Optimal Dimension Order: A Generic Technique for the Similarity Join

Christian Böhm[1], Florian Krebs[2], and Hans-Peter Kriegel[2]

[1] University for Health Informatics and Technology, Innsbruck
Christian.Boehm@umit.at
[2] University of Munich
{krebs,kriegel}@dbs.informatik.uni-muenchen.de

Abstract. The similarity join is an important database primitive which has been successfully applied to speed up applications such as similarity search, data analysis and data mining. The similarity join combines two point sets of a multidimensional vector space such that the result contains all point pairs where the distance does not exceed a given Parameter ε. Although the similarity join is clearly CPU bound, most previous publications propose strategies that primarily improve the I/O performance. Only little effort has been taken to address CPU aspects. In this Paper, we show that most of the computational overhead is dedicated to the final distance computations between the feature vectors. Consequently, we propose a generic technique to reduce the response time of a large number of basic algorithms for the similarity join. It is applicable for index based join algorithms as well as for most join algorithms based on hashing or sorting. Our technique, called Optimal Dimension Order, is able to avoid and accelerate distance calculations between feature vectors by a careful order of the dimensions. The order is determined according to a probability model. In the experimental evaluation, we show that our technique yields high performance improvements for various underlying similarity join algorithms such as the R-tree similarity join, the breadth-first-R-tree join, the Multipage Index Join, and the ε-Grid-Order.

1 Introduction

The similarity join is a database primitive which gains increasing importance for similarity search [2] and data mining [4]. Like the relational join, the similarity join combines tuples (vectors) of two sets into one such that a join condition is fulfilled. The join condition of the similarity join is the similarity between the two objects (vectors) according to some suitable metric $\|\cdot\|$. In most cases, the Euclidean distance metric is used. Formally, the similarity join $P \underset{sim}{\bowtie} Q$ between two finite sets $P = \{p_i, \ldots, p_n\}$ and $Q = \{q_i, \ldots, q_m\}$ is defined as the set

$$P \underset{sim}{\bowtie} Q := \{(p_i, q_j) | \|p_i - q_j\| \leq \varepsilon\}$$

and can also be expressed in a SQL like fashion as

$$\textbf{SELECT * FROM } P, Q \textbf{ WHERE } \|P.\text{vector} - Q.\text{vector}\| \leq \varepsilon\,.$$

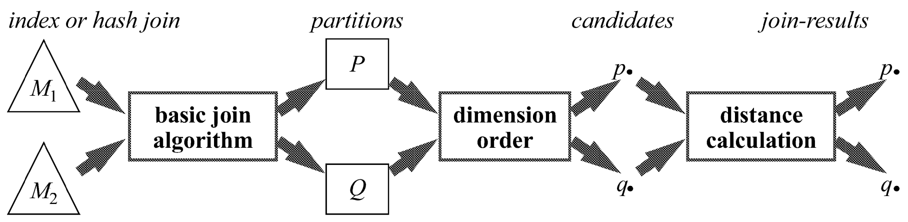


Fig. 1. Integration of the dimension-order-algorithm

Due to its high importance, many different algorithms for the similarity join have been proposed, operating on multidimensional index structures [9,14,11], multidimensional hashing [15,16], or various sort orders [19,12,7]. In contrast to algorithms for simple similarity queries upon a single data set (such as *range queries* or *nearest neighbor queries*), all of these algorithms are clearly CPU bound. In spite of the filtering capabilities of the above algorithms the evaluations cost are dominated by the final distance calculations between the points. This is even true for index structures which are optimized for minimum CPU cost [8].

Therefore, in the current paper, we propose a technique for the effective *reduction* of the high number of the distance calculations between feature vectors. Our method shows some resemblance to the paradigm of plane-sweep algorithms [17] which is extended by the determination of an optimal order of dimensions. A design objective of our technique was generality, i.e. our method can be implemented on top of a high number of basic algorithms for the similarity join such as R-tree based joins [9,14,11], hashing based methods [15,16], and sort orders [19,12,7]. A precondition for our optimal dimension order is to have some notion of *partitions* to be joined having a position and extension in the data space. This is given for all techniques mentioned above. Our technique is not meaningful on top of the simple nested loop join [20].

Organization Section 2 is dedicated to our technique, the *Optimal Dimension Order*. The experimental evaluation of our approach is presented in section 3 and section 4 concludes the paper.

2 Optimal Dimension Order

In this section we will develop a criterion for ordering the dimensions to optimize the distance computations. We assume that our join algorithm with optimal dimension order is preceded by a filter step based on some spatial index structure or spatial hash method which divides the point sets that are to be joined into rectangular partitions and only considers such partitions that have a distance to each other of at most ε. Suitable techniques are depth-first-and breadth-first-R-tree-Join [9,11], Spatial Hash Join [15,16], Seeded Trees [14], the ε-dB-tree [19], the Multidimensional Join (MDJ) [13], or the ε-grid-order [7].

