MTA SZTAKI Közlemények 32/1985 27-49

OPTIMIZATION OF SELECTIONS IN RELATIONAL EXPRESSIONS

J. DEMETROVICS, HO THUAN

Computer and Automation Institute,
Hungarian Academy of Sciences
Budapest, Hungary

NGUEN THANH THUY

Hanoi Polytechnical Institute
Vietnam

ABSTRACT

One operation met usually in the relational expressions is the selection of a relation R on a conditional expression $E=\bigwedge_{i=1}^{n} E_i$. In this paper, basing upon the estimations of probability of the tuples $r \in R$ satisfying E_i, we shall show one simple O(nlogn) algorithm, where n is the length of E, rearranging the sub-expressions of E and so, the average probabilistic complexity of the algorithm for finding

$$\sigma_E(R)=\{r \in R / r \text{ satisfies } E\}$$

is minimal.

§O. INTRODUCTION

On operation met usually in expressions of relational algebra is the selection of a relation $\mathcal{R}$ on a conditional expression $\mathcal{E}$. In general, it requires time O(N), where N is the number of the tuples in $\mathcal{R}$, to perform that selection. However, when it is discussed in the common situation with respect to other operations, for instance, projection, join, cartesian product ... the following principle is in priority: "*Perform the selections and the projections as early as possible.*"

The transformation

$$\sigma_{\mathcal{E}}(\mathcal{R}) \Longrightarrow \sigma_{\mathcal{E}_1}(\sigma_{\mathcal{E}_2}(\ldots\sigma_{\mathcal{E}_m}(\mathcal{R})\ldots))$$

when $\mathcal{E}$ is of the form

$$\mathcal{E} = \bigwedge_{i=1}^{m} \mathcal{E}_i ,$$

is performed for the above principle. When an initial parse tree of a relational expression is reduced to a better form by the general optimization principles for relational expressions [1,3,4], it is possible that in the obtained parse tree there is a conjunctive selection

$$\sigma_{E_1 \wedge \ldots \wedge E_n}(R)$$

of a relation R on

$$E = E_1 \wedge \ldots \wedge E_n .$$

Due to the commutativity and the associativity of the operation $\wedge$, the relational expression

$$\sigma_{\bigwedge_{n=1}^{n} E_i}(R)$$

can be reduced to

$$\sigma_{\bigwedge_{i=1}^{n} E_{\tau_i}}(R)$$

where $\tau=\{\tau_1,\ldots,\tau_n\}$ is a permutation of $\{1,\ldots,n\}$. That is why we want to find the best permutation $\tau=\{\tau_1,\ldots,\tau_n\}$ such that the time complexity (cost) to find $\sigma_E(R)$ is minimal. In this paper, basing upon the estimations of probability of the tuples $r \in R$ satisfying the logical expressions E_i and the definition of the average probabilistic complexity of an algorithm, the best ordering $\tau=\{\tau_1,\ldots,\tau_n\}$ of the subexpressions E_i, $i=\overline{1,n}$ will be obtained such that the average probabilistic cost (complexity) of the algorithm finding $\sigma_E(R)=\{r\ R/r \text{ satisfies } E\}$ is minimal.

When E is an arbitrary a logical expression (as defined in §1) it can be reduced to the conjunctive - disjunctive normal form

$$E = \bigvee_{i=1}^{n} \bigwedge_{j=1}^{n_i} E_j^i$$

where E_j^i is of the form either $A\Theta B$ or $A\Theta c$ or $c\Theta A$, where A, B are attributes, c is a constant and Θ is a comparision operator $\Theta \in \{\geq, >, <, \leq, =, \neq\}$.

As the algorithm for a conjunctive selection can be used for a disjunctive selection with some modifications, so for a given arbitrary logical expression E, it is possible to find the best ordering ofthe subexpressions of E such that the average probabilistic cost of the algorithm finding $\sigma_E(R)$ is minimal.

It is interesting that when the cost to find the best ordering is added to the cost of the algorithm finding

$$\sigma_E(R) = \sigma_{\bigwedge_{i=1}^{n} E_{\tau_i}}(R) ,$$

the total cost remains desirable i.e. is less than the cost of the algorithm finding

$$\sigma_E(R) = \sigma_{\bigwedge_{i=1}^{n} E_i}(R)$$

with large N . The algorithm shown here for finding the best ordering τ of the subexpressions of E can be implemented in the computers as a subroutine without any access to the secondary memory devices containing the file R. Its time complexity is of O(nlogn) where n is the length of E.

