Query Processing Method for Multi-Attribute Clustered Relations

Lilian HARADA, Miyuki NAKANO, Masaru KITSUREGAWA, Mikio TAKAGI

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

Abstract

In this paper we introduce a new query processing method for multi-attribute clustered relations. Many proposals on multi-attribute clustered relations have been done so far. However, an efficient query processing method for these relations has not been proposed and analyzed yet. The multi-attribute clustered relations treat all attributes symmetrically, at the cost of losing the sequential data order between some specific pages. Thus, if the naive query processing methods used for single-attribute clustered relations, which rely on the sequential order of the clustered attribute, are straightly used for the multi-attribute clustered relations, it results in a high I/O cost.

Here, aiming at reducing this high I/O cost caused by the problem of no total order presented by the multi-attribute data, we introduce a query processing method which emphasizes the page loading strategy. In this query processing method we introduce a new concept called *wave*. Wave is a set of pages which represents the unit of loading from the secondary storage to the main memory. Our query processing method uses the information of the multi-attribute clustering index to group pages, whose data are not ordered, into waves, which are ordered and must fit in the memory size as much as possible. Thus the execution of the relational operation for the tuples in the waves results in the execution of the whole multi-attribute clustered relation with the minimum I/O cost. We evaluate the proposed model using the KD-tree and the Grid-file, which are representative multi-attribute clustering methods. Simulation results show that this query processing method is efficient and the queries for multi-attribute clustered relations can be executed with almost one relation scan, which is the lowest I/O bound.

Proceedings of the 16th VLDB Conference
Brisbane, Australia 1990

1. Introduction

In this paper we introduce a new model of relational query processing for multi-attribute clustered relations. The multi-attribute clustering method provides a file structure that treats all attributes symmetrically, that is, avoids the distinction between primary and secondary keys. The base space of the relation is considered as the Cartesian product of the attribute domains and is partitioned along the axis of multiple attributes by the multi-attribute clustering methods. The multi-attribute clustering methods can be classified into two broad categories : the adaptable method which partitions the base space according to the data distribution [1,2,3,6,8,11,18,21], and the fixed method which partitions the base space at fixed places [5,12,16,17,19,20]. Now that there are so many proposals of data structures for multi-attribute clustering, it is time to study the relational query processing for these data structures.

Concerning the query processing for multi-attribute clustered relations, the above-mentioned papers intensively discuss the cost of the retrieval (selection) operation, without examining the other relational operations such as join, aggregate by group and projection with duplicate elimination. The difficulty presented by multi-attribute clustered relations for query processing is that natural total orders of multi-attribute data do not exist. When the relational queries are applied to the key attribute for which the relations are ordered, the query execution, which requires comparison of tuples with the same key attribute value, can be done with only one scan of the ordered relations. For example, when relations have B-trees which provide clustering on a single key attribute, their join on this key attribute can be done with one scan of each relation using the sort-merge algorithm. On the other hand, multi-attribute clustered relations are partitioned in pages according to the tuple's values for multiple attributes, which results in pages without sequential data order. Thus, the simple processing methods used for single-attribute clustered relations, such

as the ones using B-trees that has pages in sequential order of the clustering attribute, can not be applied to the multi-attribute clustered relations with a reduced I/O cost.

A query processing model which efficiently utilizes the information of the multi-attribute clustering index has not been proposed and evaluated yet. In this paper we aim at reducing the high I/O cost of relational processing caused by the problem of no total order presented by multi-attribute data. We introduce a query processing model for the multi-attribute clustered relations, which emphasizes the page loading strategy. The new concept of *wave* is the key point of the query processing model. Because the whole relation does not fit in the memory at once, the base space of the relation is divided into subspaces which are used as processing units. Wave is a set of pages which actually holds the tuples within these subspaces, and represents the unit of loading from the secondary storage to the main memory. In our query processing model pages, whose data are not ordered, are grouped into waves, which are ordered according to the processing attribute, that is, the attribute referred to by the relational operation, and must fit in the memory size as much as possible. Thus, the execution of the relational operation for the tuples in the waves results in the execution of the whole multi-attribute clustered relation in a sequential order for the processing attribute, with the minimum I/O cost. The waves are determined from the information of the multi-attribute clustering index, which means that this query processing model is flexible in terms of the multi-attribute clustering data structure and does not request any specific data structure.

In this paper, using the concept of wave, we describe in detail this query processing model which aims at reducing the I/O cost. Then, in order to validate the efficiency of the query processing model we concentrate on the join operation, which is the relational operation of highest I/O cost. Based on the proposed model, we present join algorithms which determine the waves from information contained in the multi-attribute clustering index by using the KD-tree (adaptable method) and Grid-file (fixed method). We verify the efficiency of these algorithms by simulations which show that with the best proposed algorithms, the wave tuples to be processed are always on memory and thus the query processing for non-resident relations can be performed with only one scan of the relations, which is the lowest I/O bound. We conclude that the presented query processing model provides an efficient framework for multi-attribute clustered relations.

Section 2 briefly describes the multi-attribute clustering method. Section 3 introduces the concepts of wave, processing range and cluster which are the base of the query processing model. Based on this query processing model, sections 4 and 5 describe and evaluate the join algorithms for the adaptable and fixed multi-attribute clustered relations, using the KD-tree and Grid-file, respectively. Finally, section 6 presents our conclusions.

2. Multi-Attribute Clustering Method

In this section we first introduce some notation and then we present the multi-attribute clustering of the adaptable and fixed methods.

Let R be a relation having k attributes $A_1, A_2, ..., A_k$ composed of tuples $t = (a_1, a_2, ..., a_k)$. 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.,

$$D = \prod_{i=1}^{k} dom(A_i) = \prod_{i=1}^{k} [MIN_i, MAX_i).$$

R is a set of physical tuples and is a subset of D.

The relation is partitioned in pages, which are its unit of access and consists of disjoint sets of actual tuples. Let relation R be partitioned in |R| pages P_j, j = 1, ..., |R|. The space p_j of each page P_j is given by

$$p_j = \prod_{i=1}^{k} [\alpha_{ij}, \beta_{ij})$$

$(\alpha_{ij} < \beta_{ij}, \alpha_{ij}, \beta_{ij} \varepsilon [MIN_i, MAX_i)$ for *all* i).

While relation R is the actual set of data, composed of tuples grouped in pages, the corresponding base space D of relation R contains the page spaces. Thus, we have that $R = \sum_{j=1}^{|R|} P_j$ and $D \geq \sum_{j=1}^{|R|} p_j$.

The determination of the pages spaces, i.e., the values of α_{ij} and β_{ij}, differs for the adaptable and fixed multi-attribute clustering methods. As stated in the introduction, the adaptable method partitions the base space according to the data distribution, and the fixed method does it at fixed places.

