ON THE SIZE OF THE SEARCH SPACE OF JOIN OPTIMIZATION[*]

Cong-cong Xing
Department of Mathematics and Computer Science
Nicholls State University
Thibodeaux, LA 70310
cmps-cx@nicholls.edu

Bill P. Buckles
EECS Department
Tulane University
New Orleans, LA 70118
buckles@eecs.tulane.edu

ABSTRACT

A fundamental issue when we deal with the analysis of the join optimization problem in database systems is: How large is the search space? This paper presents a complete answer to this question. The result shows that the size of the search space is far beyond exponential, which poses a great challenge for finding an e_cient polynomial time algorithm for solving the join optimization problem. On the other hand, it prompts attempts to prove that the join optimization problem is NP-complete. Our work is the initial step for further study on the join optimization problem.

1. INTRODUCTION

Join optimization [3] is a critical issue in distributed database systems, where the (total) cost of a sequence of join operations over a set of relations depends on the order in which join operations are carried out, but the cost will not be affected if any two relations involved in one join operation are exchanged. Formally, let $R_1, \ldots, R_n$, $n \in \mathbf{N}$, $n \geq 1$ be a sequence of relations, $\bowtie$ be the binary join operation over these relations, and $cost(R_i \bowtie R_j)$ denote the cost of joining any two relations R_i and R_j. We know that $cost(R_i \bowtie R_j) = cost(R_j \bowtie R_i)$ for any $i, j \in \{1, \ldots, n\}$, but we do not know whether $cost((R_i \bowtie R_j) \bowtie R_k) = cost(R_i \bowtie (R_j \bowtie R_k))$ for any $i, j, k \in \{1, \ldots, n\}$. The join optimization problem can be stated as: find the minimum-cost joining sequence (or order) of $R_1, \ldots, R_n$ among all possible ones. For example, when $n = 2$, there is only one joining sequence $R_1 \bowtie R_2$ (or equivalently $R_2 \bowtie R_1$) which clearly has the minimum cost. When $n = 3$, there are 3 possible distinct joining sequences

$$(R_1 \bowtie R_2) \bowtie R_3, \qquad (R_1 \bowtie R_3) \bowtie R_2, \qquad R_1 \bowtie (R_2 \bowtie R_3)$$

in terms of the cost (any other joining sequence has the same cost as one of them due to the "commutativity" of joining cost). A trivial brute-force comparison can determine the winner, with a $O(n)$ time complexity.

Naturally and necessarily, we wonder the cases when n is (sufficiently) large. Can brute-force still be effective when n is large? How many different joining sequences are there for large n? This question leads to the study of the size of the search space and results in this paper.

2. SEARCH SPACE

To facilitate the formulation of the size of the search space, we represent joining sequences graphically as binary trees with some constraints.

Definition 1 Let S be a non-empty finite set. An optimization tree over S, denoted by *opt(S)*, is inductively defined as follows:

- If S is a singleton set $\{a\}$, then *opt(S)* is a single node labelled by a.
- Otherwise, *opt(S)* consists of a root $\bowtie$ and a *set* of two sub-optimization trees *opt(S_1)* and *opt(S_2)*, where $\{S_1, S_2\}$ is a partition of S.

Note that we intentionally require that the subtrees *opt(S_1)* and *opt(S_2)* form a set so that the positions (left and right) of the two subtrees in an optimization tree do not matter. That is, exchanging the two subtrees in an optimization tree does not make any difference. Note also that normal binary trees do not possess this property. We need this property in optimization trees so that they can isomorphically represent the joining sequences.

For small-cardinality finite sets, we can easily enumerate their optimization trees. Figure 1 lists the optimization trees for finite sets with cardinalities ranging from 1 to 4. A careful enumeration shows that the number of optimization trees for a 5-element set is 105! These 105 trees are omitted in Figure 1 for obvious reasons.

