Metric indexing to improve distance joins

Niels Nes, Wilko Quak, Martin Kersten

University of Amsterdam
{niels,quak,mk}@wins.uva.nl

Keywords: Multi-dimensional vectors, index structures, similarity, fuzzy queries, DBMS, Image databases

Abstract

Database applications using large vector data are often supported by spatial index structures to locate spatially related objects. An important query class deals with finding related object pairs under a distance function. In this paper we demonstrate that a light-weight indexing structure, based on the metric properties derived from the distance function, is often sufficient to support this important class. It is of particular importance as a temporary search accelerator while processing complex queries. Moreover, it can be used to speed up point and region queries for low selectivities and, presumably, highly-skewed spaces.

keywords: *Multi-dimensional vectors, index structures, similarity, fuzzy queries.*

1 Introduction

An emerging class of database applications heavily relies on spatial information, e.g. geographical and multi-media information systems. The majority of queries involves exploration of the spatial information, such as finding elements in a given area. Furthermore, join queries over a spatial domain often boil down to calculation of a distance function to locate point pairs within close vicinity of one-another. They are conventionally implemented using a spatial index (like R-tree) to filter candidate pairs.

In this paper we introduce a cheap and fast index structure to speed up such distance joins. In particular, we demonstrate that an index structure based on the distance metric is both cheap to construct and competitive in performance.

Because the index structure can be built very fast and its storage overhead is minimal, it is it a good candidate for on the fly construction as part of a query processing plan. Since the index structure captures the metric information to answer distance joins directly, it reduces the number of times the expensive distance function is called considerably, compared to a naive spatial join evaluation.

The key property exploited is that distance joins involve a well-defined, but expensive, (Euclidean) distance function. Moreover, these functions satisfy the mathematical triangular inequality property, i.e. the distance between two points is always smaller or equal than the distance between these points and a third point.

Based on the triangular inequality it is possible to build an index structure to speed up several query classes, such as:

- **point-match**, for each A find the points B positioned at the same location (e.g. A.pos=B.pos).
- **k-nearest neighbor**, for each A find the k nearest elements B under the distance function.
- **δ-search**, find all B points within delta δ range away from A.
- **δ-join**, find all A,B point pairs within delta δ range.

In this paper we ignore the point-match and k-nearest neighbor queries, because they are special cases of the δ-search. The point-match can be re-phrased as a delta-join with $\delta = 0$, while the k-nearest neighbor can be implemented using a binary traversal over delta values until k values have been obtained. The δ-search is part of the inner-loop of the δ-join algorithm.

The remainder of this paper is organized as follows. In Section 2 we review index structures that deal with spatial joins in high dimensions. Section 3 explains the use of the triangular inequality in a multi-dimensional space. In Section

4 the metric index data structure and algorithms are explained. Section 5 provides a mathematical estimation of the effectiveness of the new index structure. Section 6 reports on experimentations to validate the approach taken. Finally, in section 7 we draw our conclusions and pointers for future research.

2 Index Structures for Spatial Joins

Most previous work on searching in multi-dimensional spaces is concentrated on low dimensional data-structures, such as R-tree [4] and and K-D-trees [1]. These structures can be extended to higher dimensions, but this results in two problems, a performance and effectivity degration. The performance degrades because as the dimension increases the querying cost often increases exponentially; the so-called *dimensionality curse.* The index structures deployed become less effective as a pre-filter for selections and join operations.

This curse also stems from the metric effects in higher dimensions, which leads to a clustering of objects at the 'edge' of the n-dimensional space, while all points theoretical become placed at 'equal' distance of any other point in this space.

The prime route explored in literature to tackle the former deals with the scalability limitations of most data structures. Examples considered here are the X-tree the SS-tree and the TV-tree:

The X-tree [2] tackles the dimensionality problem by observing that the performance degradation in the R-tree based index structures is mainly due to the high overlap between the nodes in the R-tree itself. This overlap causes an increased number of nodes to be visited when querying the R-tree. The X-tree solves this problem by allowing nodes of the X-tree to be bigger than one disk block (the so-called supernodes) if a split node would generate a high overlap. This technique makes an X-tree behave like an R-tree in low dimensions, while in higher dimensions the join behavior converges to that of a nested loop.