Fig.1(a) shows an example of relation partitioning for the adaptable method. In this example the base space of the relation is two-dimensional and each page can contain a maximum of 3 tuples. Initially there is only one page a containing tuples #1,#2,#3. The insertion of tuple #4 causes page a to overflow and, thus the base space to be split with a boundary line at value a_1 of the tuple that evenly divides the whole set of tuples. In this example the value is determined as μ_1, that is the value a_1 of tuple #2. Then tuple #5 is inserted without overflow. The insertion of tuple #6 causes the split of page b with a boundary line at v_2, which is the value a_2 of tuple #3, generating a new page b'. Thus the respective page spaces for a, b and b' are given by :

$A = [MIN_1, \mu_1) \times [MIN_2, MAX_2)$,

$B = [\mu_1, MAX_1) \times [MIN_2, \nu_2)$ and
$B' = [\mu_1, MAX_1) \times [\nu_2, MAX_2)$.

Tuples #7,#8 are inserted with no page overflow. Thus, the insertion of more tuples in the relation causes recursive partitioning of its base space until the number of tuples contained in each page is less than or equal to 3 tuples (i.e., the page size).

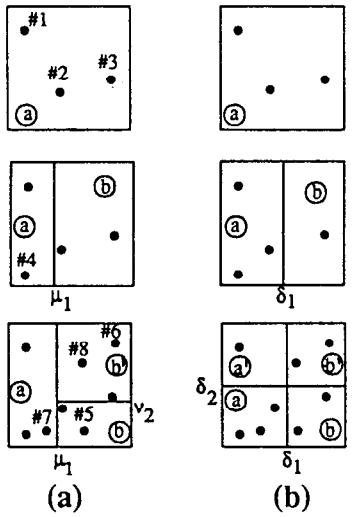

Fig. 1 Multi-Attribute Clustering Methods
(a) Adaptable (b) Fixed

For the fixed method, the boundary lines partition the base space of the relation at fixed values of each attribute domain and cut the entire base space as shown in Fig.1(b). A single page a of tuples #1,#2,#3 is initially assigned to the base space and overflows when tuple #4 is inserted, which causes the splitting with boundary line at the fixed value δ_1, generating a new page b. Then tuples #5,#6 are inserted with no page overflow. The insertion of tuples #7,#8, however, causes the overflow of pages a and b which are split with boundary line at value δ_2, generating pages a' and b', respectively. Thus the respective page spaces for a, a', b and b' are given by :

$A = [MIN_1, \delta_1) \times [MIN_2, \delta_2)$,
$A' = [MIN_1, \delta_1) \times [\delta_2, MAX_2)$,
$B = [\delta_1, MAX_1) \times [MIN_2, \delta_2)$ and
$B' = [\delta_1, MAX_1) \times [\delta_2, MAX_2)$.

Comparing the fixed and adaptable methods, we see that they basically differ in two points : first, for the fixed method the value of the boundary lines is predetermined, while for the adaptable method it depends on the data distribution; furthermore, the boundary lines cut the entire base space of the relation for the fixed method, while they become progressively shorter for the adaptable method. As can be easily observed from Fig.1(a) and (b), these differences imply that the load-factor for the fixed method is lower than that for the adaptable method.

3. Model of Query Processing for Multi-Attribute Clustered Relations

In this section we present a query processing model for multi-attribute clustered relations which emphasizes the page loading strategy. We consider relations clustered on k attributes $A_1, A_2, ..., A_k$, and relational operations referring only one of the k attributes, i.e., referring attribute A_i, for $i \leq k$. From now A_i is called processing attribute.

When the memory size is small and the relation scarcely fits in it, the index information can be very useful for efficiently accessing the relation. However, the multi-attribute clustered relations are characterized by treating all attributes symmetrically at the cost of losing the sequential data order between some specific pages. Thus, for example, the search for a partial range of a single attribute in a multi-attribute clustered relation requires much more page accesses than in a relation clustered on this single attribute [19]. In Fig. 2 we illustrate an example of a relation symmetrically clustered on attributes A_1 and A_2, with pages whose multi-attribute data are out of order. In this case the relation is composed of 4 pages a, a', b and b'. We can see that between pages a' (composed of tuples (1,4) and (3,7)) and b (composed of tuples (6,3) and (9,2)), for example, there is a sequential order of the values of attribute A_1. However, between pages a (composed of tuples (2,3) and (4,1)) and a' (composed of tuples (1,4) and (3,7)), a sequential order of attribute A_1 values is not maintained.

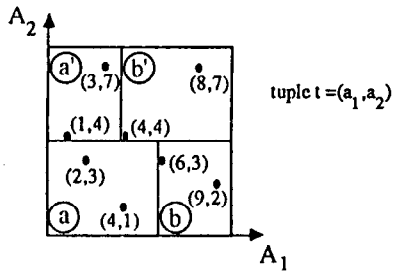

Fig. 2 Page Spaces for a Multi-Attribute Clustered Relation

Thus, the processing methods used for single-attribute clustered relations, which rely on the total order of the processing attribute value among pages and require only one scan of the relations, can not be efficiently applied for the multi-attribute clustering case.

In our query processing model, because the relation is larger than the memory, we use information of the

multi-attribute clustering index to divide the base space of the relation into subspaces which are used as processing units. These units are determined so that a sequential order of the processing attribute between them is maintained. In this query processing model, the determination of these processing units is fundamental and is based on new concepts which we introduce here : the processing range, cluster and wave. The domain of the processing attribute A_i is divided in disjoint parts, each of which is called *processing range*. *Cluster* is the logical subspace of the relation which may contain the tuples whose attribute value a_i is within the processing range. *Wave* is the set of physical pages of the relation which contains the cluster, that is, that actually holds the tuples whose attribute value a_i is within the processing range. Cluster is a subspace of the relation which is logically determined, and is independent of the physical pages of the relation, while the waves are the actual set of physical pages that are accessed from the secondary storage. In this query processing model, in order to minimize the number of page accesses from the secondary storage, the processing range is determined so that each wave fits in the memory and each cluster overlaps with the space of the pages in the correspondent wave as much as possible. For each processing range, the correspondent wave is loaded in the memory and its cluster is computed; the operation is repeated for all disjoint processing ranges and computation of the whole relation is done with a reduced number of page accesses.

In the next subsection 3.1., we briefly present the notations of cluster and wave. Then, in subsection 3.2., we discuss the processing flow of the model, showing how the processing ranges, the clusters and the waves may be determined to reduce the number of page accesses.