The 1-1 correspondence between joining sequences and optimization trees is trivial. For example, the joining sequences $(a \bowtie b) \bowtie c$ and $a \bowtie (b \bowtie c)$ (where a, b, and c are regarded as relations) are represented by the first and the third optimization trees respectively for $S = \{a, b, c\}$ in Figure 1. It is easy to see that the search space size for join optimization for relations $R_1, \ldots ,R_n$ is equal to the number of different optimization trees over a finite set with cardinality n.

It is worth noting that the number we are looking for is not the Catalan number (e.g., [4, 1]) in the literature. Catalan number $c_n = \frac{1}{n+1}\binom{2n}{n}$ can be interpreted, among other ways, as the number of ways of placing parentheses around a string of n+1 characters to form a multiplication expression. For example, given strings *abc* and *abcd*, Catalan number yields $\frac{1}{2+1}\binom{4}{2} = 2$ and $\frac{1}{3+1}\binom{6}{3} = 5$ ways to build a multiplication respectively; but there are 3 and 15 different optimization trees over $\{a, b, c\}$ and $\{a, b, c, d\}$ respectively (Figure 1). It looks like that the number of optimization trees over a

n-element set is greater than the Catalan number (over a string of length n). But we do not know how they are exactly related unless the former is resolved.

3. SOLUTION

We now calculate the number of different optimization trees over a finite set. We first present a recursive solution to the problem and then solve the recursive equation to obtain a closed form formula.

Lemma 1 *Let ϵ_n be the number of different optimization trees over a finite set S with $|S| = n$, $n \geq 1$. Then*

$$\epsilon_1 = 1 \tag{1}$$

$$\epsilon_n = \frac{1}{2}\sum_{i=1}^{n-1}\binom{n}{i}\epsilon_i\epsilon_{n-i} \quad (n > 1) \tag{2}$$

Proof. Case $n = 1$ is trivial. For case $n > 1$, note, by the definition of optimization trees, that each optimization tree over the set S contains two subtrees. One of the subtrees is an optimization tree over a non-empty proper subset of S, say $S_1 \subset S$ with $|S_1| = i$, $1 \leq i \leq n - 1$. The other subtree is an optimization tree over the complement of S_1 with respect to S, $S - S_1$, with $|S - S_1| = n - i$. So the total number of optimization trees over S with one subtree being over S_1 and the other subtree being over $S - S_1$ is $\epsilon_i\epsilon_{n-i}$. Since we have $\binom{n}{i}$choices to construct the i-element set S_1 and i can range from 1 to n - 1, so $\sum_{i=1}^{n-1}\binom{n}{i}\epsilon_i\epsilon_{n-i}$ includes all possible optimization trees over S. Finally, note that we do not distinguish the positions (left and right) of subtrees in optimization trees, so in $\sum_{i=1}^{n-1}\binom{n}{i}\epsilon_i\epsilon_{n-i}$, each optimization tree is counted twice. It is first counted in $\binom{n}{i}\epsilon_i\epsilon_{n-i}$ and then in $\binom{n}{n-i}\epsilon_{n-i}\epsilon_i$. $\binom{n}{i}\epsilon_i\epsilon_{n-i}$ and $\binom{n}{n-i}\epsilon_{n-i}\epsilon_i$ actually denote the number of the same set of optimization trees. Hence, $\epsilon_n = \frac{1}{2}\sum_{i=1}^{n-1}\binom{n}{i}\epsilon_i\epsilon_{n-i}$. □

While the recursive solution for on is useful in computing ϵ_n, it does not directly tell how large ϵ_n is. It is desirable to find a straightforward closed form formula for ϵ_n.

