Title: Optimizing Data Page Placement for Similarity Join in Metric Spaces

By: Ving Ian Lei

Abstract

In this paper, we address the similarity join problem in metric space. We describe how page locality may be improved by reorganizing disk pages using an algorithm borrowed from numerical methods concerning the reorganization of the rows and columns of matrices to form banded matrices. The model is further exploited as a method to schedule the operations of similarity join for finding all pairs of neighbors in a metric space. Experimental results demonstrate significant performance improvements over the symmetric clustering join algorithm proposed in [15] and the cost-based clustering algorithm presented in [7].

1. Introduction

Current databases are growing rapidly. Many works in the literature focus on the retrieval, range queries, and the nearest neighbor problems. Similarity join is an important primitive on such databases and deserves much attention also.

One example can be found in computational biology. In molecular biology, DNA and protein sequence are the basic object of study. We usually index a given sequence of characters inside a longer sequence. In this case, an exact match is unlikely, and computational biologists are more interested in the problem that given a short sequence, identify parts of a longer sequence which are similar to it. Genetic streams that describe sequences of similar functionality, evolution etc. often differs slightly. This makes the search not exact. The measure of similarity used depends on the type of differences one is interested in.

Another example can be found in audio and video compression. The transmission of audio and video over a narrow-band channel, such as internet-based audio and video conferencing, is an important problem. One can think of a frame as formed by a set of sub-frames. To solve the problem of audio and video transmission, we can send the first frame as is, and for each subsequent frame, we only send those sub-frames that differ a lot from the previously sent sub-frames. This poses the need to find sub-frames that are similar to the one that is about to be sent among a collection of recently sent sub-frames.

In this paper, we address the similarity join in the context of general metric space. Joining two large datasets are expensive. It involves two major costs: I/O cost and CPU cost. The CPU cost is fixed since there is a fixed cost for the joining of each pair of data objects and the number of pairs to be joined is fixed. On the other hand, I/O cost is not fixed. The buffer space in the memory is limited and so some of the pages may have to be read

several times from the disk. Therefore, the schedule of reading the pages from disk determines the I/O cost.

Traditional techniques for doing joins do not work well in these large databases. The major bottleneck of doing joins is the disk I/O. When the database is much larger than the available buffer, most of the pages that will be used later may be removed from the buffer. In this case, most of the pages need to be fetched a multiple of times. Some approaches to this problem have been proposed. However, most of these algorithms address the special case for spatial datasets only. Different indexing schemes have also been proposed to speed up similarity searches.

Our contribution

In this paper, our contribution is twofold. First, observe that if pages containing data that are close together in the data space are stored close together on disk, the seek operations during a join operation will be reduce. We present a method for improving page locality in metric-space. This is achieved by first constructing a page-connectivity graph (PCG) on the MVP-tree index of a dataset based on self-join. After that, we reorder the rows and columns of the adjacency matrix of this graph to bring in as many marked entries near the diagonal as possible using a heuristic for band-diagonalization from [15]. The disk pages of the MVP-tree are then reorganized according to the order of the rows (or columns) of the symmetric adjacency matrix to improve page locality. We also propose an algorithm for joining two datasets in metric-space. A PCG is first constructed on the two datasets based on the given join threshold. A marked entry in the matrix indicates that joining the corresponding page pair may potentially contribute to the join of these datasets. We then schedule the processing of the marked entries of the matrix in blocks that fit into the available number of buffer pages. Our goal is to minimize the total I/O cost.

1.1 Page-Connectivity Graph

A page-connectivity graph (PCG) [10] describes the join relationship between two datasets at the page level. It is a bipartite graph $G_B = (V_1, V_2, E)$ where vertex set V_1 represents the pages from the first dataset, and vertex set V_2 represents the pages from the second dataset. There is an edge between page v_i in V_1 and page v_j in V_2 if and only if there is at least one object from v_i and one object from v_j that satisfy the join predicate.

We can partition the nodes in $V = V_1 \cup V_2$ of the graph $G_B = (V_1, V_2, E)$ into disjoint subsets while minimizing the number of edges between two different partitions. This is called a min-cut node partition. The set of edges whose incident nodes are in two different partitions is the cut-set of the min-cut partition. We will use an efficient min-cut partition algorithm to cluster the pages in a PCG.

The rest of the paper is organized as follows. In section 2, we present a brief overview of

our techniques. In section 3, we explain our approach to improve page locality. We describe the construction of a PCG, and the band-diagonalization heuristic algorithm that we used. In section 4, we present our join algorithm. Section 5 gives the experimental results. We conclude our paper in section 6.

