Centrum voor Wiskunde en Informatica
REPORT*RAPPORT*

Probabilistic Bottom-up Join Order Selection --- Breaking the Curse of NP-completeness

F. Waas, J. Pellenkoft

Information Systems (INS)

INS-R9906 April 30, 1999

Report INS-R9906
ISSN 1386-3681

CWI
P.O. Box 94079
1090 GB Amsterdam
The Netherlands

Probabilistic Bottom-up Join Order Selection — Breaking the Curse of NP-completeness

Florian Waas Arjan Pellenkoft

CWI
P.O.Box 94079, 1090 GB Amsterdam, The Netherlands
arjan@cwi.nl

ABSTRACT

Join-ordering is known to be NP-complete and therefore a variety of heuristics have been devised to tackle large queries which are considered computational intractable otherwise. However, practitioners often point out that typical problem instances are not difficult to optimize at all.

In this paper we address that seeming discrepancy. We present a probabilistic bottom-up join-ordering technique that is distinguished by the high-quality results achieved, the extremely short running time and—most notable—its independence of the search space's size. The subsequent thorough analysis of the algorithm's principle confirm our experimental results.

1991 Computing Reviews Classification System: [H.2.4] Query Processing
Keywords and Phrases: randomized query optimization
Note: Funded by the HPCN/IMPACT project.

1. Introduction

Join-ordering is one of the most persistent problems in query optimization. Ibaraki and Kameda proved its NP-completeness for the restricted case of linear execution orders [6], and recently Scheufele and Moerkotte established the proof for the general case [15]. These results clearly indicate that it is very unlikely to find an algorithm that solves the problem in polynomial time.

However, these facts are contrasted with the experiences made by practitioners in the development of commercial database systems. They found the optimization of even large queries to be not as difficult a problem as the theoretical results may suggest. Similar observations were also reported on a series of well-known NP-complete problems—including k-colorability of graphs and k-SAT—where typical cases are *easy* to solve [21].

In this paper we investigate this seeming discrepancy for the join ordering. We present a technique that overcomes the major hurdle posed by NP-completeness as it is independent of the problem's size. Based on a mapping of permutations of the query's predicate, we devise a random sampling which makes use of the favorable ratio of good to poor solutions. In contrast to transformation-based algorithms like *Iterative Improvement* or *Simulated Annealing* which navigate through the search space state by state [10, 19, 17], no running time is wasted

to escape local minima and make up for poor intermediate results. Hence, the technique presented converges quicker and delivers more stable results.

For a better understanding of both the problems peculiarities and the algorithm, we carefully analyze the underlying cost distributions. The experimental evidence we provide, shows that cost distributions are limited in shape to what is best approximated with Gamma distributions. By abstracting the cost model step by step finally arriving at a bare Cartesian model, we show that these shapes are caused by the tree structure of the query plan and common to all cost models.

We conclude that NP-completeness alone is not sufficient a criterion to describe the difficulty of join-ordering problems as it describes only the potential worst case complexity.

Road-Map. In Section 2 the problem together with some fundamental considerations about quality measures is outlined briefly. The optimization algorithm and its quantitative assessment are presented in Section 3 followed by the analysis of costing techniques and cost distributions in Section 4.

2. Preliminaries

Since the join-ordering problem has been discussed in detail in previous work we give only a short outline of the basic setting here. More detailed descriptions can be found e.g. in [16, 19, 9, 15, 17]. Given a join query, a query plan is a binary tree where each inner node corresponds to a predicate of the query; the leaves correspond to the base relations. Each such query plan is of certain costs, computed according to a cost model[1]. Both components together make up the join-ordering problem of finding the query plan with the least costs.

An important point often neglected when discussing query optimization techniques is the accuracy of the costing. As pointed out in [7], result size estimate errors propagate exponentially through the query plan, rendering comparisons of plans whose costs differ by a few percent only, meaningless. Hence, a sound demand for comparing plans is to respect the *resolution* of the cost computation.

Therefore, we need a quality measure based on the actual costs value that is abstracting yet meaningful. In [18], Swami proposed a possible classification. Plans are divided into three groups: *good*, *acceptable*, and *bad* query plans. Plans are considered *good* if they have costs below twice the minimal costs c_{min}, *acceptable* if they are no more expensive than 10 times c_{min}, and *bad* otherwise.

However, this schema suffers from the severe drawback to be not invariant under additive translation. Consider the two case of Figure 1 that two queries have very similar shaped cost distributions, i.e. the distribution of costs for all query plans that answer the query, are of similar shape. In this example we assume the distribution to be an exponential distribution for both queries, i.e. $\phi(t) = e^{-t}$. The only difference between the two is a shift of the x values.