2.1 Algorithm

The integration of the dimension order is shown in figure 1. Our dimension order algorithm receives partition pairs (P, Q) from the basic technique for the similarity join and generates point-pairs as candidates for the final distance calculations. The general idea of the dimension order is as follows:

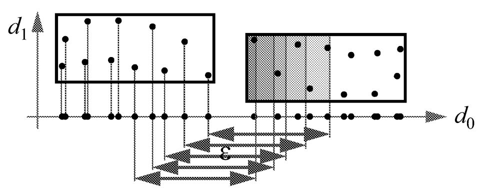


Fig. 2. Idea of the Dimension Order

If the points of one partition, say Q are sorted by one of the dimensions, then the points of Q which can be join mates of a point p of P form a contiguous sequence in Q (cf. figure 2). A large number of points which are excluded by the sort dimension can be ignored. In most cases, the points which can be ignored, are located at the lower or upper end of the sorted sequence, but it is also possible, that the sequence of points that must be processed are in the middle of the sequence. In the latter case, the start and end of the sequence of relevant points must be searched e.g. by binary search, as depicted in the algorithm in figure 3. In the other cases, it is actually not necessary to determine the first (last) point before entering the innermost loop. Here, we can replace the search by a suitable break operation in the innermost loop.

2.2 Determining the Optimal Sort Dimension

We will show in section 3 that the proposed algorithm gains much performance only if the optimal sort dimension is selected for each partition pair individually. If two partitions are joined by the basic technique, we can use the following information in order to choose the optimal dimension:

- The distance of the two partitions with respect to each other or the overlap (which we will interpret as *negative* distance from now on) in each case projected on the one single dimension of the data space. We observe that the overall distance of the two partitions as well as the distance projected on each of the dimensions cannot exceed ε otherwise the whole partition pair would have been eliminated by the *preprocessing step* in the basic technique.
- The extent of the two partitions with respect to each of the single dimensions

In order to demonstrate in a 2-dimensional example that both distance as well as extent, really do matter, see figure 4: In both cases the distance of the two partitions is the same. For simplification we show one exemplary point with its

```
algorithm optimal_dimension_order_join (index $M_1$, $M_2$)
    the similarity-join basic method
    generates partition pairs from $M_1$ and $M_2$ ;
        for all parition pairs ($P$,$Q$) with dist($P$,$Q$) $\leq \varepsilon$
        determine best sort dimension $s$ according to Eq. (9) ;
        sort (indirectly) points in $Q$ according to dimension $s$ ;
        for all points $p \in P$
            determine the first point $a \in Q$: $|a_s - p_s| \leq \varepsilon$ ;
            determine the last point $b \in Q$: $|b_s - p_s| \leq \varepsilon$ ;
            for all points $q \in Q$ with $a_s \leq q_s \leq b_s$
                if dist($p$,$q$) $\leq \varepsilon$
                    output ($p$,$q$) ;
end ;
```

Fig. 3. Algorithmic Scheme

ε-neighborhood in partitions Q and Q' although in our further discussion we assume uniform distribution of points within the two partitions i.e. we will not consider one specific point.

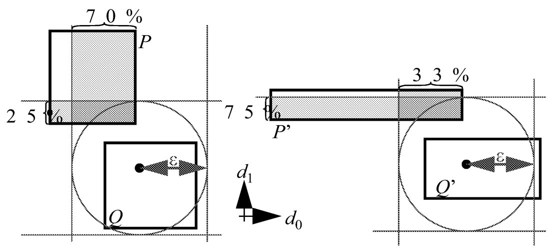


Fig. 4. Distance and extension of partitions

On the left side of figure 4 both of the partitions are roughly square. Let us first look at the projection on the d_0-axis: We observe that about 70% of the projected area of P lies within the projected ε-neighborhood of our sample point in Q. If we were to choose d_0, as sort-dimension only about 30% of the points can be excluded as joinmates for our sample point in the first step. For the remaining 70% we still have to test dimension d_1 i.e. we have to compute the overall point distance. If we now look at the projection on dimension d_1: here only 25% of the area of P lies within the ε-neighborhood of our sample point in Q. If we choose d_1 as start-dimension as much as 75% of the points are already eliminated in the first step of our dimension ordering algorithm. In the case of quadratic partitions it is thus advisable to choose the dimension within which the partitions have the largest distance with respect to each other as this minimizes the area of the projected ε-neighborhood.

The right side of figure 4 shows the two partitions P' and Q' which have a much larger extent in dimension d_0, than in dimension d_1. For this reason the projection on the d_0-axis, with a portion of 33% of the area, is much better than the projection on the d_1-axis (75%). In this case the dimension d_0, should be chosen as sort-dimension.