§1. BASIC DEFINITIONS

Definition 1: A relation R with a set of attributes $U=\alpha(R)=\{A_1,\ldots,A_k\}$ and the corresponding ranges $D_1,\ldots,D_k$ is defined as follows

$$R \overset{def}{=\!=} \{r: U \rightarrow \bigcup_{i=1}^{k} D_i \mid \forall i\ 1\leqslant i\leqslant k\ \ r(A_i)\in D_i\}$$

or

$$R \overset{def}{=\!=} \{r= <t_1,t_2,\ldots,t_k> \mid \forall i\ 1\leqslant i\leqslant k\ t_i\in D_i\}$$

Each $r\in R$ is called a tuple of R.

Definition 2: A logical expression E in R with the set of attributes $U=\alpha(R)$ and the ranges $D_1,\ldots,D_k$ can be defined recursively as follows:

1) An expression of the form $A\theta B$, $A\theta c$, $c\theta A$, $A,B\in U$, $c\in \bigcup_{i=1}^{k} D_i$, $\theta\in\{\geqslant, >, \leqslant, <, =, \neq\}$, is a simple logical expression.

2) If E_1, E_2 are logical expressions, then $E_1 \vee E_2$, $E_1 \wedge E_2$, $\neg E_1$ are also logical expressions.

Definition 3: A logical expression E in R which is of the form $E=E_1\wedge \ldots \wedge E_n$ is called conjunctive logical expression.

Definition 4: Given a logical expression E in a relation R with the set of attributes ν and the ranges D_i, $i=\overline{1,k}$, and a tuple r of R.

We say that r satisfies E if when subtituting the names of the attributes A in E by the value r. $A\in \bigcup_{i=1}^{k} D_i$ of the tuple $r\in R$, the obtained logical expression has the value "true".

Definition 5: Given a relation R and a logical expression E. The selection of the relation R on the condition E, denoted by $\sigma_E(R)$ is defined as follows:

$$\sigma_E(R) = \{r \in R \mid r \text{ satisfies } E\} .$$

If E is a conjunctive logical expression, the selection $\sigma_E(R)$ is called a conjunctive selection.

Definition 6: Let Ω be a probability space of finite cardinality, i.e. in Ω is defined a probability measure $\rho: 2^\Omega \longrightarrow [0,1]$ satisfying the probability axioms. Put

$$\Omega = \{\omega_1, \omega_2, \ldots, \omega_g\}$$

and

$$\rho_i = \rho(\{\omega_i\}),\ i=\overline{1,g} .$$

Then, the average probabilistic value of the real valued function f defined on Ω corresponding to the probability measure ρ is defined as

$$\bar{f}_\rho = \sum_{i=1}^{g} f(\omega_i)\rho_i .$$

§2. AN APPROACH TO THE PROBABILITY ESTIMATIONS

Let a relation R be given with the set of attributes υ and the ranges D_i, $i=\overline{1,k}$. As defined in definition 2, a logical expression can be constructed by the simple logical expressions and the logical operators $\{\wedge, \vee, \neg\}$.

The following statistical parameters obtained and collected during the manipulation of R can be estimated:

1) The distribution of values of the attributes in υ.
2) The exact upper bound and lower bound for the attributes in υ during the manipulation.
3) The number of distinct values of an attribute. One of the simple assumptions is that the values of every attribute $X \in \upsilon$ are uniformly distributed in the segment $[X_m, X_M]$ where $X_m = \min R[X]$, $X_M = \max R[X]$. (H1).

With the assumption (H1), it is easy to give the probability estimations Pr(E) for the tuple $r \in R$ satisfying E.

For instance, for $X, Y \in \upsilon$; $a, b \in R[X]$, we have

$$Pr(X=a) = \frac{1}{\text{card } R[X]} ; \tag{2.1}$$

$$Pr(X \geq a) = \begin{cases} \dfrac{X_M - a}{X_M - X_m}, & X_m \leq a < X_M \\ \dfrac{1}{\text{card } R[X]}, & a = X_M \end{cases} \tag{2.2}$$

$$Pr(X > a) = Pr(X \geq a) - \frac{1}{\text{card } R[X]} \tag{2.3}$$

$$Pr(a \leq X < b) = \begin{cases} \dfrac{b-a}{X_M - X_m}, & X_m \leq a < b < X_M \\ \dfrac{X_M - a}{X_M - X_m} - \dfrac{1}{\text{card } R[X]}, & X_m \leq a < b = X_M \end{cases} \tag{2.4}$$

