Partition-Based Similarity Join Processing in High Dimensional Data Spaces

Hyoseop Shin[1], Bongki Moon[2], and Sukho Lee[1]

[1] School of Computer Science and Engineering
Seoul National University
Seoul, Korea
{hsshin@db,shlee@cse}.snu.ac.kr
[2] Department of Computer Science
University of Arizona
Tucson, AZ 85721
bkmoon@cs.arizona.edu

It is not desirable in the performance perspective of search algorithms to partition a high dimensional data space by dividing all the dimensions. This is because the number of cells resulted from partitioning explodes as the number of partitioning dimensions increases, thus making any search method based on space partitioning impractical. To address this problem, we propose an algorithm to dynamically select partitioning dimensions based on a data sampling method for efficient similarity join processing. Futhermore, a disk-based plane sweeping method is proposed to minimize the cost of joins between the partitioned cells. The experimental results show that the proposed schemes substantially improve the performance of the partition-based similarity joins in high dimensional data spaces.

1 Introduction

Efficient processing of similarity joins in a high dimensional space is crucial in many application areas such as image and multimedia databases, time-series databases, and data warehouse and data mining.

For two data sets, R and S, of d-dimensional data points, a similarity join query is formulated as:

$$R \bowtie S = \{(r,s) | \left(\sum_{i=1}^{d} |r_i - s_i|^p \right)^{1/p} \leq \varepsilon, r \in R, s \in S\} \quad (1)$$

where r and s are represented as $[r_1, r_2, \ldots, r_d]$ and $s = [s_1, s_2, \ldots, s_d]$, respectively. The formula returns point pairs of R and S whose distances in the data space are less than or equal to a cut-off similarity value, ε. The Euclidean distance, in which p is set to 2, is generally used to compute the distance between points.

Similarity join queries are similar to spatial join queries [4, 7] or distance join queries [6, 13] in the spatial databases in that both make use of the *overlap* or

distance between objects in a data space to determine whether to pair them together or not. The difference mainly lies in the dimensionality. In a low dimensional space, many index structures including R-tree [5] and its variants [2, 3] are available and these indexes can be useful in processing spatial joins efficiently. Even without indexes available, some partition-based spatial join algorithms [11, 10] are known to be competitive to the algorithms using indexes. Basically, most of these spatial join methods are based on the assumption that a data space can be partitioned in an efficient way. Though space partitioning methods in the low dimensions may be applicable to similarity join queries in high dimensional spaces, its direct application may cause serious problems. First of all, it is not possible in practical point of view to partition a high dimensional space by employing all the dimension axes. This is because the number of cells resulted from partitioning explodes as the number of dimension axes participating in the partitioning increases, thus making any search method based on space partitioning impractical. For example, let each dimension axis be divided into 10 continuous sub-intervals, then the number of cells generated after being partitioned will be $10^8, 10^{16}, 10^{32}, 10^{64}$ for 8, 16, 32, 64 dimensions, respectively, and it is likely that these numbers are usually larger than the number of the points in the original data sets before being partitioned. Another problem of the partition-based methods for similarity join processing in high dimensional data spaces is that most of the cells after being partitioned are so sparse that the cost of disk I/O for join processing between cells is likely to be very high. Lastly, there are so many spatially-neighbored cells for each partitioned cell that the number of joins between cells becomes larger, too. For these reasons, it is not desirable to employ all the dimension axes in partitioning high dimensional data spaces.

To address this problem, in this paper we propose an algorithm to dynamically select partitioning dimensions based on a data sampling method for efficient partition-based similarity join processing. The algorithm not only determines the number of dimensions for use in partitioning but also yields dimension axes which partition the data spaces to most efficiently process similarity joins for given data sets and a cut-off similarity value, ε. Furthermore, a disk-based plane sweeping method is proposed to minimize the cost of joins between skewed cells.

This paper is organized as follows. In Section 2, related work is described. Section 3 explains the overview of the partition-based similarity join. Section 4 presents the dimension selection algorithm for efficient partition-based similarity join processing. Section 5 presents a disk-based plane-sweeping method for join processing between skewed cells. Experimental results for the proposed methods are reported in Section 6. In Section 7, the conclusion of this paper is presented.