3.1. Cluster and Wave

Consider a processing range $[\chi_i, \delta_i)$, where $\chi_i \leq \delta_i$ and $\chi_i, \delta_i \in [MIN_i, MAX_i)$. We define *cluster of* $[\chi_i, \delta_i)$ ($C_{[\chi_i, \delta_i)}$) as the space given by :

$$C_{[\chi_i, \delta_i)} = [MIN_1, MAX_1) \times \cdots \times [\chi_i, \delta_i) \times \cdots \times [MIN_k, MAX_k).$$

Thus the cluster of a processing range $[\chi_i, \delta_i)$ for an attribute A_i represents the subspace of the relation whose tuples have the attribute value a_i in this range. Fig. 3(a) shows an example for a 2-attribute clustered relation where $dom(A_1) = [0,10)$ and $dom(A_2) = [0,8)$, illustrating a cluster of [4,6) for attribute A_1.

Now we define *wave of* $[\chi_i, \delta_i)$ ($W_{[\chi_i, \delta_i)}$) as the set of pages which contains the cluster $C_{[\chi_i, \delta_i)}$ i.e.,

$$W_{[\chi_i, \delta_i)} = \{P_j \mid p_j \cap C_{[\chi_i, \delta_i)} \neq \phi, \text{ for } j = 1,\ldots,|R|\}.$$

Thus the wave of a processing range $[\chi_i, \delta_i)$ for an attribute A_i represents the set of pages which can actually contain tuples with the value of attribute A_i in this range. Fig. 3(b) shows the example of wave of [4,6) for attribute A_1.

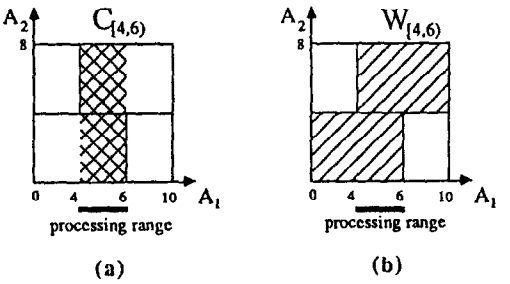

Fig. 3 (a) Cluster (b) Wave

3.2. Query Processing for Multi-Attribute Clustered Relations

For the base space of a relation, it is easy to partition it in clusters which maintain a mutual sequential order of the processing attribute and can be varied in size and number. When considering the clusters, we are only considering a logical space of tuples. We are not considering the actual tuples which are contained in this logical space and are physically housed in pages. In order to reduce the number of page accesses for the query processing, however, it becomes important to take into account the pages and their space, which are registered in the multi-attribute clustering index of the relation. In the following, we illustrate the query processing model flow and show how the processing ranges, clusters and waves are determined with an example for the join operation.

Fig. 4 shows an example of the join operation for relations R and S clustered on attributes A_1 and A_2, and joined on attribute A_1. In this example, both relations R and S are composed of 4 pages and the available memory space is 3 pages. From the multi-attribute clustering index of relation R, the processing range of the first step is determined as [0,4) as shown in Fig. 4(a). First, the wave $W_{[0,4)}$ of relation R, which is composed of 2 pages, is loaded in the memory. The wave $W_{[0,4)}$ of relation S, also composed of 2 pages, is loaded in the remaining single memory page, one by one, as shown in Fig. 4(b). The join of the tuples of clusters $C_{[0,4)}$ of relations R and S on memory can be processed using any of the conventional join algorithms, such as nested-loop, sort-merge or hash-based. In this example, the whole cluster $C_{[0,4)}$ of relation R is loaded in the memory at once. On the other hand, the cluster $C_{[0,4)}$ of relation S is loaded in the memory in two stages, since only one

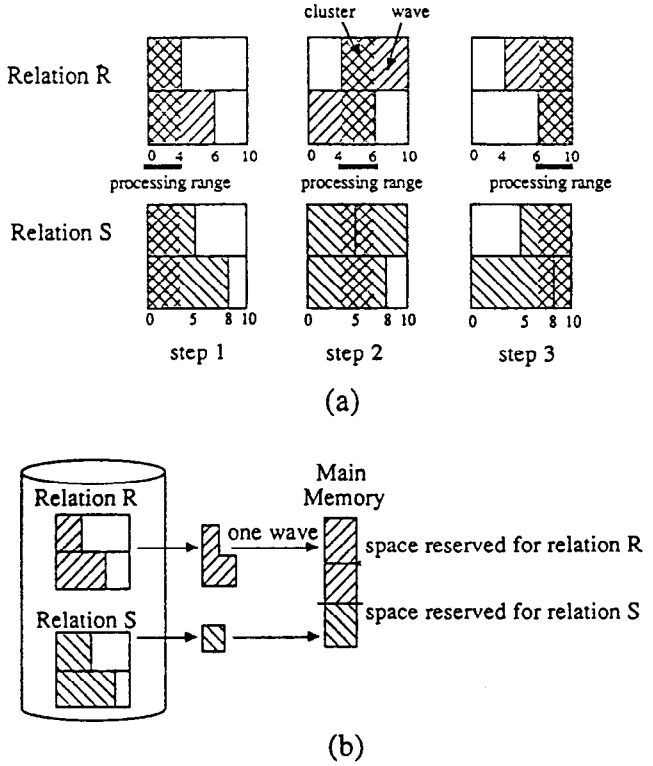

Fig. 4 Query Processing Model Flow for the Join Operation (a) Processing Flow (b) Wave Loading

memory page is available for the loading of the two pages of the wave $W_{[0,4)}$ of relation S. Now, let analyze how would the pages be accessed in the case of other processing ranges. First, let consider the case of processing range [0,3). The waves $W_{[0,3)}$ of relations R and S would be the same as $W_{[0,4)}$. Thus, the tuples of cluster $C_{[3,4)}$ of relations R and S would be loaded in the memory but not processed, which would increase the number of page accesses of relation S since these pages would be re-accessed for the computing of the processing range of the next step. Now, let consider the case of the processing range [0,6). The waves $W_{[0,6)}$ of relations R and S would be composed of 3 pages. Thus, neither of them could be entirely loaded in the memory, resulting in their being read from the secondary storage in a nested-loop way with too many page accesses. Thus, in the example shown in Fig. 1, the determination of the processing range of the first step as [0,4), so that the wave $W_{[0,4)}$ of relation R could be loaded in the memory at once, and the cluster $C_{[0,4)}$ was the largest cluster entirely contained in wave $W_{[0,4)}$ of relation R, was efficient to reduce the number of page accesses.