We can note the following as a first rule of thumb for the selection of the dimension: for approximately square partitions choose the dimension with the greatest distance, otherwise the dimension with the greatest extent.

2.3 Probability Model

In the following we will propose a model which grasps this rule of thumb much more precisely. Our model is based on the assumption that the points within each partition follow a uniform distribution. Data from real life applications such as those used in our experimental evaluation are far from being uniformly distributed. In our previous work, however, it has been shown that the conclusions which are drawn from such models with respect to optimization in multidimensional index structures are good enough to clearly improve the performance of multidimensional query processing [5].

Our model determines for each dimension the probability $W_i[\varepsilon]$ (termed *mating probability* of dimension d_i) that two points in partitions P and Q with given rectangular boundaries $P.\text{lb}_i$, $P.\text{ub}_i$, $Q.\text{lb}_i$, $Q.\text{ub}_i$ $(0 \leq d$; lb and ub for lower bound and upper bound, respectively) have at most the distance ε with respect to dimension d_i.

Definition 1
Given two partitions P and Q, let $W_i[\varepsilon]$ denote the probability for each dimension d_i that an arbitraty pair of points (p, q) with $p \in P$ and $q \in Q$ has a maximal distance of ε with respect to d_i:

$$W_i[\varepsilon] := W(|p_i - q_i| \leq \varepsilon), (p, q) \in (P, Q) \tag{1}$$

If $P.\#$ denotes the number of points within partition P, then the expectation of the number of point pairs which are excluded by the optimal dimension order equals to

$$E_i[\varepsilon] := P.\# \cdot Q.\# \cdot (1 - W_i[\varepsilon]) \tag{2}$$

This means that exactly the dimension d_i should be chosen as sort dimension that minimizes the mating probability $W_i[\varepsilon]$.

We will now develop a universal formula to determine the mating probability. We assume uniform distribution within each of the partitions P and Q. Thus the i-th component p_i of the point $p \in P$ is an arbitrary point from the uniform interval given by $[P.\text{lbi}_i \ldots\ P.\text{ub}_i]$. The pair (p_i, q_i) is chosen from an independent and uniform distribution within the two-dimensional interval $[P.\text{lb}_i \ldots\ P.\text{ub}_i] \times [Q.\text{lb}_i \ldots\ Q.\text{ub}_i]$ because of the independence of the distributions within P and Q, which we can assume for $P \neq Q$. Hence the event space of dimension d_i is given by

$$F_i = (P.\text{ub}_i - P.\text{lb}_i) \cdot (Q.\text{ub}_i - Q.\text{lb}_i) \tag{3}$$

$W_i[\varepsilon]$ is therefore given by the ratio of the portion of the area of F_i where p_i and q_i have a distance of at most ε to the whole area F_i. This can be expressed by the following integral:

$$W_i[\varepsilon] = \frac{1}{F_i} \cdot \int\limits_{P.\mathrm{lb}_i}^{P.\mathrm{ub}_i} \int\limits_{Q.\mathrm{lb}_i}^{Q.\mathrm{ub}_i} \begin{cases} 1 \; for |x-y| \le \varepsilon \\ 0 \; otherwise \end{cases} \quad dy\, dx \tag{4}$$

We can now simplify the integral of formula (4) by case analysis looking at the geometric properties of our configuration, i.e. we can transform our problem into d distinct two-dimensional geometric problems. To illustrate this, we look at the join of the two partitions P and Q in two-dimensional space as shown on the left hand side of figure 5. In this case, it is not directly obvious which dimension yields better results. The projection on d_0, which is the transformation that is used to determine $W_0[\varepsilon]$ is shown on the right hand side of figure 5. The range with respect to d_0, of points which can be stored in P is shown on the x-axis while the range with respect to d_0, of points which can be stored in Q is shown on the y-axis. The projection (p_0, q_0) of an arbitrary pair of points $(p, q) \in (P, Q)$ can only be drawn inside the area denoted as event space (cf. equation 3), as all points of P with respect to dimension d_0, are by definition within $P.\mathrm{lb}_0$ and $P.\mathrm{ub}_0$. The same holds for Q.

The area within which our join condition is true for dimension d_0, i.e. the area within which the corresponding points have a distance of less than ε with respect to d_0, is marked in gray in figure 5. All these projections of pairs of points which fall into the gray area are located within a stripe of width 2ε (the ε-stripe) which is centered around the 45° main diagonal. All projections outside this stripe can be excluded from our search as the corresponding points already have a distance with respect to d_0 that exceeds our join condition. The intersection of this stripe with the event space represents those point pairs that cannot be excluded from our search using d_0 alone. The mating probability is given by the ratio of the intersection to the whole event space which equals 18% in our example.

2.4 Efficient Computation

In the previous section, we have seen that the exclusion probability of a dimension d_i corresponds to the proportion of the event space which is covered by the ε-stripe. In this section, we show how this proportion can be efficiently determined. Efficiency is an important aspect here because the exclusion probability must be determined for each pair of mating pages and for each dimension d_i.