2. An Overview of Our Technique

In this paper, we will assume a finite buffer of B pages and an affine disk model. For each dataset, we assume that a multi-vantage point tree (MVP-tree) [2] index structure has been built on it.

We choose to use an affine disk model [14] because it is a more realistic model than simply counting the number of disk I/O's. It is based on the observation that it may be cheaper to read a few empty pages than to skip them. The cost (i.e. the elapsed time) of reading a page mainly consists of two parts: seek time and transfer time. The seek time is the time to move the arm of the disk to the right position in the file. The transfer time is the time to transfer a page from disk into main memory. Let s be the seek time and t be the transfer time. The cost of a read request transferring n pages is $s + t * n$. The average seek time is usually much higher than transfer time. Therefore, if two or more required pages are located close to each other, the total retrieval time may be reduced if all of them are read in one request, even if some not required pages are read too, instead of reading each page one by one. Of course, this requires additional buffer space. A sequence of not required pages is called a gap. In this paper, we assume that the gap size $g = s / t$.

To improve page locality of a dataset, we start by constructing a PCG based on self-join. We represent the PCG with an adjacent matrix. Each entry m_{ij} of this matrix associates the ith page of the dataset to the jth page. If the bounding predicate of page i intersects with that of page j, the corresponding entry m_{ij} in the matrix is marked. A marked entry in the adjacency matrix indicates that the corresponding pages in the row and the column are closely related. We then try to permute the rows and columns of the matrix so as to bring in as many marked entries near the diagonal as possible. This is achieved by a heuristic for band-diagonalization presented in [15]. This approach exploits global information across the entire PCG. The band-diagonalization heuristic consists of two steps. First, we use a min-cut partition algorithm, Metis [8], to divide the nodes of the PCG into disjoint subsets while minimizing the number of edges between different partitions. After that, we use a greedy algorithm to order the partitions. From this reordered adjacency matrix, we can determine a good ordering of disk pages since all the similar pages are brought close to each other along the diagonal of the matrix.

When joining two different datasets, we can use the same idea of a PCG described above. First, we construct a PCG on the two datasets based on the given join predicate (i.e. a join threshold). This gives us an estimate of what page pairs will join. Based on the adjacency matrix of this PCG, we then devise a way to schedule the page reads from the datasets so

as to minimize the I/O costs. This algorithm is given in section 4.

3. Improvement on Page Locality

Page locality plays an important role in determining the total I/O cost for similarity search and similarity join in metric-space databases. However, determining which data pages are holding closely related data and thus should be placed near each other on disk is not easy. We propose a method to improve page locality of a dataset through a global approach based on band-diagonalization of the adjacency matrix of the PCG derived from the dataset. This adjacency matrix is constructed based on self-join, and thus each marked entry in it denotes the similarity between the corresponding page-pair under the join range used. We further band-diagonalize the matrix so as to maximize the number of marked entries near the diagonal. This has the effect of clustering closely related pages together. We use a heuristic for band-diagonalization presented in [15]. Based on the row order (or column order, these are the same since the matrix is symmetric) of the band-diagonalized adjacency matrix, we reorder the data pages on disk accordingly so as to achieve page locality. Next, we will describe the construction of PCG and the band-diagonalization heuristic.

3. 1 Construction of the Page-Connectivity Graph

In order to find out which pages may contain objects that are likely to join together given a join range, we construct a PCG. Figure 1 presents the algorithm that constructs the adjacency matrix of the PCG. It gives a general algorithm that works for two datasets (As we will see later in Section 4, this algorithm will be used again when we present our similarity join algorithm). In the case of self-join, the same dataset is used for both arguments. We assume that an MVP-tree index structure has been built on each of the datasets. We also assume that the size of each leaf node is the same as one disk page. This means that each node access from disk takes one disk I/O. This algorithm takes in the join threshold and the root nodes of the two MVP-trees of the joining datasets as inputs. The two trees are being traversed recursively, and the adjacency matrix is marked accordingly when two leaf node overlaps.

In this algorithm, there are four cases to consider. When we are processing two internal nodes, we first calculate the distances between each pair of vantage points of the two nodes. After that, we decide for each pair of children of the two internal nodes whether they overlap or not based on the Pruning Conditions (I will describe the Pruning Conditions later in this section). If they overlap, the algorithm will be called again on those two overlapping nodes for further processing; otherwise, those two nodes are pruned, meaning that they will not be considered further. We get to the second and the third cases when we are processing an internal node from one tree and a leaf node from the other. In this case, we do not have to do much. We simply recurs the algorithm on the children of the internal node with the leaf node. When we are processing two leaf node,

which is the last case, we know that they do overlap and so we can mark the appropriate entry in the adjacency matrix of the PCG accordingly.