Now, let consider the processing of the second step. Once the computing over the processing range [0,4) is finished, the page of relation R whose space was entirely contained on cluster $C_{[0,4)}$ (i.e., has had all its tuples processed), is unloaded. Then, determining the processing range as [4,6) and by loading a new page of relation R into the memory, the wave "propagates" to $W_{[4,6)}$ for the second step. Again, the wave size of relation R is 2 pages and only 1 memory page is reserved to load the 3 pages of the wave $W_{[4,6)}$ of relation S. After the processing of clusters $C_{[4,6)}$ of relations R and S, the page of relation R whose space is entirely contained in the clusters of the processing ranges [0,4) and [4,6) (i.e., has had all its tuples processed) is unloaded from the memory. In the last step, the processing range is determined as [6,10) and the last page of relation R is loaded in the memory. The computing of the clusters $C_{[6,10)}$ of relations R and S is executed in the same way as the previous steps. This procedure is repeated until computing over the entire domain of A_1 is finished.

In the proposed query processing model, by overlapping as much as possible the physical pages of the wave to be loaded in memory with the logical space of the cluster to be processed, processing goes on in the attribute sequential order, with a low number of page accesses. Using this query processing model, relation R, from whose multi-attribute clustering index the processing ranges are determined, is accessed only once and the number of page accesses of relation S can be minimized.

Because the join operation is the most I/O costly and the most intensively studied among the relational operations, in the next two sections we detail join algorithms for adaptable and fixed multi-attribute clustered relations focusing on the join operation, in order to validate the efficiency of this query processing model which emphasizes the page loading strategy.

4. Join Processing for Adaptable Multi-Attribute Clustered Relations

The KD-tree indexing is one of the most known multi-attribute clustering mechanisms based on the adaptable method [1,2]. Using the concepts of cluster, wave and processing range (for the join operation, *join range*) introduced in the previous section, we present join algorithms for non-resident relations indexed by totally balanced KD-trees. We call them KD-join algorithms. In [15] we have evaluated the KD-join algorithms in detail and thus, we only present some representative results here. In the following we present some notations and assumptions of KD-trees, and then introduce two basic and two extended KD-join algorithms with their respective simulation results.

4.1. KD-Tree

For the KD-tree indexing, at each partitioning step,

the base space of the relation is partitioned with boundary lines determined so as to evenly divide the set of tuples. Let us consider a KD-tree indexed on k attributes. The partitioning step is recursively applied with boundary lines at values of attributes changed in cyclic order as $A_1,..., A_k, A_1,....$ For each partitioning step, a node of the KD-tree is constructed. When the total number of partitioning steps is d, we say that the depth of the KD-tree is d. For simplicity, here we consider totally balanced KD-trees. This means that when the depth of the KD-tree is d, the relation is partitioned into $|R| = 2^d$ pages. Thus, there are $|R|$ pages P_j (j = 1,...,$|R|$) with space p_j such that $D = \sum_{j=1}^{|R|} p_j$.

Consider a value $\gamma_i \in [MIN_i, MAX_i)$. From the definition of wave presented in the previous section, $W_{[\gamma_i,\gamma_i)}$ is the minimum wave size that is possible and equals $2^{(d-\frac{d}{k})}$ pages. From now we denote it $mws = 2^{(d-\frac{d}{k})}$ pages.

Now let us introduce the definition of *wave rear* and *wave front*. Consider a wave of range $[\gamma_i, \delta_i)$ ($W_{[\gamma_i,\delta_i)}$). For this wave, we define wave rear W_r and wave front W_f as :

wave rear $W_r = \bigcup \alpha_{ij}$ for $P_j \in W_{[\gamma_i,\gamma_i)}$; and

wave front $W_f = \bigcup \beta_{ij}$ for $P_j \in W_{[\delta_i,\delta_i)}$.

Fig. 5 illustrates an example for the wave $W_{[4,6)}$. In this case, $W_r = \{0,4\}$ and $W_f = \{6,10\}$.

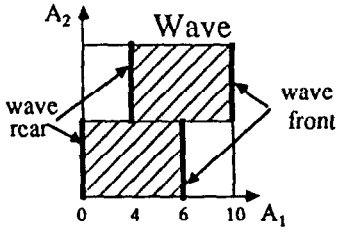

Fig. 5 Wave Rear and Wave Front

4.2. Basic KD-Join Algorithms

Let us consider the join of two relations R and S with $|R|$ and $|S|$ pages, respectively, so that $|R|$ ($= 2^d$ pages) $\leq |S|$, relation R and S indexed by a KD-tree of dimension k, and the memory size is $|M|$ pages ($|M| << |R|$).

4.2.1. Description of Basic KD-Join Algorithms

In the basic KD-join algorithms, the join ranges are determined from the KD-tree index of the smaller relation R. Considering a wave W of relation R, its wave rear $(W_r)_t$ and wave front $(W_f)_t$ at a certain step t, the join range to process at this step t is given by :

$[\max(\alpha_{ij} \in (W_r)_t), \min(\beta_{ij} \in (W_f)_t))$, so that $\max(\alpha_{ij} \in (W_r)_t) = \min(\beta_{ij} \in (W_f)_{(t-1)})$.

Fig. 6 illustrates it with an example. For each step, the join range is determined as the largest "inner" range of the wave of relation R. Successive join ranges are contiguous in the domain of attribute A_1.

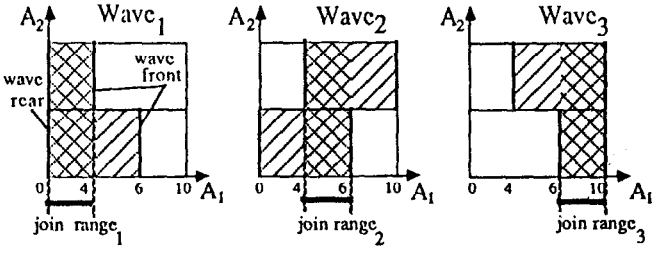

Fig. 6 Join Ranges for KD-tree Indexed Relations

We define two basic KD-join algorithms : algorithm KD-N (Narrow) and algorithm KD-W (Wide).

Algorithm KD-N is the one that uses a narrow wave, that is, the wave of relation R has the minimum wave size which is given by $mws = 2^{(d-\frac{d}{k})}$ pages. The idea of algorithm KD-N is to load the smallest wave of relation R in memory, and to reserve all the remaining memory space to load the wave of relation S.

Algorithm KD-W is the one that uses the wide wave, that is, the wave of relation R has the maximum size that can be loaded in the memory, which means, $|M|$-1 pages. The idea of algorithm KD-W is to load the largest wave of relation R in memory, and to reserve only one memory page to load the wave of relation S.

In the query processing model, in each step, the wave of relation R has priority to be loaded in the memory. Thus, the pages of relation R are loaded in the memory only once, while some pages of relation S are reloaded, depending on the available memory space. Following, we illustrate the determination of the join ranges, the waves and the clusters of relation R for algorithms KD-N and KD-W with the examples in Fig. 7 and 8, respectively. In the examples, the relations R and S, which are composed of 16 pages and indexed by 2-dimensional KD-trees, are joined on attribute A_1. Let us suppose that memory size $|M| = 7$ pages.