Throughout this section we will use the shortcut PL for $P.\mathrm{lb}_i$ and similarly PU, QL, and QU. Considering figure 6 we can observe that there exists a high number of different shapes that the intersection of the event space and the ε-stripe can have. For each shape, an individual formula for the intersection area applies. We will show

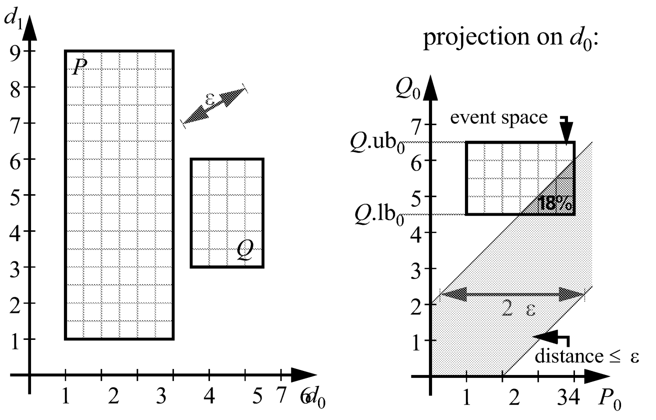


Fig. 5. Determining the mating probability $W_0[\varepsilon]$

- that exactly 20 different shapes are possible,
- how these 20 cases can be efficiently distinguished, and
- that for each case a simple, efficient formula exists.

Obviously, the shape of the intersection is determined by the relative position of the 4 corners of the event space with respect to the ε-stripe. E.g. if 3 corners of the event space are *above* (or *left* from) the ε-stripe, and 1 corner is *inside* the ε-stripe, the intersection shape is always a triangle. For the relative position of a corner and the ε-stripe, we define the following cornercode *cc* of a point:

Definition 2 Cornercode (*cc*) of a point in the event space
A point (p, q) in the event space has the corner code $cc(p, q)$ with

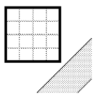
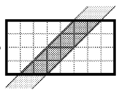
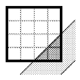
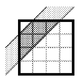
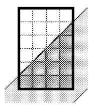
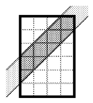
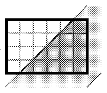
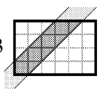
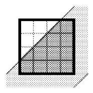
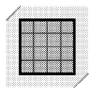
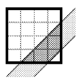
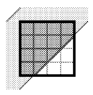
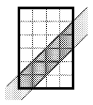
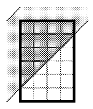
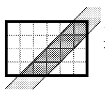
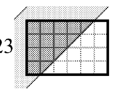
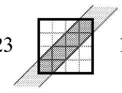
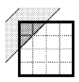
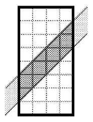
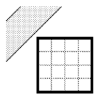
$$cc(p,q) = \begin{cases} 1 \text{ if } q > p + \varepsilon \\ 2 \text{ otherwise} \\ 3 \text{ if } q < p - \varepsilon \end{cases} \tag{5}$$

Intuitively, the cornercode is 1 if the point is left (or above) from the ε-stripe, 3 if it is right (or underneath) from the ε-stripe, and 2 if it is inside the ε-stripe (cf. figure 8). For an event space given by its upper and lower bounds (PL,PU,QL,QU), the corners are denoted as C_1, C_{2a}, C_{2b}, and C_3, as depicted in figure 7. We induce the cornercode for the event space given by lower and upper bounds.

Definition 3 Cornercode $cc(ES)$ of the event space
The cornercode of the event space ES given by the lower and upper limits $ES =$ (PL,PU,QL,QU) is the 4-tuple:

$$cc(ES = (cc(C_1), cc(C_{2a}), cc(C_{2b}), cc(C_3)) \tag{6}$$

Formally, there exist $3^4 = 81$ different 4-tuples over the alphabet {1,2,3}. However, not all these 4-tuples are geometrically meaningful. For instance it is not

Code	Figure	Formula	Code	Figure	Formula
1111		0.0	1313		$\frac{2\varepsilon}{PU - PL}$
1112		$\frac{(PU - QL + \varepsilon)^2}{2(PU - PL)(QU - QL)}$	1333		$\frac{(QU - PL + \varepsilon)^2}{2(PU - PL)(QU - QL)} - (\%)$
1122		$\frac{(PU + PL)/2 + \varepsilon - QL}{QU - QL}$	1233		$\frac{QU - (PU + PL)/2 + \varepsilon}{QU - QL} - (\%)$
1212		$\frac{PU - (QU + QL)/2 + \varepsilon}{PU - PL}$	1323		$\frac{(QU + QL)/2 + \varepsilon - PL}{(PU - PL)} - (\%)$
1222		1.0 – (%)	2222		1.0
1113		$\frac{(PU - QL + \varepsilon)^2}{2(PU - PL)(QU - QL)} - (*)$	2223		1.0 – (*)
1123		$\frac{(PU + PL)/2 + \varepsilon - QL}{QU - QL} - (*)$	2233		$\frac{QU - (PU + PL)/2 + \varepsilon}{QU - QL}$
1213		$\frac{PU - (QU + QL)/2 + \varepsilon}{PU - PL} - (*)$	2323		$\frac{(QU + QL)/2 + \varepsilon - PL}{(PU - PL)}$
1223		1.0 – (%) – (*)	2333		$\frac{(QU - PL + \varepsilon)^2}{2(PU - PL)(QU - QL)}$
1133		$\frac{2\varepsilon}{QU - QL}$	3333		0.0

