Join strategies on KD-tree indexed relations

Conference Paper · March 1989

DOI: 10.1109/ICDE.1989.47203 · Source: IEEE Xplore

CITATIONS
26

READS
71

3 authors, including:

Masaru Kitsuregawa
The University of Tokyo
358 PUBLICATIONS **3,056** CITATIONS
SEE PROFILE

Lilian Harada
Fujitsu Ltd.
34 PUBLICATIONS **187** CITATIONS
SEE PROFILE

Join Strategies on KD-Tree Indexed Relations

Masaru KITSUREGAWA, Lilian HARADA, Mikio TAKAGI

Institute of Industrial Science
University of Tokyo
22-1, Roppongi 7, Minato-ku, Tokyo
Japan

ABSTRACT

In this paper we present efficient join algorithms on very large relations indexed by KD-trees. There are previous works proposing the join on multi-attribute clustered relations based on hashing and also on grid-partitioning, whose shortcomings are non-order preservation and low load-factor, respectively. KD-tree indexed relations are characterized by preserving data order and maintaining high load-factors. However, KD-tree indexing has the disadvantadge of generating clusters which are overlapped in the join attribute domain, what causes a very high I/O cost for naive join algorithms. Here we analyze strategies to deal with this problem and introduce efficient algorithms to join two non-resident relations indexed by KD-trees.

First we introduce the concept of wave, which is a set of pages that is the object of join processing and that propagates over the relation space in the direction of the join attribute axis. Based on this new concept, we present five join algorithms and also four extended algorithms with a garbage collection mechanism to increase the effective space of the main memory. We extensively evaluate these join algorithms with analytical formulas and simulation results. It is shown that the join of very large relations indexed by KD-trees can be performed with one scan of the relations.

1. Introduction

Recently, the trend of research on very large relations is towards efficient multidimensional clustering of data on secondary memory in order to reduce processing cost, principally I/O cost.

The multidimensional clustering methods can be classified into two broad categories : the fixed methods which partition the data space at fixed places, and the adaptable methods which partition the data space according to the data distribution. Some examples of the fixed methods are the Grid File[10], DYOP[13], PLOP[9], Colored Binary Trie[16], Multidimensional Digital Hashing[12], Multidimensional Extendible Hashing[11,7,8] and some examples of the adaptable methods are the KD-Tree[1,2], Extended KD-Tree[3], KDB-Tree[15], GKD-Tree[5] and R-Tree[6]. However, almost all the analysis based on these multidimensional clustering methods are restricted to the selection and aggregation operations.

Concerning the join operation, some of the recent works are based on Predicate-Tree[18,4], Superjoin Algorithm[17] and DYOP[14]. The first two works use clustering based on multidimensional hashing and the last, on grid-partitioning, all of them being multi-attribute clustering mechanisms based on the fixed method. Fixed clustering methods are proper for join operations because all tuples with a discriminator attribute value share the same cluster and, the join of two relations is thus reduced to the join of their correspondent clusters. However, there are some shortcomings : the methods based on hashing do not preserve data order, which makes them inefficient for range-queries, and those based on grid-partitions have low load-factors, which increases the I/O cost.

The KD-tree indexing is one of the most known multi-dimensional clustering mechanisms based on the adaptable method, with the advantages of order-preservation and high load-factor but the disadvantage that tuples with a given discriminator attribute value are not in a single cluster. A naive join algorithm on very large relations indexed by KD-trees presents a high I/O cost and until now there is no work investigating join algorithms using KD-trees.

In this paper we propose join algorithms on KD-tree indexed relations. The new join algorithms are based on a new concept called wave. Wave is a set of pages that is the object of joining and that propagates over the relation space in the direction of the join attribute axis. Here we propose four basic join algorithms that determine the wave from one of the relations, and one algorithm that determines the wave from both relations. After describing these algorithms, we extensively analyze them with analytical formulas and simulation results. Then we introduce a garbage collection mechanism that discards the unnecessary data loaded in the main memory and extends the previous basic algorithms with an efficient memory management. We extensively evaluate these algorithms with analytical formulas and simulation results and show that the proposed algorithms perform the join of very large relations with one scan.

Section 2 provides some background to the proposed algorithms, describing some notation and assumptions used in our analysis. Section 3 describes how, using the KD-tree indexing mechanism, the join can be performed on very large relations. It presents five join algorithms for KD-tree indexed relations, their analytical and simulation results. Section 4 describes how the dynamic removal of unnecessary tuples from the main memory increases its effective space and improves the previous algorithms. If there is a suitable memory management, each relation is read into the main memory once. We present four extended versions of the join algorithms presented in section 3, their cost formulas and simulation results. Section 5 concludes the paper.

2. Notations and Assumptions

Let R be a relation having k attributes $A_1,...,A_k$ and composed of tuples $t = (a_1,...,a_k)$.

D is the base space of relation R and denotes the Cartesian product of the domains of attributes referred to by the relation, i.e.,

$$D = \prod_{i=1}^{k} [\, MIN_i , MAX_i \,]$$

For a given relation, the KD-tree method first divides D into two subspaces, using the A_1 value of the tuple so that it evenly divides the whole set of tuples. Here A_1 is called discriminator attribute. This step is then recursively applied to the subspaces with the discriminator attributes changed in the cyclic order as A_2, A_3, ...,

A_k, A_1, ... until each produced subspace can be contained in a disk page. The KD-tree is gradually constructed while dividing D. When a subspace is divided, a new node, which contains the discriminator attribute, its value at the recursive step, as well as a right and left pointers to the two new subspaces, is added to the KD-tree.