Algorithm KD-N :

In this example the narrow wave of relation R has $mws = 4$ pages. As shown in Fig. 7, the join of relations

R and S is processed in six steps. In the first step, the join range is determined as [0,3) and wave $W_{[0,3)}$ of relation R composed of four pages is loaded in the memory. The wave $W_{[0,3)}$ of relation S is loaded in the remaining three pages of the memory, and the join of the clusters $C_{[0,3)}$ of relations R and S on memory is processed using any of the conventional join algorithms. Once the processing of the first step is finished, the two pages of relation R of which all the tuples have been processed are unloaded. For the second step, the join range is determined as [3,5). Thus, two new pages of the wave $W_{[3,5)}$ of relation R are loaded. The wave $W_{[3,5)}$ of relation S is loaded in the remaining memory and the processing of clusters $C_{[3,5)}$ is done. As shown in the figure, this procedure is repeated for join ranges [5,7), [7,11), [11,14), [14,16) when R and S are joined in the entire domain [0,16) of the join attribute A_1.

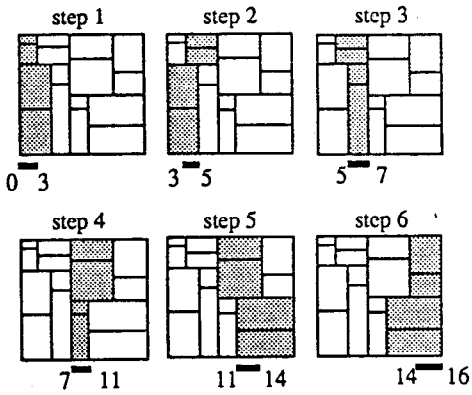

Fig. 7 Basic KD-Join Algorithm KD-N

Algorithm KD-W :

In this example the wide wave of relation R has |M|-1 = 6 pages. As shown in Fig. 8, the join is processed in four steps. The processing is analogous to algorithm KD-N, the difference being in the join ranges determined as [0,5), [5,7), [7,14) and [14,16). After loading a wave of 6 pages of relation R, the correspondent wave of relation S is loaded in the remaining one memory page.

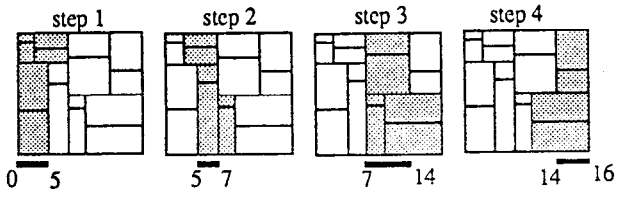

Fig. 8 Basic KD-Join Algorithm KD-W

4.2.2. Performance Evaluation of Basic KD-Join Algorithms

Because the most expensive operation in the join processing of non-resident relations is the I/O operation, we evaluate the basic KD-join algorithms by the number of page accesses. Following, we present the simulation results for relations R and S with 64 Ktuples, indexed by 2-dimensional KD-trees and with uniform data distribution as the one exemplified in Fig. 9. The page size is 16 tuples.

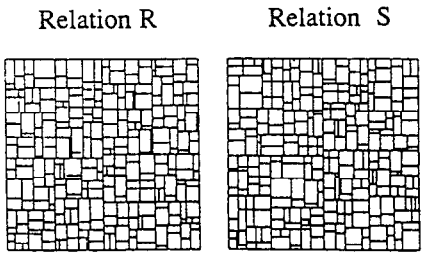

Fig. 9 An Example of Uniform Data Distribution

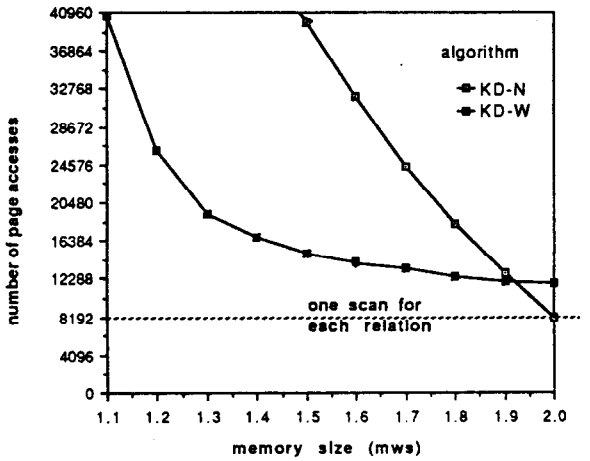

Fig.10 Simulation Results of Basic KD-Join Algorithms

In Fig. 10, the vertical axis is the total number of page accesses and the horizontal axis is the memory size. The memory size is varied from $1.1mws$ to $2mws$, where mws = 64 pages. The simulation results show that when the memory size increases, the number of page accesses decreases. We can observe that algorithm KD-W which uses the wide wave presents the best performance. Therefore, 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. On the other hand, we also verify that the join can be performed with one scan of each relation as the wave is narrow and the available memory space is sufficient to

hold at least the pages of relation S to be used again in the next step. Thus algorithm KD-N achieves the ideal number of page accesses for the range of memory size variation.

4.3. Extended KD-Join Algorithms

4.3.1. Description of Extended KD-Join Algorithms

Based on the analysis and performance results of the basic KD-join algorithms, we introduce some modifications in these algorithms with efforts to utilize the memory space more efficiently in order to :
(a) enlarge the available memory space in each step to hold more data of relation S, for algorithm KD-N; and
(b) enlarge the wave size to decrease the number of steps for algorithm KD-W.

In order to load and unload data in page units, the basic KD-join algorithms maintain tuples which have already been processed and are not necessary any more in the memory. Here we introduce a garbage collection mechanism which dynamically discards the tuples that are no longer necessary after the processing of each step, increasing the effective space of the memory for each step and allowing the loading of more pages in this free space. Here we call these new algorithms extended KD-N and extended KD-W.

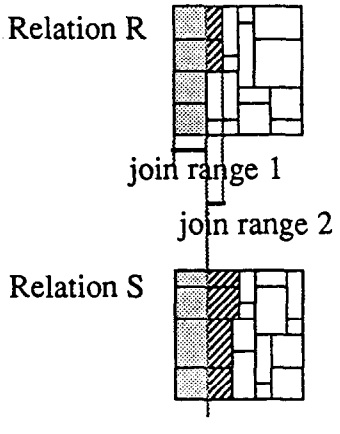

Fig.11 Garbage Collection Mechanism on Extended KD-Join Algorithms

As exemplified in Fig.11 for the case of algorithm extended KD-N, utilization of this dynamic garbage collection mechanism, after processing of the join range in the first step, allows the effective memory space for the second step to be enlarged if the dotted portion is discarded as garbage and only the dashed portion is maintained in the memory.