Another data structure for indexing high-dimensional vectors is the SS-tree [6]. The SS-tree is an R*-tree based structure using bounding (hyper)balls instead of rectangles. In 2-dimentions bounding circles are more appropriate for performing similarity queries. Furthermore, they store some additional statistical data in the nodes to support various operations used in image retrieval.

The TV-tree [5] reduces the size of internal nodes by projecting the data in internal nodes to a lower dimension. By using different projections in different parts of the tree, all parts of the original vectors are used. If some dimensions of the input data are more important than others a big speedup can be gained. This is done by first projecting the data to these important dimensions. It is unclear how well the TV-tree performs when all dimensions are equally important.

Despite the progress reported in reducing the storage/processing cost in moving to higher dimensional indices, these data structures are focussed on the spatial organization. We focus on point and region-based retrieval operations. Our key operation, δ-joins, requires a relaxation of the spatial joins supported by several systems. It behaves more like a theta-join within a spatial context. The role of the index structures in this case are primarily aimed at reducing the number of candidates for consideration.

3 Triangular Inequality

As mentioned before, distance functions play an important role in real life applications, e.g. GIS, CAD/CAM, Image Retrieval and multimedia applications. For example in GIS and CAD/CAM applications require spatial queries, like "find me the closest restaurant to a given location" and "find objects that are so closely placed that they generate electro-static interference". Many content based text, image and multi-media applications use similarity based queries, like "find similar colored objects". To illustrate, the functions encountered in the areas considered are:

1. The Great Circle Distance is used in GIS to calculate the 'as the crow flies' distance between two places in the world.

2. A distance function amongst customer profiles (time series) in the datamining area.

3. The Weighted Euclidean Distance over a vector space:
   $d(V,W) = (V-W)^T A(V-W) = \sum_i \sum_j A_{i,j}(V_i - W_i)(V_j - W_j)$

4. Histogram Intersection

$$d(V,W) = 1 - \frac{\sum_i min(V_i, W_i)}{\sum_i (V_i)}$$

Most distance functions found are expensive to calculate and, because they are called repeatedly they contribute considerably to the total querying cost. Our index structure aims to reduce the number of calls to these functions in a naive implementation of the δ-join. This is achieved by

using the metric properties. The key to the solution proposed relies on the *triangular inequality* relationship.

The mathematical properties for a metric, $|xy|$, where x, y and z are multi-dimensional vectors, are:

- **Positivity** $|xy| \geq 0 \wedge |xx| = 0$
- **Symmetry** $|xy| = |yx|$
- **Triangular inequality** $|xy| \leq |xz| + |zy|$

It enables to set-up an index that is both effective (on low selectivities) and fast to construct.

3.1 Using the Triangular Inequality

The naive implementation of the envisioned δ-join is a nested loop. For each pair considered the distance should be calculated. Since this results in many expensive calculations it becomes mandatory to reduce candidate pairs to re-use results being calculated. This is achieved by taking a reference point (or set of reference points) and to calculate the distances between the reference point and each vector in a join operand first.

Now consider a query looking for all points within distance δ from a query point q, i.e. all points p with $|pq| < \delta$. The metric properties enables reuse of distances calculated between p and reference point r.

Assume that for points in our space we know its distance to a reference point r. Then the query $|pq| < \delta$ could use this information as follows. The triangular inequality provides us with $|rq| \leq |rp| + |pq|$. So we have an upper bound $|rq| \leq |rp| + \delta$. Since also $|pq| \leq |pr| + |rq|$ holds, we also know that $|pr| - \delta \leq |rq|$ holds. Using the metric symmetry we can use $|rp|$ directly, else we could also store $|pr|$. Figure 1 shows the use of the triangular inequality in the 2 dimensional case.

If the query point is already close to the reference point, we can remove all possible points with large distance from r from consideration. They typically fall behind the horizon of $2 * \delta$. Alternatively, if the query point is far away from the reference point we can remove all possible points which are close to the reference point r.

4 Metric index structure

In this section we will explain how the metric index structure is built and how it is used in the select and join algorithms.

