

Optimization of Queries Including ADT Functions

Kenichi Yajima*, Hiroyuki Kitagawa**,
Kazunori Yamaguchi**, Nobuo Ohbo**, and Yuzuru Fujiwara**

*Program in Engineering Science

** Institute of Information Science and Electronics

University of Tsukuba

Tsukuba, Ibaraki 305, Japan

ABSTRACT

Inclusion of ADTs (*Abstract Data Types*) has been studied by many researchers as a promising feature to make the relational database meet requirements from non-traditional advanced applications. In databases which support ADTs, the execution cost of a selection involving ADT functions (shortly, *ADT selection*) may be equal to or possibly more than that of a join, since the evaluation of a computationally and expensive complex function is often required in the test of the selection condition. In such environments, ADT selections become dominant cost factors in query optimization. The conventional query optimization heuristics sometimes does not work well in such databases, since it focuses on joins as dominant cost factors. In this paper, we propose an optimization method that takes ADT selections as well as joins into consideration. Based on some simulation results, we show that our method will find query execution plans superior to those obtained by the conventional heuristics, in particular, 1) when ADT selections are considerably more costly than joins, and/or 2) when there exist more than one ADT selection in a given query. We also propose some schemes to reduce the optimization cost of our method.

1. INTRODUCTION

Query optimization has been intensively studied in the relational database research community, and various types of heuristics have been developed so far. They include the syntactic rearrangements [ULL89], the query decomposition [WYO76], and the optimal usage of indices based on database statistics [SEL79]. Further, some studies [SWA88, SWA89, KBZ86, IBA84, WIO87] focus on the join operation, which is usually the most costly operation in select-project-join queries, and show how to determine the optimal join order.

Conventionally, the relational database management system has supported only built-in data types (e.g. integer, character) and built-in operators (e.g. +, -, *, /). It has been pointed out that this is a serious limitation when the system is used in an advanced application field such as CAD and

CAM. One promising solution to this problem is inclusion of abstract data types (shortly, ADTs) in the database management system [STO86a, STO86b, OSH86, JIA89]. An ADT is defined by a set of ADT functions to manipulate its instances. For example, the domain of the attribute CHEM-GRAPH of relation R in Fig. 1 is an ADT GRAPH. A sample selection condition is given as follows:

$R.CHEM-GRAPH \text{ isomorphic } \bigcirc$.

where the operator *isomorphic* is an ADT function defined on the ADT GRAPH. This query selects tuples that have CHEM-GRAPH occurrences isomorphic to the given ADT GRAPH instance: $\bigcirc$.

relation R:

ID	NUM-OF-NODES	NUM-OF-EDGES	CHEM-GRAPH (ADT)	FORMULA	FUNCTION-GROUP
O1	9	10		(C 9 H 16)	1100...
O2	5	5		(C 5 H 10)	0000...
1	1	1	1	1	1

Fig.1. Relation Involving ADT

The inclusion of ADTs gives rise to some serious problems in the context of query optimization in the relational database system. In the conventional query optimization strategy, the cost of a selection operation is regarded as much cheaper than that of a join operation. However, the selection involving evaluation of a computationally expensive ADT function, may be more costly than the join, since selection is performed by inspecting each tuple to determine whether the tuple satisfies the expensive ADT function or not. For example, one evaluation of the ADT function *isomorphic* for non-trivial graphs usually takes as much time as some disk block accesses. In order to solve this problem, [STO83, LYN88] proposes the use of secondary indices as a way to access tuples efficiently. However, it is not straightforward to prepare an index for such a complex ADT function as the above example. This is because 1) there exists no standard total order between the ADT GRAPH instances, and 2) it is computationally very expensive to find the unique representation of a given instance. We have noticed the fact that a join operation also contributes to reduction of the cardinality of a relation, if the join selectivity is less than a given certain value. So, there is a possibility that the total query processing cost is reduced if some joins are performed before such a costly selection. Therefore, in the optimization of a query which has selections involving ADT functions (shortly, *ADT selections*), the optimizer should not exclude

query execution plans in which some ADT selections are executed after joins. This contrasts with the conventional query optimization strategy of "pushing selections down" [ULL89].

In this paper, we propose a new optimization method which takes ADT selections into consideration as potentially dominant cost factors as well as joins. Our method determines the optimal execution order of ADT selections and joins based on their execution cost estimation. Here, the term "optimal" means "cost minimum." Our method makes use of the KBZ method [KBZ86] as its component procedure. The KBZ method was originally proposed to derive the optimal linear join sequence for a query consisting of only join operations.

We apply our method to several sample queries involving ADT selections. The simulation results indicate that our method will find drastically superior query execution plans, in particular, 1) if ADT selections are considerably more costly than joins, and/or 2) if a given query includes more than one ADT selection.

The remainder of this paper is organized as follows. Section 2 gives definition of the basic terminology and clarifies the target problem. Section 3 shows basic assumptions and develop the cost model for joins and ADT selections. Section 4 describes the optimization method, and shows its applicability by some examples. Section 5 discusses evaluation and extension of the optimization method. Section 6 is devoted to the conclusion.

