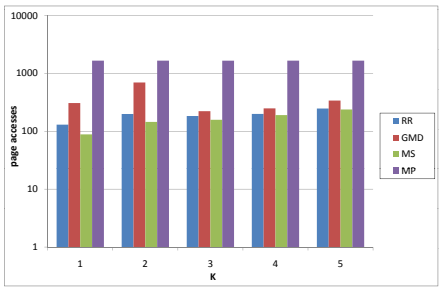
Fig. 4. Page accesses on the CLUSTERED data set, varying subspace dimensionalities.



(a) $d = 1$.



(b) $d = 2$.



(c) $d = 3$.

Fig. 5. Page accesses on the FOREST data set, varying subspace dimensionalities.

ACKNOWLEDGEMENTS

This research has been supported in part by the THESEUS program in the MEDICO and CTC projects. They are funded by the German Federal Ministry of Economics and Technology under the grant number 01MQ07020. The responsibility for this publication lies with the authors.

REFERENCES

[1] A. Asuncion and D. J. Newman. UCI Machine Learning Repository. `http://www.ics.uci.edu/~mlearn/MLRepository.html`, 2007.

[2] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The R*-Tree: An efficient and robust access method for points and rectangles. In *Proc. SIGMOD*, pages 322–331, 1990.

[3] S. Berchtold, D. A. Keim, and H.-P. Kriegel. The X-Tree: An index structure for high-dimensional data. In *Proc. VLDB*, 1996.

[4] A. P. de Vries, N. Mamoulis, N. Nes, and M. Kersten. Efficient k-NN search on vertically decomposed data. In *Proc. SIGMOD*, 2002.

[5] R. Fagin. Combining fuzzy information from multiple systems. *JCSS*, 58:83–99, 1999.

[6] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. *JCSS*, 66(4):614–656, 2003.

[7] A. Guttman. R-Trees: A dynamic index structure for spatial searching. In *Proc. SIGMOD*, 1984.

[8] I. Guyon and A. Elisseeff. An introduction to variable and feature selection. *J. Mach. Learn. Res.*, 3:1157 – 1182, 2003.

[9] U. Güntzer, W.-T. Balke, and W. Kießling. Optimizing multi-feature queries for image databases. In *Proc. VLDB*, 2000.

[10] X. He. Incremental semi-supervised subspace learning for image retrieval. In *Proc. ACM MM*, 2005.

[11] N. Katayama and S. Satoh. The SR-tree: An index structure for high-dimensional nearest neighbor queries. In *Proc. SIGMOD*, 1997.

[12] M. Koskela, J. Laaksonen, and E. Oja. Use of image subset features in image retrieval with self-organizing maps. In *Proc. CIVR*, 2004.

[13] H.-P. Kriegel, P. Kröger, M. Schubert, and Z. Zhu. Efficient query processing in arbitrary subspaces using vector approximations. In *Proc. SSDBM*, 2006.

[14] H.-P. Kriegel, P. Kröger, and A. Zimek. Clustering high dimensional data: A survey on subspace clustering, pattern-based clustering, and correlation clustering. *ACM TKDD*, 3(1):1–58, 2009.

[15] X. Lian and L. Chen. Similarity search in arbitrary subspaces under Lp-norm. In *Proc. ICDE*, 2008.

[16] S. Nepal and M. V. Ramakrishna. Query processing issues in image (multimedia) databases. In *Proc. ICDE*, 1999.

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

[18] T. Seidl and H.-P. Kriegel. Optimal multi-step k-nearest neighbor search. In *Proc. SIGMOD*, 1998.

[19] R. Weber, H.-J. Schek, and S. Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *Proc. VLDB*, 1998.