Denoting the |R| pages generated by dividing relation R as p_j ($1 \leq j \leq |R|$), the space P_j of pages p_j is represented by :

$$P_j = \prod_{i=1}^{k} [\, \alpha_{ij}, \beta_{ij} \,] \quad (\alpha_{ij} < \beta_{ij}, \alpha_{ij}, \beta_{ij} \in [\, MIN_i, MAX_i \,] \text{ for } \forall i).$$

We define "cluster of an attribute value a_i" the set of pages p_j ($1 \leq j \leq cs$) whose space contains the attribute value a_i, i.e.,

$$\text{cluster of } a_i = \{\, p_j \mid a_i \in [\, \alpha_{ij}, \beta_{ij} \,] \,\}.$$

When the KD-tree is perfectly balanced, the cluster size cs equals $|R|^{(1-1/k)}$ pages. An example of a cluster of a_1 for a 16-page relation and a 2-dimensional KD-tree is illustrated in Fig. 1.

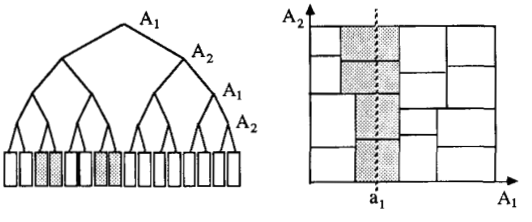


Fig. 1 KD-tree Indexed Relation and a Cluster

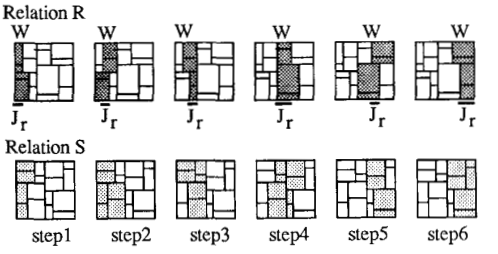
3. Join Algorithms on KD-tree Indexed Relations

In this section we present four algorithms to join two non-resident relations R and S of sizes |R| and |S| pages on attribute A_i. Our analysis is restricted to perfectly balanced KD-trees. Here we consider $|S| \geq |R| >> M$, where M is the main memory size.

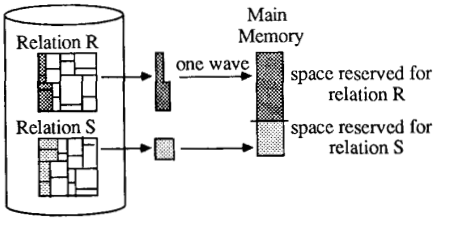
3.1 Description of the Basic Algorithms

A join of two KD-tree indexed relations R and S is illustrated in Fig. 2. In this example, the relations which are composed of 16 pages and indexed by a 2-dimensional KD-tree are joined on attribute A_1. As shown in the figure, the join of R and S is processed in six steps. In each step, four pages of relation R are loaded in the main memory and the range of the join attribute value to process is determined. Here we call this range as "Join Range Jr". The pages of relation S which contain tuples whose join attribute values a_1 are in the join range are loaded in the remaining space of the main memory in a nested-loop way. When the processing in this join range finishes, the two pages of relation R, all of whose tuples have already been processed, are unloaded and then two new pages are loaded. A new join range is determined and the join is processed in the same way as in step 1. As shown in Fig. 2, this procedure is repeated for six steps, when R and S are joined in the entire range of the join attribute A_1.

The set of pages of relation R loaded in the main memory is the unit of processing in each step and because of its shape, we denote it as "Wave W". The average speed of wave propagation over the relation space in the direction of the join attribute axis is given by 1/(number of join steps). In the example of Fig. 2 the wave size is 4 pages, the number of join steps is 6 and the wave propagation speed is 1/6. In this example, the waves of consecutive join steps have 2 pages in common, i.e., 2 pages of a wave in a join step overlap with the wave in the next join step.



(a) Processing Pages of Relations R and S in Each Join Step



(b) Join Processing in Step 1

Fig. 2 Processing Overview of Join Algorithms on KD-Tree Indexed Relations

In the join algorithms presented here, the wave size and the join range are two important parameters. Here we consider that, in a wave, all the tuples whose attribute value is in the join range are processed in each step. Also, in order to hold down the I/O cost of relation S, the wave, in each step, contains at least one cluster of an attribute value in the join range. This implies that the minimum wave size is cs pages and the maximum wave size is (M-1) pages. Also, as shown in Fig. 3, the join range can be determined in two ways : Jr_{in} and Jr_{out}. Considering a wave W propagating in the direction of the join attribute A_i, we call "Wave Rear Line Wr" the minimum join attribute values for the cs pages of the rear of the wave W, and "Wave Front Line Wf" the maximum join attribute values for the cs pages of the front of the wave W. Therefore the two possible join ranges Jr_{in} and Jr_{out} can be expressed as :

$$Jr_{in} = [\, \max(\alpha_{ij} \in Wr), \min(\beta_{ij} \in Wf) \,]$$

and

$$Jr_{out} = [\, \min(\alpha_{ij} \in Wr), \max(\beta_{ij} \in Wf) \,].$$

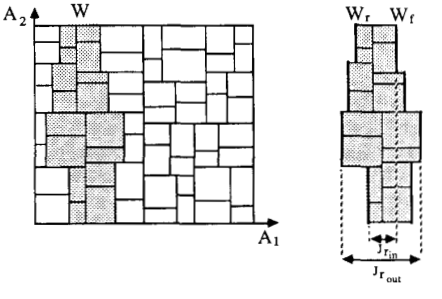


Fig. 3 Waves W and Join Ranges Jr

When the join range is Jr_{in}, waves of consecutive join steps overlap in some pages. On the other hand, when the join range is Jr_{out}, all the pages of waves of consecutive steps are different.