As mentioned above, we prune internal nodes in the first case based on the Pruning Conditions. These are the conditions under which two nodes can be considered safely as non-overlapping. Figure 2 and Figure 3 illustrated these conditions. When the distance *d* between two vantage points is greater than the sum of the distances from each vantage point to its outer bound, $R1_{max}$ and $R2_{max}$, the two nodes do not overlap. This is Pruning Condition 1. Two nodes also do not overlap if the sum of *d* and the distance from one vantage point to its outer bound, $R1_{max}$ or $R2_{max}$, is less than the distance of the other vantage point to its inner bound, $R2_{min}$ or $R1_{min}$. This is Pruning Condition 2. Incorporating the join threshold into these conditions, we have the following conditions for pruning nodes:

1) $d > R1_{max} + R2_{max} + \varepsilon$

2) $d + R1_{max} + (\varepsilon /2) < R2_{min} - (\varepsilon /2)$ OR $d + R2_{max} + (\varepsilon /2) < R1_{min} - (\varepsilon /2)$

If either one of these conditions is satisfied, the two nodes will not be processed further.

```
/*
   Let R be the root of the MVP-tree of the first dataset.
   Let S be the root of the MVP-tree of the second dataset.
   Let ε be the join threshold.
   Let M = ( m_{i,j} ) be the adjacency matrix of the PCG (originally, all entries are 0's).
*/
Algorithm PCG ( R, S, ε )

Case 1. R is an internal node and S is an internal node
1. Calculate the distances between each pair of vantage points of the two nodes
2. Check if any children of R overlap with any children of S
     For each child i of R
                For each child j of S
                   For each VP of R
                              For each VP of S
                     If the Pruning Conditions are satisfied
                        PRUNE i, j
                     else
                        PCG ( i, j, ε )

Case 2. R is an internal node and S is a leaf node.
    For each child i of R
```

PCG (i, S, ε)

Case 3. *R* is a leaf node and *S* is an internal node.
For each child *i* of S
PCG (R, i, ε)

Case 4. *R* is a leaf node and *S* is a leaf node.
Mark *M(R, S)* = 1

Figure 1. Algorithm for constructing the adjacency matrix of the PCG.

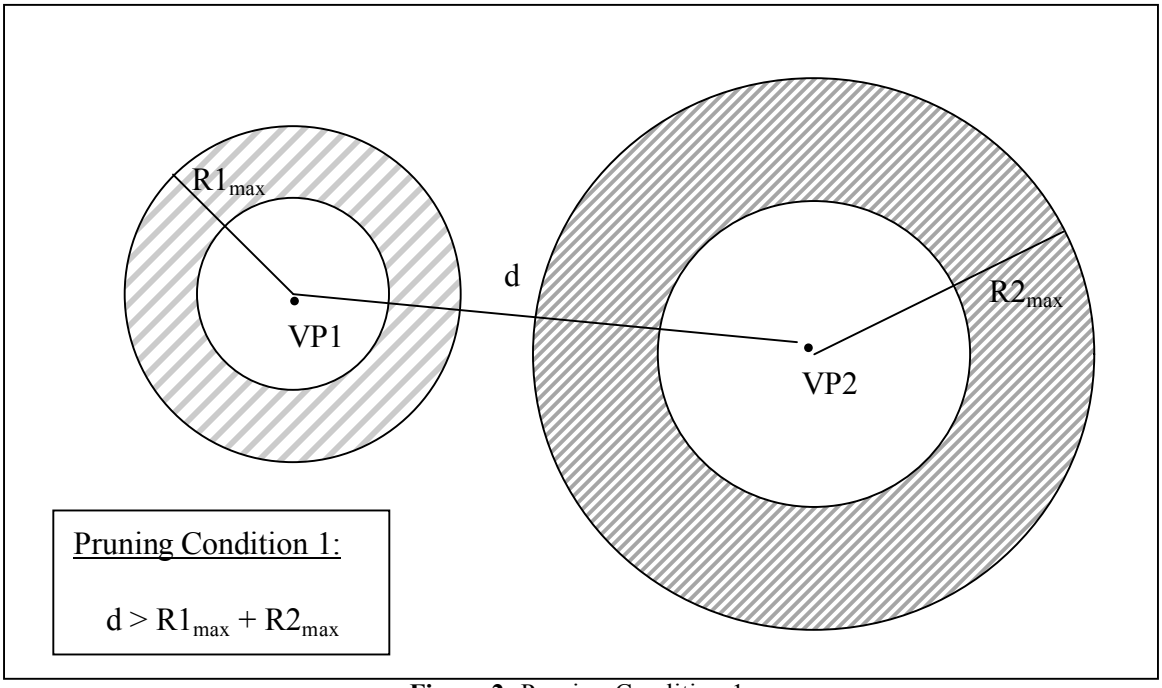


Figure 2. Pruning Condition 1.