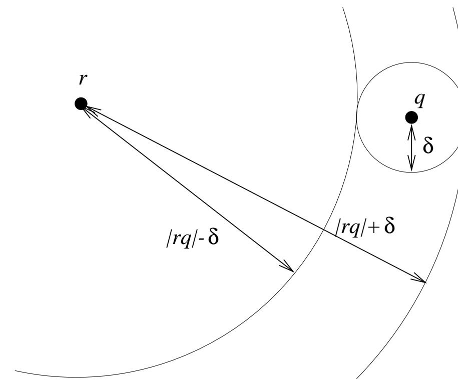

Figure 1: Circles

4.1 The reference points

There are various ways to select a reference point: random, center of gravity, or middle point. A randomly selected reference point would be the prime choice considered when it is a priori known that the space is large. The center of gravity would be of interest if the space contains several clusters. Then each cluster leads to a reference point. Unfortunately, detection of clusters and, subsequently, the reference point is expensive to calculate. Instead we use the heuristic to take a simple reference point.

Given a reference point, we calculate its distance with each point in the table. The points are subsequently sorted by distance using a tree-like structure. This will speed-up searching later for elements at a given distance from the reference point.

To illustrate the algorithms, we use on the index structure for a table of n-dimensional points with a single reference point.

4.2 The optimized distance select

Selection using the metric index follows the traditional route of pre-filtering; the index is used to reduce the candidates to consider to solve the δ-select. The following pseudo code routine explains how this can be done.

```
delta-select ( ps, q, δ ){
        select p from
                select p from  ps
                where |pr| − δ < |rq| < |pr| + δ
        where |pq| < δ
}
```

The inner **select** selects all points p form the point set *ps* where the distance of the reference point r to the query point q is between $|pr| - \delta$ and

$|pr| + \delta$. This identifies candidates in a cylinder around the reference point. It can be solved with a single lookup because we know the distance to r. The outer **select** filters this set by checking for the actual distance between p and q.

Similarly, we can use the metric index to speed-up the δ-join using the generalized triangular inequality.

$$|pq| \leq |pr_0| + |r_0 r_1| + \ldots + |r_n q|$$

```
delta-join ( ps, qs, δ ){
        select p, q from
                join p, q from  ps, qs
                where ||pr_p| − |r_p r_q| − δ| < |r_q q|
                        < |pr_p| + |r_p r_q| + δ
        where |pq| < δ
}
```

5 Effectiveness of the metric index

In this section we give an estimate on the expected hit ratio of the candidates selected, i.e. "Is the metric index a good filter?". This estimate is only given for the case where one reference point is chosen. All estimates in this section are further based on the (simplifying) assumption that the vectors are uniformly distributed in space; this means that the size of a query result is linear with the volume covered by the query. A formula for the volume of a hyperball with dimension d and volume r, denoted as $V_{r,d}$. First we give a formula for balls with $r = 1$. This formula is defined recursively where the volume in one dimension is expressed in volumes of the lower dimensions. The volumes in dimensions 1 and 2 are given:

$$V_{1,1} = 2, \qquad V_{1,2} = \pi, \qquad V_{1,d} = \frac{2\pi}{d} V_{1,d-2}$$

In fact $V_{1,1}$ is the length of the interval $[-1, 1]$ and $V_{1,2}$ is the area of the circle with radius 1. The formula for balls with given r becomes:

$$V_{r,d} = r^d \; V_{1,d}$$

The next step is to estimate the size of the query result and the size of the filter set of the range query with range δ around query point q with reference point r. The two dimensional case of this query is depicted in Figure 1. Due to the uniform distribution the size of the query result is equivalent to the volume of a ball with radius δ around point q. The candidate points (points which pass the filter step) are all the points in the (hyper)disc of all points with distance between $|rq| - \delta$ and $|rq| + \delta$. Now the effectiveness of the filter is the number of hits divided by the number of candidates.