[1] We will discuss different models and their effects in Section 4

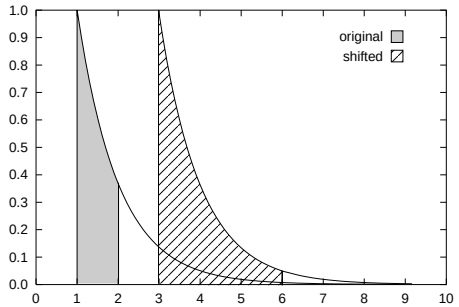


Figure 1: c_{min}–Classification

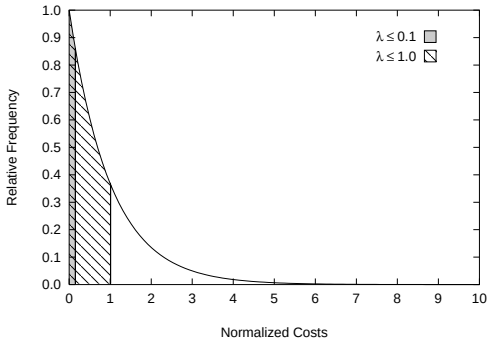


Figure 2: λ–Classification

Let us assume that for the original distribution the cost of the cheapest plan is $c_{min} = 1$ and that the average cost of a plan is $c_\mu = 2c_{min}$. The ratio of good plans in the search space, i.e. plans with costs below $c_{good} (= 2c_{min} = c_\mu)$, computes to 0.63

We would expect the ratio to keep being this way as long as the distributions are shaped similarly no matter what the actual cost values are. The previous classification, however, falls short of this invariance.

Translating the original distribution by adding a factor $2c_{min}$ to the costs, yields $c'_{min} = 3c_{min}$ and $c'_{good} = 2c'_{min} = 6c_{min}$ for the shifted cost distribution. Now, the ratio of good plans increases to 0.95!

Although the distribution stayed the same—the whole range of costs did not change either—the ration of good plans increased by about 50%. In Figure 1 the area of good plans is shaded and hatched, respectively. For the original distribution, the interval $[1, 2]$ comprises the good plans, whereas for the shifted, the whole interval $[3, 6]$ is classified good.

Clearly, the cause for the insufficient valuing is that only one single reference point, namely c_{min}, is taken into account. To overcome this drawback, we classify plans with respect to the two parameters c_{min} and c_μ. We shall denote the quality of a plan q by its normalized costs

$$\lambda(q) = \frac{C(q) - c_{min}}{c_\mu - c_{min}}.$$

This measure obviously is translation invariant. For the optimum, the normalized costs equal 0 while $\lambda(q)$ is 1 for plans of average costs. Plans above c_μ have normalized costs greater than 1, accordingly. In principle, the maximal cost value could also serve as a reference point, however, incorporating c_μ into the quality measure links it automatically to the particular distribution. In Figure 2, the areas of plans with $\lambda(q) \leq 0.1$ and $\lambda(q) \leq 1.0$ are shown for the same distribution as before.

In our experience high quality plans show a λ of less than 0.1, although greater values are justified with respect to large join queries. Hence, the optimization goal we are aiming at is to find a plan with λ below 0.1. In Figure 2 this target cost range is shaded.

Algorithm QuickPick
Input $G(V, E)$ join graph
Output q_{best} best query plan found

```
r ← ∞                                   // initialize lowest costs so far
E' ← E
q ← G'(V, ∅)                            // initialize query plan
repeat
    choose e ∈ E'                       // random edge selection
    E' ← E' \ {e}
    AddJoin(q, e)
    if E' = ∅ or c(q) > r do            // either plan complete or costs exceeded
        if c(q) < r do                  // check for new best plan
            q_best ← q
            r ← c(q)
        done
        E' ← E
        q ← G'(V, ∅)                    // reset query plan
    done
until stopping criterion fulfilled
return q_best
```

Figure 3: Algorithm QuickPick

3. Probabilistic Bottom-up Join Order Selection

Every permutation of a join query's predicates can be mapped to a query plan as follows. For each new predicate we add a join operator to the tree and if not present yet we also add the base tables required. In case, both of the predicate's join partner are already present, i.e. already connected by a previous join, we add the predicate to this very join. Obviously, this mapping works for arbitrarily shaped query graphs without any restrictions.

3.1 The Algorithm

Based on this mapping a random sampling can be implemented by generating random permutations of the predicates. In Figure 3 an outline of our algorithm, called QuickPick is given. we choose predicates—i.e. edges from the associated query graph—randomly one by one and add them to the query plan. After each addition, we compute the costs of the partial plan so far. We make use of the fact that cost computation proceeds also bottom-up and therefore can be done simultaneously with the generation of the plan. If the partial costs exceed the costs of the best plan found so far, the current plan is discarded and a new permutation is generated.

This procedure is repeated until a certain stopping criterion is fulfilled.