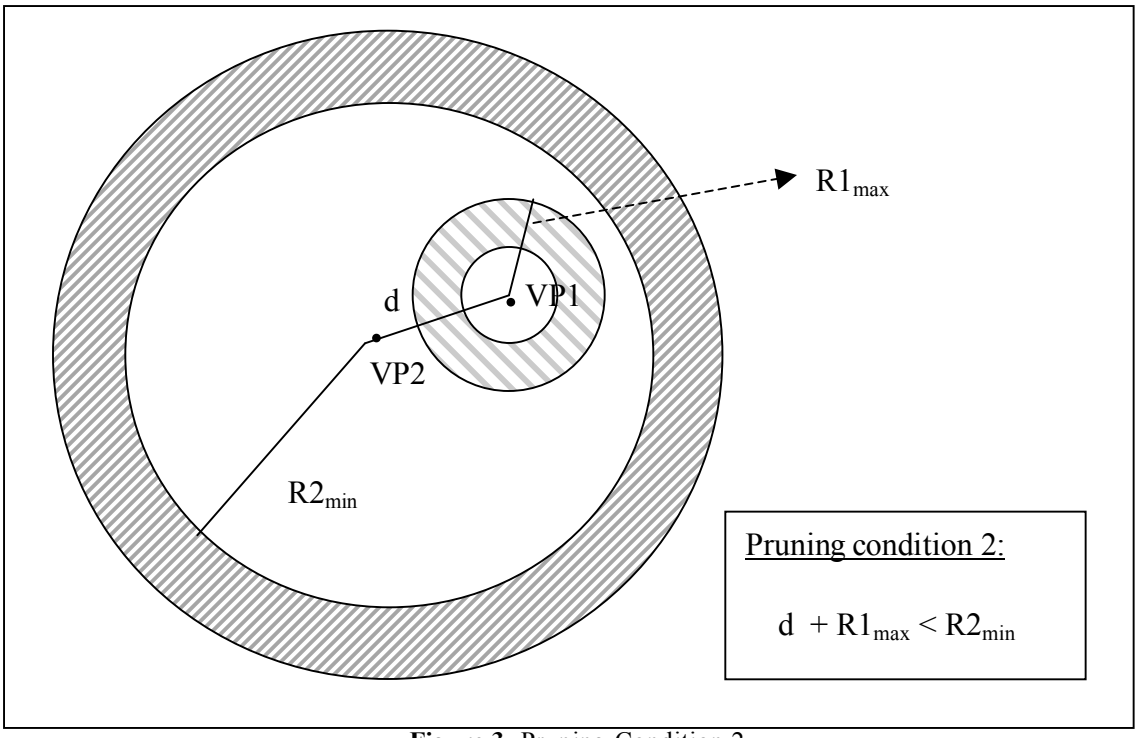


Figure 3. Pruning Condition 2.

3.2 Band-diagonalization

After we have constructed the adjacency matrix of the PCG, we rearrange the rows and columns of the matrix to bring in as many marked entries as possible near the diagonal. This is achieved by the band-diagonalization algorithm presented in [15].

Band-diagonalization can be based on various algorithms such as specialized envelope-reduction algorithms [1], [3], or min-cut graph partition algorithms [8], [9].
In this paper, we use the min-cut graph partition algorithms, just as in [15]. This heuristic approach for band-diagonalization is given in Figure 4.

/*
Let $M = (m_{i,j})$ be the adjacency matrix of a page-connectivity graph.
*/
Algorithm *BD (M)*

1. Partition M by a graph partitioning software, e.g. Metis.
Partitions_Set = Graph_Partition(M)

2. Order the partitions in Partitions_Set by the number of cut-edges between the nodes in different partitions in descending order.

Figure 4. Heuristic for band-diagonalization of the page-connectivity graph.