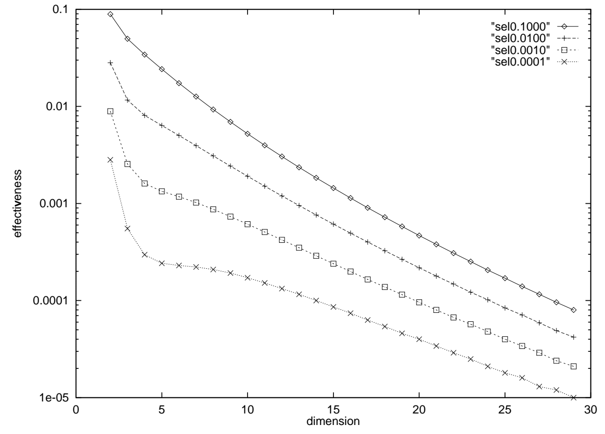

Figure 2: Effectiveness

$$\frac{\#\text{hits}}{\#\text{candidates}} = \frac{V_{|rq|,\delta}}{V_{d,|rq|+\delta} - V_{d,|rq|-\delta}} =$$

$$\frac{\delta^d}{(|rq| + \delta)^d - (|rq| - \delta)^d}$$

In Figure 2 the effectiveness of a few selectivity values is shown. In this Figure we keep the answer set constant by increasing δ for higher dimensions. As can be seen, the filter effectiveness degrades for high dimensions. But there are also some aspects to take into account to make life bearable in practice.

- In this analysis only one reference point is taken into account. Improved gain comes when more reference points are used, because they break the cylinders into pieces. This analysis will be done experimentally.

- The effectiveness of the filter is still good for small query results. A situation likely to occur in large multi-dimensional applications.

- The uniform distribution assumption is not likely to hold in practice. Clustered spaces will lead to more opportunities to filter out irrelevant points.

To assess the impact of the choice of the refence point on the effectiviness, wel calculated the expected performance while varying the distance $|rq|$ between 0 and 0.1. In Figure 3 we plot the effectiveness for various dimensions while fixing δ on 0.001.

Although the analysis in this section reinforced the existence of the dimensionality

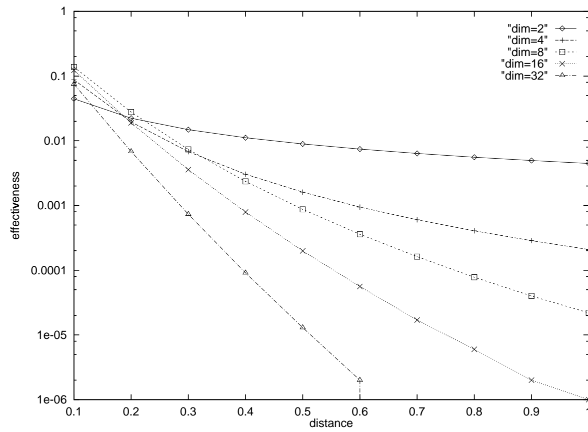

Figure 3: Impact of location reference point

curse when dealing with distance joins in high-dimensional spaces, it also indicates good behavior for low selectivity values and a small δ. We conjuncture that it will further improve with sparsely (skewed) populated in real-life applications.

One way to improve the filter effectiveness of the metric index is the use multiple reference points. For two reference point, the filtering step becomes a windowing query on points in $\mathbb{R}^2$. See Figures 4 and 5. Adding more reference points yields windowing queries in n-dimensional space. In fact this leads to filter step which converts an n-dimensional range query into a windowing query of any dimension (depending on the number of reference points).

6 Experimentation

To assess the performance of the metric index in a real setting, we have extended the Monet [3] system with a software module for δ-joins, δ-select, and a metric index. Subsequently, we conducted experiments on data sets generated using a standard pseudo-random number generator. All vector fields domains are [0..1).

The first experiment conducted was geared to get a handle on the cost of the distance function. Therefor, we measured the execution time of a naive implementation –with a simple loop– of the distance select. The results are shown in Figure 6. It shows the execution time of a distance select for databases sizes ranging from 10 to 100k with vectors of dimensions 2,4,8,16,32 and 64. All selects were done with an equal distance of 0.1. So only very close points were retrieved.