Using the specified wave sizes and join ranges as parameters, we can define four join algorithms as summarized in Table 1.

wave size \ join range	Jr_{out} $[\min(\alpha_{ij} \in W_r), \max(\beta_{ij} \in W_f)]$	Jr_{in} $[\max(\alpha_{ij} \in W_r), \min(\beta_{ij} \in W_f)]$
$\|R\|^{(1-\frac{1}{x})}$	algorithm 1	algorithm 3
M-1	algorithm 2	algorithm 4

Table 1 Wave Size and Join Range for the Basic Algorithms 1, 2, 3 and 4

Following, we detail how the wave W and the join range Jr are determined in each algorithm.

Algorithm 1 :

In this algorithm, the wave is determined as a cluster of relation R and the wave size is cs pages. After loading a wave in the main memory, the join range is determined as Jr_{out}. The pages of relations S whose join attribute value are in the join range are loaded in the remaining (M-cs) pages of the main memory. When processing of the join range Jr_{out} is finished, all the tuples of the wave have already been processed and are thus unloaded. Following, a new wave of cs pages is loaded and the procedure is repeated until the wave propagates over the entire space of relation R. The waves, the corresponding join ranges and their propagation over the relation space are exemplified in Fig. 4. Because the clusters generated by a KD-tree index overlap in the join attribute domain and, in algorithm 1 the join range is Jr_{out}, the waves of consecutive steps are not overlapped but the join ranges are.

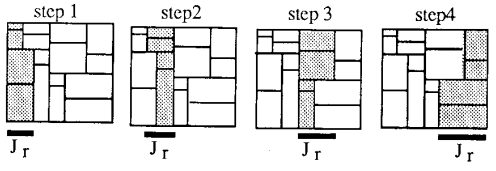


Fig. 4 Waves and Join Ranges for Algorithm 1

Algorithm 2 :

In this algorithm, in order to speed up the wave propagation in comparison with algorithm 1, the wave size is increased to (M-1) pages. Only one page is reserved for relation S. Similarly to algorithm 1, the join range is determined as Jr_{out}, and so waves in consecutive steps are not overlapped while the join ranges are. The waves and corresponding join ranges are exemplified in Fig. 5, for M taken as 7 pages.

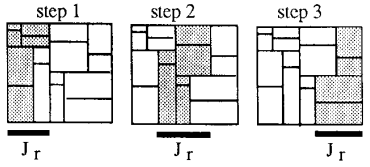


Fig. 5 Waves and Join Ranges for Algorithm 2

Algorithm 3 :

In the same way as in algorithm 1, the wave is determined as one cluster and (M-cs) pages are reserved for relation S. However, in this algorithm, the join range is Jr_{in}, so that waves of consecutive steps overlap and the corresponding join ranges do not. Fig. 6 illustrates the waves and the respective join ranges in algorithm 3.

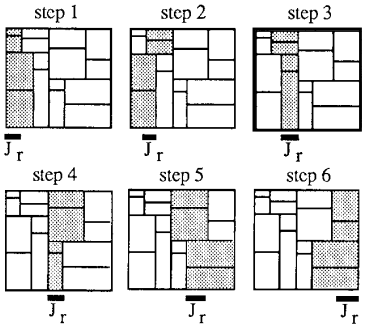


Fig. 6 Waves and Join Ranges for Algorithm 3

Algorithm 4 :

In this algorithm the wave size is (M-1) pages and only one page is reserved for relation S. The join range is determined as Jr_{in}, like in algorithm 3, so that waves in successive steps overlap and the join ranges do not. The wave propagation speed is higher than in algorithm 3. The wave and the join ranges for this algorithm are illustrated by an example in Fig.7.

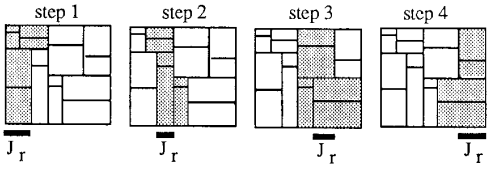


Fig. 7 Waves and Join Ranges for Algorithm 4

In the four algorithms above, the wave and corresponding join ranges are determined from relation R, which is read only once.

Comparing the wave propagation speed for the four algorithms, we have that :

(a) the wave propagation speed for algorithms 2 and 4 is higher than that for 1 and 3, respectively, because the first two have larger wave size (M-1 > cs);

(b) the wave propagation speed for algorithms 1 and 2 is higher than that for 3 and 4, respectively, because the first two have larger join range ($Jr_{out} > Jr_{in}$).

In order to clarify the characteristics of the four basic join algorithms presented above, we present their evaluation results as follows.

3.2 Evaluation for Uniform Data Distribution

3.2.1 Analytical Evaluation

In the previous four algorithms, relation R is read only once and thus, considering $|R| = |S| = N = 2^p$ pages, the number of page

accesses for relation R, π_R, is N pages. Following, we present the expressions which estimate the number of page accesses for relation S, π_S, which is given by :

π_S= (number of join steps) * (pages of relation S within the join range and not in the main memory).....[1]

For simplification we will use cs to denote the cluster size, which equals $N^{(1-1/k)}$ pages.

Algorithm 1 :

With this algorithm, the number of page accesses of relation S shown in [1] is given by :

π_S= (number of join steps) * {(pages within the join range) - (pages remaining in the main memory)}.....[2]

- the number of join steps is equal to the number of clusters, i.e., N/cs ;
- the number of pages of S within the join range is, on average, 3cs ; and
- the number of pages of S which can remain in the main memory is (M - cs).