Given the adjacency matrix of a PCG, we use the min-cut graph-partitioning software Metis [8] to partition this bipartite graph. A min-cut partition algorithm divides the nodes of the PCG into disjoint subsets while minimizing the number of edges between different partitions.

After we have partitioned the PCG, we determine an order for the partitions so as to maximize the number of marked entries along the diagonal. This is achieved by putting partitions that share a higher number of cut-edges next to each other through a greedy algorithm. First, we construct a partition-interaction matrix M [15]. The rows and columns of M represent the partitions. Each entry in M such as $M[P_i, P_j]$ lists the number of cut-edges between the partition P_i and P_j. After that, we sort the entries in M in descending order, breaking ties arbitrarily. Then we pick an entry with the largest value. This gives us a partition order with two partitions. We then extend this partition order on both sides greedily, choosing the highest value of $M[P_2, P_i]$ and $M[P_3, P_j]$ in M. This process gives us an order of the partitions. We can then reorder the rows and columns of the adjacency matrix of the PCG according to this partition order. The rows and columns within each partition are arranged arbitrarily.

4. Proposed Similarity Join Algorithm

There are two major costs involved in similarity join queries. These are the I/O cost for reading pages from disk and CPU cost for joining the objects after the pages are read into main memory. The cost of reading a set of disk pages highly depends on the location of these pages. This is because the random seek cost is much higher than a sequential page transfer cost. Therefore, reading a set of pages is cheaper if they are located close to each other. We have presented a method to improve page locality in the previous section, and now we present a similarity join algorithm that aims at minimizing the random seek cost further through the scheduling of page accesses. This algorithm is given in Figure 5.

/*
Let R be the root of the MVP-tree of the first dataset.
Let S be the root of the MVP-tree of the second dataset.
Let ε be the join threshold.
Let B be the total buffer pages available.
Let B_R be the reserved buffer pages.
Let B_D be the buffer pages dedicated to each dataset.
Let $M = (m_{i,j})$ be the adjacency matrix of the PCG (originally, all entries are 0's).
*/
Algorithm *Similarity_Join (R, S, ε, B_R)*

1. $M = PCG\ (R, S, \varepsilon)$
2. $B_D = \text{floor}(B - B_R) / 2$
3. While not finish processing all marked entries of M
   Read B_D marked pages from dataset 1
   Read B_D marked pages from dataset 2
   Process the pages and remove the corresponding marked entries from M
   While there are marked entries overlapping any of those B_D pages from dataset 1
      Read in B_R of those pages
      Process the pages and remove the corresponding marked entries from M
   While there are marked entries overlapping any of those B_D pages from dataset 2
      Read in B_R of those pages
      Process the pages and remove the corresponding marked entries from M

Figure 5. Algorithm for our proposed similarity join.

This algorithm is inspired by merge join and the Symmetric Clustering algorithm proposed in [15]. We reserve B_D buffer pages for each dataset for the processing of pages along the diagonal. The rest of the B_R pages are used for processing pages that are connected with the pages in the buffer.

5. Empirical Results

For the experimental evaluation of our similarity join algorithm, we tested its performance on the following datasets:

- ∞ Synthetic uniform vector dataset: This dataset consists of one million vectors randomly selected from a 5-D [0,1] super-cube. Euclidean distance is used for comparing the similarity between data objects.
- ∞ Image dataset: It consists of 10221 images. Each image is represented by three vectors corresponding to its properties in structure, color, and texture. A metric distance function is defined for each feature vector [4, 5]. In our experiments, we use a linear combination of the three metric distance functions as the distance function. The distance values are continuous in [0, 1].
- ∞ Mass spectra signatures: This is the SWISS-PROT database downloaded from[11] on 27 April 2004. We use a collaborator's protein digestion program to compute theoretical spectra for each protein. Each spectra is represented as a very high dimensional binary vector. A fuzzy measure of the shared peaks count (SPC) can be interpreted as a modified form of cosine distance [5, 6]. Two peaks are marked as being equal if the absolute difference of their m/z values lies within certain tolerance. Based on the cosine distance, the distance between two data points is the angle between their vector representations. Thus, the distance values are continuous in $[0, \pi/2]$.
- ∞ Protein sequence fragments: This protein sequence dataset was downloaded from GenBank in July 2003 [12]. The dataset contains FASTA formatted amino acid

translations extracted from GenBank/EMBL/DDBJ records that are annotated with one or more CDS features. We split the sequences into overlapping fixed length fragments or q-grams. The fragment length is 5. There is one q-gram for each sequence element. The distance between two fragments is their global alignment [13] score. The substitution matrix used is mPAM [17], which makes the distance function metric.