2. BASIC TERMINOLOGY

In this section, we define the basic terminology required to construct the query optimization method. We also make the problem investigated in this paper more clear.

In this paper, select-project-join queries have been investigated in most studies on query optimization. Conventional selections which include no ADT function in the selection conditions are much cheaper than joins and ADT selections, and ignored. Also projections are ignored because we do not include duplicate eliminations in projections. Therefore, we view a given query just consisting of joins and ADT selections.

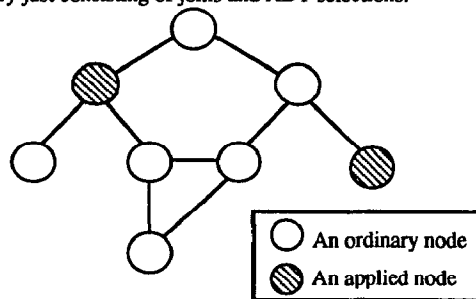


Fig.2. Query Graph

We construct a *query graph* for a given query as follows. An example is shown in Fig. 2. Each node of the query graph corresponds to a relation involved in the query.* The nodes of the query graph are categorized into two types. Nodes to which ADT selections are applied are called *applied*

* We use the terms "relation" and "node" interchangeably in the following part of the paper.

nodes, and the others are called *ordinary nodes*. A node in the query graph is connected to another node by an undirectional edge iff there exists a join predicate between them in the query. A query is a *tree query* iff the query graph is connected and acyclic. In this paper, we consider only a tree query. A tree query is called a *rooted tree query* iff the root node of the tree is specified. In the rooted tree query, the first join is assumed to be applied to the root. Further, the query graph of a tree query is called a *join tree* iff the query includes only joins.

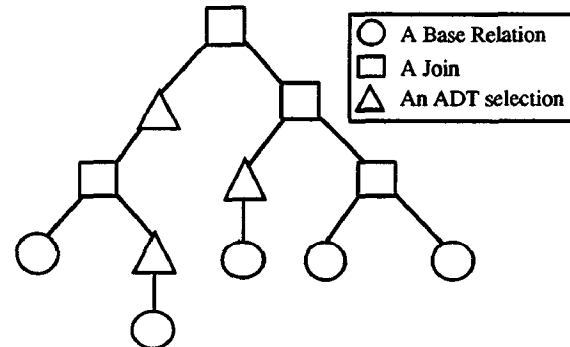


Fig.3. QLPT

A *processing tree* (shortly, *PT*) is a tree which specifies the execution order of operations involved in a given query. An example of a PT is shown in Fig. 3. In PTs, circles, rectangles, and triangles represent base relations, joins, and ADT selections, respectively. Operations in a PT are executed from the bottom to the top. A variety of PTs exist for a given query. A PT is called a *linear PT* (shortly, *LPT*) iff all operations appear in a linear sequence, in other words, they are totally ordered.

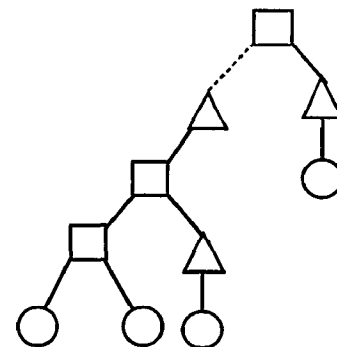


Fig.4. QLPT

PTs considered as query execution plans in this paper are *quasi linear processing trees* (shortly, *QLPTs*) as shown in Fig. 4. A QLPT is composed of one *trunk* and *n branches* ($n \geq 0$). The trunk is a linear sequence of joins and ADT selections and their operand base relations. A branch consists of a single ADT selection and its operand base relation. ADT selections on branches are called *type-1*, and those on the trunk are called *type-2*. Joins appear only on the trunk. Most query optimization heuristics proposed and implemented so far consider only linear join sequences to reduce the search space [IBA84, KBZ86, SEL79, WYO76, SWA88, SWA89, TAY90]. This is the reason why we restrict joins to appear only on the trunk. The conventional optimization heuristics of "pushing selections down" bring QLPTs as shown in Fig. 5, where all selections are directly applied to base relations. We intend our optimization method to generate the same query execution plan as the conventional one,

Therefore, the problem studied in the paper is to find the optimal QLPT for a given tree query composed of joins and ADT selections. Because of the above mentioned constraints, the root always appears at the bottom of the trunk in QLPTs if a given tree query is a rooted tree query.

In this section, we establish cost formulas for joins and ADT selections, and derive a cost model for OLPTs.