4.3.2. Performance Evaluation of Extended KD-Join Algorithms

Fig. 12 shows the number of page accesses of the extended KD-join algorithms for a uniform data distribution.

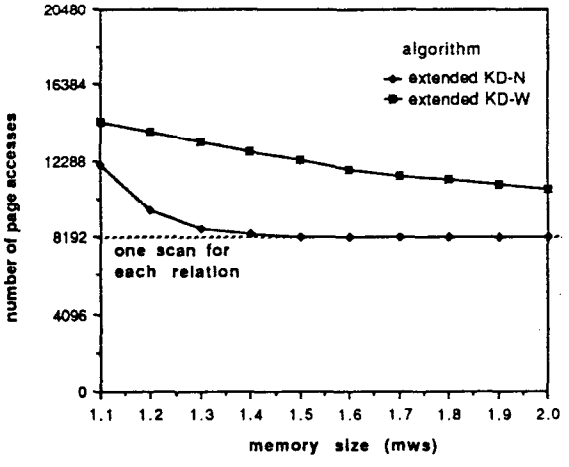

Fig.12 Simulation Results of Extended KD-Join Algorithms

As expected, all the extended algorithms reduce the number of page accesses in comparison with the basic KD-join algorithms. Algorithm extended KD-N presents the best performance and reduces the I/O cost to one scan for each relation in most of the memory size variation range. As opposed to the best basic algorithm (KD-W), the best extended algorithm (extended KD-N) is the one with narrow wave size, i.e., the one with the largest number of steps. With this algorithm extended KD-N, the effective memory size is almost enough to hold the narrow clusters of both relations to be processed in each step and thus the join can be performed with almost one scan of each relation, which is the lowest I/O bound. Here we evaluate the algorithms with the number of page accesses only. However, we have implemented these algorithms and have already reported in [9] that the overhead implied by the garbage collection mechanism was negligible in comparison with the disk read time, and the cost of the join operation was completely I/O bound. Therefore, the extended KD-join algorithm showed to be very efficient since it reduces the total execution time to one scan I/O cost.

5. Join Processing for Fixed Multi-Attribute Clustered Relations

In order to detail the join processing for multi-attribute clustered relations of the fixed method, we choose the Grid-file, which is the most representative fixed method [19]. In the following we present some notations of Grid-files and then two basic and two

extended join algorithms. We call them Grid-join algorithms.

5.1. The Grid-File

In the Grid-file, the insertion of tuples causes partitioning of the base space at pre-fixed values, generating grid subspaces. The assignment of grid subspaces to the data pages is the task of the *grid directory*. A grid directory consists of two parts : first, a dynamic k-dimensional array called *grid array*; its elements (pointers to data pages) are in one-to-one correspondence with the grid subspaces of the partition; and second, k one-dimensional arrays called *linear scales*; each scale defines a partition of a domain of the relation. For the sake of notational simplicity we present the case $k = 2$. A grid directory G for a two-dimensional space is characterized by two integers $nx > 0, ny > 0$ which gives the extent of the directory. The grid directory consists of the grid array, which is a two-dimensional array $G(0...,nx-1,0...,ny-1)$ and two linear scales which are one-dimensional arrays $X(0...,nx)$, $Y(0...,ny)$.

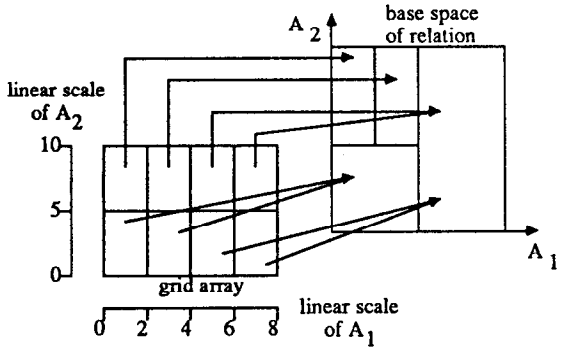

Fig. 13 Grid Directory

Efficiency dictates that only a subset of all possible assignments of grid subspaces should be allowed to pages. All tuples in one grid subspace must be stored in the same page. However, if each grid subspace has its own data page, page occupancy becomes low. Hence in the Grid-file several grid subspaces share a page. Fig. 13 shows a typical assignment of grid subspaces to pages.

5.2. Basic Grid-Join Algorithms

Based on the query processing model introduced in section 3, here we present the join algorithms for Grid-files, considering Grid-files R and S with |R| and |S| pages, respectively, such that $|R| \leq |S|$. The available memory size is |M| pages such that $|M| << |R|$.

5.2.1. Description of Basic Grid-Join Algorithms

In the Grid-join algorithms, the join ranges are determined from the grid directory of the smaller relation R. We define two basic Grid-join algorithms : algorithm G-N (Narrow) and algorithm G-W (Wide).

Algorithm G-N is the one that uses a narrow join range. The narrow join range is determined by consecutive elements of the linear scale of the attribute A_i being joined. Thus, for example, if the linear scale of A_i is given by $X(0,1,...,nx)$, the narrow join range of step t is given by :

$[X(t-1)_t, X(t)_t)$, such that $X(t-1)_t = X(t-1)_{(t-1)}$, for $t=1,...,nx$.

Thus, there are nx narrow join ranges which are determined as :

$[X(0), X(1)), [X(1), X(2)), ..., [X(nx-1), X(nx))$.

The idea of algorithm G-N is to load the smallest wave of relation R in memory, and to reserve all the remaining memory space to load the correspondent wave of relation S.

Algorithm G-W is the one that uses the wide join range. The wide join range is determined by elements of the linear scale of the attribute A_i, such that the respective wave is the largest one that can fit in memory. Thus, for example, if the linear scale of A_i is given by $X(0,1,...,nx)$, the wide join range of step t is given by :

$[X(i)_t, X(j)_t)$, such that $\text{sizeof}(W_{[X(i)_t, X(j)_t)}) \leq |M|-1$ and $\text{sizeof}(W_{[X(i)_t, X(j+1)_t)}) > |M|-1$; and $X(i)_t = X(j)_{(t-1)}$.

Following, we illustrate the determination of the join ranges, the waves and the clusters of relation R for algorithm G-N and G-W with the examples in Fig. 14 and 15, respectively. In the examples, the Grid-files R and S are clustered on two attributes A_1 and A_2, and R and S are joined on attribute A_1. Let us suppose that memory size |M| = 7 pages.