We compare our algorithm with the Cost-based Clustering (CC) algorithm proposed in [7] and the Symmetric Clustering (SC) algorithm presented in [15]. However, both papers assume a linear I/O model. We have implemented a modified version of these algorithms to fit in our affine I/O model. We assume a seek cost $s = 0.1$ sec. and a transfer cost $t = 0.02$ sec. in our experiments.

Given a dataset, we generate an adjacency matrix of the PCG based on self-join. The join radius used for each dataset is shown in Table 1. In general, different join radius will affect the clustering of the PCG. A larger join radius will result in more marked entries in the adjacency matrix, and so more neighbors of a page may not be in the memory buffer. In our experiments, we just choose a convenient join threshold to work with. In the future, one may explore the effect of this component on the I/O cost.

Dataset	No. of partitions	Join threshold
Uniform vectors	500	0.0001
Images	100	1.0E-5
Mass-spectra signatures	200	0.05
Protein sequence fragments	500	1000000

Table 1. The number of partitions and the join threshold used for each dataset.

After a PCG is generated, we use Metis [8], a graph partitioning software, to partition it. The number of partitions used for each dataset is shown in Table 1. We just choose a convenient value for each dataset in our experiments. The number of partitions will affect the clustering of marked entries along the diagonal of the adjacency matrices. Once we have partitioned the PCG, we reorder the partitions so as to maximize the number of cut-edges between partitions. Once we have determined this order, we permute the rows and columns of the adjacency matrices accordingly. These resulting adjacency matrices are used as input to the join algorithms.

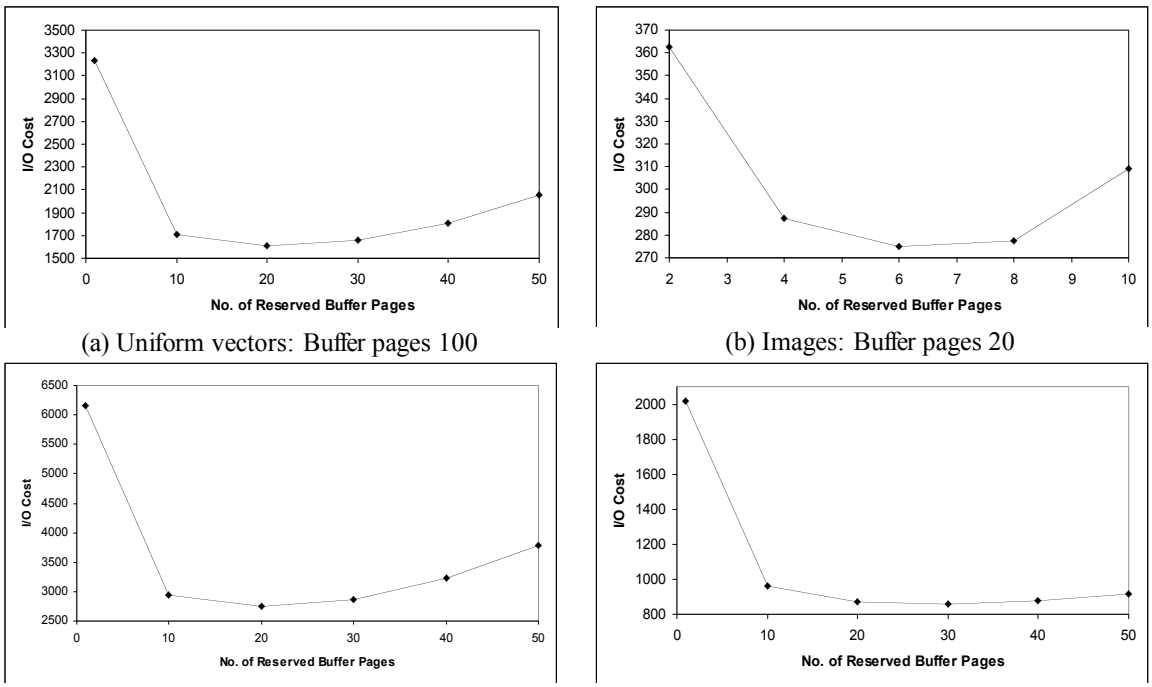
We will explore the effect of the number of reserved buffer pages of our join algorithm, and the effect of varying the number of buffer pages for the various join algorithms.