where $(\%) = \frac{(QU - PL - \varepsilon)^2}{2(PU - PL)(QU - QL)}$ and $(*) = \frac{(PU - QL - \varepsilon)^2}{2(PU - PL)(QU - QL)}$

Fig. 6. Relative Positions of Event Space and ε-Stripe and Corresp. Probability Formulas

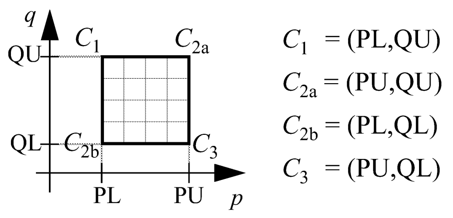


Fig. 7. Identifiers for the Corners of the Event Space

possible that simultaneously C_1 is below and C_3 above the ε-stripe. As C_1 is left from C_{2a} and C_{2a} is above C_3 we have the constraint:

$$cc(C_1) \leq cc(C_{2a}) \leq cc(C_3) \tag{7}$$

And as C_1 is above C_{2a} and C_{2b} is left from C_3 we have the constraint:

$$cc(C_1) \leq cc(C_{2b}) \leq cc(C_3) \tag{8}$$

The corner code of C_{2a} may be greater than, less than, or equal to the corner code of C_{2b}. The following lemma states that there are 20 different 4-tuples that fulfill the two constraints above.

Lemma 1. Completeness of Case Distinction
There are 20 different intersection shapes for the event space and the ε-tripe.

Proof. By complete enumeration of all four-tuples: There are 3 tuples where $cc(C_1) = cc(C_3)$: 1111, 2222, and 3333. If the difference between $cc(C_1)$ and $cc(C_3)$ is equal to 1 (i.e. tuples like 1??2 or 2??3), we obtain 2 possibilities for each of the corner Codes $cc(C_{2a})$ and $cc(C_{2b})$, i.e. $2 \cdot 2^2 = 8$ different tuples. For a difference of two between $cc(C_1)$ and $cc(C_3)$ which corresponds to tuples like 1??3, we have a choice out of three for each of the corners C_{2a} and C_{2b}, i.e. $3^2 = 9$ tuples. Summarized, we obtain 20 different tuples □

Note that the cornercodes 1111 and 3333 which are associated with a probability of 0.0 actually are never generated because the corresponding partitions have a distance of more than ε and, thus, are excluded by the preceding filter step.

Each corner code of the event space is associated with a geometric shape of the intersection between event space and ε-stripe. The shape varies from a triangle (e.g. $cc = 1112$) to a six-angle ($cc = 1223$). The fact that only 45° and 90° angles occur facilitates a simple and fast computation. Figure 6 shows the complete listing of all 20 shapes along with the corresponding corner codes and the formulas to compute the intersection area.

The concept of the cornercodes is not only a formal means to prove the completeness of our case distinction but also provides an efficient means to implement the area determination. Our algorithm computes the corner code for each of the 4 corners of the event space, concatenates them using arithmetic operations and performs a case analysis between the 20 cases.

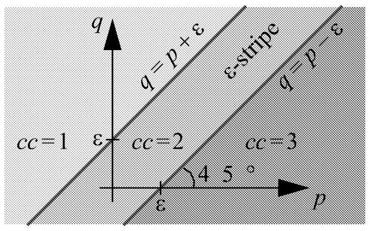


Fig. 8. The ε-stripe

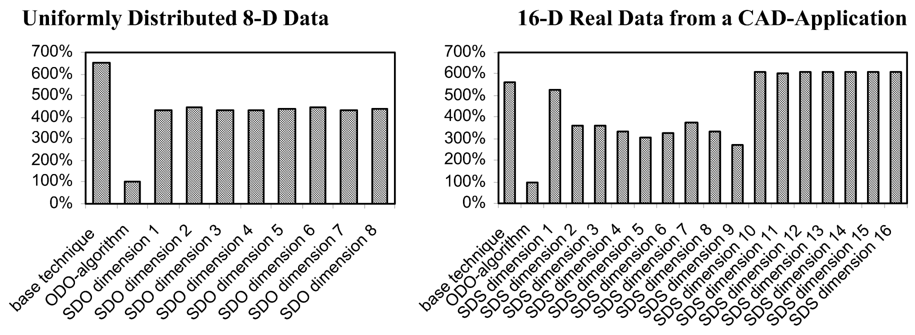


Fig. 9. Experimental Results for MuX: Plain Basic Technique, ODO and Simple DS

2.5 Determining the Optimal Sort Dimension

Our algorithm determines the sort dimension such that the mating probability $W_i[\varepsilon]$ is minimized. Ties are broken by random selection, i.e.