The cost of this naive loop could be invested in construction of a metric index. Once it is available, it can be used as a pre-filter. Figure 7 shows

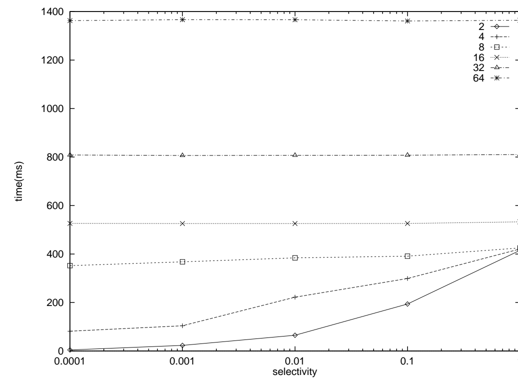

Figure 10: increasing selectivity

the results against the same query using the index structure. The benefit from the index is evident. For the low selectivity considered it leads to an overall improvement performance. Since the investement in the index structure are already recovered with 2 δ-selects.

For the δ-join a similar experiment was conducted. Figures 8 and 9 show the results of the naive nestedloop- and metric index based implementations. The cost of metric index construction is neglectable compared to the gain. Again the benefit of the index is evident.

To assess the degradation caused by increasing the result size we conducted an experiment with fixed database size of 100k and with dimensions 2 to 64, but with increasing query selectivity rangeing from 0.01% to 1.0%. Figure 10 shows the execution times of the distance selects using the index structure. It clearly shows the reduced in usability of the index structure for larger answer sets. Only for low dimensions the index structure seems effective. This stems from the uniform generated data.

To show that the index structure is cheap alternative for spatial queries in low dimensions we also compared our method with the R-tree data structure. Because our current version of the R-tree only works on two-dimension vectors, this test is only run on two dimensional data. The results of this experiment are show in Figures 11 and 12. They show the construction and execution times of the δ-join for naive (nor), optimized (opt), optimized with 2 reference points (opt2) and Rtree (rtree). From the figure we can conclude that the metric index with two reference points is overall better and that for relative low selectivity also the single reference point performs well.