$$Pr(X=Y) = 0 \qquad \text{if } X_m < X_M < Y_m < Y_M \text{ or } Y_m < Y_M < X_m < X_M \tag{2.5}$$

$$Pr(X=Y) = \frac{1}{\text{card } R[X] \cdot \text{card } R[Y]} \qquad \text{if } X_m < X_M = Y_m < Y_M \text{ or } Y_m < Y_M = X_m < X_M \tag{2.6}$$

$$Pr(X=Y) = \frac{1}{\max\left(\frac{X_M-Y_m}{X_M-X_m}\text{ card } R[X],\ \frac{X_M-Y_m}{Y_M-Y_m}\text{ card } R[Y]\right)} \qquad (2.7)$$

if $X_m<Y_m<X_M<Y_M$

or $Y_m<X_m<Y_M<X_M$

$$Pr(X=Y) = \frac{1}{\max(\text{card } R[X],\ \text{card } R[Y])} \qquad (2.8)$$

if $X_m = Y_m<X_M = Y_M$

(a special case of (2.7)).

Remark 1. To compute Pr(E) for a non simple logical expression, we use the following rules:

a) $Pr(E_1 \wedge E_2) = Pr(E_1)\cdot Pr(E_2)$ where E_1, E_2 are independent.

b) $Pr(E_1 \vee E_2) = Pr(E_1) + Pr(E_2) - Pr(E_1 \wedge E_2)$;

c) $Pr(\neg E_1) = 1 - Pr(E_1)$.

Remark 2. In [2], for the computation of Pr(X=Y), the authors gave only formula (2.8), with the assumption (H1). Of course, when taking attention to the relative positions of the segments $[X_m, X_M]$ and $[Y_m, Y_M]$, this simple formula is not complete.

Remark 3. For other distribution types, the idea of this paper is also useful. One must only considered an appropriate method for computing the probability estimations.

§3. THE MAIN PROBLEM

Given a relation R with the set of attributes $U=\alpha(R)$ and the ranges D_i, $i=\overline{1,k}$. E is a logical expression in R. The selection $T=\sigma_E(R)$ can be obtained by the following algorithm:

$$\left.\begin{array}{l} T:=\emptyset \\ \underline{\text{for}}\ \text{each}\ r\in R\ \underline{\text{do}} \\ \quad \underline{\text{if}}\ \text{test}\ (r,E)\ \underline{\text{then}}\ \text{add}\ r\ \text{to}\ T \end{array}\right\} \qquad (1)$$

The function test (r,E) performs two operations:

i) Replaces the names of attributes $A\in U$ by the values r.A of the tuple $r\in R$.

ii) Computes the obtained logical expression and assigns the result to the function test.

Given a tuple $r\in R$. Denote cost (r,E) the cost paid to perform the function test (r,E). In this section, we always consider that E is a conjunctive logical expression

$$E = \bigwedge_{i=1}^{n} E_i \ .$$

The function test (r,E) can be computed by two different ways:

Way 1: Compute all functions test (r,E_i) and then set

$$\text{test}(r,E) = \bigwedge_{i=1}^{n} \text{test}\ (r,E_i).$$

We have the algorithm:

$$\begin{array}{l} \text{test}\ (r,E):=\underline{\text{true}}\ ; \\ \underline{\text{for}}\ i:=1\ \underline{\text{to}}\ n\ \underline{\text{do}}\ \text{test}\ (r,E)=\text{test}(r,E)\ \wedge \\ \qquad\qquad \wedge\ \text{test}(r,E_i). \end{array} \qquad (2)$$

The one-pass compilers compute often the conjunctive logical expressions in this way, for instance the compiler on the hypothetical computer P-code for language PASCAL-S.

Way 2: It is obvious that if there is some i_o during the computation of test (r,E_i) such that test(r,E_{io})=false then it is possible to conclude immediately that test(r,E)= false and to halt the calculation of test (r,E). This is expressed in the algorithm as follows.

```
Test (r,E): = false
  for i : = 1 to n do
    if ¬ test (r,Ei) then goto L;
                test (r,E) : = true ;      (3)
L:
```

Intuitively, it is easy to see that the method in the algorithm (3) is very natural. (Of course it is better than the algorithm (2).) However, it is very interesting if we know the probabilities of the tuples $r \in R$ satisfying the expressions E_i in R and so we can expect that there exist a best ordering of the subexpressions E_i such that the average probabilistic cost of the algorithm (3) is minimal.