3.2 Quantitative Assessment

The experiments were carried out with a prototypical implementation of an optimizer framework that comprises also transformation-based optimization

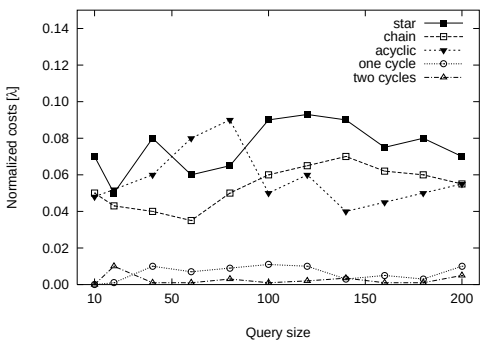


Figure 4: Result quality

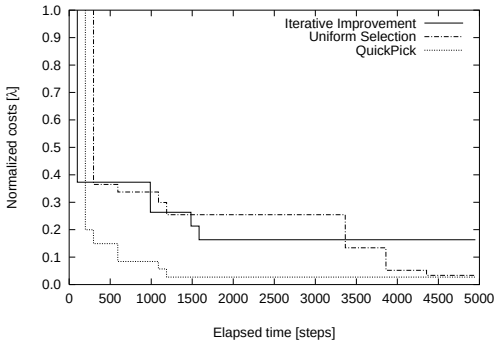


Figure 5: Convergence

techniques and enumeration methods. The platform used is a SGI/Origin2000 250MHz.

The cost model we use is I/O-based and comparable to the techniques described in [11, 17]. Both catalogs and queries were generated at random in analogy to [9], an issue we will discuss more detailed in Section 4.

For each query we first determine the cost distribution by taking a uniform sample —all plans have the same probability of being selected— from the search space. Such a uniform selection can be obtained by performing a long random walk on the search space [9]. In our experiments we used a sample of 10^6 plans.

From this very expensive analysis, we gain an approximation of the minimal cost value as well as the mean of the distribution enabling a later classification in terms of λ.

In Figure 4 the optimization results for 4 query suites of different query graph topology are shown. Every query was optimized with a limit of 5000 steps and repeated 50 times to also assess the stability of the results.

In all experiments QuickPick found results of high quality ($\lambda \leq 0.1$) reaching the resolution of the cost model (cf. Sec. 2). Two points become evident from the plots: 1) obviously, the "difficulty" of the problem is constant for the whole range of query sizes, and 2) cyclic queries appear to be easier to optimize than acyclic ones. We attribute this to a larger cost range and the stronger restriction of the additional predicates. A trend we found continuing for greater number of cycles as well.

The deviation of the single experiments as a measure of the stability was below any significant value, through-out all experiments.

In Figure 5 the convergence behavior of QuickPick is compared to Iterative Improvement and Uniform Selection, for the restricted case of an acyclic query. The results for arbitrary queries are very similar, however, a comparison with Uniform Selection is only possible for the acyclic case due to the restrictions of its applicability.

On the y-axis, the normalized costs are given, the number of optimization steps on the x-axis. All three algorithms eventually find plans of high quality, however, QuickPick converges much quicker than any of the others, finding its best plan in step 1228.

4. Cost distributions

QuickPick is able to find high quality results in a very short time with enormous stability because it relies only on the cost distribution in the search space. Memoryless, it selects trees independent of the result achieved in the previous attempt. This raises the question whether cost distributions are always of favorable shape and if so, why.

In this section we analyze the principles underlying costing techniques for join queries to substantiate the claim that cost distributions are limited to certain shapes. First of all, we identify facts query representations and costing techniques have in common.

4.1 Observations

Query plans are rooted binary trees where each inner node corresponds to a unary or binary relational algebraic operator. Given an input of size n, the output result size is in $O(n)$ for unary and $O(n^2)$ for binary operators. Due to the nature of relational algebraic operators the work that has to be done is in $O(inputsize) + O(outputsize)$ which in turn is in $O(outputsize)$ in both cases. Thus all reasonable cost functions for such an operator will be of the same category. The costs of all operators are computed bottom-up observing the dependencies between them. Finally, the single per-operator costs are summed up.

Cost models used in todays applications differ not only in the precision—some incorporate elapsed CPU time and memory management overhead while others focus on the bare I/O costs—but may be accommodated to special requirements of the processing environment the queries are run on afterwards. A typical example is the granularity of data passed between operators. Some query engines process single tuples in order to exploit pipelining effects while others use bulk processing to increase the throughput.

Despite those differences, standard cost models have several important points in common:

1. The cost computation is done per operator, observing the data dependencies along the binary tree structure. The total cost value is the sum of the per-operator costs.