Let $R_1, R_2, \dots, R_n$ be relations, and let $|R_i|$ be the cardinality of R_i . Then, the *join selectivity* denoted by S_{ij} represents the expected ratio of the number of tuple pairs from R_i and R_j that meet the join condition. That is,

$$S_{ij} = \frac{|R_i \bowtie R_j|}{|R_i| * |R_j|}$$

Similarly, let D_i be the selectivity giving the expected ratio of the number of tuples from R_i that are selected by ADT selection $\sigma(R_i)$. That is,

$$D_i = \frac{|\alpha(R_i)|}{|R_i|}$$

Here, we make the following assumptions required for constructing the cost formula for QLPTs. Similar assumptions are made in [KBZ86, SWA88, SWA89]:

- 1) A given query is restricted to be a tree query, as we previously mentioned.
- 2) Database is memory resident, and the cost formulas of a join and an ADT selection are based on the in-memory processing costs.*
- 3) The size of the main memory is infinite.
- 4) Joins and selections in a given query do not interact with each other as far as their selectivities are concerned. For example, the same join selectivity S_{ij} is not only applicable to $R_i \bowtie R_j$ but also to the joins between R_i and $E(R_j)$, where $E(R_j)$ stands for a regal combination of joins and ADT selections involving R_j and other relations. A similar

* We discuss application of our cost model to the disk resident database in Section 5.

6) The tuple matching cost of a join operation $R_1 \bowtie R_2$ is of the form $N_1 * g_2(N_2)$. Here, N_1 is the cardinality of R_1 , and $g_2(N_2)$ is the cost incurred per tuple of relation R_1 . The nested loop join method and the hash join method have cost formulas of this form. Note that the comparison cost incurred for each pair of tuples in joining two relations is regarded as a unity. So far, we have not explicitly discussed joins including ADT functions (shortly, ADT joins). Note that inclusion of ADT joins can be managed properly in the above cost formula by setting a huge value to $g_2(N_2)$.

7) All of the parameters required for the execution cost estimation are given in advance, i.e. the cardinality of each relation, the cost function g , the join selectivity, and the ADT selection selectivity, and so forth.

First, we derive the cost formula for a linear sequence of joins, $(... (R_1 \bowtie ... \bowtie R_n))$. Here, the selectivity $S_{(i-1)}$ is simply denoted by S_i . The cost formula for a join between R_1 and R_2 is of the form $N_1 * g_2(N_2) + c * N_1 * N_2 * S_2$. The constant c caribrates a comparison cost and an insertion cost. Note that the cardinality of the result is $N_1 * N_2 * S_2$. If we use the dummy selectivity S_1 whose value is a unity, the above formula becomes $N_1 * g_2(N_2) + c * N_1 * N_2 * S_1 * S_2$. Similarly, the cost of a join between $R_1 \bowtie R_2$ and R_3 is of the form $N_1 * N_2 * S_1 * S_2 * g_3(N_3) + c * N_1 * N_2 * N_3 * S_1 * S_2 * S_3$. Also, the cardinality of the result is $N_1 * N_2 * N_3 * S_1 * S_2 * S_3$. In general, the total cost of the join sequence $(... (R_1 \bowtie R_2) \bowtie ... \bowtie R_n)$ is the following form:

$$\sum_{i=2}^k \{ [\prod_{i=1}^{i-1} (N_j^* S_i)] * [g(N_i) + c * N^* S_i] \} \quad (3.1)$$

Then, we derive the cost formula for an ADT selection. From the above assumption 5), an applied relation must be scanned sequentially. Therefore, the selection cost of $\sigma(R_i)$ is of the form $N_i * \lambda_i + c * N_i * D_i$. Here, λ_i is the average cost of evaluating the ADT selection condition for each tuple in R_i , and D_i represents the ADT selection selectivity. The cardinality of the result relation is $N_i * D_i$.

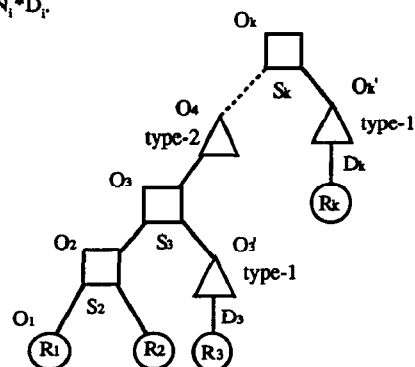


Fig.6. OLPT

Combining the above cost formulas for joins and ADT selections, we get the cost formulas for QLPTs. As shown in Fig. 6, we assume that operations on the trunk are sequentially indexed by integers 2, ..., k from the bottom to the top and that indices for type-1 ADT-selections on branches and base relations are determined based on the indices for

operations on the trunk. Those indices are also used to designate their selectivities and cost parameters. (Note that the operation O_1 and the selectivity S_1 are dummy and $S_1=1$.) The cost for a QLPT is expressed as follows:

$$\text{cost}(PT) = \sum_{R_j \in ST} (N_j * \lambda_j + c * N_j * D_j) + \sum_{i=2}^k \left(\prod_{j=1}^{i-1} X_j \right) * Y_i \quad (3.2)$$

where

ST: a set of relations which appear on branches.

$$X_j = \begin{cases} N_j * D_j * S_j & \text{if } O_j \text{ is a join applied to a type-1 ADT selection result: } O_j(R_j). \\ D_j & \text{if } O_j \text{ is a type-2 ADT selection.} \\ N_j * S_j & \text{otherwise.} \end{cases}$$

$$Y_i = \begin{cases} N_i * D_i + c * N_i * D_i * S_i & \text{if } O_i \text{ is a join applied to a type-1 ADT selection result: } O_i(R_i). \\ \lambda_i + c * D_i & \text{if } O_i \text{ is a type-2 ADT selection.} \\ g(N_i) + c * N_i * S_i & \text{otherwise.} \end{cases}$$