2 Related Work

Several methods for similarity join processing in high dimensional data spaces have been reported in the literature. Shim et al. [12] proposed an indexing structure, ε-kdB-tree, to process similarity join queries. The data space is subdivided into a series of stripes of the width, ε, along one selected dimension axis. And

then, each stripe is constructed as a main memory data structure, the ε-kdB-tree in which dimension axes are chosen in order to partition the space recursively so that the search space is reduced while processing joins between nodes of the ε-kdB-trees. The main drawback of this method is that severe performance decrease occurs due to possible random disk accesses if each tree does not fit in main memory. Koudas and Sevick [8] proposed to use space filling curves to partition the high dimensional spaces. Each data point is converted as a hypercube of the side-length, ε, and then each cube is assigned a level value corresponding to the size of the largest cell that contains the cube. In joining step, each cube is associated with the cubes in the other set of which level is higher than or same as its own level. This method also has been reported to suffer performance decrease with increasing dimensions, as most of the cubes are mapped to a very high level [1]. Böhm et al. [1] proposed the epsilon grid order algorithm for the similarity join problem. A regular grid of the side-length, ε, is laid over the data space, anchored in the origin. Then, the lexicographical order is defined on the grid cells. With this ordering of data points, it can be shown that all join mates of a point p lie within an interval of the file. The lower and upper limit of the interval is determined by subtracting and adding the vector $[\epsilon, \epsilon, \ldots, \epsilon]^T$ to p, respectively. This is illustrated in Fig. 1. In Fig. 1, the striped grid cells are considered to be computed for pairing with the specified grid cell, p. The number of grid cells in an interval, however, tends to get larger as the dimension increases.

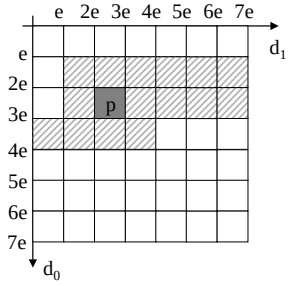


Fig. 1. Epsilon Grid Order-Based Similarity Join

The TV-tree [9] has been proposed as a structure to index the high dimensional data using only part of the entire dimensions. The TV-tree is mainly for efficient search on single data set, while the dimension selection method in this paper dynamically selects the dimensions for efficient similarity joins on two associated data sets.

3 Partition-Based Similarity Joins

In this section, the general approach of the partition-based similarity join is described and the difficulties in applying the approach for high dimensional data are explained. The partition-based similarity join consists of two steps, partition

step and join step. In the partition step, the entire data space is partitioned into cube-shaped cells by dividing each dimension. We assume without loss of generality that all the data points are within an unit hypercube. As each dimension value ranges between [0,1], each dimension is partitioned into $\lceil 1/\varepsilon \rceil$ intervals of the width, ε. And then, each point in the data sets, R and S, which participate in the similarity join, is assigned to the cell to which it belongs. Note that two separate sets of cells exist for the two data sets. The Fig. 2 illustrates the partitioned data space after the partition step. The small rectangles in the figure represent the resulted cells.

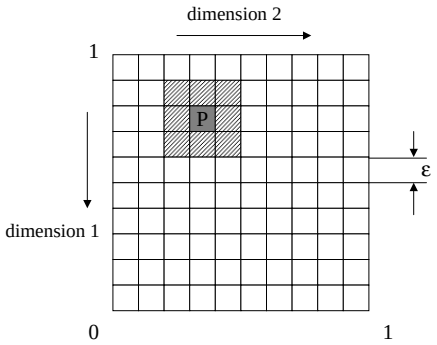


Fig. 2. Partition-Based Similarity Join : The Partition Step

In the join step of the partition-based similarity join, actual join computations are performed between cells from the two input data sets. Each cell from R does not need to be paired with every cell from S, but is paired only with the cells which it overlaps or neighbors in the data space. For example, in the Fig. 2, a cell, P shall be paired with one cell overlapping with it and eight cells surrounding it. Generally, in d dimensional data space, a cell in a data set that is not located at a border of the unit hypercube should be paired with 3^d cells in the other set.