$$d_{\text{sort}} = \text{some}\{d_i | 0 \le i < d, \quad W_j[\varepsilon] \forall j, 0 \le j < d\}\,. \tag{9}$$

Thus, we have an easy way to evaluate the formula for the sort dimension. As $W_i[\varepsilon]$ merely is evaluated for each dimension d_i, thus keeping the current minimum and the corresponding dimension in local variables, the algorithm is linear in the dimensionality d of the data space and independent of all remaining parameters such as the number of points stored in the partitions, the selectivity of the query, etc. Moreover, the formula must be evaluated only once per pair of partitions. This constant (with respect to the capacity of the partition) effort is contrasted by potential savings which are quadratic in the capacity (number of points stored in a partition). The actual savings will be shown in the subsequent section.

3 Experimental Evaluation

In order to show the benefits of our technique we implemented our optimal dimension order on top of several basic similarity join methods and performed an extensive experimental evaluation using artificial and real data sets of varying size and dimensionality. For comparison we tested our algorithm not only against plain basic techniques, but also against a simple version of the dimension-order algorithm which does not calculate the best dimensions for each partition pair, but chooses one dimension which then is used globally for all partition pairs. In the following we will not only observe that our algorithm can improve CPU-efficiency by an important factor, but we will also see that it performs much better than the simple dimension-ordering algorithm – even if this algorithm chooses the best global dimension.

We integrated the ODO-algorithm into two index-based techniques, namely the *Multipage Index Join (MuX)* [8] and the *Z-order-RSJ* which is based on the *R-tree Spatial Join (RSJ)*[9] and employs a page scheduling strategy using Z-ordering. The latter is very similar to the *Breadth-First-R-tree-Join (BFRJ)* proposed in [11]. We also implemented the ODO-algorithm into the recently proposed *Epsilon Grid Order (EGO)* [7] which is a technique operating without preconstructed index.

The Multipage Index (MuX) is an index structure in which each page accommodates a secondary main-memory search structure which effectively improves the CPU performance of the similarity join. We implemented ODO on top of this secondary search structure, i.e. we measured the improvement that ODO brings on top of this secondary search structure. For comparison, we used the original MuX code which also exploited the secondary search structure.

All our experiments were carried out under Windows NT4.0 on Fujitsu-Siemens Celsius 400 machines equipped with a Pentium III 700 MHz processor and 256 MB main memory (128 MB available). The installed disk device was a Seagate ST3 10212A with a sustained transfer rate of about 9 MB/s and an average read access time of 8.9ms with an average latency time of 5.6ms.

Our 8-dimensional synthetic data sets consisted of up to 800,000 uniformly distributed points in the unit hypercube. Our real-world data set is a CAD database with 16-dimensional feature vectors extracted from geometrical parts.

The Euclidean distance was used for the similarity join. We determined the distance parameter ε for each data set such that it is suitable for clustering following the selection criteria proposed in [18] obtaining a reasonable selectivity.

Figure 9 shows our experiments comparing the overall runtime i.e. I/O- and CPU-time for the plain basic technique MuX either to MuX with integrated ODO or integrated simple dimension-order (SDO) for all possible start dimensions. The left diagram shows the results for uniformly distributed 8-dimensional artificial data while the right diagram shows results for 16-dimensional real data from a CAD-application. The database contained 100,000 points in each case. The SDO-algorithm depends heavily on the shape of the page regions i.e on the split algorithm used by the index employed by the basic technique. For uniformly distributed artificial data the loading procedure used by MuX treats all dimensions equally and therefore the results for the simple dimension-ordering algorithm are roughly the same for all start dimensions. ODO performs 6 times faster than plain MuX and 4 times faster than the best SDO while SDO itself is about 1.5 times faster than plain MuX. Note again that our algorithm chooses the most suitable dimension *for each pair of partitions*. Therefore, it is possible that ODO clearly outperforms the simple dimension ordering technique (SDO) even for its best dimension. For our real data set SDO shows varying performance with varying start dimension. We can even observe that for some start dimensions the overhead of SDO outweighs the savings and overall performance degrades slightly compared to the plain basic technique. This shows that it can be disadvantageous to apply dimension-ordering for one fixed start dimension. MuX with integrated ODO is about 5.5 times faster for the real data set than plain MuX while it is still 3 times faster than the SDO with the best performance, however it is more than 6 times faster than SDO with the worst performance.