In (3.2), the first term is the total cost of type-1 ADT selections in the QLPT, namely the cost of operations on branches. The second term is the total cost of joins and ADT selections on the trunk of the QLPT. Here, let N_i' be as follows:

$$N_i' = \begin{cases} N_i * D_i & \text{if } R_i \in ST. \\ N_i & \text{otherwise.} \end{cases}$$

Then, the second term of the formula (3.2), denoted by cost-2, is simplified as follows:

$$\text{cost-2}(PT) = \sum_{i=2}^k \left(\prod_{j=1}^{i-1} X_j \right) * Y_i \quad (3.3)$$

$$X_j = \begin{cases} D_j & \text{if } O_j \text{ is a type-2 ADT selection.} \\ N_j * S_j & \text{if } O_j \text{ is a join.} \end{cases}$$

$$Y_i = \begin{cases} \lambda_i + c * D_i & \text{if } O_i \text{ is a type-2 ADT selection.} \\ g(N_i') + c * N_i' * S_i & \text{if } O_i \text{ is a join.} \end{cases}$$

4. THE OPTIMIZATION METHOD

In this section, we first describe a basic strategy to obtain the optimal QLPT from a given tree query. Then, we show an optimization algorithm based on that strategy, and demonstrate its applicability by some examples.

4.1. Basic Strategy

The algorithm here is to find the QLPT which gives the minimum value for the formula (3.2). Note that the cost formula (3.2) for a QLPT consists of two terms. Once, in a query graph, we arbitrarily select a set of applied nodes which appear on branches, the first term of (3.2) is determined. Note that there are 2^m possible choices if there exist m ADT selections in a given query. Then, the type-1 ADT selections are applied to these selected nodes, and they are replaced with nodes which are the results of the type-1 ADT selections as shown in Fig. 7. The remaining problem is to find the minimum value for the second term of (3.2), which is determined by the sequence of operations on the trunk to process the

modified query graph. Then, the issue is to obtain the optimal linear ordering of joins and type-2 ADT selections on the trunk. If we can determine such an optimal linear ordering for each 2^m possible choices of the applied nodes, we get the optimal QLPT based on such 2^m minimal cost values given by (3.2).

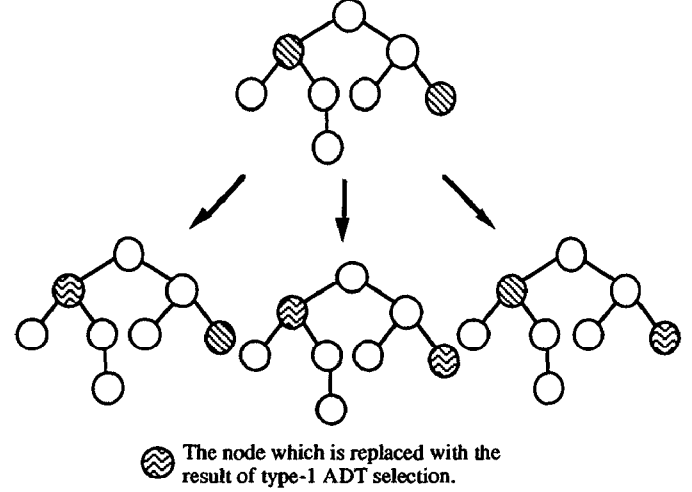


Fig.7. Modification of Query Trees

We show a method to determine the optimal linear ordering of joins and ADT selections on the trunk based on the KBZ method. The KBZ method was originally proposed in [KBZ86]. It determines the optimal linear execution ordering of joins for a given join tree. The cost function used in the KBZ method is same as that used here for joins. The cost for a linear processing tree, as shown in Fig. 8, consisting of k joins is formulated as:

$$\sum_{i=2}^k \left[\left(\prod_{j=1}^{i-1} (N_j * S_j) \right) * \{g(N_i) + c * N_i * S_i\} \right] \quad (4.1)$$

where the meanings of N_i , S_i , g_i are same as in our context. Note that (3.3) is essentially identical to (4.1), if there is no type-2 ADT selection. In order to apply the KBZ method to our problem of finding the optimal linear sequence of joins and ADT selections, we regard an ADT selection as a kind of "join." Namely, we look on each type-2 ADT selection σ_i with given D_i and λ_i as a dummy relation R_i which meets the constraints $N_i * S_i = D_i$ and $g_i(N_i) = \lambda_i$. Then, all type-2 ADT selections can be completely substituted by "joins," one operand of which is such a dummy relation. In parallel with this, the tree query is further translated into a join tree as shown in Fig. 9.