4 Partitioning Dimensions

4.1 Motivation

In case of high dimensions, if a data space is divided by employing all the dimensions during the partition step, the number of cells resulted from the partitioning may explode so that the data skew problem can be serious. This consequently causes to lessen the effect of search space reduction by space partitioning, while increasing additional costs during the join step.

Theoretically, under an assumption that data points are uniformly distributed in the data space, given the two input data sets, R and S, and the number of partitioning dimensions, d_p, the CPU cost of the partition-based similarity join,

which is computed by counting the number of distance computations between data points, is formulated as follows:

$$Cost(CPU) = |R| \times |S| \times (\frac{3}{\lceil 1/\varepsilon \rceil})^{d_p} \tag{2}$$

Meanwhile, the disk I/O cost, which is computed by counting the number of disk blocks visited, is formulated as follows:

$$Cost(IO) = |R|_{block} + 3^{d_p} \times |S|_{block} \tag{3}$$

for the total number of disk blocks, $|R|_{block}$ for R and $|S|_{block}$ for S.

According to the Equation 2 and 3, as the partitioning dimension, d_p, increases, the CPU cost of the partition-based similarity join decreases(if we assume that $\lceil 1/\varepsilon \rceil > 3$), while the I/O cost increases. This implies that there is a trade-off between the CPU cost and the I/O cost in regard to the performance of the partition-based similarity joins and it is desirable to determine the converging dimensionality. In this aspect, we propose methods for the partition-based similarity joins to determine the number of partitioning dimensions and to select partitioning dimensions.

4.2 The Number of Partitioning Dimensions

As mentioned in the previous section, all the dimensions cannot be used in dividing the data space. To determine the number of partitioning dimensions, we pre-estimated average cell size to be same as the size of a disk block. Then, the number of cells, N_p, to be generated after a partition step is computed as:

$$N_p = \frac{Min(|R|_{block}, |S|_{block})}{blockSize} \tag{4}$$

for the size of a disk block, $blockSize$. And also, the number of cells after the partition step can be computed in other way for given the number of partitioning dimension, d_p, and a partition sub-interval, ε:

$$N'_p = (\lceil 1/\varepsilon \rceil)^{d_p} \tag{5}$$

From the equation, $N_p = N'_p$, we can obtain the number of partitioning dimensions, d_p, as follows:

$$d_p = \frac{log \frac{Min(|R|_{block}, |S|_{block})}{blockSize}}{log \lceil 1/\varepsilon \rceil} \tag{6}$$

4.3 Selection of Partitioning Dimensions

With increasing dimensions, the domain cardinality of a data space grows exponentially ($O(c^d)$ for a dimension d), and accordingly data points in a high dimensional data space are likely to be distributed sparsely and skewedly in most regions of the data space. This data skewness can also happen when data points are projected into each dimension axis. So, it is desirable to select dimension axes that show more uniform data distributions for efficient partitioning of a data space. As the similarity join processing associate two input data sets, a dimension which yields rather uniform data distribution on one input data set may yield non-uniform data distribution on the other input data set. Futhermore, the degree of the uniformity even for one data set can change according to the cut-off similarity value. For these reasons, the partitioning dimensions for the similarity joins should be selected under the consideration of the associated two input data sets as well as the cut-off similarity value.

In this paper, to determine partitioning dimensions, we pre-computed the expected join cost for each dimension when data points of the input data sets are projected onto each dimension axis. After the join cost for each dimension compared, the d_p dimensions for which the join costs are smallest are selected as the partitioning dimensions. Note that the number of partitioning dimensions, d_p, is determined by the Equation 6. Algorithm 1 represents the algorithm, *DimSelect* which selects partitioning dimensions for the partition-based similarity joins.

The join cost for a dimension is computed as follows. First, as the space of a dimension axis is divided by a similarity cut-off value, ε, the space is divided into $\lceil 1/\varepsilon \rceil$ cells of length, ε. Second, the number of data points to be included in each cell from the input data sets are counted(line 4-6, 7-9). Third, each cell in one input data set is paired with three cells in the other input data set, a cell at the left side of it, one at the right side of it, and one overlapping with it. This is illustrated in Fig. 3 for the two input data sets, R and S.

