Constructing Optimal Bushy Processing Trees for Join Queries is NP-hard

(extended abstract)

W. Scheufele* G. Moerkotte

Lehrstuhl für praktische Informatik III
Universität Mannheim
Seminargebäude A5
68131 Mannheim, Germany
email: *ws*|*moer*@pi3.informatik.uni-mannheim.de
phone: +49 621 292 5403 | 5579
fax: +49 621 292 3394

Abstract

We show that constructing optimal bushy processing trees for join queries is NP-hard. More specifically, we show that even the construction of optimal bushy trees for computing the cross product of a set of relations is NP-hard.

1 Introduction

Ever since the invention of relational database systems, query optimization has been an important issue. One of the problems the query optimizer has to deal with is the ordering of joins. Given a set of relations $R_1, \ldots, R_n$ and a set of binary join predicates $p_1, \ldots, p_k$, the *join ordering problem* asks for an operator tree to evaluate the join $\bowtie_{p_1 \wedge \ldots \wedge p_k} (R_1, \ldots, R_n)$, so that the estimated evaluation cost is minimal. An operator tree is a labeled regular binary tree whose leaves represent base relations and whose internal nodes denote binary join operators. Trees where all internal nodes have a leaf as its right child are called *left-deep trees* and otherwise *bushy trees*. A bushy tree and a left-deep tree are shown in Figure 1. The sets $\{R_1, \ldots, R_n\}$ and $\{p_1, \ldots, p_k\}$ define an undirected graph—the

*Research supported by the German Research Association (DFG) under contract Mo 507/6-1.

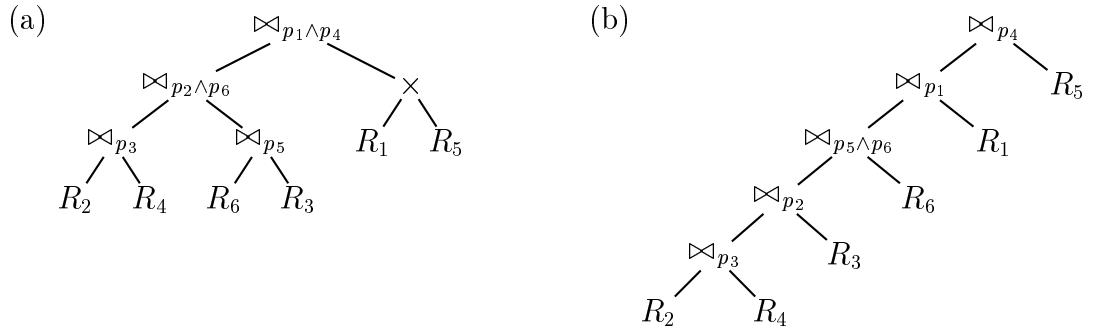


Figure 1: (a) Bushy Tree; (b) Left-deep Tree

join graph. In the join graph two relations are connected by an edge if and only if there exists a join predicate relating them. If the join graph has c connected components then any operator tree for the query contains at least $c - 1$ cross products. Therefore, when we say that an algorithm for the join ordering problem *does not consider cross products*, we mean that the algorithm does not consider operator trees with at least as many cross products as connected components. An example of a join graph is shown in Figure 2.

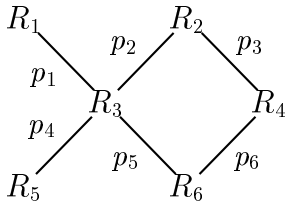


Figure 2: Join Graph

The first approach used to solve the join ordering problem applies dynamic programming [8]. This algorithm uses two heuristics in order to cut down the search space. First, only left-deep trees are considered. Second, no cross products are considered. Nevertheless, the number of alternatives considered can be exponential.

Often, the larger search space where cross products are allowed contains much cheaper processing trees [7]. But then, the dynamic programming approach investigates $n2^{n-1} - n(n+1)/2$ different plans if left-deep trees are considered. Similarly, bushy trees can also lead to cheaper plans. For bushy trees, the number of considered alternatives is even higher, namely $(3^n - 2^{n+1} + 1)/2$ [7].

A problem one is confronted with immediately is whether this exponential run time is inherent to the problem or whether smart polynomial algorithms exist. For constructing optimal left-deep processing trees, some answers have been given so far.