To make clear this idea we do as follows:

At first, basing on the probability estimations $s_i = Pr(E_i)$, $i=\overline{1,n}$ of E_i in R and the costs $c_i = cost(r,E_i)$ $i=\overline{1,n}$ paid to compute the functions test (r,E_i), we can compute the average probabilistic cost of the algorithm (3). Then, analyzing the mathematical expression cost (r,E) represented by c_i, s_i $i=\overline{1,n}$, we try to find the best ordering $\tau=\{\tau_1,\dots,\tau_n\}$– a permutation of $\{1,\dots,n\}$, such that the value of the expression cost (r,E) is minimum.

Note that to compute test (r,E) for a given E and $r \in R$, the replacements in step i) are necessary and the time cost is the same for every $r \in R$.

It is obvious that:

$$\text{cost}\ (r,E_i) = c_i > 0\ \text{(constant)}\ \forall r \in R \qquad (4)$$

Assume that for each E_i, by the rules as in §2 we can define $s_i = Pr(E_i)$, $i=\overline{1,n}$ and the logical subexpressions are independent of each other.

The relation R is partitioned into T and T_i, $i=\overline{1,n}$ as follows:

$$R = T \cup \left(\bigcup_{i=1}^{n} T_i \right)$$

$$T = \sigma_E(R) = \{r \in R / \text{test}\ (r,E) = \underline{\text{true}}\}$$

$$T_i = \left\{ r \in R / \forall j,\ j=\overline{1,i-1}\ \ \text{test}(r,E_j) = \underline{\text{true}},\ \ \text{test}(r,E_i) = \underline{\text{false}} \right\}$$

it is evident that

$$T \cap T_j = \emptyset, j = \overline{1,n}$$

$$T_i \cap T_j = \emptyset, j \neq i\ .$$

Define

$$\rho^*(T) = \prod_{i=1}^{n} s_i,\quad \rho^*(T_i) = \prod_{j=1}^{i-1} s_j(1-s_i),\ i=\overline{2,n}$$

$$\rho^*(T_1) = 1-s_1$$

We have

$$\rho^*(T) + \sum_{i=1}^{n} \rho^*(T_i) = \sum_{i=1}^{n} \prod_{j=1}^{i-1} s_j(1-s_i) + \prod_{j=1}^{n} s_j = 1$$

Indeed, set $h_i = \prod_{j=1}^{i} s_j$, $h_o=1$

$$\rho^*(T_i) = \prod_{j=1}^{i-1} s_j(1-s_i) = h_{i-1} - h_i,\quad i=\overline{2,n}$$

$$\rho^*(T_1) = 1-s_1 = h_o - h_1$$

$$\sum_{i=1}^{n} \rho^*(T_i)+\rho^*(T)= \sum_{i=1}^{n} (h_{i-1}-h_i)+h_n = h_o = 1$$

Moreover in T i.e. for the tuples $r \in R$ for which test(r,E)=<u>true</u>, it is necessary to compute all test (r,E_i), $i=\overline{1,n}$ for the final result of test (r,E), therefore it requires $\sum_{i=1}^{n} c_i$.

In T_i, because test $_i(r,E_i)=$ <u>false</u>, the computation of test(r,E) halts and it takes $\sum_{j=1}^{i} c_j$. By the definition 6, the average probabilistic cost of the algorithm (3) is

$$\overline{\text{cost}_3}(r,E) = \rho^*(T)\text{cost}(r\in T,E)+ \sum_{i=1}^{n} \rho^*(T_i)\text{cost}(r\in T_i,E) =$$

$$= (\sum_{i=1}^{n} c_i) \prod_{i=1}^{n} s_i + \sum_{i=1}^{n} \prod_{j=2}^{i-1} s_j(1-s_i)(\sum_{j=1}^{i} c_j) \qquad (5)$$

Return to the algorithm' (2) computing the function test(r,E), the worst-case cost and the average probabilistic cost are the same. We have:

$$\text{cost}_2(r,E)= \sum_{i=1}^{n} c_i \qquad (6)$$

The following result is obvious.

<u>Proposition 1</u>.

$$\overline{\text{cost}_3}(r,E) \leqslant \text{cost}_2(r,E)$$

where $\overline{\text{cost}_3}(r,E)$ and $\text{cost}_2(r,E)$ are expressed by the formulae (5), (6) respectively.