Theorem 1 *Equations (1) and (2) are equivalent to*

$$\epsilon_{n+1} = \frac{n!}{2^n}\binom{2n}{n} \quad (n \geq 0) \tag{3}$$

Proof. The technique of generating functions [2] is used in this proof. From equation (2), we have

$$\begin{aligned} 2\epsilon_n &= \sum_{i=1}^{n-1}\binom{n}{i}\epsilon_i\epsilon_{n-i} \\ &= \sum_{i=1}^{n-1} n!\frac{\epsilon_i}{i!}\frac{\epsilon_{n-i}}{(n-i)!}. \end{aligned}$$

That is,

$$2\frac{\epsilon_n}{n!} \;=\; \sum_{i=1}^{n-1}\frac{\epsilon_i}{i!}\frac{\epsilon_{n-i}}{(n-i)!}.$$

For each i, let $\delta_i = \frac{\epsilon_i}{i!}$, then

$$\begin{aligned} 2\delta_n &= \sum_{i=1}^{n-1}\delta_i\delta_{n-i} \\ &= \delta_1\delta_{n-1}+\delta_2\delta_{n-2}+\cdots+\delta_{n-1}\delta_1. \end{aligned}$$

Let g be a generating function defined as

$$g(x) \;=\; \delta_1 x+\delta_2 x^2+\cdots+\delta_n x^n+\cdots\;. \tag{4}$$

We have

$$\begin{aligned} g^2(x) &= (\delta_1\delta_1)x^2+(\delta_1\delta_2+\delta_2\delta_1)x^3+\cdots+(\delta_1\delta_{n-1}+\cdots+\delta_{n-1}\delta_1)x^n+\cdots \\ &= 2(\delta_2x^2+\delta_3x^3+\cdots+\delta_nx^n+\cdots) \\ &= 2(g(x)-x). \end{aligned}$$

Thus

$$g^2(x) - 2g(x) + 2x = 0.$$

Solving this equation, we have roots $1\pm\sqrt{1-2x}$. We pick the root $1-\sqrt{1-2x}$. (The reason for us to do so will be clear in the next two steps.) Hence

$$g(x) = 1-\sqrt{1-2x}.$$

Let $h(x)=\sqrt{1-2x}$. It is easy to compute the n-th ($n > 1$) derivative $h^{(n)}(x)$ of $h(x)$ at $x = 0$,

$$h^{(n)}(0) = -\prod_{i=2}^{n}(2i-3).$$

So, the Taylor expansion of $g(x)$ is

$$\begin{aligned} g(x) &= 1-[1+(-1)x+\frac{(-1)}{2!}x^2+\cdots+\frac{h^{(n)}(0)}{n!}x^n+\cdots] \\ &= x+\frac{1}{2!}x^2-\cdots-\frac{h^{(n)}(0)}{n!}x^n-\cdots\;. \end{aligned} \tag{5}$$

(We can see, now, that if we pick the root $1+\sqrt{1-2x}$ for $g(x)$, then the lowest-order term of $g(x)$ would be the constant 2, which would contradict the definition of $g(x)$ in equation (4) where there is no constant term.)

Comparing the coefficients of equations (4) and (5), we see

$$\begin{aligned}\delta_n &= -\frac{h^{(n)}(0)}{n!}\\ &= -\frac{1}{n!}(-1)\prod_{i=2}^{n}(2i-3)\\ &= \frac{1}{n!}\prod_{i=2}^{n}(2i-3)\end{aligned}$$

Therefore,

$$\begin{aligned}\epsilon_n &= \delta_n n!\\ &= \prod_{i=2}^{n}(2i-3)\\ &= \frac{[2(n-1)]!}{2^{n-1}(n-1)!}\\ &= \frac{(n-1)!}{2^{n-1}}\binom{2(n-1)}{n-1} \qquad (n\geq 1)\end{aligned}$$

That is,

$$\epsilon_{n+1}=\frac{n!}{2^n}\binom{2n}{n} \qquad (n\geq 0)$$

□

Now that ϵ_n is resolved, from the standpoint of algorithm analysis, we would like to know its asymptotic order. The following corollary sketches an upper bound and a lower bound for ϵ_n.

Corollary 1

$$n! < \epsilon_n < n^n$$

when $n > 7$. That is, $\epsilon_n = O(n^n)$ and $\epsilon_n = \Omega(n!)$.

