数理解析研究所講究録
第 695 巻 1989 年 108-114

Complexity of the Optimum Join Order Problem in Deductive Databases

京都大学工学部・宇野　裕之
(Yushi UNO, Faculty of Engineering, Kyoto University, Kyoto, Japan)

京都大学工学部・茨木　俊秀
(Toshihide IBARAKI, Faculty of Engineering, Kyoto University, Kyoto, Japan)

1 Introduction

The roll of inference in a deductive database system is to generate new facts from given rules and facts. These rules and facts constitute a database (knowledge base). In the practical applications, it is important to execute the inference process efficiently. A main cause of hampering inference efficiency is that the size of intermediate data may explode before the answer is derived. Therefore, it is important to carry out inference process so that the size of intermediate data is kept small.

Now, assume that the rules are represented by Horn clauses, as used in Prolog. In this case, each predicate can be considered as a relation, and given facts are explicitly kept in the form of extensional relational tables (i.e., extensional database). Other predicates are determined by the inference procedures based on the given rules and extensional tables. [1, 2, 7, 10]. In other words, the inference procedure can be understood as executing operations of relational databases on many relational tables. Among these operations applied to databases, the join operation is most important in the sense that it consumes most of the computational time. Therefore, this paper will concentrate on the join operations and evaluate the size of the intermediate data generated by them.

Define a graph, called a query graph, by denoting each relational table by a node and by connecting two nodes to be joined by an edge. Theoretically, a query graph can be an arbitrary graph, but in practice it is often a tree. The amount of intermediate data generated by join operations relies on the execution order of join operations. If we use merge-scan method to execute join operations, determining a join order that minimizes the sum of the intermediate data sizes is equivalent to find an optimum selection order of all edges in the query graph.

In this paper, we examine the computational complexity of this problem, and show that the problem is NP-hard even if query graphs are restricted to be trees. This fact suggests that no algorithm can compute an optimum join order in polynomial time in the size of query trees.

However, it is also observed that, if we further restrict query trees to be certain special types of trees, relatively efficient polynomial time algorithms exist.

2 Definitions

2.1 Relational Tables and Selectivity Factors

Join is an operation which constructs a new relational table from two given relational tables. The size of the resulting table is determined by the sizes of original tables and the selectivity factor between them. The computing cost is also determined by the size of the resulting table, as will be discussed in the next section.

When we execute join operations, the size of each relational table and the selectivity factor between two relational tables to be joined determine its computational time. Introduce the following definitions.

R_i : the i-th relational table,

N_i : the size of R_i. i.e., the number of lines which are stored in R_i,

f_{ij} : the selectivity factor between R_i and R_j.

The join operation on an attribute a of a relation R_i and b of R_j, where a and b have the same domain, produces the relation that has all attributes in R_i and R_j, and contains a concatenation of a line in R_i and a line in R_j if and only if these lines have the same value in a and b, respectively. We denote this operation

$$R_i.a \bowtie R_j.b.$$

The size of the new relational table is denoted by $f_{ij}N_iN_j$. i.e., this is the definition of selectivity factor.

2.2 Query Graph and Query Tree

Queries which are given to a deductive database can be answered by executing operations of relational algebra such as projection, selection, join, sum of sets and difference of sets in some order. Among these operations, the join operation is the most important in the sense that it requires a dominating computational time. In view of this, we pay attention in the following discussion only to the join operations.

When we are asked to execute a set of join operations on relational tables $R_1, R_2, \cdots, R_n$, denote each relational table R_i as a node R_i and connect two nodes R_i and R_j by an edge (R_i, R_j) if they are to be joined. The resulting graph is called a query graph. Associate with each node R_i its size N_i, and with each edge (R_i, R_j) its selectivity factor f_{ij}. When a query graph is a tree, it is called a query tree. As we encounter only query trees in most of the applications, we consider query trees in the rest of this paper.

3 Definition of OPTJOIN

3.1 Computational Cost of a Join Operation

As efficient methods of joining two relational tables, two methods using sorting technique and using hashing technique are known. Operation $R_i.a \bowtie R_j.b$ by the first method proceeds as follows.

- Sort all lines in R_i by the values of attribute a, and R_j by the values of attribute b.
- Scan the lines in both relations in ascending order while outputting the join results.

These steps respectively require the computational time

- $O(N_i \log N_i + N_j \log N_j)$,
- $O(N_i + N_j + f_{ij} N_i N_j)$.

The latter method of hashing proceeds as follows.

- Construct the hash table of the values of attribute b in R_j.
- Scan the lines in R_i, while examining the existence of lines in R_j that have the same value of attribute b as the current value of attribute a, by making use of the hash table, and output the join results.

These steps respectively require the average computational time

- $O(N_i + N_j)$,
- $O(N_i + N_j + f_{ij} N_i N_j)$.