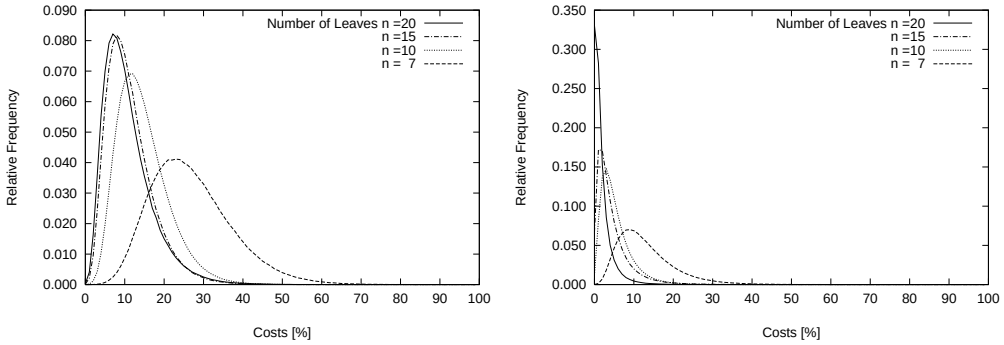
2. The cost function establishes a partial order on the space of query plans.

3. For a pair of query plans t and t', different cost functions C_1 and C_2 do not define the same relation in general, however, for plans with extremal costs we often observe:

$$C_1(t) << C_1(t') \Rightarrow C_2(t) << C_2(t')$$

4. The majority of query plans have costs lower than the mean μ_c [8]. Moreover, the distributions found bear strong resemblance with the Gamma distribution having shape parameters between 1 and 2.

While points 1 through 3 are more or less what we should expect, the fourth needs special attention. This effect was first spotted by Ioannidis and Kang in



a) $\sigma = 0.2\mu$ b) $\sigma = 0.4\mu$

Figure 6: Cost distributions for the binary tree cost model

[9] but also reported on by Steinbrunn et. al. in [17], not explicitly mentioned though. Experiments with different cost models showed that this effect does not occur reliably for queries of size smaller than approximately 10.

To isolate the influential parameters that cause this effect we conducted experiments using different cost models including the ones proposed in [3],[11], and [9] as well as the one used in DBS3 [1]. We reduced the number of parameters to the cost function gradually, finally arriving at the Cartesian model, the simplest possible model where all binary operators are abstracted as Cartesian products. Throughout this process the phenomenon could be observed changing only in extent, but not in quality (cf. [9]).

The following experiment provides deeper insight by abandoning even queries and catalogs, the last remaining imponderabilities.

4.2 Beyond Queries and Catalogs

To examine the nature of cost computation along tree structures we devise a cost model for mere binary trees: Given a binary tree t the costs of an inner node v are determined as $C(v) = C(v_l) \cdot C(v_r)$ where v_l and v_r denote its left and right son, respectively. In case v is a leaf, the costs amount to $C(v) = Y_v$ where Y_v are independent identically distributed random variables, widely used in statistics. The total costs of t compute to $C(t) = \sum_{v \in t} C(v)$.

For a given number of leaves n, we generate all non-isomorphic binary trees, i.e. trees that are not isomorphic under commutative exchange of subtrees. For every such tree we take a sample of 1000 cost computations, that is, we generate 1000 vectors of random values Y_i according to a certain distribution and compute the costs of the tree for each vector. For the implementation of the Y_i we used various standard distributions. The differences, however, turned out to be of no significance. Thus, we present only the results for Normal distributed values with deviation σ, here. Further experiments can be found in Appendix 1.

Figure 6 shows two samples with a different ratio of mean μ to deviation σ. The abscissa corresponds to the complete cost range of an experiment and the frequency of each cost value is plotted against the ordinate. Note, the x-axis is always relative to the particular experiment, i.e. every curve has non-zero

y-value for x being 0% and 100%. In each plot the experimentally determined distributions for $n = 7, 10, 15, 20$ for the same mean and deviation for Y_i are shown. For queries of smaller size the distribution is not that compact as the according search space contains only few elements. Due to the exponentially growing number of trees no assessment is possible for larger n. However, several distinct tendencies are evident and appear to extrapolate.

Firstly, the cost distribution is not of an arbitrary shape but shows a concentration in dependency of the underlying parameters. Secondly, in each experiment, the majority of plans have costs lower than the mean of the over all distribution approving proposition 4. Finally, the distribution shifts to the left with both increasing number of joins and increasing deviation of the Y_i.

What do these results now imply for standard cost models? All of them use the mechanism we scrutinized in this section, that is, their resulting cost distributions are *modulations* of the ones above. Differences occur because of the obviously coarse simplification by Cartesian products and with other distributions realized by the catalog and the query structure. Furthermore, not all non-isomorphic tree shapes maybe valid for a given join graph. However, according to our experience, these modulations do not change the characteristics as pointed out above. Finally, σ, as the main parameter responsible for the shifting, can be interpreted as the variance of the catalog. Similar to our binary tree model, conventional cost models show the same shifting behavior for high and low catalog variance [9].