Therefore :

$$\pi_S = N\left(4 - \frac{M}{cs}\right)$$

Algorithm 2 :

In this case, only one page is used for relation S, so :

π_S = (number of join ranges) * (pages within the join range).....[3]

- the number of join steps is

$$\frac{N}{M-1};$$

and
- the number of pages of S within the join range is, on average,

$$\left(\left\lceil\frac{M-1}{cs}\right\rceil + \frac{M-1}{cs}\right) cs$$

Therefore :

$$\pi_S = N\left(\left\lceil\frac{M-1}{cs}\right\rceil \frac{cs}{M-1} + 1\right)$$

Algorithm 3 :

In the same way as algorithm 1, π_S can be expressed by [2]. For this case :

- the number of join steps is

$$\sum_{i=1}^{\frac{p}{k}-1} 2^{ki} + 2;$$

- the number of pages of S within the join range is, on average, cs ; and
- the number of pages of S saved in the main memory which can be used again in the next join range is

$$M - 1 - cs - 2^{k-1}$$

Therefore :

$$\pi_S = \left(\sum_{i=1}^{\frac{p}{k}-1} 2^{ki} + 2\right)\left(2cs - M + 1 + 2^{k-1}\right).$$

Algorithm 4 :

In the same way as algorithm 2, π_S can be given by [3]. Here :

- the number of join steps is

$$\frac{N}{(M - 1 - cs) + 2^{k-1}};$$

and
- the number of pages of S within the join range is (M-1) pages.

Therefore :

$$\pi_S = \frac{N(M-1)}{M - cs - 1 + 2^{k-1}}$$

3.2.2 Simulation Results

Following, we present the relative performance of the join algorithms described above, for relations R and S with 64 K pages, 8 tuples/page, 2-dimensional KD-tree and a random data distribution as the one exemplified in Fig. 8.

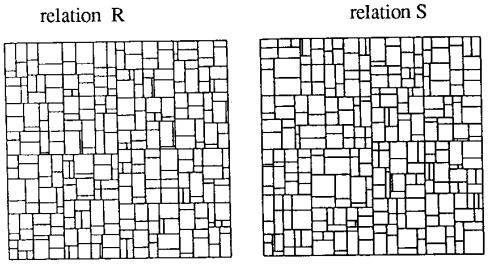


Fig. 8 An Example of Uniform Data Distribution for Relations R and S

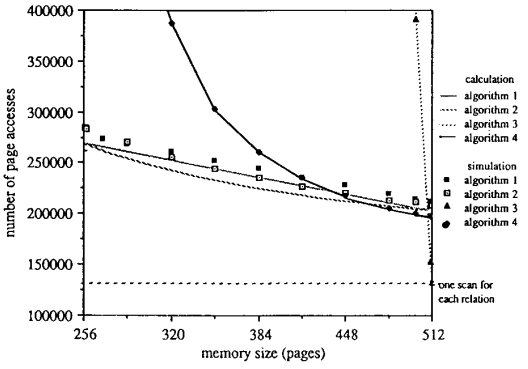


Fig.9 Analytical and Simulation Results of Algorithms 1~4 for Uniformly Distributed Relations

Fig. 9 shows the results of the simulation and analytical evaluation. The vertical axis is the total number of page accesses, π (= $\pi_R + \pi_S$). The horizontal axis is the main memory size M, which is varied from (cs+1) to 2cs pages. We can see that the cost formulas given in section 3.2.1 estimate the number of page accesses quite satisfactorily. Following we compare the algorithm performances for small and large wave sizes.

(a) Small wave size :

From Fig.9 we can observe that, in algorithms 1 and 3, the number of I/O decreases linearly with the memory size M. As shown in the cost formula [1], the number of page accesses for relation S, π_S, depends on the number of join steps and pages of S to read in each step. Given a relation, for algorithms 1 and 3, the number of join steps is constant, and the pages of S to read in each step decreases linearly when M enlarges. The inclination of the curve of algorithm 3 is much greater than that of algorithm 1. As shown by the analytical formulas, this inclination is given by the number of join steps. The number of join steps is much greater in algorithm 3 because the join range is given by Jr_{in}. For small M the I/O cost of algorithm 3 is much higher than that of algorithm 1. On the other hand, when M is enlarged and taken as 2cs pages, in algorithm 3 the I/O cost is reduced to one scan of each relation, while in algorithm 1 this I/O cost is greater.

(b) Large wave size :

Fig. 9 shows that the I/O cost is inversely proportional to M, for algorithms 2 and 4. As given in [1], π_S is proportional to the number of join steps and the pages of S to read. The number of join steps is inversely proportional to the wave size which is (M-1) pages for algorithms 2 and 4. Concerning the inclination of the curves and the I/O cost for small M, the inclination for algorithm 2 is smaller and the performance for small M is better than that for algorithm 4, analogously to algorithm 1 and 3, respectively. However, the difference of algorithms 2 and 4 in comparison with algorithms 1 and 3 is that the I/O cost does not reduce to one scan of the relations, in the range of variation of M shown in Fig. 9.

Observing the four curves we find that the higher the wave propagation speed over the space of relation R, the better the algorithm performs for most range of M. That is, when the main memory space is not enough to maintain the necessary pages of relation S in each step, the performance is better when the number of join steps is smaller. As shown by the formulas, the number of join steps decreases in the sequence of algorithms 3, 4, 1, 2, i.e.,

$$\sum_{i=1}^{\frac{p}{k}-1} 2^{ki} + 2 > \frac{N}{(M - 1 - cs) + 2^{k-1}} > \frac{N}{cs} > \frac{N}{M - 1}$$

for k=2, cs<M<2cs, and so the speed of propagation of the waves increases and the algorithms performance improves in this sequence.

3.3 Algorithm for Non-Uniform Data Distribution

From the analysis of the four basic join algorithms presented above we conclude that :

(1) for small memory sizes, the performance is better when the wave propagation speed is higher.

(2) the only algorithm to reduce the number of page accesses to one scan of the relations, is the one with small wave size and short join range (Jr_{in}) - algorithm 3.