<u>Proof</u>. It is not difficult to see that:

$$\overline{\text{cost}_3}(r,E) \leqslant (\sum_{i=1}^{n} c_i)\left[\prod_{i=1}^{n} s_i + \sum_{i=1}^{n} \prod_{j=1}^{i-1} s_j(1-s_i)\right] = \sum_{i=1}^{n} c_i =$$

$$= \text{cost}_2(r,E)$$

The above proof shows thatthe algorithm (3) has always the cost less than the cost of algorithm (2).

Now, (5) can be transformed in the following way:

$$\overline{\text{cost}_3}(r,E) = (\sum_{i=1}^{n} c_i) \prod_{i=1}^{n} s_i + \sum_{i=1}^{n} (1-s_i) \prod_{j=1}^{i-1} s_j (\sum_{j=1}^{i} c_j)$$

Set

$$g_i = \sum_{j=1}^{i} c_j, \; g_o = 0$$

$$\overline{\text{cost}_3}(r,E) = g_n h_n + \sum_{i=1}^{n} (h_{i-1} - h_i) g_i =$$

$$= g_n h_n + \sum_{i=1}^{n} h_{i-1} g_i - \sum_{i=1}^{n} h_i g_i =$$

$$= \sum_{i=1}^{n} h_{i-1} g_i - \sum_{i=1}^{n-1} h_i g_i = \sum_{i=1}^{n} h_{i-1} g_i - \sum_{i=1}^{n} h_{i-1} g_{i-1} =$$

$$= \sum_{i=1}^{n} h_{i-1} (g_i - g_{i-1}) = \sum_{i=1}^{n} c_i \prod_{j=1}^{i-1} s_j$$

From here the following problem can be formulated:

Given the numbers $c_i > 0, \; i=\overline{1,n}$

$$1 \geqslant S_i \geqslant 0$$

Find the best permutation $\tau = \{\tau_1, \ldots, \tau_n\}$ of $\{1,\ldots,n\}$ such that

$$A(\tau) = \sum_{i=1}^{n} c_{\tau_i} \prod_{j=1}^{i-1} s_{\tau_j} \longrightarrow \min .$$

If it is possible to find the best permutation τ such that $A(\tau) \longrightarrow \min$, then by the commutativity and the associativity of the logical operator $\wedge$, we have

$$\sigma_E(R) = \sigma_{\bigwedge_{i=1}^{n} E_i}(R) = \sigma_{\bigwedge_{i=1}^{n} E_{\tau_i}}(R) .$$

This transformation should allow us to calculate the function test (r,E) by the algorithm (3) not with the ordering $\{1,\dots,n\}$ of E_i but with the ordering $\{\tau_1,\dots,\tau_n\}$. And so, the algorithm (3) becomes:

```
        Test (r,E) : = false ;
    for  i : = 1 to  n  do
    if ¬ test (r,E_τi) then  go to L ;
    test (r,E) : = true .

    L :
```
(3.1)

For this algorithm, the function test (r,E) can be computed with the average cost

$$\sum_{i=1}^{n} c_{\tau_i} \prod_{j=1}^{i-1} s_{\tau_j} \longrightarrow \min .$$

The following proposition will show the way to find the best τ.

<u>Proposition 2</u>.

Let $s_i > 0, i = \overline{1,n}$, τ, τ' be two permutations of $\{1,\dots,n\}$ whose i_o-th and (i_o+1)-th elements are changed with each other, i.e.

$$\tau'_{io} = \tau_{i_o+1}, \; \tau'_{i_o+1} = \tau_{i_o} .$$

If

$$t_{\tau_{i_o}} = \frac{u_{\tau_{i_o}}}{c_{\tau_{i_o}}} < t_{\tau_{i_o+1}} = \frac{u_{\tau_{i_o+1}}}{c_{\tau_{i_o+1}}}$$

$$u_i = 1-s_i, \quad i=\overline{1,n} .$$

Then

$$A(\tau') < A(\tau)$$

Proof.