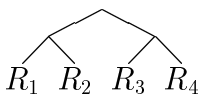
Constructing optimal left-deep trees not containing any cross products for

general join graphs is known to be NP-hard. This has been proven for a special—quite complex—block-wise nested-loop join cost function [3] and for a very simple cost function counting just the number of tuples in the intermediate results [1]. When restricting the join graph to be acyclic, constructing optimal bushy trees for those cost functions which fulfill the ASI property [6] can be done in polynomial time [3, 5]. If cross products are considered, then even if the join graph has the form of a star, the problem of constructing an optimal left-deep tree has been shown to be NP-hard [1].

As far as we know, no such result exists for constructing bushy processing trees. Hence, this paper discusses this question. More specifically, we show that constructing optimal bushy trees for a set of relations whose cross product has to be computed is NP-hard. This contrasts the left-deep case where the optimal left-deep tree can easily be computed by sorting the relations on their sizes. Moreover, since taking the cross product is a very special case in which all join selectivities are set to one, constructing optimal bushy trees for any join problem—independent of the join graph—is NP-hard. Thus, any hope of constructing optimal bushy trees in polynomial time has to be abandonded for whatever join problem at hand. The only possible exceptions are those where no cross products are considered and special join graphs exhibit a polynomial search space. An example are chain queries.

2 Finding optimal bushy trees is NP-hard

We assume that the cross product of n relations $R_1, \ldots, R_n$ has to be computed. This is done by applying a binary cross product operator $\times$ to the relations and intermediate results. An expression that contains all relations exactly once, can be depicted as a regular binary tree, where the intermediate nodes correspond to applications of the cross product operator. For example, consider four relations $R_1, \ldots, R_4$. Then, the expression $(R_1 \times R_2) \times (R_3 \times R_4)$ corresponds to the tree



For each relation R_i, we denote its size by $|R_i| = n_i$. As the cost function we count the number of tuples within the intermediate results. Assuming the relation's sizes $n_1 = 10$, $n_2 = 20$, $n_3 = 5$ and $n_4 = 30$, the cost of the above bushy tree would be $10 * 20 + 5 * 30 + (10 * 20 * 5 * 30) = 30350$. Since the final result is always the same for all bushy trees—the product of all relation sizes—we often neglect it.

First, we need the following lemma.

Lemma 2.1 Let $R_1, \ldots, R_n$ be relations with their according sizes. If $|R_n| > \prod_{i=1,n-1} |R_i|$, then the optimal bushy tree is of the form $X \times R_n$ or $R_n \times X$ where X is a bushy tree containing relations $R_1, \ldots, R_{n-1}$.

The proof of this lemma is quite obvious and hence omitted.

Definition 2.2 (cross product optimization, XR)
The problem of constructing minimal cost bushy trees for taking the cross product of n relations is denoted by *XR*.

In order to prove that *XR* is NP-hard, we need another problem known to be NP-complete for which we can give a polynomial time Turing reduction to *XR*. We have chosen to take the exact cover with 3-sets (*X3C*) as the problem of choice. The next definition recalls this problem which is known to be NP-complete [2].

Definition 2.3 (exact covering with 3-sets, X3C)
Let S be a set with $|S| = 3q$ elements. Further let C be a collection of subsets of S containing three elements each. The following decision problem is called *X3C*: Does there exist a subset C' of C such that every $s \in S$ occurs exactly once in C'?

We are now prepared to prove our claim.

Theorem 2.4 The problem *XR* is NP-hard.

In order to understand the proof, it might be necessary to state the underlying idea explicitly. An optimal bushy tree for a set of relations is as balanced as possible. That is, a bushy tree $(T_1 \times T_2) \times (T_3 \times T_4)$ with subtrees T_i is optimal, if $\text{abs}(|T_1 \times T_2| - |T_3 \times T_4|)$ is minimal and cannot be reduced by exchanging relations from the left to the right subtree. This is not always true, since a left-deep tree can be cheaper even if this criterion is not fulfilled. In order to see this, consider the following counterexample. Let $R_1, \ldots, R_4$ be four relations with sizes $n_1 = 2$, $n_2 = 3$, $n_3 = 4$, and $n_4 = 10$. The optimal "real" bushy tree is $(R_1 \times R_4) \times (R_2 \times R_3)$ with cost $2 * 10 + 3 * 4 = 32$. Its top-level difference is $20 - 12 = 8$. But the left-deep tree $((R_1 \times R_2) \times R_3) \times R_4$ has lower cost $(2 * 3 + 2 * 3 * 4 = 30)$ although it has a higher top-level difference $(24 - 10 = 14)$. Considering our lemma, it becomes clear that it is a good idea to add some big relations at the top to fix the shape of an optimal tree. Further, these additional relations (named T and D in the following proof) are needed to guarantee the existence of a fully balanced and optimal tree.