In either method, $O(f_{ij} N_i N_j)$ is dominant in most cases. Also this value is important because it gives the size of the resulting relational table. For this reason, we define the cost $c(e)$ needed for joining an edge $e = (R_i, R_j)$ by

$$c(e) = f_{ij} N_i N_j.$$

3.2 Merge-Scan Method and OPTJOIN

In addition to the straightforward method [9], called nested-scan, there are two known methods of executing join operations in more efficient manner; nested-loops method [4], and the merge-scan method [9]. For the nested-loops method applied to a given query, a polynomial time algorithm is known to determine an optimal execution order of joins that minimizes computational cost of join operations [4]. Therefore, we concentrate here on the merge-scan method.

The merge-scan method repeats the operation of selecting an edge in a query tree and joins its end nodes (i.e., the corresponding relational tables) until all the edges disappear. The join of two relational tables connected by an edge is executed by an appropriate method, as will be discussed later, and the new relational table is explicitly constructed. The final relational table obtained by joining all the edges in the query tree is the desired result.

The problem of finding an optimal join order for a given query tree (i.e., finding an optimal order of edges), which minimizes the total computational cost of the merge-scan method, is stated as follows.

OPTJOIN
Input: a query tree $T = (V, E)$, $V = \{R_1, R_2, \cdots, R_n\}$, $E = \{e_1, e_2, \cdots, e_m\}(m = n-1)$, size N_i of each node R_i, and selectivity factor f_{ij} of each edge $e = (R_i, R_j)$.
Output: a permutation $\pi = (e_{i_1}, e_{i_2}, \cdots, e_{i_m})$ of all edges in E that minimizes cost $C(\pi)$, where $C(\pi) = \sum_k c_\pi(e_{i_k})$ and $c_\pi(e_{i_k})$ is the computational cost of joining two end nodes of e_{i_k} after having joined edges $e_{i_1}, e_{i_2}, \cdots, e_{i_{k-1}}$.

4 NP-hardness of OPTJOIN

In order to show that a problem A is NP-hard, we have to indicate that an already known NP-complete problem B is (polynomially) reducible to A [3], i.e.,

$$B \prec A.$$

Here, $B \prec A$ means that an arbitrary problem instance Q of B can be transformed into a problem instance P of A in polynomial time in the size of Q, such that P and Q have the same answer.

In our discussion, we employ 0-1 KNAPSACK in the place of B, and show that

$$\text{0-1 KNAPSACK} \prec \text{OPTJOIN}.$$

0-1 KNAPSACK is defined in the following.

0-1 KNAPSACK
Input: positive integers $a_0, a_1, \cdots, a_n$ (without loss of generality, we assume $a_0 \geq a_i, i = 1, 2, \cdots, n$).
Output: YES if there exist $x_i \in \{0, 1\}, i = 1, 2, \cdots, n$, such that $a_0 + \sum_{i=1}^n a_i x_i = A/2 + 1$, where $A = \sum_{i=0}^n a_i$; otherwise NO.

Theorem 1 *OPTJOIN is NP-hard.* □

proof See [8].

5 Special Query Trees

Even if there is no polynomial time algorithm to compute optimum join orders for arbitrary query trees, there may exist such algorithms for special query trees. Some possible restrictions are;

(1) restrict the depth of a query tree less than k from an appropriate root node,

(2) restrict the degree of each node less than k.

However, restriction (1) does not make OPTJOIN easier, because the query trees in the proof of NP-hardness (Section 4) have depth $k \leq 2$.

Moreover, we can state the following theorem.

Theorem 2 *OPTJOIN with restriction (2) under the condition that $k \leq 3$ is NP-hard.* □

proof See also [8].

6 Query Trees Solvable in Polynomial Time

As a result of previous sections, it became clear that query trees must be severely restricted to have polynomial time algorithms. Here, we give polynomial time algorithms to the following two kinds of query trees.

6.1 Stars

A star is a tree which has one center node to which all the other nodes are connected, as shown in Fig.1. Node weights N_i and edge weights f_{0i} are also given in the figure, where they satisfy $0 < f_{0i} \leq 1$ and $N_i \geq 1$.

Let $M_i = f_{0i}N_i$ $(i = 1, 2, \cdots, m)$, and denote the edge with selectivity factor f_{0i} by e_i. Furthermore, suppose $M_i < M_j$ for $i < j$ without loss of generality. Now consider an arbitrary join order π:

$$\pi = (e_{i_1}, \cdots, e_{i_a}, \cdots, e_{i_b}, \cdots, e_{i_n}).$$

Here, if $i_a > i_b$ holds in π, construct the corresponding join order π' obtained by exchanging e_{i_a} and e_{i_b} in π:

$$\pi' = (e_{i_1}, \cdots, e_{i_b}, \cdots, e_{i_a}, \cdots, e_{i_n}),$$

Then their costs c_π and $c_{\pi'}$ become

$$c_\pi = N_0(M_{i_1} + \cdots + M_{i_1} \cdots M_{i_a} + \cdots + M_{i_1} \cdots M_{i_a} \cdots M_{i_b} + M_{i_1} \cdots M_{i_a} \cdots M_{i_b} \cdots M_{i_n}),$$

$$c_{\pi'} = N_0(M_{i_1} + \cdots + M_{i_1} \cdots M_{i_b} + \cdots + M_{i_1} \cdots M_{i_b} \cdots M_{i_a} + M_{i_1} \cdots M_{i_b} \cdots M_{i_a} \cdots M_{i_n}).$$

and obviously satisfy $c_{\pi'} < c_\pi$. This shows that joining the edges in the increasing order of $f_{0i}N_i$ is optimum.