Proof. From equation (3), we know

$$\begin{aligned}\epsilon_n &= \frac{(n-1)!}{2^{n-1}}\binom{2(n-1)}{n-1} \\ &= \frac{n!}{n2^{n-1}}\binom{2(n-1)}{n-1}.\end{aligned}$$

That is

$$\begin{aligned}\frac{\epsilon_n}{n!} &= \frac{1}{n2^{n-1}}\binom{2(n-1)}{n-1} \\ &= \frac{1}{n2^{n-1}}\cdot\frac{(2n-2)(2n-3)\cdots(n+1)n(n-1)\cdots 1}{(n-1)!(n-1)!} \\ &= \frac{1}{2^{n-1}}\left[\frac{2n-2}{n-1}\cdot\frac{2n-3}{n-2}\cdot\cdots\cdot\frac{n+1}{2}\right]. \qquad (6)\end{aligned}$$

In (6), each term in the bracket is of the form $\frac{2n-i}{n-(i-1)}$ with $2 \le i \le n - 1$. Note that

$$\begin{aligned}\frac{2n-i}{n-(i-1)} &= \frac{2n-2(i-1)+i-2}{n-(i-1)} \\ &= 2+\frac{i-2}{n-(i-1)}.\end{aligned}$$

Since $2 \le i \le n - 1$, so $\frac{i-2}{n-(i-1)} \ge 0$. Therefore

$$\frac{2n-i}{n-(i-1)} \ge 2 \qquad (7)$$

for each i. Note also that $n > 7$, which assures

$$\frac{n+1}{2} > 2^2. \qquad (8)$$

Combining (7) and (8), we have

$$\frac{2n-2}{n-1}\cdot\frac{2n-3}{n-2}\cdot\cdots\cdot\frac{n+1}{2} > 2\cdot 2\cdot\cdots\cdot 2^2 = 2^{n-1},$$

and $\epsilon_n > n!$ follows from (6).

The proof of $\epsilon_n < n^n$ is similar. We briefly describe it. By equation (3), we can manage to get

$$\frac{\epsilon_n}{n^n} = \frac{1}{2^{n-1}} \left[\frac{2n-2}{n} \cdot \frac{2n-3}{n} \cdot \cdots \cdot \frac{n+1}{n} \cdot \frac{1}{n} \right]. \tag{9}$$

Note

$$2 > \frac{2n-2}{n} > \frac{2n-3}{n} > \cdots > \frac{n+1}{n} > \frac{1}{n},$$

therefore

$$\frac{2n-2}{n} \cdot \frac{2n-3}{n} \cdot \cdots \cdot \frac{n+1}{n} \cdot \frac{1}{n} < 2^{n-1},$$

and $\epsilon_n < n^n$ follows immediately from (9). □

4. FINAL REMARKS

The size ϵ_n of the search space is a fundamental issue of the join optimization problem in database systems. Theorem 1 provides a complete solution to this issue and corollary 1 addresses its asymptotic order. ϵ_n and the Catalan number c_n are not on the same level of magnitude as shown by the fact: $n^n > \epsilon_n > n! > c_n > 2^n$ for large n. Regarding the join optimization problem, ϵ_n, because of its prohibitively high order, eliminates brute-force as a feasible search strategy, poses a great challenge for discovering a polynomial time search algorithm, and suggests the possibility of NP-completeness. The solution for ϵ_n in this paper is the basis for further studies on the join optimization problem.

REFERENCES

[1] RobertM. Dickau. *Catalan Numbers*. http://mathforum.org/advanced/robertd /catalan.html, 1996.

[2] Donald E. Knuth. *Fundamental Algorithms, The Art of Computer Programming, Vol. 1*. Addison-Wesley, Reading, MA, 1973.

[3] Dennis Shasha and T. Wang. Optimizing equijoin queries in distributed databases where relations are hash partitioned. *ACM Trans. on Database Systems*, 16:279–308, 1991.

[4] Thomas A. Standish. *Data Structure Techniques*. Addison-Wesley, Reading, MA, 1980.