Proof We prove the claim by reducing *X3C* to *XR*. Let (S, C) with $|S| = 3q$ be an instance of *X3C*. Without loss of generality, we assume that $|C| > q$. Obviously, if $|C| < q$ there exists no solution. If $|C| = q$, the problem can be decided in polynomial time.

We start by coding $X3C$. First, we map every element of S to an odd prime. Let $S = \{s_1, \ldots, s_{3q}\}$, then s_i is mapped to the i-th odd prime. Note that we can apply a sieve method to compute these primes in polynomial time. Subsequently, we identify s_i and p_i.

Every element $c = \{s_{i_1}, s_{i_2}, s_{i_3}\} \in C$ is mapped to the product $s_{i_1} * s_{i_2} * s_{i_3}$. Again, we identify c with its product.

Note that this coding allows to identify uniquely the s_i and c. Each c will now become a relation R of size c. In addition, we need two further relations T and D with sizes $\tilde{T}$ and $\tilde{D}$ defined as follows:

$$\begin{aligned}
\tilde{S} &:= \prod_{s \in S} s \\
\tilde{C} &:= \prod_{c \in C} \prod_{c' \in c} c' \\
\tilde{H} &:= \mathrm{lcm}\left(\tilde{S}, \frac{\tilde{C}}{\tilde{S}}\right) \\
K &:= 2\tilde{C}^2 \\
\tilde{T} &:= \frac{\tilde{H}}{\tilde{S}} K \\
\tilde{D} &:= \frac{\tilde{H}\tilde{S}}{\tilde{C}} K
\end{aligned}$$

where $lcm(x, y)$ denotes the least common multiple of the numbers x and y.

Without loss of generality, we assume that $\tilde{C} \equiv 0 \pmod{\tilde{S}}$. If this is not the case, obviously no solution to *X3C* exists.

We will now show that

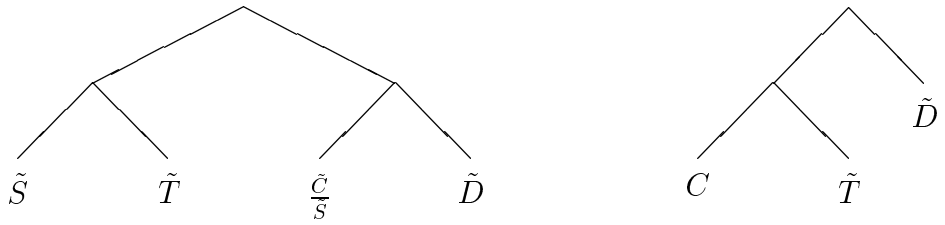
> *there exists a solution to X3C if and only if the optimal solution has the form* $(A \times T) \times (B \times D)$ *where* A *and* B *are subtrees and* T *and* D *are the special relations from above. Further,* $|A| = \tilde{S}$ *and* $|B| = \frac{\tilde{C}}{\tilde{S}}$ *must hold.*

Of course, the above must be seen with respect to possible interchanges of sibling subtrees which does not result in any cost changes.

Clearly, if there is no solution to the X3C problem, no bushy tree with these properties exists. Hence, it remains to prove that, if X3C has a solution, then the above bushy tree is optimal.

Within the following trees, we use the sizes of the intermediate nodes or relation sizes to denote the corresponding subtrees and relations. To proof the remaining thesis, we distinguish three cases. Within the first case, we compare our (to be shown) optimal tree with two left-deep trees. Then, we consider the case where both T and D occur in either the left or the right part of a bushy tree. Last, we assume one part contains T and the other part contains D.

For the first case, the left tree of the following figure must be cheaper than the right left-deep tree.



As mentioned, the tree shows only the sizes of the missing subtrees. If some $C' \subseteq C$ is a solution to *X3C*, then it must have a total size $\tilde{C'} = \tilde{S}$.

Note that we need not to consider any other left-deep trees except where T and D are exchanged. This is due to the fact that the sizes of these relations by far exceed the C. (Compare with the above lemma.)