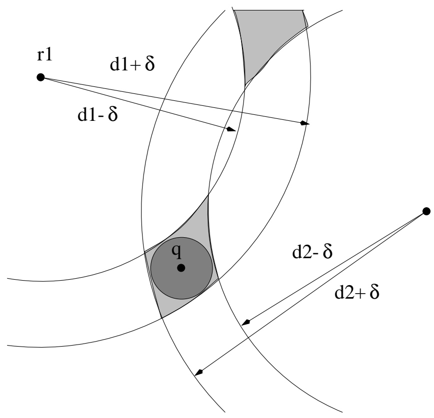

Figure 4: Distance Query

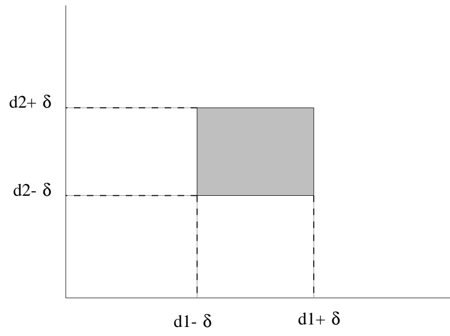

Figure 5: Range Query

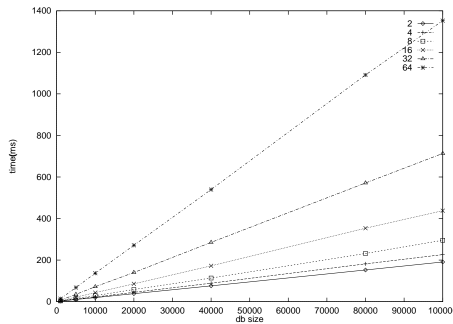

Figure 6: Naive δ-search

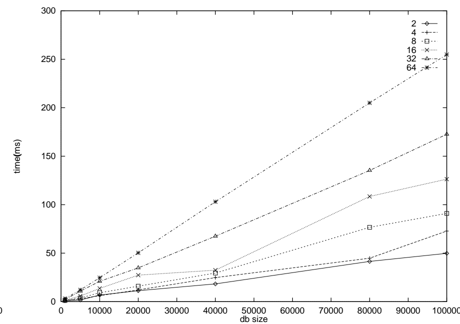

Figure 7: δ-search using Metric Index

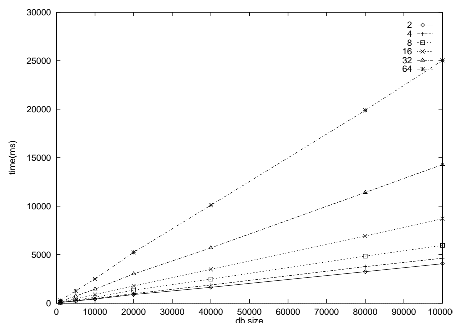

Figure 8: Naive δ-join

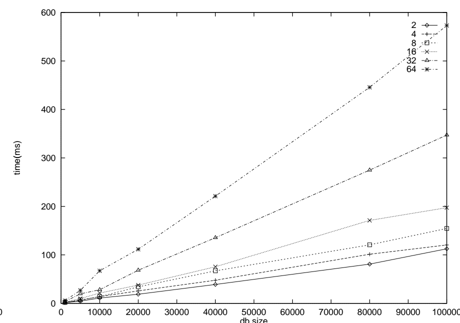

Figure 9: δ-join using Metric Index

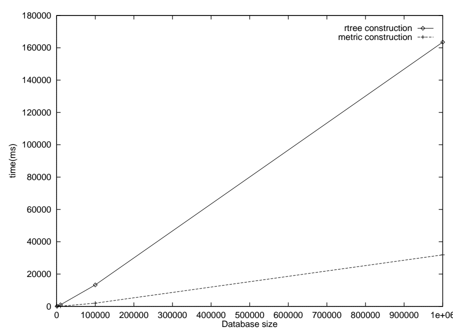

Figure 11: Construction Cost

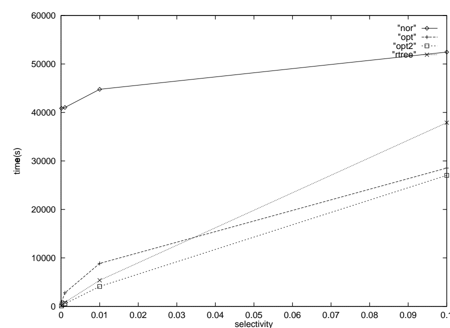

Figure 12: Two Dimensions

7 Conclusions

In this paper we presented a cheap index structure to improve the query performance of joins involving a distance metric. This index structure works on any distance metric, as long as it obeys the triangular inequality. There is no need for a full metric. We showed that the index structure is profitable in higher dimensions for small selectivities.

Several areas require further investigation. First, our assumption of uniform distribution of points in the space leads to a worst-case behavior, especially in high dimensions. All points appear at the border of the space and are equally spaced. We conjuncture that data obtained from real-life applications are extremely sparse and that clustering of points (the focus of the query) lead to good performance for acceptable ranges of selectivity.

Second, the implementation of the n-dimensional R-tree in Monet should be finished to balance the results obtained so far. We conjuncture that the effects of the dimensional curse for such data structures are worse than those experienced in our metric index. Experiments in both directions are under way.

References

[1] J.L. Bentley. Multidimensional binary search trees used for associative searching. *Comm. ACM*, 18:509–517, 1975.

[2] Stefan Berchtold, Daniel A. Keim, and Hans-Peter Kriegel. The X-tree: An index structure for high-dimensional data. In *Proceedings of the 22nd VLDB Conference*, 1996.

[3] Peter A. Boncz, Wilko Quak, and Martin L. Kersten. Monet and its Geographic Extensions: a novel Approach to High Performance GIS Processing. In *EDBT proceedings*, 1996.

[4] A. Guttman. R-trees: a dynamic index structure for spatial searching. *Proc. ACM SIGMOD*, pages 47–57, 1984.

[5] King-Ip Lin, H. V. Jagadish, and Christos Faloutsos. The TV-tree: An index structure for high-dimensional data. *VLDB Journal*, 3(4):517–542, Januari 1994.

[6] David A. White and Ramesh Jain. Similarity indexing with the SS-tree. *Proc. 12th IEEE International Conference on Data Engineering*, pages 516–523, 2 1996.