Algorithm G-N :

In the example shown in Fig. 14, the linear scale of attribute A_1 for relation R is given by (0,2,4,8,10,12,14,16). Thus, in the first step, the narrow join range is [0,2) and the waves $W_{[0,2)}$ of relations R and S are loaded in the memory and the clusters $C_{[0,2)}$ are processed. After this processing, the two pages of relation R, all of whose tuples have been processed, are unloaded and then the waves of both relations on the second join range, that is, [2,4) are loaded and processed. This procedure is repeated for the join ranges [4,8), [8,10), [10,12), [12,14) and [14,16), when R and S are joined in the entire domain [0,16) of the join attribute A_1.

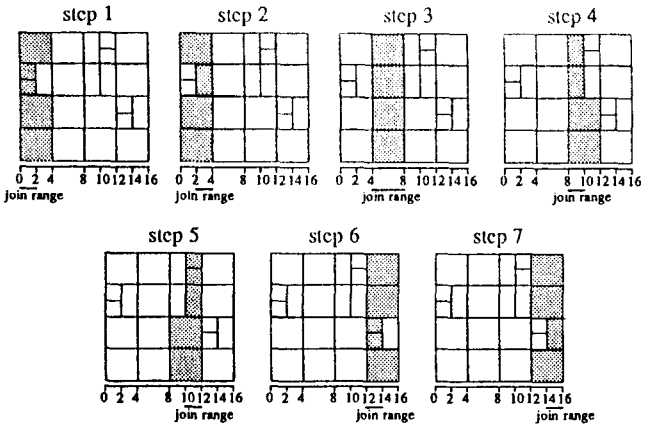

Fig. 14 Basic Grid-Join Algorithm G-N

Algorithm G-W :

In this example the wave of relation R can be sized at maximum |M|-1 = 6 pages. As shown in Fig. 15, the first join range is [0,4) and the 6 pages of the wave $W_{[0,4)}$ of relation R are loaded in the memory. The correspondent wave $W_{[0,4)}$ of relation S is loaded in the remaining 1 memory page, and the clusters $C_{[0,4)}$ are joined. In the same way, the join ranges [4,8), [8,10), [10,12), [12,16), whose correspondent wave sizes for relation R are 4, 4, 5 and 6 pages, respectively, are processed in the next 4 steps.

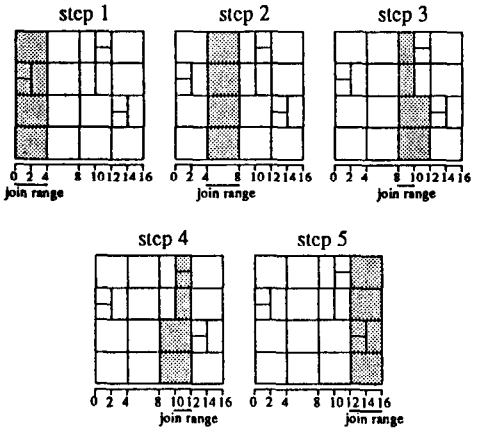

Fig. 15 Basic Grid-Join Algorithm G-W

5.2.2. Performance Evaluation of Basic Grid-Join Algorithms

The simulation results are for Grid-files R and S which are clustered on 2 attributes, have 64 Ktuples and random data distribution as the one exemplified in Fig. 16. The page size is 16 tuples.

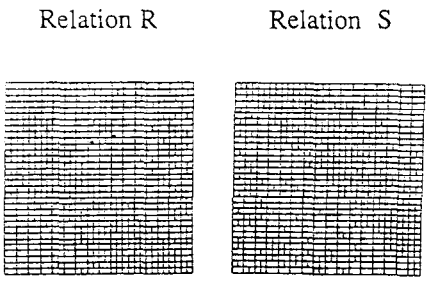

Fig. 16 An Example of Uniform Data Distribution

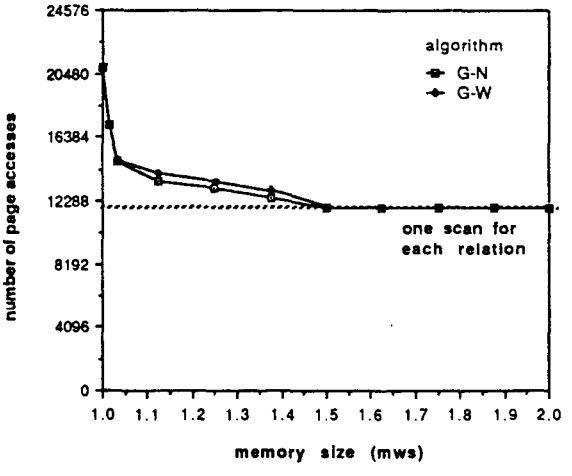

Fig. 17 Simulation Results of Basic Grid-Join Algorithms

In Fig. 17, the vertical axis is the total number of page accesses and the horizontal axis is the memory size. The memory size is varied from *mws* to 2*mws*, where *mws* = 64 pages and means the minimum wave size *mws* when considering the relation R indexed by a KD-tree. From Fig. 17 we see that the curves of algorithms G-N and G-W almost coincide. For memory size larger than 1.5*mws*, the number of page accesses is reduced to one scan of each relation, because the waves of both relations fit in the memory. When the memory size is smaller than 1.5*mws*, the curves show a linear increase with decreasing memory size. This is because, in each step, the wave of relation R fits in the memory, and thus the number of page accesses is inversely proportional to the memory space reserved to relation S. When the memory size is reduced to *mws* pages, the curves indicate an abrupt increase because the memory size is not enough to maintain the waves of both relations, which are read from the secondary storage and processed in a nested-loop way.

5.3. Extended Grid-Join Algorithms

5.3.1. Description of Extended Grid-Join Algorithms

As in the case of the KD-join algorithms, we introduce a garbage collection mechanism which discards the unnecessary tuples as soon as possible in order to increase the effective memory space. In algorithm extended G-N, the garbage collection mechanism increases the effective memory space used for the larger relation. On the other hand, in algorithm extended G-W, it increases the effective memory space used for the smaller relation and thus, enlarges the wide join range.

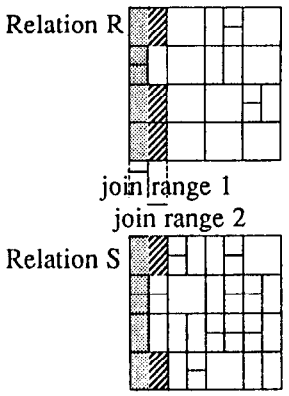

Fig. 18 Garbage Collection Mechanism on Extended Grid-Join Algorithms

As exemplified in Fig.18 for the case of algorithm extended G-N, with this dynamic garbage collection mechanism, after processing the join range in the first step, the effective memory space for the second join step is enlarged if the dotted portion is discarded as garbage and only the dashed portion is maintained in the memory.