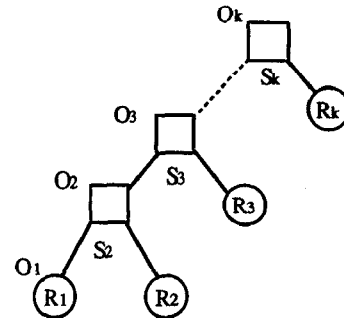


Fig.8. Linear Sequence of Joins by the KBZ method

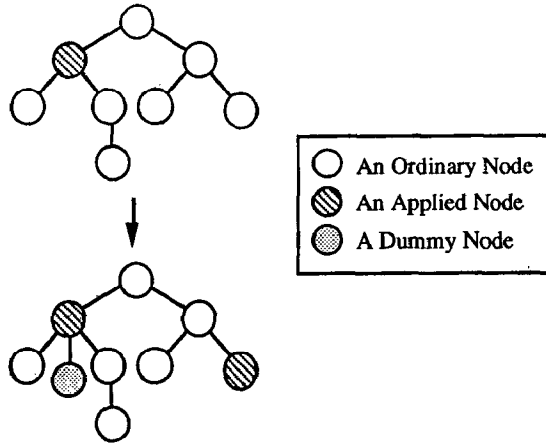


Fig.9. Join Tree with a Dummy Node.

Now we can directly apply the KBZ method so as to obtain the optimal linear sequence of joins for the join tree. After the execution of the KBZ method, we replace "joins" involving dummy relations with original ADT selections on the trunk. Then, it is the optimal linear sequence of joins and ADT selections.

4.2. The Overview of the KBZ Method

In this subsection, we give an overview of the KBZ method [KBZ86]. This method is composed of two levels of algorithms. The first algorithm KBZ-1 decides the optimal linear sequence of joins for a given rooted join tree. Here, the first join is applied to the root relation of the input tree. The second algorithm KBZ-2 inputs a join tree, and finds the optimal linear join sequence by invoking the KBZ-1 algorithm for each selection of the root.

The KBZ-1 algorithm works on the rooted join tree in a bottom-up manner. All nodes in subtrees are merged to form a linear sequence based on the value of the rank, which is defined as $(N_i * S_i - 1) / \{g_i(N_i) + c * N_i * S_i\}$. This process is recursively continued from the bottom to the top. When this process stops, we get a linear sequence consisting of all nodes, which gives the optimal join sequence for a given rooted join tree.

The KBZ-2 algorithm invokes the KBZ-1 so as to find an optimal linear join sequence for each selection of the root in the input join tree. If there exist n nodes in a join tree, it calls the KBZ-1 n times. Then, the KBZ-2 determines the optimal linear join sequence for a given join tree. It is shown in [KBZ86] that the computational complexity of the KBZ method is $O(n^2)$.

4.3. The Optimization Algorithm

Based on the strategy described in Section 4.1, we can construct an algorithm to find the optimal QLPT for a given query including joins and ADT selections. The optimization algorithm is as follows:

ALGORITHM:

INPUT: a tree query Q including m applied nodes $A_1, \dots, A_m$.

OUTPUT: the optimal QLPT for Q .

begin

```

Costmin = ∞;
for i=0 to 2m-1
  begin
    Qtemp = Q;
    Costtemp = 0;
    INDEX = i;
    /* Determine the set of applied nodes which appear on branches. */
    for j=1 to m
      begin
        if (INDEX(j) == 1) then
          /* INDEX(j) stands for the j-th bit of INDEX */
          begin
            Include the ADT selection  $\sigma_j$  into TYPE-1;
            Replace the applied node  $A_j$  in  $Q_{temp}$  with the
              result of  $\sigma_j(A_j)$ ;
            Costtemp = Costtemp + the cost of  $\sigma_j(A_j)$ ;
          end;
        end;
      /* Obtain the optimal linear sequence of joins and type-2
      ADT selections for the query  $Q_{temp}$ , using the KBZ method. */
      Connect dummy nodes to the remaining applied nodes in
         $Q_{temp}$ ;
      Determine the optimal linear join sequence SEQ by applying
        the KBZ method to  $Q_{temp}$ , and add its cost to Costtemp;
      if (Costtemp < Costmin) then
        begin
          Costopt = Costtemp;
          Construct the QLPT  $PT_{opt}$  such that the trunk is a se-
            quence of joins and ADT-selections based on SEQ,
            and that ADT selections in TYPE-1 appear on branches;
        end;
      end;
    return ( $PT_{opt}$ );
  end.

```

The above optimization algorithm invokes the KBZ method 2^m times, if m applied nodes exist in a given query. It is usually the case that at most two or three ADT selections are included in a query. Therefore, this exponential factor will not cause no serious problems in most situations. However, the optimization cost becomes prohibitive in case many applied nodes should appear in a query. We discuss solutions to this problem in Section 5.

4.4. Examples

In this section, we apply our optimization method to three sample queries including ADT selections. In order to evaluate its effectiveness, we compare our method with the conventional query optimization heuristics. Here, we assume that under the conventional heuristics, 1) all ADT selections are performed before joins under the principle of "pushing selections down," and 2) the KBZ method is applied to determine the optimal linear sequence of the remaining joins. We have done a number of experimental measurements to determine some constants which appear in our cost formula. The ratio of a comparison cost to an insertion cost, denoted by c , is

set here to 20 based on the actual measurements on SUN 3. The selection cost λ drastically differs depending on computation involved in ADT functions. It is not unusual for computationally expensive ADT functions such as graph isomorphism checking to have λ values greater than 10^6 . In the following examples, we consider two cases of $\lambda \approx 10^6$ and $\lambda \approx 10^2$.