These conclusions and analysis are for relations whose data are uniformly distributed over the relation space and so, for the case in which the wave propagation speed is constant. However, the join range depends on the data distribution and for the case of non-uniformly distributed data, the wave does not propagate with a constant speed.

Following, we introduce a new algorithm for relations with non-uniform data distribution. For this algorithm we use a small wave size and short join range and, using the information of the data distribution for relations R and S, we choose the waves consulting the KD-trees of both relations, aiming at reducing the number of join steps, i.e., maximizing the average wave propagation speed in direction of the join attribute axis.

Algorithm 5 :

In algorithm 5 the wave is taken as a cluster from one relation and (M-cs) pages are reserved for the other relation. The join range is taken as Jr_{in}. In each join step, the wave is taken from one of the relations in order to maximize the wave propagation speed. The waves, their corresponding join ranges and their propagation are illustrated in Fig.10. For the exemplified relations, six join steps are determined with algorithm 3, while algorithm 5 reduces this number to three.

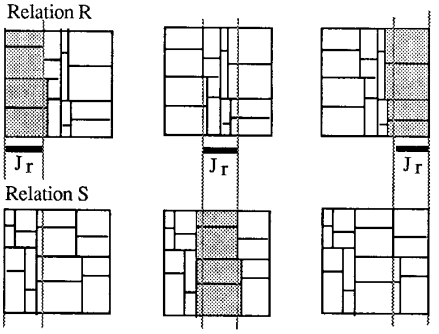


Fig. 10 Waves and Join Ranges for Algorithm 5

3.4 Simulation Results for Non-Uniform Data Distribution

In order to analyze the performance of relations whose data are biased in some regions, we used relations with normal and inverse normal distributions for the simulation. An example of such relations R and S is shown in Fig.11. The sizes of the relation and page used for the simulation are the same as in the uniform data distribution case.

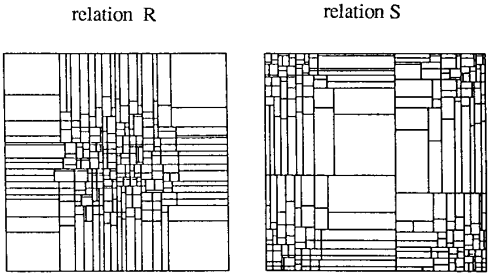


Fig. 11 An Example of Non-Uniform Data Distribution for Relations R and S

The simulation results are shown in Fig.12. As in the case of uniform data distribution, on the first four algorithms the number of page accesses increases in the sequence of algorithms 2, 1, 4, 3, which is the sequence of decreasing the wave propagation speed. Concerning algorithm 5, whose speed of propagation of wave is higher than that of algorithm 3, the performance is improved as expected. Although it is not satisfactory compared to the other algorithms, as shown below in section 4, the extended version of algorithm 5 introduce improvements so that it will show the best performance.

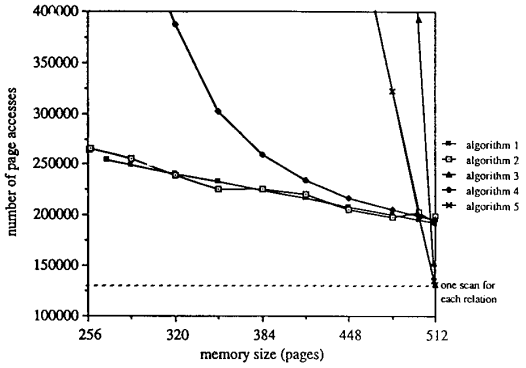


Fig. 12 Simulation Results of Algorithms 1~5 for Non-Uniformly Distributed Relations

4. Extended Join Algorithms

4.1 Description of the Extended Algorithms

From the analysis of the previous five join algorithms, we conclude that, due to insufficient memory space to hold all the necessary pages in each step, the fewer the number of steps, the better the performance. Therefore, algorithms 2 and 4, whose wave propagation speed is higher present better performance. On the other hand, we also verify that the join can be performed with one scan of each relation if the wave propagation speed is low enough. In this case, the join range is too short and the corresponding pages of relation S can be hold in the memory to be used again in the next step. Algorithms 3 and 5 are the only ones to achieve the ideal number of page accesses. We will now modify the join algorithms described above with efforts to :

(a) utilize the memory space efficiently, in order to speed up the propagation of the wave of algorithms 2 and 4 ;

(b) after the wave taken from one relation is loaded in the memory, utilize the remaining memory space efficiently, in order to hold more data of the another relation, for algorithms 1, 3 and 5.

In order to load and unload data in page units, the algorithms presented until now maintain tuples which have already been processed and are not necessary any more in the main memory. We can introduce a garbage collection mechanism which dynamically discards the tuples that are no longer necessary after the processing of each step. This increases the effective space of the main memory for each join step, allowing the loading of more pages in this free space. Following we will analyze how the tuples of relations R and S can be discarded in the previous five algorithms.

First, consider the removal of tuples from relation R. In algorithm 1 and 2, successive waves overlap when propagating over the space of relation R. Thus, a wave of R is loaded in the main memory, used in the processing of one join range and then unloaded, which is then followed by the loading of the next wave. Therefore, for both algorithms the pages of R are discarded as soon as they have been processed in the join range, so that there is no garbage of relation R remaining in the memory. On the other hand, successive waves in algorithm 3 and 4 overlap when propagating over the space of relation R. After processing a join step, it is not necessary to maintain the whole page which overlaps with the successive wave, but only that portion that has not been processed yet.

Now, consider the removal of tuples from relation S. In algorithms 1 and 3, which reserve (M-cs) pages to relation S, there are tuples which have already been processed and are still maintained in the memory. On the other hand, algorithms 2 and 4 reserve only one page for relation S, so that the pages are loaded and then unloaded without garbage maintained in the memory. For algorithm 5, which determines the wave from both relations R and S, it is possible to discard tuples of both relations.