Figure 10 shows all results for the uniformly distributed artificial data set for varying database size, including the diagram with distance calculations. We can see that the plain MuX performs up to 50 times more distance calculations than with ODO. The diagrams for the real data set are left out due to space restrictions.

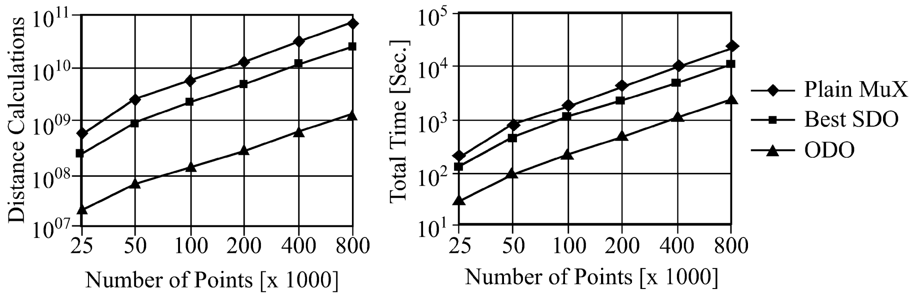


Fig. 10. Experimental Results for MuX: Uniformly Distributed 8-D Data

In order to show that the optimal dimension-ordering algorithm can be implemented on top of other basic techniques as well, we show the results for the *Z-order-RSJ* with uniformly distributed data in figure 11. Z-order-RSJ without ODO is up to 7 times slower than with integrated ODO and performs up to 58 times more distance calculations. The results for *Z-order-RSJ* with real data are shown in figure 12. We can see a speedup factor of 1.5 for SDO vs. plain *Z-order-RSJ* with respect to total time and of 1.8 with respect to distance calculations. ODO performs 3.5 times faster and performs 17 times fewer distance calculations than SDO while it performs 5.5 times faster and up to 25 times less distance calculations than SDO.

EGO was used to demonstrate integration of ODO with a basic technique that does not use a preconstructed index. The results are given in figure 13 where

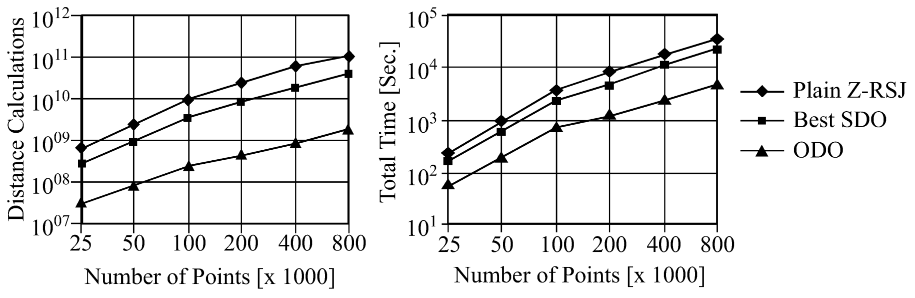


Fig. 11. Experimental Results for Z-RSJ: Uniformly Distributed 8-D Data

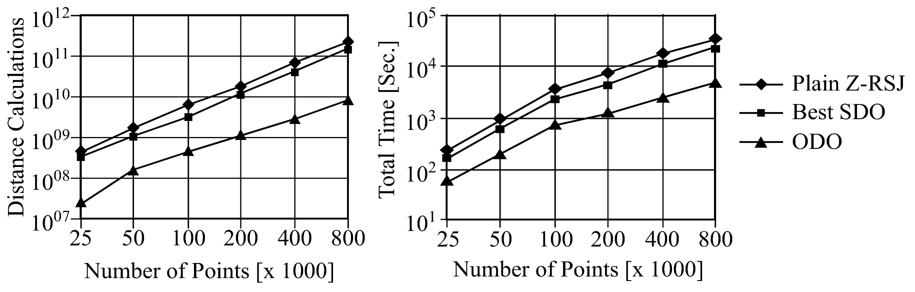


Fig. 12. Experimental Results for Z-RSJ: 16-D Real Data from a CAD-Application

EGO with SDO, as well as plain EGO clearly perform worse than ODO i.e. SDO is about 1.5 times faster than plain EGO, but ODO is twice as fast as SDO and outperforms plain EGO by a factor of 3.5.