$$A(\tau) = \sum_{i=1}^{n} c_{\tau_i} \prod_{j=1}^{i-1} s_{\tau_j}$$

$$A(\tau') = \sum_{i=1}^{n} c_{\tau'_i} \prod_{j=1}^{i-1} s_{\tau'_j}$$

when $i<i_o$

$$c_{\tau'_i} = c_{\tau_i}$$

$$\prod_{j=1}^{i-1} s_{\tau'_j} = \prod_{j=1}^{i-1} s_{\tau_j}$$

when $i=i_o$

$$c_{\tau'_{i_o}} = c_{\tau_{i_o+1}}$$

$$\prod_{j=1}^{i_o-1} s_{\tau'_j} = \prod_{j=1}^{i_o-1} s_{\tau_j}$$

when $i=i_o+1$

$$c_{\tau'_{i_o+1}} = c_{\tau_{i_o}}$$

$$\prod_{j=1}^{i_o} s_{\tau'_j} = \prod_{j=1}^{i_o-1} s_{\tau_j} \cdot s_{\tau_{i_o+1}}$$

when $i>i_o+1$

$$c_{\tau'_i} = c_{\tau_i}$$

$$\prod_{j=1}^{i-1} s_{\tau'_j} = \prod_{j=1}^{i_o-1} s_{\tau'_j} \cdot s_{\tau'_{i_o}} \cdot s_{\tau'_{i_o+1}} \cdot \prod_{j=i_o+2}^{i-1} s_{\tau'_j} =$$

$$= \prod_{j=1}^{i_o-1} s_{\tau_j} \cdot s_{\tau_{i_o+1}} \cdot s_{\tau_{i_o}} \cdot \prod_{j=i_o+2}^{i-1} s_{\tau_j} =$$

$$= \prod_{j=1}^{i-1} s_{\tau_j}$$

$$A(\tau')-A(\tau) = c_{\tau_{i_o+1}} \prod_{j=1}^{i_o-1} s_{\tau_j} + c_{\tau_{i_o}} \prod_{j=1}^{i_o-1} s_{\tau_j} \cdot s_{\tau_{i_o+1}}$$

$$- c_{\tau_{i_o}} \prod_{j=1}^{i_o-1} s_{\tau_j} - c_{\tau_{i_o+1}} \prod_{j=1}^{i_o-1} s_{\tau_j} \cdot s_{\tau_{i_o}}$$

$$= \prod_{j=1}^{i_o-1} s_{\tau_j} (c_{\tau_{i_o+1}} + c_{\tau_{i_o}} s_{\tau_{i_o+1}} - c_{\tau_{i_o}} - c_{\tau_{i_o+1}} s_{\tau_{i_o}})$$

$$= \prod_{j=1}^{i_o-1} s_{\tau_j} (c_{\tau_{i_o+1}} (1-s_{\tau_{i_o}}) - c_{\tau_{i_o}} (1-s_{\tau_{i_o+1}}))$$

$$= \prod_{j=1}^{i_o-1} s_{\tau_j} \cdot c_{\tau_{i_o}} \cdot c_{\tau_{i_o+1}} \left(\frac{1-s_{\tau_{i_o}}}{c_{\tau_{i_o}}} - \frac{1-s_{\tau_{i_o+1}}}{c_{\tau_{i_o+1}}}\right) < 0$$

$\longrightarrow A(\tau') < A(\tau)$.

Proposition 3.

Given the numbers

$$\left.\begin{matrix} 0 \leqslant s_i \leqslant 1 \\ 0 < c_i \end{matrix}\right\} i = \overline{1,n}$$

Set

$$t_i = \frac{1-s_i}{c_i}$$

If $\tau_o = \{1,\dots,n\}$ satisfies $t_i \geqslant t_{i+1}$, $i=\overline{1,n-1}$ then $A(\tau_o)$ is the minimum.

<u>Proof.</u>

Let $\tau=\{\tau_1,\dots,\tau_n\}$ be a permutation of $\{1,\dots,n\}=\tau_o$. We have to prove that $A(\tau_o) \leqslant A(\tau)$.

First we remark that from $\{t_{\tau_i}\} \overset{n}{\underset{i=1}{\otimes}}$ the sequence $\{t_i\}, i=\overline{1,n}$ (corresponding to τ_o) can be obtained by

(i) the bubble sorting algorithm permuting sequentially the adjacent elements $t_{\tau_{i_o}}$, $t_{\tau_{i_o+1}}$ satisfying $t_{\tau_{i_o}} < t_{\tau_{i_o+1}}$ and

(ii) the permutations (if necessary) of the elements with equal values in the obtained sequence.

By proposition 2, if (i) should be carried out then we should obtain τ^1 satisfying $A(\tau^1) < A(\tau)$.

Basing upon the proof of the proposition 2, we have: The permutations of the elements with equal values in the sequence $\{t_{\tau_i^1}\}_{i=1,n}$ do not change the value of A, i.e. $A(\tau_o)=A(\tau^1)$.