Algorithm	remove tuples of relation R	remove tuples of relation S
1m	X	O
(2m)	X	X
3m	O	O
4m	O	X
5m	O	O

Table 2 Garbage Collection on Extended Algorithms

Excluding algorithm 2, which does not maintain unnecessary tuples in the main memory, we introduce a garbage collection mechanism in the algorithms 1, 3, 4, 5 described above. Here these new algorithms are called algorithms 1m, 3m, 4m and 5m, as shown in Table 2.

The removal of tuples of relation R in algorithm 1m, and of both relations R and S in algorithms 3m and 5m increases the effective space used by relation S. The removal of unnecessary tuples of relation R in algorithm 4m enlarges the effective space used by relation R which means the enlargement of the wave size and the speeding up of the wave propagation. Fig. 13 (a), (b), (c), (d) exemplifies each of these algorithms. After processing the join range in the first step the effective memory space for the next join step is enlarged : the dotted portion is discarded by the garbage collection mechanism and only the dashed portion is maintained in the main memory. In order to clarify this effect, we will present the evaluation of the extended algorithms.

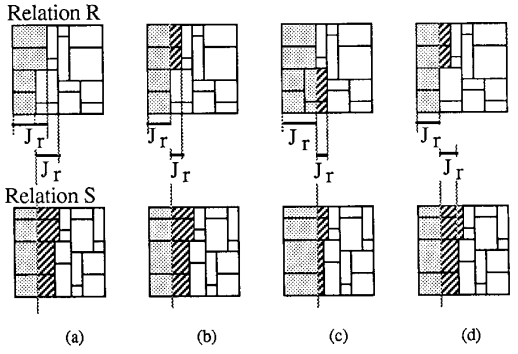


Fig. 13 Garbage Collection on Algorithms (a) 1m (b) 3m (c)4m (d) 5m

4.2 Evaluation for Uniform Data Distribution

4.2.1 Analytical Evaluation

Following, we present the cost formulas for the algorithms presented above. First, we explain the analytical model used for our evaluation. We assumed a simplified model in which :

(1) the KD-tree is perfectly balanced;

(2) the tuples are uniformly distributed within a page; and

(3) for a cluster, denoting μ the mean width of the cluster and σ, the variance, we consider that:

$$\sigma = \frac{(\frac{cs}{2^{k-1}} - 1)\,\delta}{2}, \quad \delta \text{ being a constant.}$$

This means that we consider each of the cs pages of a cluster, except those of the first and last clusters, as having width μ and being dislocated by δ in relation to the previous 2^{k-1} pages in the cluster. This is illustrated by an example for $k = 2$ in Fig.14 .

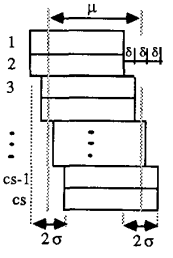


Fig. 14 A Cluster and its mean μ and variation σ

Assuming the model stated above, we can estimate the cost formulas for the extended join algorithms as presented below. For all of them, relation R is read only once, that is, π_R = N pages. To simplify the notation we will use $K = 2^{k-1}$.

Algorithm 1m :

In the same way as algorithm 1, the number of page accesses for relation S is given by expression [2] in section 3.2.1, which we rewrite here :

π_S= (number of join steps) * {(pages within the join range) - (pages remaining in the main memory)}

The part of a cluster of relation S which is processed in a join range and may be maintained in the main memory to be also processed in the next join range, ρ, is represented by the dashed area in Fig. 15 and is calculated as :

$$\rho = \frac{K\delta}{\mu} \sum_{i=1}^{\frac{cs}{K}-1} i \text{ pages} = \frac{\sigma}{\mu} cs \text{ pages}$$

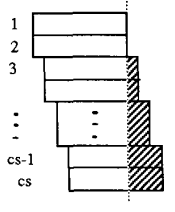


Fig. 15 ρ for Algorithm 1m

Therefore, π_S may be considered for two cases :

(1) ρ fits in the main memory, i.e.,

$$M - cs \geq \frac{\sigma}{\mu} cs$$

In this case, at least the tuples of the dashed area are maintained in the main memory. For this case :

- the number of join steps is equal to the number of clusters, i.e., N/cs ;
- the number of pages of S within the join range is, on average, 2cs ; and
- the number of pages of S which can remain in the main memory is (M - cs - ρ).

Therefore :

$$\pi_S = N\,(3 - \frac{\sigma}{\mu} - \frac{M}{cs})$$

(2) ρ does not completely fit in the main memory, i.e.,

$$M - cs < \frac{\sigma}{\mu} cs$$

For this case :

- the number of join steps is equal to the number of clusters, i.e., N/cs ;
- the number of pages of S within the join range is, on average, 3cs ; and
- the number of pages of S which can remain in the main memory, γ, is calculated by

$$M - cs = \frac{K\delta}{\mu} \sum_{i=1}^{\frac{\gamma}{K}-1} i$$

resulting in :

$$\gamma = \frac{K + \sqrt{K^2 + \frac{4\mu}{\sigma}(M-cs)(cs-K)}}{2}$$

Therefore :

$$\pi_S = N\,(3 - \frac{K + \sqrt{K^2 + \frac{4\mu}{\sigma}(M-cs)(cs-K)}}{2cs})$$

Algorithm 3m :

The join ranges of this algorithm are the same as those of algorithm 3. In this algorithm, unnecessary tuples of both relation R and S are discarded so its cost formula is very complex. Because its expression is too long we omit it here and just show the results in the next subsection.