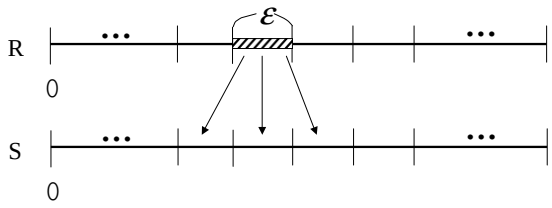


Fig. 3. The Computation of the Join Cost for a Partitioning Dimension

The cost of a join between two cells is computed by counting the number of distance computations between data points from the two cells. The number of distance computations between cells are computed by multiplying the number of data points of the cells(line 10-14). The total join cost for a dimension is obtained by the summation of the join costs between cells.

Note here that the data points of the two input data sets are not required to be actually partitioned into each cell to examine the number of data points belonging to each cell on a dimension. Moreover, to avoid a sequential scan of the entire data sets in estimating the join costs of the dimensions, we sampled small part of the data sets.

5 Disk-Based Plane Sweeping

The d_p partitioning dimensions chosen, the $(\lceil 1/\varepsilon \rceil)^{d_p}$ cells are generated for each input data set after the partition step. Each cell in one data set participates in cell-based join computations with 3^{d_p} cells in the other input data set which overlap or intersect with it in the data space. Cell-based joins are processed through a plane-sweeping method to reduce join costs. For a plane-sweeping processing, data points in each cell are sorted on a dimension axis during the partition step.

As balanced data distributions among cells are not guaranteed, some cells may not fit in the memory buffer allowed in a system. In this case, a memory-based plane-sweeping method assuming that two sides participating in the plane-sweeping should be resident in the main memory may cause excessive disk I/O costs because data points remaining in the disk space should be read into the

Algorithm 1: *DimSelect*: Partitioning Dimension Selection Algorithm

set the number of cells, $n_p \leftarrow \lceil 1/\varepsilon \rceil$;
initialize cell arrays, $P_R[1 \ldots d][1 \ldots d] \leftarrow 0, P_S[1 \ldots d][1 \ldots d] \leftarrow 0$;
initialize the number of distance computations for each dimension, $JoinCost[1 \ldots d] \leftarrow 0$;

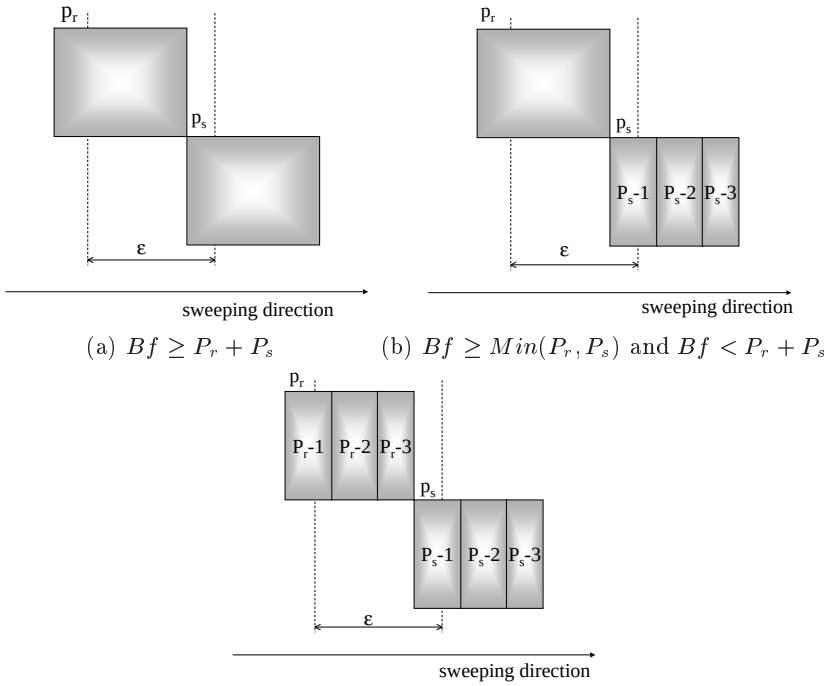
// compute the number of entities for each partition
for *each entity* $(e_1, e_2, \ldots, e_d)$ *in* R_s **do**
    for *each dimension* i *in* $[1 \ldots d]$ **do**
        $P_R[i][\lceil n_p \times e_i \rceil] + +$;
    end
end
for *each entity* $(e_1, e_2, \ldots, e_d)$ *in* S_s **do**
    for *each dimension* i *in* $[1 \ldots d]$ **do**
        $P_S[i][\lceil n_p \times e_i \rceil] + +$;
    end
end