4.3 Uniform Sampling

With the results obtained so far, we can put random sampling on solid formal grounds and compute probabilities for a successful search in dependency of the running time invested.

Let X be the random variable

$$X := \textit{costs of a query plan chosen at uniform probability.}$$

The probability to obtain a plan of costs lower than x under a cost distribution[2] ξ is

$$P(X \leq x) = \int_0^x \xi(t)dt.$$

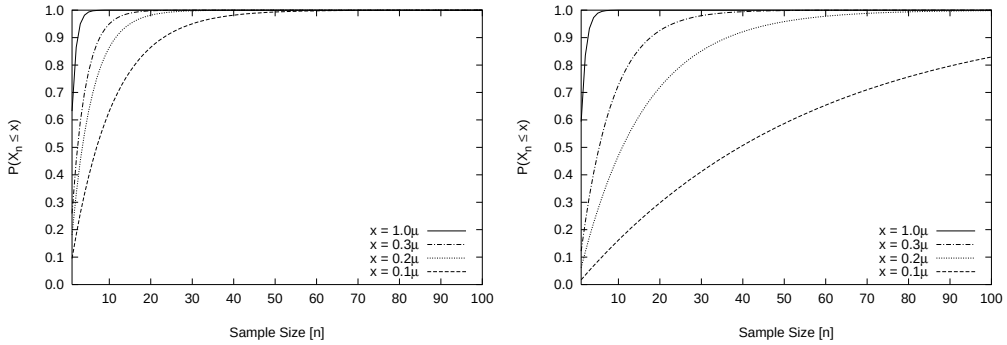
We assume that ξ is already translated in the way that c_{min} equals zero. Let X_n be the random variable

$$X_n := \textit{lowest costs in a sample of } n \textit{ plans chosen at uniform probability.}$$

Obviously, the following holds:

$$P(X_n \leq x) = 1 - \big[P(X > x)\big]^n = 1 - \big[\int_x^\infty \xi(t)dt\big]^n$$

[2]For the remainder of this section, we assume that the query is of sufficient size, so that the cost distributions can be well approximated with a continuous function (cf. Sec. 4.2).



a) $\nu = 1.0$ b) $\nu = 2.0$

Figure 7: Probability to select plan with costs below certain threshold

	$\lambda \leq 0.1$	$\lambda \leq 0.2$	$\lambda \leq 0.3$	$\lambda \leq 1.0$
$\nu = 1.0$	47	24	16	5
$\nu = 2.0$	982	261	123	16

Table 1: Sample size needed for $P(X_n \leq x) \geq 0.99$

To facilitate the formal treatment of the actual distribution we abstract them by Gamma distributions with shape parameter ν between 1 and 2, according to Observation 4. The Gamma distribution, given by

$$P(\nu, x) = \frac{1}{, (\nu)} \int_0^x e^{-t} t^{\nu-1} dt$$

coincides with the exponential distribution for $\nu = 1$, corresponding to the case of high variance catalogs (see Fig. 6b). Conversely, for $\nu = 2$ we obtain a distribution that corresponds to the case of low variance (see Fig. 6a). Other cases relate to a ν between those two values.

In Figure 7, the probability $P(X_n \leq x)$ is shown for various x. The sample size is given on the abscissa and the probability is plotted against the ordinate. As both diagrams show, finding a plan better than average ($\lambda \leq 1.0$) is almost certainly achieved by a sample of only as many as 10 plans. For $\nu = 1$ the probability to obtain a plan with $\lambda \leq 0.2$ within a sample of size 20 is already beyond 0.95. For a sample larger than 47 plans, the probability for plans better than $\lambda \leq 0.1$ is higher than 0.99 (cf. Fig. 7a). As Figure 7b shows, larger sample sizes are needed to achieve the same quality in case of $\nu = 2$. In particular, to reach below $\lambda \leq 0.1$ with a probability greater than 0.99 requires n to be at least 982. Table 1 shows the the necessary values of n to achieve $P(X_n \leq x) \geq 0.99$ are depicted. Note, those figures are by far smaller than the widely accepted limits used for transformation-based probabilistic optimization or even genetic algorithms.

4.4 Sampling with QuickPick

Galindo-Legaria et. al. pointed out that *uniform* random sampling for cyclic query graphs appears to be much more difficult than for acyclic graphs [5] and no method is known for it, so far. Typical applications and benchmark suites like the TPC-D however, demand processing cyclic join graphs. But also for the acyclic graphs, the technique proposed in [4] is—though polynomial—time consuming. QuickPick overcomes both disadvantages at the expense of uniformity. In this section, we examine this biased cost distributions as to how they related to the original, unbiased.