(Case 1)

We apply the optimization method to a tree query as shown in Fig. 10. This query graph has only one applied node. We assume that the cardinality, $g_i(N_i)$, and the selectivity are given by Table 1. In this case, the cost for λ of the ADT selection per tuple is set to 10^6 , which is much greater than $g_i(N_i)$ values for joins. In other words, ADT selections are considerably costly than joins. Also, since relations R_3 and R_5 satisfies $N_3^*S_1 < 1$, joining R_3 or R_5 with another relation contributes to reduction of the cardinality of the relation. The optimal QLPTs which are derived by our method and the conventional heuristics are shown in Fig. 11 (a) and (b), respectively. The cost of the QLPT obtained by our method is less than half of that obtained by the conventional heuristics. In this case, joins $R_3 \bowtie (R_5 \bowtie R_6)$ reduce the cardinality of the intermediate result drastically. In addition, the ADT selection is much more expensive than joins. Therefore, the total cost is reduced by executing these joins before the ADT selection.

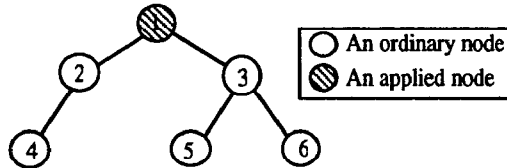


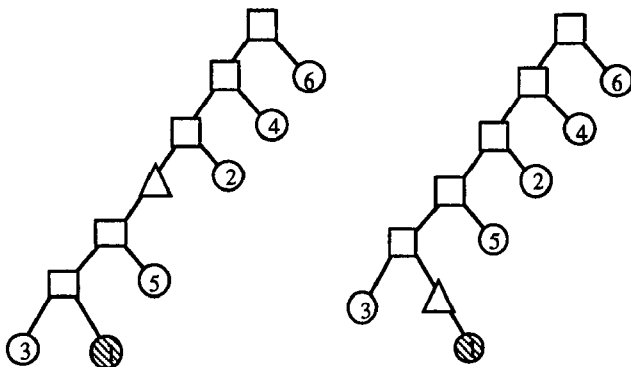
Fig.10. Tree Query with One Applied Node.

Table 1. Cost Parameters for Fig.10

node	1	2	3	4	5	6
cardinality: N_j	1000	1000	1000	1000	1000	1000
$g_i(N_i)$	100	150	100	200	200	150
selectivity: S_{ij}		$S_{12}=0.0015$	$S_{13}=0.0006$	$S_{24}=0.002$	$S_{35}=0.0008$	$S_{36}=0.002$

For applied node 1:
 $\lambda = 10^6$
 $Dc = 0.1$

The ratio of a comparison cost and an insertion cost:
 $c = 20$



The cost = 0.48×10^9

(a) The Optimal QLPT by our Optimization Method

The cost = 1.0×10^9

(b) The Optimal QLPT by the KBZ-method

Fig.11. Optimal QLPTs

(Case 2)

The optimization method is applied to the same tree query as in (Case 1). The cost parameters are same as in (Case 1), except that λ is set to 10^2 . In this case, the optimal QLPT derived by our method is same as that derived by the conventional heuristics as shown in Fig. 12. It means that, so as to reduce the total processing cost, it is wise to reduce the cost of subsequent join sequence by executing the ADT selection first. The conventional heuristics of "pushing selections down" works well in this case.

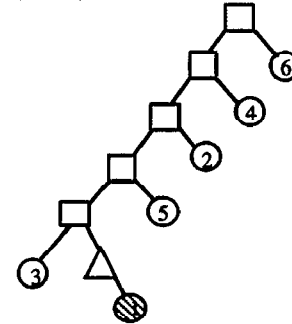


Fig.12. Optimal QLPT

(Case 3)

The optimization method is applied to a tree query shown in Fig. 13. This query graph has two applied nodes. We assume that the cost parameters are given by Table 2. No join in the tree query contributes to reduction of the cardinality of the intermediate result, and both ADT selections have the same costs as in (Case 2). In this case, the cost of the optimal QLPT derived by our method is less than two thirds of what is derived by the conventional heuristics as shown in Fig. 14. In this case, the ADT selection on R_2 drastically decreases the cardinality of the intermediate result. Because of this contribution, the cost of $\sigma(R_2) \bowtie (R_2 \bowtie (R_1 \bowtie \sigma(R_2)))$ becomes less than that of $\sigma(R_2) \bowtie (R_2 \bowtie (R_1 \bowtie \sigma(R_2)))$.