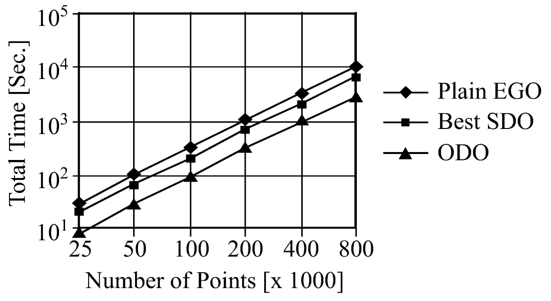


Fig. 13. Experimental Results for EGO (16d CAD data)

4 Conclusions

Many different algorithms for the efficient computation of the similarity join have been proposed in the past. While most well-known techniques concentrate on disk I/O operations, relatively few approaches are dedicated to the reduction of the computational cost, although the similarity join is clearly CPU bound. In this paper, we have proposed the Optimal Dimension Order, a generic technique which can be applied on top of many different basic algorithms for the similarity join to reduce the computational cost. The general idea is to avoid and accelerate the distance calculations between points by sorting the points according to a specific dimension. The most suitable dimension for each pair of pages is carefully chosen by a probability model. Our experimental evaluation shows substantial performance improvements for several basic join algorithms such as the multipage index, the ε-grid-order and the breadth-first-R-tree join.

References

1. Ankerst M., Breunig M.M., Kriegel H.-P., Sander J.: *OPTICS: Ordering Points To Identify the Clustering Structure*, ACM SIGMOD Int. Conf. on Management of Data, 1999.
2. Agrawal R., Lin K., Sawhney H., Shim K.: *Fast Similarity Search in the Presence of Noise, Scaling, and Translation in Time-Series Databases*, Int. Conf. on Very Large Data Bases (VLDB), 1995.
3. Arge L., Procopiuc O., Ramaswamy S., Suel T., Vitter J.S.: *Scalable Sweeping-Based Spatial Join*, Int. Conf. on Very Large Databases (VLDB), 1998.
4. Böhm C., Braunmüller B., Breunig M.M., Kriegel H.-P.: *Fast Clustering Based on High-Dimensional Similarity Joins*, Int. Conf. on Information Knowledge Management (CIKM), 2000.
5. Berchtold S., Böhm C., Jagadish H.V., Kriegel H.-P., Sander J.: *Independent Quantization: An Index Compression Technique for High Dimensional Spaces*, IEEE Int. Conf. on Data Engineering (ICDE), 2000.
6. Berchtold S., Böhm C., Keim D., Kriegel H.-P.: *A Cost Model For Neurest Neighbor Search in High-Dimensional Data Space*, ACM Symposium on Principles of Database Systems (PODS), 1997.
7. Böhm C., Braunmüller B., Krebs F., Kriegel H.-P.: *Epsilon Grid Order: An Algorithm for the Similarity Join on Massive High-Dimensional Data*, ACM SIGMOD Int. Conf. on Management of Data, 2001.
8. Böhm C., Kriegel H.-P.: *A Cost Model und Index Architecture for the Similarity Join*, IEEE Int. Conf. on Data Engineering (ICDE), 2001.
9. Brinkhoff T., Kriegel H.-P., Seeger B.: *Efficient Processing of Spatial Joins Using R-trees*, ACM SIGMOD Int. Conf. on Management of Data, 1993.
10. Brinkhoff T., Kriegel H.-P., Seeger B.: *Parallel Processing of Spatial Joins Sing R-trees*, IEEE Int. Conf. on Data Engineering (ICDE), 1996.
11. Huang Y.-W., Jing N., Rundensteiner E. A.: *Spatial Joins Using R-trees: Breadth-First Traversal with Global Optimizations*, Int. Conf. on Very Large Databases (VLDB), 1997.
12. Koudas N., Sevcik C.: *Size Separation Spatial Join*, ACM SIGMOD Int. Conf. on Managern. of Data, 1997.

13. Koudas N., Sevcik C.: *High Dimensional Similarity Joins: Algorithms and Performance Evaluation*, IEEE Int. Conf. on Data Engineering (ICDE), Best Paper Award, 1998.
14. Lo M.-L., Ravishankar C.V.: *Spatial Joins Using Seeded Trees*, ACM SIGMOD Int. Conf., 1994.
15. Lo M.-L., Ravishankar C.V.: *Spatial Hash Joins*, ACM SIGMOD Int. Conf, 1996.
16. Patel J.M., DeWitt D.J., *Partition Based Spatial-Merge Join*, ACM SIGMOD Int. Conf., 1996.
17. Preparata F.P., Shamos M.I.: 'Computational Geometry', Chapter 5 ('Proximity: Fundamental Algorithms'), Springer Verlag New York, 1985.
18. Sander J., Ester M., Kriegel H.-P., Xu X.: *Density-Based Clustering in Spatial Databases: The Algorithm GDBSCAN and its Applications*, Data Mining and Knowledge Discovery, Vol. 2, No. 2, 1998.
19. Shim K., Srikant R., Agrawal R.: *High-Dimensional Similarity Joins*, Int. Conf. on Data Engineering, 1997.
20. Ullman J.D.: *Database and Knowledge-Base Systems*, Vol. II, Computer Science Press, Rockville MD, 1989