QuickPick generates complete plans only in case of a new record and so we need to modify the algorithm to build complete plans no matter their costs. In Figure 8, the distribution generated with the modified QuickPick is compared to the original obtained by a random walk. The queries used were of size 100, i.e. involved 100 base relations. The two figures show typical situations we encountered in large series of experiments for queries of different sizes and query graphs. In Figure 8a, both distributions are almost the same, whereas in 8b the distribution of QuickPick has a significantly lower mean. The divergence depends not only on the catalog parameters but also on the shape of the join graph, e.g. for a star graph—and only in this case—both distributions coincide as QuickPick achieves uniformity in this case.

The correlation coefficient is a statistical tool to measure the correlation of two distributions. Figure 9 shows the coefficients for variable query size. Each point is the arithmetic mean of the coefficients for 50 queries of the respective size. Though very close to 1 for small queries, we observe a declining tendency with increasing query size. However, the distribution of QuickPick shifted in every observed case to the left increasing the number of plans with costs lower than the mean as shown in Figure 10. We attribute this difference to QuickPick's property of selecting bushy plans with higher probability than linear trees.

5. Related Work

The join-ordering problem continuously received attention during the past two decades. Besides enumeration techniques for small query sizes (cf. e.g. [16, 22, 14]), heuristics have been developed in order to tackle larger instances [12, 20]. However, as Steinbrunn et. al. pointed out, heuristics yield only mediocre results as the queries grow in size [17].

On the other hand, beginning with [10], randomized techniques have been introduced and attracted particular interest ever since. Swami and Gupta as well as Ioannidis and Kang proposed transformation-based frameworks where—after creating an initial plan—alternative plans are derived by application of transformation rules. The two most prominent representatives of this class of algorithms *Iterative Improvement* and *Simulated Annealing* can be proven to converge toward the optimal query plan for infinite running time. A theoretical result which is of limited use for practical applications as it does not describe the speed of convergence. As we pointed out above, these algorithms spend most of the running time on escaping local minima and making up for poor intermediate results, reaching high quality results eventually though.

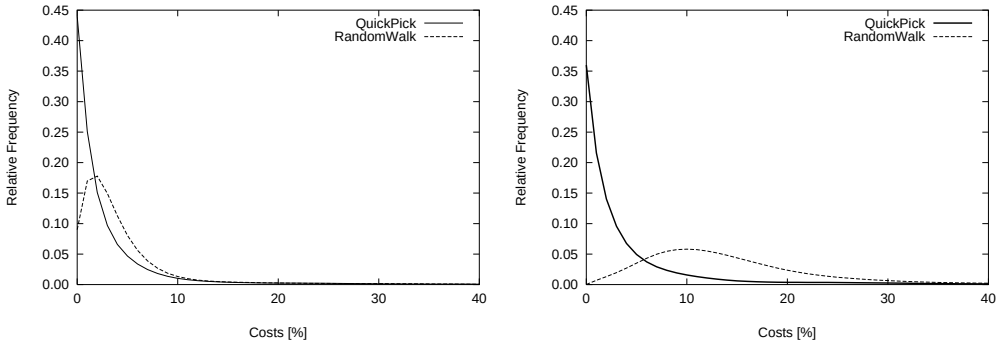


Figure 8: Distributions of Random Edge Selection compared to original

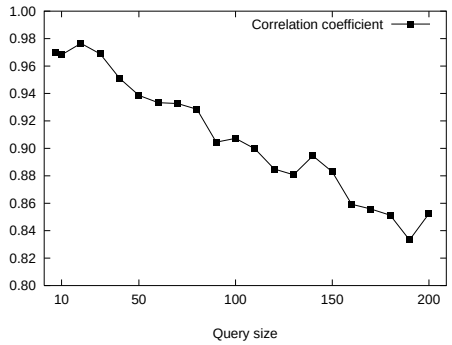


Figure 9: Correlation coefficient

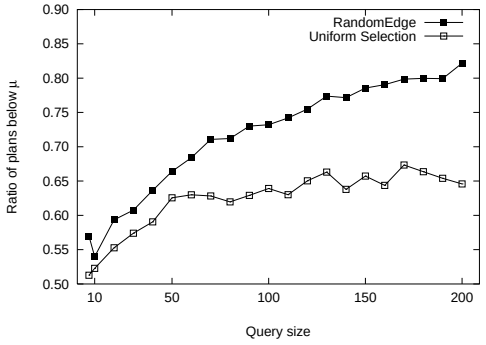


Figure 10: Ratio of plans below c_μ

In addition, navigating algorithms like Iterative Improvement or Simulated Annealing depend to a certain degree on the quality of the initial processing tree which affects the stability of the results obtained and requires careful parameter tuning: if the convergence is urged too firmly, the algorithm may get stuck in a local minimum at an early stage, if the convergence is not forced valuable running time is given away. To mitigate this problem, hybrid strategies like *Toured Simulated Annealing* and *Two-Phase Optimization* were developed [13, 9].