5.1 Effect of the Number of Reserved Buffer Pages

The number of reserved buffer pages (R) affects the number of consecutive pages that can be read into the available memory buffer in one read request. A larger number of reserved

buffer pages may be advantageous if there are many marked entries in the adjacency matrix that are far away from the diagonal. However, a larger number of reserved buffer pages also means a smaller number of available buffer pages for processing entries near the diagonal of adjacency matrices. Figure 5 shows the effect of different values of the number of reserved buffer pages on the I/O cost.

We see that there is an optimal value for *R* in each case. At first, increasing *R* just a little bit decreases the I/O cost steeply. After *R* has reached its minimum, increasing *R* leads to the increase of I/O cost first slowly and then rapidly. The optimum value for *R* depends on a number of factors, such as the clustering of marked entries along the diagonal, the clustering and the density of marked entries away from the diagonal, the seek cost and transfer cost etc.



(a) Uniform vectors: Buffer pages 100

(b) Images: Buffer pages 20

(c) Mass-spectra signatures: Buffer pages 100

(d) Protein sequence fragments: Buffer pages 200

Figure 6. Effect of the number of reserved buffer pages on I/O cost.

5.2 Effect of the Number of Buffer Pages

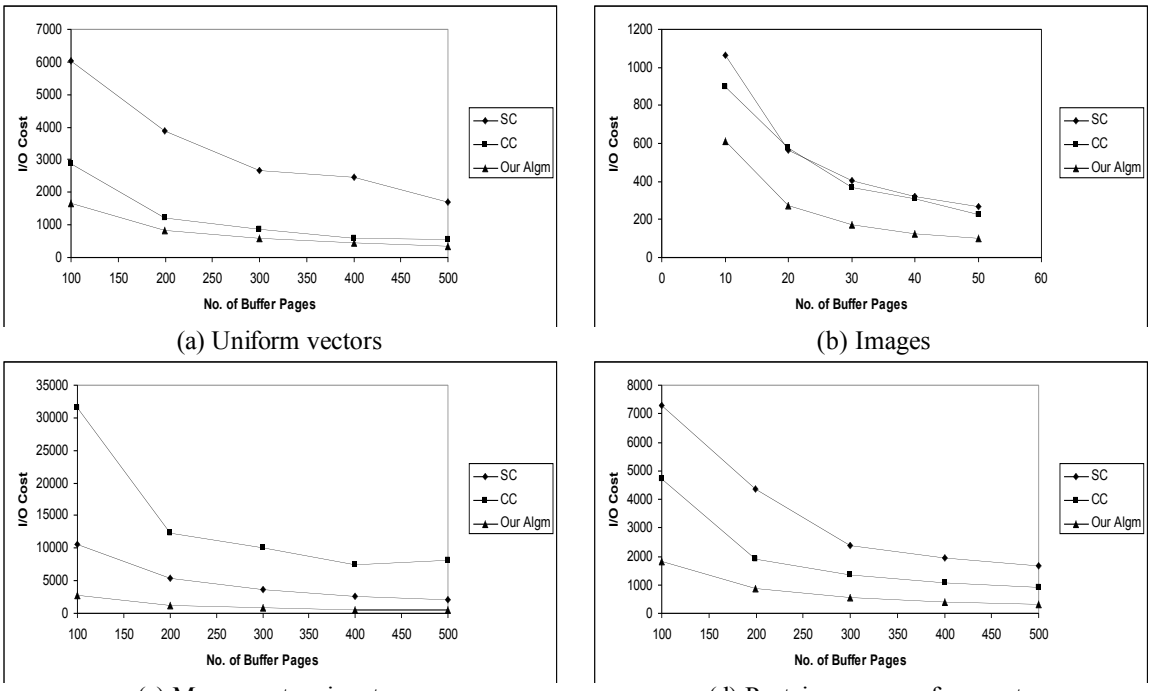
A larger number of buffer pages often decreases the number of page accesses. Figure 7 shows the effect of the number of buffer pages on the I/O cost. In this set of experiments, we use an optimal *R* for our algorithm. The values we used are shown in Table 2.

No. of Buffer Pages	Image
10	4
20	6
30	8
40	10

50		14	
No. of Buffer Pages	**Uniform vectors**	**Mass-spectra signatures**	**Protein sequence fragments**
100	30	20	20
200	30	30	30
300	40	30	30
400	40	40	40
500	50	40	50

Table 2. The value of R we used for each dataset and each number of buffer pages.

Figure 7 shows the effect of the number of buffer pages on I/O cost. Our algorithm outperforms our competitors. This difference is especially obvious when the number of buffer pages is small. SC generally gives the highest I/O cost. As the number of buffer pages increases, CC's performance gets closer to ours. However, CC has a very high processing cost which is not reflected in the I/O cost.