// compute the join cost for each dimension
for *each dimension* i *in* $[1 \ldots d]$ **do**
    for *each cell number* p *in* $[1 \ldots n_p]$ **do**
        if $i > 1$ **then** $JoinCost[i] \leftarrow JoinCost[i] + P_R[i][p] \times P_S[i][p-1]$;
        $JoinCost[i] \leftarrow JoinCost[i] + P_R[i][p] \times P_S[i][p-1]$;
        if $i < n_p$ **then** $JoinCost[i] \leftarrow JoinCost[i] + P_R[i][p] \times P_S[i][p+1]$;
    end
end

return the d_p dimensions for which $JoinCost[d]$ are the smallest;

memory buffer every time a sweeping is processed. To prevent this overhead, we propose three different schemes of plane-sweeping under considering the relationships between the cell sizes and the memory buffer size.

First, if both the cells participating in a plane-sweeping fit in the buffer, a memory-based plane-sweeping method is applied without modification. This case is illustrated in Fig. 4-(a). Both cells are processed after being fetched into the buffer. Second, if only one of the cells fit in the buffer, the cell is fetched into the buffer and the other cell is split to fit in the buffer. plane-sweeping is processed over one to many relationships. This case is illustrated in Fig. 4-(b). Last, if neither of the cells fit in the buffer, both cells are splited before fetched into the buffer. This case is illustrated in Fig. 4-(c).



(a) $Bf \geq P_r + P_s$ (b) $Bf \geq Min(P_r, P_s)$ and $Bf < P_r + P_s$

(c) $Bf < Min(P_r, P_s)$

Fig. 4. Disk-based Plane-Sweeping

Note that for the last case, plane-sweeping is needless between splited subcells for which the axis distance is larger than ε. For example, in Fig.4-(c), $P_r - 1$ is not considered for pairing with $P_s - 2$ and $P_s - 3$.

6 Experimental Results

In this section, we evaluate the proposed methods for the partition-based similarity join in high dimensional data spaces. We implemented three algorithms. The

algorithm, $NaiveSJ$, computes the distances of all possible pairs of points from the given data sets to get results. The second algorithm, $PBSJ$ uses the method proposed in this paper for the selection of partitioning dimensions but the disk-based plane sweeping method is not included. The last algorithm, $S-PBSJ$ employs the disk-based plane sweeping method as well. In the experiments, we used two sets of 1,000,000 16-dimensional points, each of which is a color histogram value of an image.

The experiments were performed by varying cut-off similarity values, ε, into 0.001, 0.01, 0.1. As Fig. 5 shows the results, our algorithms, $PBSJ$ and $P-PBSJ$ were always better than $NaiveSJ$ by more than a magnitude. This re-

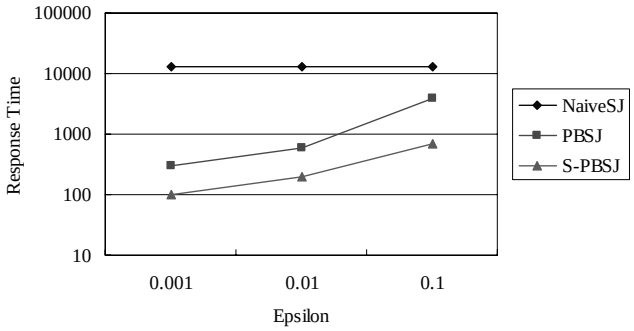


Fig. 5. Comparison of Response Times

sult implies that the partition-based similarity join using the selective partion dimensions is quite effective compared with a naive method. Note that a partition-based similarity join of partitioning all the dimensions in high dimensional data spaces are worse than $NaiveSJ$, because the number of partitions explodes. For this reason, we did not consider the implementation of that method.