Ioannidis and Kang presented a thorough analysis of the search space topology induced by transformation rules, but the impact of the query's size on the topology remains unclear. Moreover, according to these studies, navigating algorithms require more than linearly increasing running time with increasing query size.

6. Conclusion

Twenty years of research on the join-ordering problem provide the research community with a large variety of valuable facts including proofs of NP-completeness, polynomial algorithms for special cases, randomized algorithms and practitioners' experiences. In this paper we showed how to fit the puzzle

pieces together including an important, so far largely neglected, aspect in our considerations: cost distributions.

Our experimental analysis of cost models show that cost distributions are limited to certain shapes which can be effectively exploited by random sampling techniques. We presented a simple yet powerful optimization technique which relies on the cost distribution only and is completely independent from the search space's size as our experiments show. Even very large queries take only very short time for optimization.

Breaking the curse of NP-completeness? We believe, our findings put the join-ordering in the group of NP-complete problems that are "easy-to-solve", a group that received special attention recently [2]. NP-completeness as a measure of worst case complexity turns out to be not an appropriate characteristic for the difficulty of join-ordering. This aspect is not only of theoretical concern but also of particular interest for enumeration-based optimizer in today's commercial database systems. Assessing the queries difficulty is an important parameter for the trade-off between optimization results and running time spent on the optimization process.

However, not all practical implications of our work are completely clear, yet. Our results are strictly limited to join queries but real-world queries usually comprise a multitude of other relational operators. Therefore, our agenda of future research includes an experimentation as to how much of the insight about cost distributions gained in this work is transferable to more general query optimization problems.

We feel that this work opened a completely new perspective toward query optimization and are anxious to assess extensibility and applicability of the concepts presented.

References

1. B. Bergsten, M. Couprie, and M. Lopez. DBS3: A Parallel Data Base System for Shared Store. In *Proc. of the Int'l. Conf. on Parallel and Distr. Inf. Sys.*, pages 260–263, San Diego, CA, USA, January 1993.
2. P. Cheeseman, B. Kanefsky, and W. M. Taylor. Where the Really Hard Problems Are. In *IJCAI*, pages 331–337, San Mateo, CA, USA, 1991. Morgan Kaufmann.
3. E. Elmasri and S. B. Navathe. *Fundamentals of Database Sytems.* Benjamin/Cummings, Redwood City, CA, USA, 2nd edition, 1994.
4. C. A. Galindo-Legaria, A. Pellenkoft, and M. L. Kersten. Fast, Randomized Join-Order Selection – Why Use Transformations? In *Proc. of the Int'l. Conf. on Very Large Data Bases*, pages 85–95, Santiago, Chile, September 1994.
5. C. A. Galindo-Legaria, A. Pellenkoft, and M. L. Kersten. Uniformly-distributed Random Generation of Join Orders. In *Proc. of the Int'l. Conf. on Database Theory*, pages 280–293, Prague, Czechia, January 1995.
6. T. Ibaraki and T. Kameda. Optimal Nesting for Computation N-Relational Joins. *ACM Trans. on Database Systems*, 9(3):482–502, September 1984.
7. Y. E. Ioannidis and S. Christodoulakis. On the Propagation of Errors in the Size of Join Results. In *Proc. of the ACM SIGMOD Int'l. Conf. on Management of Data*, pages 268–277, Denver, CO, USA, May 1991.
8. Y. E. Ioannidis and Y. C. Kang. Randomized Algorithms for Optimizing Large Join Queries. In *Proc. of the ACM SIGMOD Int'l. Conf. on Management of Data*, pages 312–321, Atlantic City, NJ, USA, May 1990.
9. 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 Int'l. Conf. on Management of Data*, pages 168–177, Denver, CO, USA, May 1991.

10. Y. E. Ioannidis and E. Wong. Query Optimization by Simulated Annealing. In *Proc. of the ACM SIGMOD Int'l. Conf. on Management of Data*, pages 9–22, San Francisco, CA, USA, May 1987.

11. H. Korth and A. Silberschatz. *Database Systems Concepts*. McGraw-Hill, Inc., New York, San Francisco, Washington, DC, USA, 1991.

12. R. Krishnamurthy, H. Boral, and C. Zaniolo. Optimization of Nonrecursive Queries. In *Proc. of the Int'l. Conf. on Very Large Data Bases*, pages 128–137, Kyoto, Japan, August 1986.

13. R. S. G. Lanzelotte, P. Valduriez, and M. Zaït. On the Effectiveness of Optimization Search Strategies for Parallel Execution Spaces. In *Proc. of the Int'l. Conf. on Very Large Data Bases*, pages 493–504, Dublin, Ireland, August 1993.