(a) Uniform vectors (b) Images

(c) Mass-spectra signatures (d) Protein sequence fragments

Figure 7. Effect of the number of buffer pages on I/O cost.

6. Conclusion

In this paper, we addressed the similarity join problem in metric-space. We have proposed an algorithm to improve page locality through the construction of a PCG, which is based on self-join, and band-diagonalization. We have also introduced an algorithm for the scheduling of the similarity join operations for finding all pairs of neighbors in a metric-space so as to minimize the I/O cost. We assumed an affine I/O model throughout the paper. Experimental results show that there is a significant reduction in the I/O cost for our algorithm when compared to the SC algorithm and the CC algorithm.

In our experiments, we have used a convenient value for the number of partitions when we partition the PCG. This parameter affects the clustering of the reordered adjacency matrix and thus the resulting page locality. One may wish to explore its effect. We also fixed a join threshold for each of our dataset. The join threshold affects the degree of connectivity of the PCG, that is, the number of marked entries in the PCG. This is also a component that one may want to explore. In our proposed similarity join algorithm, we just run experiments to determine a good value of R to use for each buffer size. R depends on the clustering of the marked entries in M that are far away from the diagonal. It also depends on the seek cost and transfer cost. This is also a parameter worth addressing.

References

[1] S.T. Barnard, A. Pothen, and H.D. Simon, "A Spectral Algorithm for Envelope Reduction of Sparse Matrices," Numerical Linear Algebra with Applications, vol.2, no.4, pp. 317-334. 1995.

[2] T. Bozkaya, and M. Ozsoyoglu, "Distance-Based Indexing for High-Dimensional Metric Spaces", Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data, May 1997, pp. 357-368.

[3] A. George and A. Pothen, "An Analysis of Spectral Envelope-Reduction via Quadratic Assignment Problems," SIAM J. Matrix Analysis and Its Applications, vol. 18, no. 3, pp. 706-732, 1997.

[4] D.S. Hochbaum, and D.B. Shmoys, "A best possible heuristic for the k-center problem", Mathematics of Operational Research, 1985, Vol. 10(2), pp. 180-184.

[5] Q. Iqbal, and J.K. Aggarwal, "Perceptual Grouping for Image Retrieval and Classification", 3rd IEEE Computer Society Workshop on Perceptual Organization in Computer vision, Vancouver Canada, July 8, 2001, pp. 19.1-19.4.

[6] Q. Iqbal, and J.K. Aggarwal, "Image Retrieval via Isotropic and Anisotropic Mappings", Pattern Recognition Journal, December 2002, Vol. 35, no. 12, pp. 2673-2686.

[7] T. Kahveci, C. Lang, A.K. Singh, "Joining Massive High-Dimensional Databases", ICDE 2003, Banglore, India.

[8] G. Karypis and V. Kumar. Metis Home Page. http://www-users.cs.umn.edu/karypis/metis/metis/main.html.

[9] G. Karypis and V. Kumar, "Parallel Multilevel Graph Partitioning", Proceedings of Supercomputing, November 1996.

[Merrett et al. 1981] T. H. Merrett, Y. Kambayashi, H. Yasuura, "Scheduling of Page-Fetches in Join Operations", VLDB 1981, pp. 488-498

[11] NCBI Mass Spectra data website: ftp://ftp.ncbi.nih.gov/blast/db/FASTA/swissprot.gz

[12] NCBI protein data website: ftp://ftp.ncbi.nih.gov/genbank/genpept.fsa.z

[13] S. B. Needleman and C.D. Wunsch, "A general method applicable to the search for similarities in the amino acid sequence of two proteins", Journal of Molecular Biology, 1970, Vol. 48, pp. 443-453.

[14] B. Seeger, P. Larson, R. McFadyen, "Reading a Set of Disk pages", VLDB 1993, pp. 592-603.

[15] S. Shekhar, C.T. Lu, S. Chawla, S. Ravada, "Efficient Join Index Based Join Processing: A Clustering Approach", IEEE Transactions on Knowledge and Data Engineering. Vol. 14, No. 6 (Nov/Dec 2002), pp. 1400-1421.

[16] T. Wang and D. Shasha, "Query Processing for Distance Metrics", Proceedings of the 16th VLDB Conference, 1990.

[17] W. Xu, and D.P. Miranker, "A metric model for amino acid substitution", in press Bioinformatics, 2003.