Between $PBSJ$ and $P-PBSJ$, $P-PBSJ$ performed better than $PBSJ$ by up to several times. This implies that the disk-based plane sweeping is quite effective in computing joins between partitioned cells. The disk-based plane sweeping caused to reduce the IO cost of joins as well as the CPU cost. Specifically, the reduction of the IO cost is caused by efficient accesses to the cells whose sizes are larger than main memory size during disk-based plane sweeping processing.

7 Conclusion

Partition-based approaches are not directly applicable to the similarity join processing in high dimensional data spaces. This is mainly because the number of cells resulted from partitioning is too large and thus unreasonable CPU and IO costs are spent. This paper proposed a dimension selection method which dynamically selects the most efficient dimensions to be partitioned for the similarity join. Furthermore, a disk-based plane sweeping method was proposed to

minimize the cost of joins between partitioned cells. The experimental results showed that the proposed methods enhanced the partition-based similarity join in high dimensional data spaces by more than a magnitude. Future work includes an optimized soluiton to the disk-based plane sweeping as the data distributions among partitioned cells are very skewed.

References

1. C. Böhm, B. Braunmuller, F. Krebs, and H.-P. Kriegel. Epsilon grid order: An algorithm for the similarity join on massive high-dimensional data. In *Proceedings of the 2001 ACM-SIGMOD Conference*, 2001.
2. Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. The R^*-tree: An efficient and robust access method for points and rectangles. In *Proceedings of the 1990 ACM-SIGMOD Conference*, pages 322–331, Atlantic City, NJ, May 1990.
3. S. Berchtold, D. A. Keim, and H.-P. Kriegel. The X-tree: An index structure for high-dimensional data. In *Proceedings of the 22nd VLDB Conference*, Bombay, India, September 1996.
4. Thomas Brinkhoff, Hans-Peter Kriegel, and Bernhard Seeger. Efficient processing of spatial joins using R-Trees. In *Proceedings of the 1993 ACM-SIGMOD Conference*, pages 237–246, Washington, DC, May 1993.
5. Antonin Guttman. R-Trees: A dynamic index structure for spatial searching. In *Proceedings of the 1984 ACM-SIGMOD Conference*, pages 47–57, Boston, MA, June 1984.
6. Gisli R. Hjaltason and Hanan Samet. Incremental distance join algorithms for spatial databases. In *Proceedings of the 1998 ACM-SIGMOD Conference*, pages 237–248, Seattle, WA, June 1998.
7. Y.W. Huang, N. Jing, and E. Rundensteiner. Spatial joins using R-trees : Breadth-first traversal with global optimizations. In *Proceedings of the 23rd VLDB Conference*, 1997.
8. N. Koudas and C. Sevcik. High dimensional similarity joins: Algorithms and performance evaluation. In *Proceedings of the 1998 IEEE International Conference on Data Engineering*, 1998.
9. K. Lin, H. V. Jagadish, and C. Faloutsos. The TV-tree: An index structure for high-dimensional data. *VLDB Journal*, 3:517–542, 1995.
10. Ming-Ling Lo and Chinya V. Ravishankar. Spatial hash-join. In *Proceedings of the 1996 ACM-SIGMOD Conference*, pages 247–258, Montreal, Canada, June 1996.
11. Jignesh M. Patel and David J. DeWitt. Partition based spatial-merge join. In *Proceedings of the 1996 ACM-SIGMOD Conference*, pages 259–270, Montreal, Canada, June 1996.
12. K. Shim, R. Srikant, and R. Agrawal. High-dimensional similarity joins. In *Proceedings of the 1997 IEEE International Conference on Data Engineering*, 1997.
13. Hyoseop Shin, Bongki Moon, and Sukho Lee. Adaptive multi-stage distance join processing. In *Proceedings of the 2000 ACM-SIGMOD Conference*, pages 343–354, Dallas, TX, May 2000.