From here follows $A(\tau_o)=A(\tau^1) < A(\tau)$. When the step (i) does not take places, it is not difficult to see that $A(\tau_o)=A(\tau)$.

The following algorithm will give the best result for any conjunctive logical expression.

Algorithm Al.

Input: $E = E_1 \wedge \ldots \wedge E_n$.

$$R = \{r : U \longrightarrow \bigcup_{i=1}^{k} D_i \ / \ \forall i \ r(A_i) \in D_i\}$$

Output: $\tau = \{\tau_1, \ldots, \tau_n\}$ is the permutation of $\{1, \ldots, n\}$ such that

$$A(\tau) = \sum_{i=1}^{n} c_{\tau_i} \prod_{j=1}^{i-1} s_{\tau_j} \longrightarrow \min .$$

Method

1) For each i, estimate the probability $Pr(E_i)$ by the formulae in §2 or by the formulae given by the system programmers basing upon the statistical parameters during the manipulation of R.

2) If there exists i_o such that $Pr(E_{i_o}) = 0$, then inform $\sigma_E(R) = \emptyset$.

3) If there exist i_o such that $Pr(E_{i_o}) = 1$ then delete E_{i_o} from E.

More generally, denote $I = \{i_o / S(E_{io}) = 1\}$. Consider

$$E = \bigwedge_{i \in \{1, \ldots, n\} - I} E_i$$

Renumber the expressions E_i in

$$E = \bigwedge_{i \in \{1, \ldots, n\} - I} E_i$$

$$n := n - \text{card } I$$

4) Define c_i, $i=\overline{1,n}$ (In practice, in order to define c_i, we compute the function test (r,E_i) for any tuple and give c_i = cost (r,E_i).

<u>For instance</u>: if

$$E_i = A\Theta B \quad \text{then} \quad c_i = 2a + b$$

$$E_i = A\Theta c\,, \qquad c_i = a + b$$

$$E_i = c\Theta A\,, \qquad c_i = a + b$$

where

a is the cost to bustitute A by r.A;

b is the cost paid to compare two elements in $\bigcup_{i=1}^{k} D_i$.

5) $u_i = 1-s_i$, $i=\overline{1,n}$.

6) $t_i = u_i/c_i$, $i=\overline{1,n}$.

7) Sort $\{t_i\}$ such that $t_i \geqslant t_{i+1}$ $i=\overline{1,n-1}$
Step 7 can be performed by one of the sorting algorithms, in general, of complexity O(nlogn).

8) Print the best ordering obtained $\tau=\{\tau_1,\ldots,\tau_n\}$.

9) Print the value $A(\tau)= \sum_{i=1}^{n} c_{\tau_i} \prod_{j=1}^{i-1} s_{\tau_j}$.

<u>Costing the algorithm A1.</u>

Steps 1,2,3	are performed with the cost	K_1 n
step 4	has the complexity	K_2 n
step 5,6		K_3 n
step 7		K_4 nlogn
step 8,9		K_5 n

The complexity of the algorithm A1 is of

$$Kn + H\,n\log n \approx O(n\log n)$$
$$K = K1 + K2 + K3 + K5$$
$$H = K4 \;.$$

Remark 4.

The proof of the proposition 3 is based on the bubble sorting algorithm of complexity $O(n^2)$ but the step 7 of the algorithm A1 uses any sorting algorithm of complexity $O(n\log n)$. However, there is no matters about the correctness of the algorithm A1.

The algorithm A1 can be implemented without any access to the secondary memory devices containing the file R.

Theorem 1.

Let R be a relation, card R = N, $E = \bigwedge_{i=1}^{n} E_i$, $\tau_o = \{1,\ldots,n\}$ is the best ordering of E_i's i.e.

$$A(\tau_o) = \sum_{i=1}^{n} c_i \prod_{j=1}^{i-1} s_j \longrightarrow \min$$

Then, the cost of the algorithm (1) finding $\sigma_E(R)$ with the function test (r,E) computed by:

1) Algorithm (2) is $C^1 = N.\ \sum_{i=1}^{n} c_i$

2) Algorithm (3) with the best ordering τ_o of E_i's is

$$C^2 = N \cdot \sum_{i=1}^{n} c_i \prod_{j=1}^{i-1} s_j + F(n)$$

where $F(n) = Kn + Hn\log n$ is the cost paid to perform the algorithm A1.