5.3.2. Performance Evaluation of Extended Grid-Join Algorithms

Fig. 19 shows the number of page accesses from simulation results for the extended Grid-join algorithms using relations R and S with uniform data distribution. As shown by the figure, the introduction of the garbage collection mechanism increases the effective memory space so that the number of page accesses is reduced to one scan of each relation, which is the lowest I/O bound, in almost all the memory size variation. Comparing the result of Fig. 19 for the Grid-file with the result of Fig. 12 for the KD-tree indexed relation, we can see that although both of them minimize the number of page accesses to one scan of each relation, the KD-tree

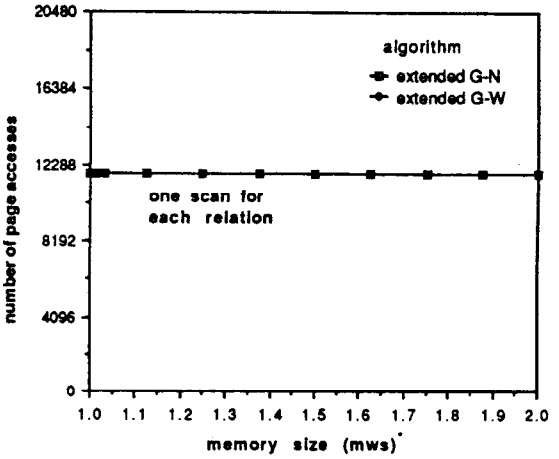

Fig. 19 Simulation Results of Extended Grid-Join Algorithms

requires more available memory space than the Grid-file to achieve this ideal I/O cost. We can also observe that the resultant minimum number of page accesses for the KD-tree is lower than that for the Grid-file. This is because, as a characteristic of the Grid-file, it presents lower load-factor and thus, more pages than the KD-tree indexed relation.

6. Conclusion

In this paper we introduced a query processing model which aims at reducing the high I/O cost of query processing for multi-attribute clustered relations caused by the problem of no total orders presented by the multi-attribute data. In this query processing model we introduced the concept of wave, which is a set of pages determined from the multi-attribute clustering index, and is used as the unit of loading from the secondary storage to the memory. The wave is determined according to a page loading strategy, so that the wave "propagates" over the base space of the relation in sequential order along the processing attribute axis with a reduced number of page accesses.

For the evaluation of the proposed model, we concentrated on the join operation which is the most costly relational operation and proposed join algorithms for multi-attribute clustered relations of the adaptable and fixed methods. These join algorithms were evaluated with simulations for the KD-tree and Grid-file, respectively. In the join processing of relations without a clustering index, the hash-based algorithms such as GRACE-hash [13,14] and Hybrid-hash [4,7], which have proved to be the best, requires an I/O cost of one write and two read scans of the relations. The simulation results of our best proposed algorithms has shown that an efficient utilization of the multi-attribute clustering index

can reduce the I/O cost of the join of very large relations to the minimum one read scan of the relations. Here we have only presented the performance evaluation for the join queries, showing results for relations with uniform data distribution and clustered on two attributes. However, this query processing model is also valid and sufficient for the efficient processing of the other relational unary queries like aggregate by group and projection with duplicate elimination, for relations with any data distribution and clustered on higher number of attributes [10].

Each multi-attribute clustering method has its strengths and weaknesses, as well as suitable environment. The selection of a multi-attribute clustering method by the user is application-dependent and also a database design problem. However, the concept of wave and the model introduced in this paper are general, so as to allow efficient processing of relational queries for all the multiple attributes, whichever clustering method is chosen.

[References]

[1] J.L.Bentley, "Multidimensional Binary Search Trees Used for Associative Searching", Communication of the ACM, pp.509-517, 18, 9, September, 1975

[2] J.L.Bentley, "Multidimensional Binary Search Trees in Database Applications", IEEE Trans. Software Eng., pp.333-340, 5, 4, 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] D.J.DeWitt et al.,"Implementation Techniques for Main Memory Database Systems," Proc. of the 1984 SIGMOD Conf., June, 1984

[5] M.Freeston, "The BANG File : A New Kind of Grid File", Proc. of the 1987 SIGMOD Conf., pp.260-269, 1987

[6] 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

[7] R.Gerber, "Dataflow Query Processing using Multiprocessor Hash-Partitioned Algorithms," PhD. Thesis, University of Wisconsin-Madison, 1986

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

[9] L.Harada, M.Kitsuregawa, M.Takagi, "Design and Implementation of Join Algorithms for KD-Tree Indexed Relations", to appear in Proc. of the Int. Conf. on Information Technology of IPSJ, October, 1990

[10] L.Harada, "Query Processing on Multi-Dimensionally Clustered Relations", PhD. Thesis, University of Tokyo, December, 1989

[11] A.Henrich, H.W.Six, P.Widmayer, "The LSD-tree: spatial access to multidimensional point and non point objects", Proc. of the 15th. Int. VLDB Conf., pp.45-53, 1989

[12] A.Hutflesz, H.W.Six, P.Widmayer, "Twin Grid Files : Space Optimizing Access Schemes", Proc. of the 1988 SIGMOD Conf., pp.183-190, 1988

[13] M.Kitsuregawa, H.Tanaka, T.Moto-oka, "Application of Hash to Database Machine and Its Architecture", New Generation Computing, 1,1, pp.64-74, 1983

[14] M.Kitsuregawa, M.Nakayama, M.Takagi, "The Effect of Bucket Size Tuning in the Dynamic Hybrid GRACE Hash Join Method", Proc. of the 15th. Int. Conf. on VLDB, 1989

[15] M.Kitsuregawa, L.Harada, M.Takagi, "Join Strategies on KD-Tree Indexed Relations", Proc. of the 5th. Int. Conf. on Data Engineering, pp.85-93, 1989

[16] H.P.Kriegel and B.Seeger, "PLOP-Hashing : A Grid File without Directory", Proc. of the 4th. Int. Conf. on Data Engineering, pp.369-376, 1988

[17] R.Krishnamurthy, K.Y.Whang, "Multilevel Grid Files", IBM Research Report, Yorktown Heights, 1985

[18] D.B.Lomet, B.Salzberg, "A Robust Multi-Attribute Search Structure", Proc. of the 5th. Int. Conf. on Data Engineering, pp.296-304, 1989

[19] 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

[20] E.A.Ozkarahan, H.Bozsahin, "Dynamic Order Preserving Data Partitioning for Database Machines", Proc. of the 11th. Int. VLDB Conf., pp.358-368, 1985

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