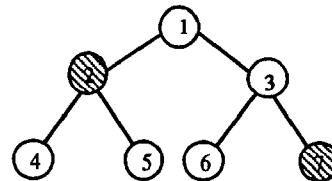


Fig.13. Tree Query with Two Applied Nodes

Table 2. Cost Parameters for Fig.13

node	1	2	3	4
cardinality: N_j	1000	1000	1000	1000
$g_i(N_i)$	100	150	300	150
selectivity: S_{ij}		$S_{12}=0.001$	$S_{13}=0.0015$	$S_{24}=0.002$
node	5	6	7	
cardinality: N_j	1000	1000	1000	
$g_i(N_i)$	100	200	150	
selectivity: S_{ij}	$S_{25}=0.003$	$S_{36}=0.002$	$S_{37}=0.002$	

For applied node 2:
 $\lambda = 100$
 $Dc = 0.05$

For applied node 7:
 $\lambda = 100$
 $Dc = 0.1$

The ratio of a comparison cost and an insertion cost:
 $c = 20$

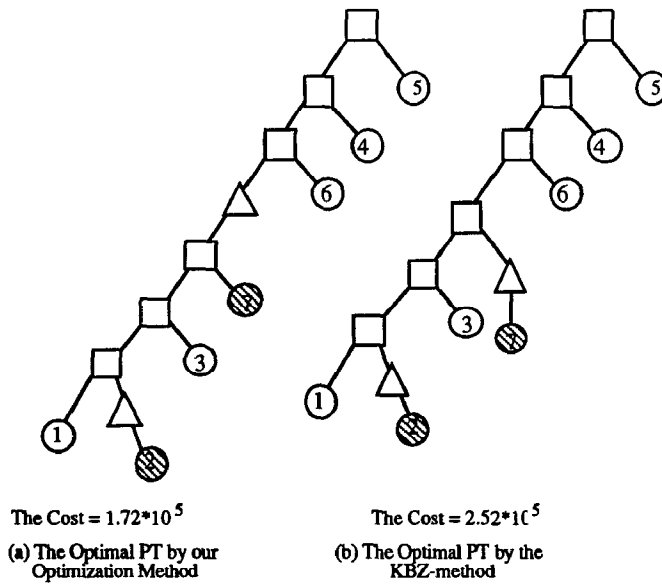


Fig.14. Optimal QLPTs

5. DISCUSSION

In this section, we evaluate our optimization method based on the simulation results presented in the previous section. Then, we discuss the extension of our method to the disk resident database. Finally, we propose some schemes to reduce the optimization cost.

5.1. Evaluation of the Optimization Method

Query execution plans obtained based on our optimization method are determined by complicated interactions among many cost parameters. Therefore, it is not easy to precisely identify cases our method is especially effective. Judging from the simulation results, our optimization method will find better query execution plans than the conventional heuristics, in particular, 1) when ADT selections are considerably more costly than joins, and/or 2) when there exist more than one ADT selection in a given tree query.

The case 1) has been exemplified by the simulation results for (Case 1) at Section 4.4. As in (Case 1), if there exists a join which meets the condition $N_i * S_j < 1$, this join contributes to reduction of the cardinality of the relation. Then, the voluminous execution cost of the ADT selection may be drastically reduced, if such joins are executed before the ADT selection. The case 2) has been exemplified by the simulation for (Case 3). In this case, it is cheaper to execute one ADT selection after joins. As exemplified by (Case 2), our method assures that the conventional heuristics of "pushing selection down" is still applicable to some queries involving ADT selections.

5.2. Application of Our Method to the Disk Resident Database

In this paper, we have assumed that the database is memory resident. However, our cost formula is basically applicable to analysis of the disk

resident database, if we properly adjust cost parameters. [IBA84] presented a similar cost formula starting from a disk resident model. One important point is to adjust join cost parameters. When we apply our cost formula to the disk resident database, joins in the disk resident model are much more costly than joins in the memory resident model. The ratio of a block access time to a memory access time is about 10^4 . Therefore, in general, the cost parameter $g_i(N_i)$ for a join has a large value in the disk resident model. However, as mentioned previously, it is not unusual for an ADT selection to have a λ value far greater than 10^6 . Therefore, our optimization method still plays a significant role in such situations.

5.3. Schemes to Reduce Optimization Cost

As we mentioned in Section 4, our method checks 2^m tree queries if a given tree query contains m applied nodes. Here, we propose two solution schemes to reduce this optimization cost. One scheme is to set termination criteria to the optimization method, and the other is to reduce the number of trees to examine in advance.

5.3.1. Termination Criteria

This scheme has been inspired by [SSD88]. We set a termination criterion to the optimization method. Then, the algorithm terminates the optimization process even in the middle of the execution when the criterion is satisfied. As in [SSD88], the following two termination criteria are promising. One is based on the ratio of the total optimization cost to the minimal query execution cost. The optimization process stops when the optimization cost spent so far reaches a certain fraction of the current minimal execution cost. The other is based on the ratio of the differential improvement of the query execution cost to the optimization cost. The optimization process stops when the optimization cost spent in the last iterative step dominates the improvement of the query execution cost brought by the step. This criterion terminates the process at a local minimum of the query execution cost.