Algorithm 4m :

In the same way as algorithm 4, here the number of page accesses of relation S is given by expression [3] in section 3.2.1, which we rewrite here :

π_S = (number of join steps) * (pages within the join range)

- the number of join steps is equal to :

$$\frac{N}{\text{new pages in the wave}} = \frac{N}{\alpha}$$

where :

α = (M-1) - (tuples of R remaining in the main memory for the

processing of next join step)

$= (M-1) - \rho$

and where ρ , which is exemplified in Fig.16 as the dashed area, can be expressed as :

$$\rho = \frac{K\delta}{\mu}\left(\sum_{i=1}^{\frac{cs}{K}-\frac{x}{K}} i - \sum_{i=1}^{\frac{x}{K}-1} i\right) + x - K$$

where x is :

$$x = \left(\frac{M-1}{cs} - 1\right) cs + K$$

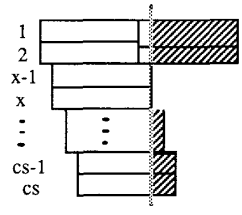


Fig. 16 ρ for Algorithm 3m

Hence, we have that :

$$\rho = \frac{(3\sigma+\mu)cs^2 + (K(\mu-\sigma) + (\mu-2\sigma)(\mu-1))cs - K\mu(\mu-1)}{\mu(cs-K)}$$

and

$$\alpha = \frac{(\mu-3\sigma)cs^2 + (2\sigma(\mu-1) - K(\mu-\sigma))cs}{\mu(cs-K)}$$

- the number of pages of S within the join range is :

(number of crammed pages of R) + (new pages in the wave)

$= \chi + \alpha$

where :

$$\chi = (M-1) - \left(\frac{M-1}{cs} - 1\right) cs + K = cs - K$$

Therefore :

$$\pi_S = N\left(\frac{(cs-K)^2\mu}{(\mu-3\sigma)cs^2 + (2\sigma(M-1) - K(\mu-\sigma))cs} + 1\right)$$

4.2.2 Simulation Results

Fig.17 shows the estimation and the respective simulation results of the extended join algorithms presented above, using the same relations R and S described in section 3.2.2. Here again, the prediction was efficient.

Fig.18 shows the simulation results for all the presented algorithms. As expected, all the extended algorithms reduce the number of page accesses in comparison with the basic join algorithms. Algorithm 3m and 5m are the best extended algorithms and they reduce the I/O cost to one scan in most range of variation of the memory size M. Among the basic join algorithms, algorithm 3 and 5 are the only to reduce the I/O cost to one scan, but for the most range of variation of M, they show the highest I/O cost because they have the lowest wave propagation speed.

The introduction of the garbage collection mechanism increases the effective memory space. For the basic join algorithms, when the

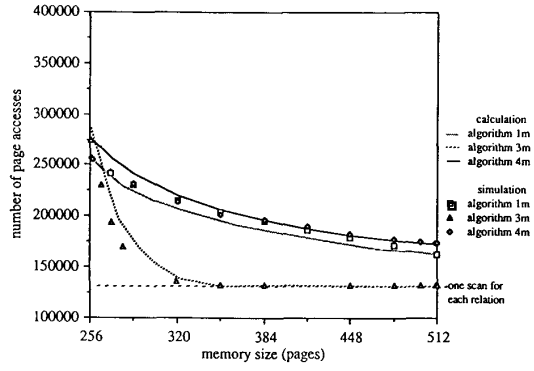


Fig.17 Analytical and Simulation Results of Algorithms 1m~4m for Uniformly Distributed Relations

memory space was not enough to save the necessary pages in each join step, the performance degraded as the number of join steps increased. Algorithms 1m, 3m and 5m increases the effective memory space to save more tuples of relation S while algorithm 4m enlarges the wave. For the extended join algorithms, the saving of tuples of S are more efficient than the decreasing of the number of join steps. The wave propagation speed increases from 5m, 3m, 1m to 4m and the I/O cost increases in this sequence.

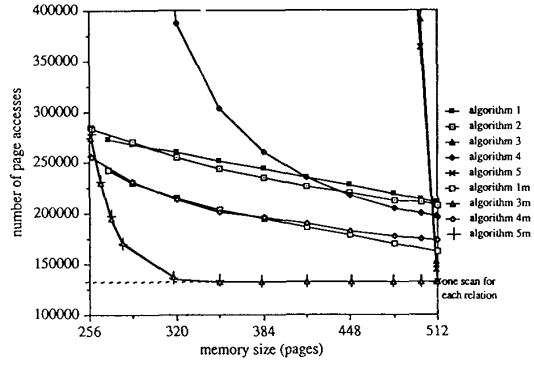


Fig.18 Simulation Results of Algorithms 1~5, 1m~5m for Uniformly Distributed Relations

4.3 Simulation Results for Non-Uniform Data Distribution

In this simulation we used the same relations described in section 3.4.1 and the results are shown in Fig.19. In the same way as for the uniform data distribution, the I/O cost increases for algorithms which reserve more memory space to be used by the wave. For the case of normal and inverse normal data distribution, the fact of choosing the appropriate wave from the KD-tree information of both relations is well utilized : algorithm 5m resulted the best in the entire range of main memory size, minimizing the I/O cost to one scan of each relation.