The following is a sequence of inequalities which hold also if T and D are exchanged in the left tree of the above figure. We have to proof that

$$\tilde{S}\tilde{T} + \frac{\tilde{C}}{\tilde{S}}\tilde{D} + cost(C') + cost(C \setminus C') \quad < \quad \tilde{C}\min(\tilde{D}, \tilde{T}) + cost(C)$$

Obviously,

$$cost(C) \quad \geq \tilde{C}$$

and

$$cost(C') \leq \frac{\tilde{C}}{2}, \qquad cost(C \setminus C') \leq \frac{\tilde{C}}{2}$$

Hence,

$$cost(C') + cost(C \setminus C') \quad < \quad cost(C)$$

Further,

$$\begin{array}{rrl} & 2\tilde{H}K & < \quad \tilde{C}\frac{\tilde{H}}{\tilde{S}}K \\ \Longleftarrow & & \\ & 2 & < \quad \frac{\tilde{C}}{\tilde{S}} \end{array}$$

and

$$\begin{array}{rrcl} & 2\tilde{H}K & < & \tilde{C}\frac{\tilde{H}\tilde{S}}{\tilde{C}}K \\ \Longleftarrow & & & \\ & 2 & < & \tilde{S} \end{array}$$

also hold. This completes the first case.

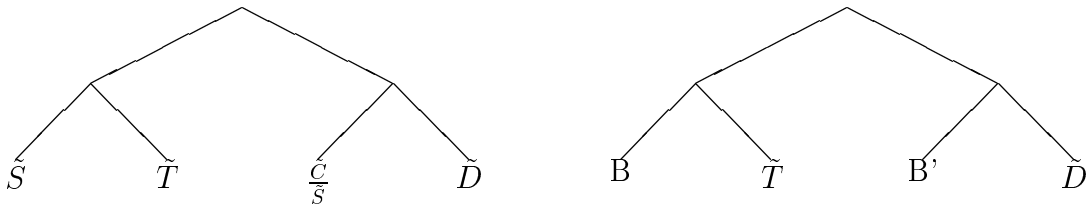
In order to follow the inequalities note that the cost of computing a bushy tree never exceeds twice the size of its outcome, if the relation sizes are greater than two, which is the case here.

If we assume T and D to be contained in either the right or the left subtree, we get the following cost estimations:

$$\begin{array}{rrcl} & \tilde{S}\tilde{T} + \frac{\tilde{C}}{\tilde{S}}\tilde{D} + cost(C') + cost(C \setminus C') & < & \tilde{T}\tilde{D} \\ \Longleftarrow & & & \\ & 2\tilde{H}K + \tilde{C} & < & \frac{\tilde{H}K}{\tilde{C}}\tilde{H}K \\ \Longleftarrow & & & \\ & 2 + \frac{\tilde{C}}{\tilde{H}K} & < & \frac{1}{\tilde{C}}\tilde{H}K \\ \Longleftarrow & & & \\ & 1 + \frac{1}{\tilde{H}\tilde{C}} & < & \tilde{H}\tilde{C} \end{array}$$

Again, the last inequality is obvious. This completes the second case.

Now consider the last case, where T and D occur in different subtrees.



Denote the size of the result of B by $\tilde{B}$ and the size of the result of B' by $\tilde{B}'$. Further, note that B and B' arize from S and $\frac{C}{S}$ by exchanging relations within the latter two. This gives us

$$\begin{array}{rrcl} & 2\tilde{H}K + cost(C') + cost(C \setminus C') & < & \tilde{B}\tilde{T} + \tilde{B}'\tilde{D} + cost(B) + cost(B') \\ \Longleftarrow & & & \\ & 2\tilde{H}K + cost(C') + cost(C \setminus C') & < & \tilde{B}\tilde{T} + \tilde{B}'\tilde{D} \\ \Longleftarrow & & & \\ & 2\tilde{H}K + \tilde{C} & < & b\tilde{S}\tilde{T} + \frac{1}{b}\tilde{D}\frac{\tilde{C}}{\tilde{S}} \\ \Longleftarrow & & & \\ & 2\tilde{H}K + \tilde{C} & < & (b + \frac{1}{b})\tilde{H}K \end{array}$$

where b is $\frac{\tilde{B}}{\tilde{S}}$.

Since all relation sizes are odd primes, and we assume that the right tree is different from our optimal tree, $\tilde{S}$ and $\tilde{B}$ must differ by at least 2. Hence, either $b \geq \frac{\tilde{S}+2}{\tilde{S}}$ or $0 < b \leq \frac{\tilde{S}}{\tilde{S}+2}$. Since the function $f(x) = x + \frac{1}{x}$ has exactly one minimum at $x = 1$, and is monotonously decreasing to the left of $x = 1$ and monotonously increasing to the right of $x = 1$, we have:

$$
\begin{array}{rcl}
\tilde{C} & < & \left(\frac{\tilde{S}+2}{2} + \frac{\tilde{S}}{\tilde{S}+2} - 2\right)\tilde{H}K \\
\Longleftarrow & & \\
4\tilde{C} & < & \frac{(\tilde{S}+2)^2+\tilde{S}^2-2\tilde{S}(\tilde{S}+2)}{\tilde{S}(\tilde{S}+2)}\tilde{H}K \\
\Longleftarrow & & \\
4\tilde{C} & < & \frac{\tilde{S}^2+4\tilde{S}+4+\tilde{S}^2-2\tilde{S}^2-4\tilde{S}}{\tilde{S}(\tilde{S}+2)}\tilde{H}K \\
\Longleftarrow & & \\
4\tilde{C} & < & \frac{4}{\tilde{S}(\tilde{S}+2)}\tilde{H}K \\
\Longleftarrow & & \\
\tilde{C} & < & \frac{2}{\tilde{S}(\tilde{S}+2)}\tilde{H}\tilde{C}^2 \\
\Longleftarrow & & \\
1 & < & \frac{2}{\tilde{S}(\tilde{S}+2)}\tilde{H}\tilde{C}
\end{array}
$$

The last inequality holds since $\tilde{H} \geq \tilde{S}$ and $\tilde{C} \geq \tilde{S} + 2$. This completes the proof.

□

The next corollary follows immediately from the theorem.

Corollary 2.5 Constructing optimal bushy trees for a given join graph is —independent of its form—NP-hard

Whereas taking the cross product of vast amounts of relations is not the most serious practical problem, joining a high number of relations is a problem in many applications. This corollary unfortunately indicates that there is no hope of finding a polynomial algorithm to solve this problem.

3 Conclusion

Since there is no hope of a polynomial algorithm for constructing optimal bushy trees, the only chance is to develop heuristics or to apply probabilistic optimization procedures. Concerning the latter, Ioannidis and Kang [4] make some important observations. In fact, they conclude that for probabilistic algorithms producing good bushy trees seems to be easier than constructing good left-deep trees. If this observation also holds for heuristics to be developed, than our result

may not be as bad as it seems to be at first sight. There is some evidence that this is the case. Even simple greedy algorithms seem to perform quite well [9]. What is needed in the future are careful studies concerning heuristics for the construction of good bushy trees. Only after, the builders of query optimizers can trust them.

Another way for research is the construction of very fast dynamic programming algorithms and possible pruning heuristics. The work by Vance and Maier is a good step in this direction [10].

Acknowledgement. We thank Bennet Vance for many fruitful hints to make the paper much more readable.

References

[1] S. Cluet and G. Moerkotte. On the complexity of generating optimal left-deep processing trees with cross products. In *Proc. Int. Conf. on Database Theory (ICDT)*, pages 54–67, 1995.

[2] M. R. Garey and D. S. Johnson. *Computers and Intractability: a Guide to the Theory of NP-Completeness.* Freeman, San Francisco, 1979.

[3] T. Ibaraki and T. Kameda. Optimal nesting for computing n-relational joins. *ACM Trans. on Database Systems*, 9(3):482–502, 1984.

[4] Y. E. Ioannidis and Y. C. Kang. Left-deep vs. bushy trees: An analysis of strategy spaces and its implications for query optimization. In *Proc. of the ACM SIGMOD Conf. on Management of Data*, pages 168–177, 1991.

[5] R. Krishnamurthy, H. Boral, and C. Zaniolo. Optimization of nonrecursive queries. In *Proc. Int. Conf. on Very Large Data Bases (VLDB)*, pages 128–137, 1986.

[6] C. Monma and J. Sidney. Sequencing with series-parallel precedence constraints. *Math. Oper. Res.*, 4:215–224, 1979.

[7] K. Ono and G. M. Lohman. Measuring the complexity of join enumeration in query optimization. In *Proc. Int. Conf. on Very Large Data Bases (VLDB)*, pages 314–325, 1990.

[8] P. G. Selinger, M. M. Astrahan, D. D. Chamberlin, R. A. Lorie, and T. G. Price. Access path selection in a relational database management system. In *Proc. of the ACM SIGMOD Conf. on Management of Data*, pages 23–34, 1979.

[9] E. J. Shekita, K.-L. Tan, and H. C. Young. Multi-join optimization for symmetric multiprocessors. In *Proc. Int. Conf. on Very Large Data Bases (VLDB)*, pages 479–492, 1993.

[10] B. Vance and D. Maier. Rapid bushy join-order optimization with cartesian products. In *Proc. of the ACM SIGMOD Conf. on Management of Data*, pages 35–46, Toronto, Canada, 1996.