5.3.2. The Reduction of Join Trees

The above scheme restricts time spent for the optimization process. The following heuristics reduces the number of join trees to be examined in the optimization process:

For each applied node in a given tree query, the selection rank $(D-1) / (\lambda + c * D)$ is calculated. If the selection rank value is smaller than a specified value, the ADT selection for this node is performed before any join.

This heuristics is based on the following observation: The selection rank for an applied node is small iff its selectivity D and its selection cost per tuple λ are both small. The conventional heuristics of "pushing selections down" says that selections should be done first if the selection rank is very small. Therefore, if the selection rank is small enough, the possibility that the best evaluation plan is to perform the ADT selection before joins.

6. CONCLUSION

In databases which support ADTs, ADT selections are sometimes equal to or possibly more costly than joins. Then, ADT selections as well as joins work as dominant cost factors in the query optimization. The conventional optimization heuristics does not work well in such databases, since it assume that only joins are dominant cost factors.

In this paper, we have proposed an optimization method for databases including ADTs. Our method equally takes joins and ADT selections into consideration, and obtains the optimal execution plan for a given tree query. Comparing our method with the conventional heuristics based on the analysis of sample cases, it has been clarified that our method will generate better query execution plans than the conventional one, in particular, 1) when ADT selections are considerably more costly than joins, and/or 2) when a given query includes more than one ADT selection. We have also proposed some schemes to reduce its optimization cost.

Now, we are planning to construct the optimization method for a cyclic query, and to implement it in the chemical structure database management system CHARM [LUA89], where various types of chemical graph ADTs appear as attribute domains of relations with complex ADT functions defined on them.

REFERENCES

- [IBA84] Ibaraki, T. and Kameda, T., *On the Optimal Nesting Order for Computing N-Relational Joins*, ACM TODS, Vol.9, No.3, Sept. 1984, pp. 482-502.
- [JIA89] Jiang, S., Kitagawa, H., Ohbo, N., Suzuki, I., and Fujiwara, Y., *Abstract Data Types in Graphics Databases*, Proc. IFIP TC 2/WG 2.6 Working Conf. on Visual Database Systems, April. 1989, Tokyo, pp. 239-255.
- [KBZ86] Krishnamurthy, R., Boral, H., and Zaniolo, C., *Optimization of Nonrecursive Queries*, Proc. 12th Int. Conf. VLDB, Aug. 1986, Kyoto, pp. 128-137.
- [LUA89] Luan, Y., Ohbo, N., Kitagawa, H., and Fujiwara, Y., *Functional Approach to Chemical Structure Database*, Int. Symposium on DASFAA, 1989, Seoul.
- [LYN88] Lynch, C., and Stonebraker, M., *Extended User-Defined Indexing with Application to Textual Databases*, Proc. 14th Int. Conf. VLDB, 1988, Los Angeles, CA, pp. 306-317.
- [OSH86] Osborn, S. and Heaven, T., *The Design of a Relational Database System with Abstract Data Types for Domains*, ACM TODS, Vol.11, Sept. 1986, pp. 357-373.
- [SEL79] Selinger, P. G., *Access Path Selection in a Relational Database Management System*, Proc. ACM SIGMOD Int. Conf. on Management of Data, 1979, Boston, pp. 23-34.
- [SSD88] Shekhar, S., Srivastava, J., and Dutta, S., *A Formal Model of Trade-off between Optimization and Execution in Semantic Query Optimization*, Proc. 14th Conf. VLDB, 1988, Los Angeles, CA, pp. 457-467.
- [STO83] Stonebraker, M., Rubenstein, B., and Guttman, A., *Application of Abstract Data Types and Abstract Indices to CAD Data Bases*, Proc. Engineering Applications Stream of 1983 Data Base week, May. 1983, San Jose, CA.
- [STO86a] Stonebraker, M., *Design of POSTGRES*, Proc. ACM SIGMOD Conf. on Management of Data, 1986, pp. 340-355.
- [STO86b] Stonebraker, M., *Inclusion of New Types in Relational Data Base Systems*, Proc. 2nd. Int. Conf. on Database Engineering, February, 1986, Los Angeles, CA.
- [SWA88] Swami, A., *Optimization of Large Join Queries*, Proc. ACM-SIGMOD Int. Conf. on Management of Data, 1988, Chicago, Illinois, pp. 8-17.
- [SWA89] Swami, A., *Optimization of Large Join Queries: Combinatorial Techniques*, Proc. ACM-SIGMOD Int. Conf. on Management of Data, 1989, Portland, Oregon, pp. 367-376.
- [TAY90] Tay, Y., *On the Optimality of Strategies for Multiple Joins*, Proc. 9th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, 1990, pp. 124-131.
- [ULL89] Ullman, J. D., *Principles of Database and Knowledge-Base Systems*, Volume II, Computer Science Press, 1989, pp. 633-733.
- [WIO87] Wong, E. and Ioannidis, Y., *Query Optimization by Simulated Annealing*, Proc. ACM-SIGMOD Int. Conf. on Management of Data, 1987, pp. 9-21.
- [WYO76] Wong, E. and Youseffi, K., *Decomposition - A Strategy for Query Optimization*, ACM TODS, Sept. 1976, pp. 223-241.