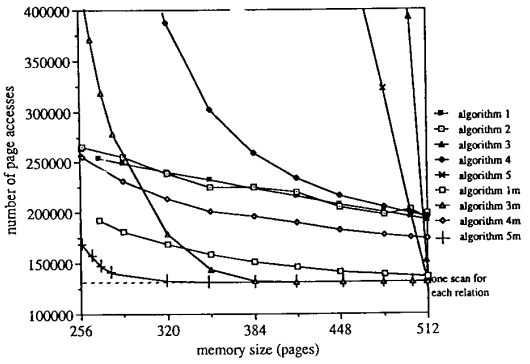


Fig. 19 Simulation Results of Algorithms 1~5, 1m~5m for Non-Uniformly Distributed Relations

5. Conclusion

In this paper we investigated join algorithms for KD-tree indexed relations, presenting their analytical and simulation results. KD-tree is a multidimensional clustering mechanism based on the adaptable method and so different clusters have tuples with a same attribute value. The join of two relations R and S can not be simply reduced to the join of one of their respective cluster and a naive join algorithm for KD-tree indexed relations requires a high I/O cost. An efficient join algorithm for KD-tree indexed relations is complex and has not been investigated until now. In this paper we propose efficient join strategies to reduce its I/O cost and extensively evaluate these algorithms with analytical analysis and simulation. With one of our proposed join algorithms which uses a suitable memory management, the join of two non-resident relations can be performed with one scan.

According the the KD-tree information, the concepts of wave and join range are introduced and, based on them, the join algorithms are proposed. Four basic algorithms determine the wave and join ranges according to the KD-tree information of relation R and another algorithm determines them with the information of both relations R and S. Analytical formulas and simulation results clarify their characteristics. For these basic join algorithms, the higher the wave propagation speed, the better the algorithm performance. This is because in each step, the memory space to maintain the necessary tuples for the next step is small and insufficient, and so the fewer the number of steps, the fewer the number of scans of relation S. Therefore, we propose extended versions of the basic algorithms, introducing a garbage collection mechanism to enlarge the effective memory size. For these algorithms, the tuples that have already been processed and are no longer necessary are discarded from the main memory at the end of each join step. These extended algorithms are analyzed by analytical formulas and simulation results. In opposition to the former basic algorithms, in these extended algorithms, as the wave propagation speed is increased, the space reserved for relation S decreases and the performance worsens. For the best algorithm, the unnecessary tuples of both relations are dynamically discarded to enlarge the effective memory space and the wave is determined so that its speed of propagation is high. With this algorithm, the join can be performed with almost one scan of each relation.

The presented figures were only for KD-trees of two dimensions but although not exposed here, analysis of higher dimensions were performed, also showing that the analytical estimation matches the simulation results.

The implementation details of these join strategies on KD-tree indexed relations are to be reported in a future paper.

References

[1] J.L.Bentley : "Multidimensional Binary Search Trees Used for Associative Searching", Commun. ACM, 18, 9, pp.509-517 (September 1975).

[2] J.L.Bentley : "Multidimensional Binary Search Trees in Database Applications", IEEE Trans. Software Eng., 5, 4, pp.333-340 (1979).

[3] J.M.Chang and K.S.Fu : "A Dynamical Clustering Technique for Physical Database Design", Proc.of the 1980 SIGMOD Conf., pp.188-199 (1980).

[4] J.P.Cheiney, P.Faudemay, R.Michel and J.M. Thevenin : "A Reliable Parallel Backend Using Multiattribute Clustering and Select-Join Operator", Proc. of 12th. Int. VLDB Conf., pp.220-227 (August 1986).

[5] S.Fushimi, M.Kitsuregawa, M.Nakayama, H.Tanaka and T.Moto-oka : "Algorithm and Performance Evaluation of Adaptive Multidimensional Clustering Technique", Proc. of the 1985 SIGMOD Conf., pp.308-318 (1985).

[6] A.Guttman : "R-Trees : A Dynamic Index Structure for Spatial Searching", Proc.of the 1984 SIGMOD Conf., pp.47-57 (June 1984).

[7] K.L.Kelley and M.Rusinkiewicz : "Implementation of Multi-Key Extendible Hashing as an Access Method for a Relational DBMS", Proc.of 2nd. Data Engineering Conf., pp.124-131 (February 1986).

[8] H.P.Kriegel and B.Seeger : "Multidimensional Dynamic Quantile Hashing is Very Efficient for Non-Uniform Record Distributions", Proc.of 3rd. Data Engineering Conf., pp.10-17 (February 1987).

[9] H.P Kriegel and B.Seeger : "PLOP-Hashing : A Grid File without Directory", Proc.of 4th. Data Engineering Conf., pp.369-376 (February 1988).

[10] J.Nievergelt, H.Hinterberger and K.C.Seveik : "The Grid-File : An Adaptable, Symmetric Multikey File Structure", ACM Trans. Database Syst., 9, 1, pp.38-71 (March 1984).

[11] E.J.Otoo : "A Mapping Function for the Directory of a Multidimensional Extendible Hashing", Proc. of 10th. Int. VLDB Conf., pp.493-506 (August 1984).

[12] E.J.Otoo : "A Multidimensional Digital Hashing Scheme for Files With Composite Keys", Proc.of the 1985 SIGMOD Conf., pp.214-229 (June 1985).

[13] E.A.Ozkarahan and M. Ouksel : "Dynamic Order Preserving Data Partitioning for Database Machines", Proc. of 11th. Int. VLDB Conf., pp.358-368 (August 1985).

[14] E.A.Ozkarahan and H. Bozsahin : "Join Strategies Using Data Space Partitioning", New Generation Computing, 6, 1, pp.19-39 (1988).

[15] J.T.Robinson : "The KDB-Tree : A Search Structure for Large Multidimensional Dynamic Indexes" Proc.of the 1981 SIGMOD Conf., pp.10-18 (1981).

[16] Y.Tanaka : "Massive Parallel Architecture for Very Large Databases", Doctoral Thesis, Univ. of Tokyo (May 1985).

[17] J.A.Thom, K.Ramamohanarao and L. Naish : "A Superjoin Algorithm for Deductive Databases", Proc.of 12th. Int. VLDB Conf., pp.189-196 (August 1986).

[18] P.Valduriez and Y. Viemont : "A Multikey Hashing Scheme using Predicate Trees", Proc.of the 1984 SIGMOD Conf., pp.107-114 (June 1984).