3) Algorithm (3) with an arbitrary ordering $\tau=\{\tau_1,\dots,\tau_n\}$ is

$$C^3 = N \sum_{i=1}^{n} C_{\tau_i} \prod_{j=1}^{i-1} s_{\tau_j}$$

we have the inequalities:

$$C^1 \geqslant C^3$$
$$C^1 \geqslant C^2 \quad \text{with large } N$$
$$C^3 \geqslant C^2 \quad \text{with large } N\ .$$

§4 Extensions

Extension 1. If E is of the form $E=E_1 \vee \dots \vee E_n$ then using the symbols as above and the De Morgan's law

$$\neg(E_1 \vee \dots \vee E_n) = \neg E_1 \wedge \dots \wedge \neg E_n$$

we have: the cost payed to compute test (r,E) is

$$\overline{\text{cost}}\ (r,E) = \sum_{i=1}^{n} c_i \prod_{j=1}^{i-1} s'_j \ \text{ where } s'_j = 1-s_j,\ j=\overline{1,n}$$

Proposition 4.

Given $s_i = Pr(E_i)$ $\quad 0 \leqslant s_i \leqslant 1$

$c_i > 0 \qquad i=\overline{1,n}$

$t_i = s_i/c_i$

If $\tau_o = \{1,\dots,n\}$ satisfies $t_i \geqslant t_{i+1}$, $i=\overline{1,n-1}$ then

$$\text{cost}\ (r,E) = \sum_{i=1}^{n} c_i \prod_{j=1}^{i-1} s'_j \longrightarrow \min.$$

Extension 2.

Algorithm A2.

Input: An arbitrary logical expression E (as defined by def.2).

Output The best ordering of the simple logical sub-expressions of E.

Method.

1) Reduce E to the conjunctive disjunctive normal form

$$r(E) = \bigvee_{i=1}^{n} \bigwedge_{j=1}^{n_i} E_j^i .$$

2) Apply algorithm A1 to

$$E_i = \bigwedge_{j=1}^{n_i} E_j^i$$

to give the best ordering τ^i of E_j^i, $j=\overline{1,n_i}$.

3) Apply the modified algorithm to $\bigvee_{i=1}^{n} E_i$ with $c_i = A(\tau^i)$ and $s_i = Pr(E_i)$ defined by the estimating formulae analogous to one's in §2.

4) Print the best ordering $E = \bigvee_{i=1}^{n} \bigwedge_{j=1}^{n_i} E^{\tau_i}_{\tau_j^i}$.

5) Print the value $C = \sum_{i=1}^{n} c_{\tau_i} \prod_{j=1}^{i-1} s_{\tau_j}$.

CONCLUSION

Independently, our approach is quite near to the Hanani's one [5]. However, our approach seems to be more straightforward, easy for extensions and the complexity analysis of the algorithm proposed is much elaborate.

ACKNOWLEDGEMENT

The authors wish to thank Dr Béla Uhrin and Katalin Fridl for their valuable comments and suggestions.

REFERENCES

[1] Ulmann, J.; Principles of database systems. Computer Science Press. Second edition, 1982.

[2] Adiba,M. and Delobel,C.: Bases de données et systèmes relationnels. Paris, Dunod, 1982.

[3] Steve Talbot: An investigation into logical optimization of relational query languages. The comp. J. Vol. 27, No.4, November 1984.

[4] Matthias Jarke and Jürgen Koch: Query optimization in Database systems. ACM Computing Surveys, Vol.16. No.2, June 1984.

[5] Hanani, M.Z.: An optimal evaluation of Boolean expression in an on line query system. Comm. ACM 20, 5, May 1977.

A relációs kifejezések kiválasztásának optimalizálásáról

J. DEMETROVICS, HO THUAN, NGUEN THANH THUY

Összefoglaló

Legyen $E = \bigwedge_{i=1}^{n} E_i$ egy feltételes kifejezés és R egy reláció. A cikkben egy O(n log n) algoritmust mutatnak be a szerzők, amely a $\sigma_E(R):=\{r\in R/r$ kielégíti az E-t$\}$ mennyiséget /átlagban/ minimális lépésszámban határozza meg /azaz, amely komplexitásának várható értéke minimális/.

Оптимизация выборок из реляциённых выражений.

Й. Деметрович, Хо Тхуан, Нгуен Тханх Тхуи

Резюме

Пустъ $E = \bigwedge_{i=1}^{n} E_i$ естъ условное выражение и R реляция. В статъе показывается O/n logn/ алгоритм который /в среднем/ минимизирует число шагов для нахождения $\sigma/R/: = \{r \in R/\ _r$ исполняет E$\}$.