The computational time of this algorithm is $O(n \log n)$ for sorting $f_{0i}N_i, i = 1, 2, \cdots, n$.

6.2 Chains

A chain is a tree, in which each node has degree less than or equal to 2, as shown in Fig.2. In this case, we can apply the following algorithm based on dynamic programming.

First of all, we define H_{ij} by

$$H_{ij} = \prod_{k=i}^{j-1} f_{kk+1} \prod_{k=i}^{j} N_k, \quad 0 \le i < j \le n.$$

Each H_{ij} denotes the weight of the node when $R_i, R_{i+1}, \cdots, R_j$ are joined into one node. Then, denoting the minimum cost of joining all nodes between R_i to $R_j, i < j$, by c_{ij}, the following recursion holds.

$$c_{ii} = 0, \quad i = 1, 2, \cdots, n,$$

$$c_{ij} = \min_{0 \le k \le j-i-1} [c_{i,i+k} + c_{i+k+1,j} + H_{ij}], \quad \text{for all } i,j \text{ such that } 1 \le i < j \le n. \qquad (6.1)$$

The above formulas expresses that the cost c_{ij} of joining a subtree from R_i to R_j into one node is computed by regarding that the last edge to be joined connects N_{i+k} and N_{i+k+1}. Since the break point k is not known in advance, minimum is taken over all k. Finally, the optimal total cost c^* for computing joins of a chain of length n is given by

$$c^* = c_{0n}.$$

Formulas (6.1) can be solved by computing c_{ij} in the nondecreasing order of $|j - i|$. As there are $n(n-1)/2$ possible pairs of i and j, the computational time of this algorithm is $O(n^2)$.

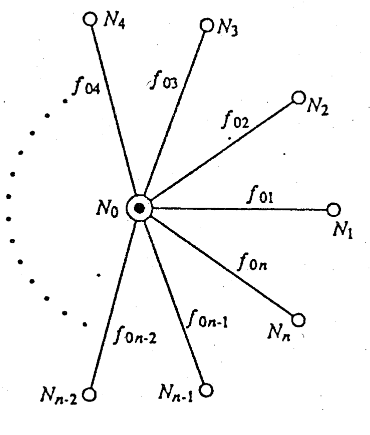

Fig.1 A star.

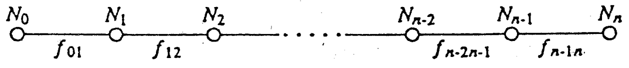

Fig.2 A chain.

7 Conclusion

In this paper, we showed that OPTJOIN is NP-hard for general query trees, but can be solved in polynomial time for some special query trees. As 0-1 KNAPSACK is used in this paper to prove that OPTJOIN is NP-hard, OPTJOIN may be only weakly NP-hard. Therefore, there may exist polynomial time approximation schemes for OPTJOIN by modifying those proposed for 0-1 KNAPSACK (e.g., [5]).

References

[1] Bayer, R., "Database Technology for Expert Systems", International GI-Kongress, 85, Wissenbasiete Systeme, Informatik Fachberichte 112, pp.1-16, 1985.

[2] Ceri, S., Tanca, L., "Optimization of Systems of Algebraic Equations for Evaluating Datalog Queries", Proceeding of the 13th VLDB Conference, Brighton 1987.

[3] Garey, M. R., Johnson, D. S., "Computers and Intractability : A Guide to the Theory of NP-Completeness", W. H. Freeman and Company, San Fransisco, 1979.

[4] Ibaraki, T., Kameda, T., "On the Optimal Nesting Order for Computing N-Relational Joins", ACM Transaction on Database System, Vol.9, No.3, pp.482-502, 1984.

[5] Ibarra, O. H., Kim, C. E., "Fast Approximation Algorithms for the Knapsack and Sum of Subset Problems", J. ACM, Vol.22, No.4, pp.463-468, 1975.

[6] Knuth, D. E., "The Art of Computer Programmings, Vol.2", Addison-Wesley, Reading, Mass., 1975.

[7] Ullman, J. D., "Implementation of Logical Query Languages for Databases", ACM Transaction on Database System, Vol.10, No.3, pp.289-321, 1985.

[8] Uno, Y., Ibaraki, T., "Complexity of the Optimal Join Order Problem", IEICE Technical Report, COMP88-81, 1989.

[9] Won, K., "A New Way to Compute the Product and Join of Relations", ACM SIGMOD, 1980.

[10] Zaniolo, C., "Safety and Compilation of Non-Recursive Horn Clauses", 1st International Conference on Expert Database Systems, pp.167-178, 1986.