14. A. Pellenkoft, C. A. Galindo-Legaria, and M. L. Kersten. The Complexity of Transformation-Based Join Enumeration. In *Proc. of the Int'l. Conf. on Very Large Data Bases*, pages 306–315, Athens, Greece, September 1997.

15. W. Scheufele and G. Moerkotte. On the Complexity of Generating Optimal Plans with Cross Products. In *Proc. of the ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 238–248, Tucson, AZ, USA, May 1997.

16. P. 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 Int'l. Conf. on Management of Data*, pages 23–34, Boston, MA, USA, May 1979.

17. M. Steinbrunn, G. Moerkotte, and A. Kemper. Heuristic and Randomized Optimization for the Join Ordering Problem. *The VLDB Journal*, 6(3):191–208, August 1997.

18. A. Swami. Distributions of Query Plan Costs for Large Join Queries. Technical Report RJ7908, IBM Almaden Research Center, San Jose, CA, USA, January 1991.

19. A. Swami and A. Gupta. Optimizing Large Join Queries. In *Proc. of the ACM SIGMOD Int'l. Conf. on Management of Data*, pages 8–17, Chicago, IL, USA, June 1988.

20. A. N. Swami and B. R. Iyer. A Polynomial Time Algorithm for Optimizing Join Queries. In *Proc. of the IEEE Int'l. Conf. on Data Engineering*, pages 345–354, Vienna, Austria, April 1993.

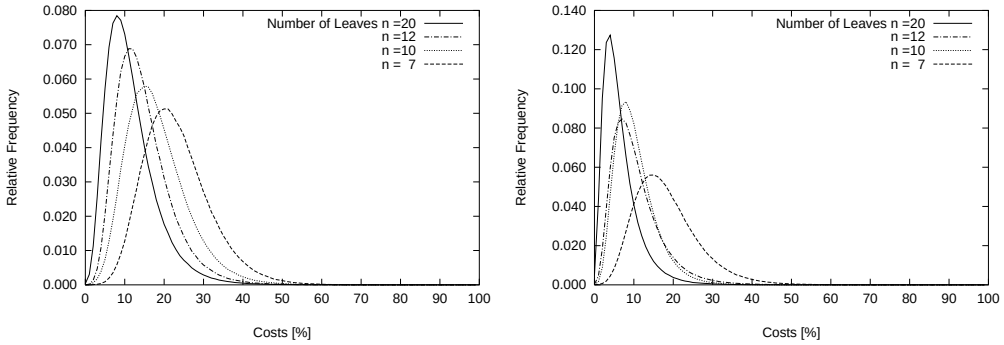
21. J. S. Turner. Almost All k-Colorable Graphs are Easy to Color. *Journal of Algorithms*, 9(1):63–82, March 1988.

22. B. Vance and D. Maier. Rapid Bushy Join-order Optimization with Cartesian Products. In *Proc. of the ACM SIGMOD Int'l. Conf. on Management of Data*, pages 35–46, Montreal, Canada, June 1996.

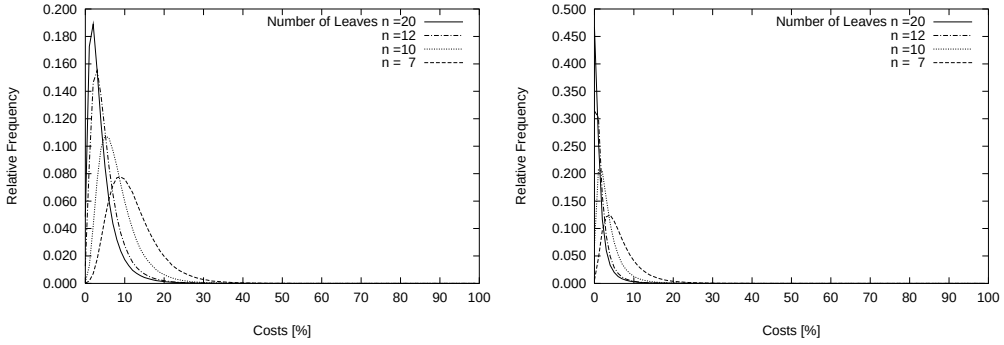
23. F. Waas and A. Pellenkoft. Exploiting Cost Distributions for Query Optimization. Technical Report INS-R9809, CWI, Amsterdam, The Netherlands, October 1998.

1. Appendix

The following graphs show the experiment described in Section 4.2 for Gamma-distributed Y_i. Experiments using the χ^2 distribution yield graph of the same shape due to the relation between Gamma and χ^2 distribution. Since not only the deviation is influenced by ν we centered the distribution at the respective mean. As a measure of deviation we use σ/μ.



a) 0.1000 b) 0.1291



c) 0.1581 d) 0.2236

Figure 11: Cost distribution for Gamma-distributed random variables