Centrum voor Wiskunde en Informatica

REPORT*RAPPORT*

Exploiting Costs Distributions for Query Optimization

F. Waas, A. Pellenkoft

Information Systems (INS)

INS-R9809 September 1998

Report INS-R9809
ISSN 1386-3681

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

Exploiting Cost Distributions for Query Optimization

Florian Waas Arjan Pellenkoft

CWI
P.O.Box 94079, 1090 GB Amsterdam, The Netherlands
⟨*firstname.lastname*⟩*@cwi.nl*

ABSTRACT

Large-scale query optimization is, besides its practical relevance, a hard test case for optimization techniques. Since exact methods cannot be applied due to the combinatorial explosion of the search space, heuristics and probabilistic strategies have been deployed for more than a decade. However, the results achieved are subject to discussion as several performance-critical effects still lack a sound explanation.

In this paper we show how the cost distribution within these search spaces can be exploited. We analyze costing techniques and their principles to assess the difficulty of optimizing queries. Our investigation points out distinct characteristics of the cost distribution caused by the tree structure of the query plans. Applying basic stochastics to those distributions shows that random plan generation qualifies as a highly effective optimization technique under the precondition that every possible query plan may be generated with a positive probability. We present a run-time optimized algorithm meeting this requirement. Our experiments confirm the analytical results and demonstrate the algorithm's efficiency.

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

1. Introduction

Despite the relentless effort devoted to it, the problem of join order optimization for large queries is still far from being solved satisfactory. Currently, the state of research in this field resembles that of an abortive rather than a successful search: The underlying combinatorial optimization problem is known to be NP-complete, thus, no efficient optimization algorithm can be expected; for problem instances of small size dynamic programming and plain enumeration of the search space are feasible techniques to find the optimal query plan within acceptable running time. As the search space grows exponentially with the query's size the application of these methods is limited. Hence, optimization techniques that yield results of *acceptable* quality as an approximation to the optimum are sought.

Besides heuristics, probabilistic algorithms attracted particular interest [9, 13, 11]. Implementing a black box-approach, they require only little knowledge about the actual optimization problem which has to be incorporated into a set of transformations. Though applied successfully to various combinatorial

optimization problems, in context of join ordering they earned only questionable reputation as different studies showed contradicting results [13, 7, 11]. These reports make them appear unreliable and very sensitive to a multitude of parameters needed for proper tuning.

In this paper we tread a new path inspired by an observation first reported on by Ioannidis and Kang [8]. The distribution of costs in the search space is left-weighted, i.e. the majority of states have costs lower than the actual mean of the distribution. The question we want to address, in this work is: *How can we exploit such cost distributions for query optimization?*

We show that the aforementioned behavior is caused by the binary tree structure of the query plans. The structure defines dependencies among the single operators that are reflected in the operators' cost values. Although cost models usually differ in the richness of system parameters they take into account, they all compute the costs according to the tree structure of the plan and its intrinsic dependencies.

Before investigating the possibilities how to make use of the cost distribution's shape, we develop a classification scheme in terms of the cost distribution's parameters that overcomes the deficiencies of previous schemes and enables a sound definition of the optimization goal.

For a further analysis, we abstract cost distributions by Gamma distributions with different parameters and show analytically that random sampling—where plans are generated with uniform probability—delivers results of high quality with a probability close to 1, for samples of moderate size. However, implementing a uniform selection limits the whole approach to special kinds of queries where appropriate techniques are known [3, 4].

To escape this dilemma, we sacrifice the uniformity of the selection process and demand only that *each* plan can be generated with probability greater than zero. A simple yet efficient technique to implement this is *Random Edge Selection*. Edges of the join graph are randomly chosen and the corresponding query plan is simultaneously built, bottom-up [14]. The cost distribution obtained with this method turns out to have a lower mean, in almost every case. Therefore, results of better quality can be achieved spending the same effort. In order to cut down on the running time we exploit the bottom-up constructing of plans: Sub-plans are discarded as soon as their costs exceed the costs of the best plan found so far. Plans are only built-up as long as they stand a chance to become the new best plan. Large series of experiments not only show that Random Edge Selection is superior to other techniques at average, it is also distinguished by its reliability. Since it exploits the distribution's feature it is independent from a particular size of query or search space.

Road-Map. The remainder of this paper is organized as follows. In Section 2 we examine the principles of standard costing techniques. Based on those results we show how cost distributions can be exploited by random plan generation in Section 4. A quantitative assessment of the techniques is given in Section 5. Section 6 concludes the paper.

2. Inside Cost Models

The role of cost models has been discussed controversially throughout the history of query optimization. One of the points at issue is the richness of the model, i.e. which parameters to cover and which not. Furthermore, in the context of parallel databases additional multi-dimensional costing was introduced lately to advance scheduling techniques one-dimensional model were incapable to handle [5]. However, to match the settings of previous work we focus on one-dimensional models which are the state-of-the-art used in commercial products.

Before we scrutinize cost distributions we try to identify the basic principles that lead us to conjecture Ioannidis' observation to be more than just an odd incident caused by a particular cost model.

2.1 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. As a typical example the granularity of data passed between operators can serve. 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 [7]. 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 expectable, the fourth needs special attention. This effect was first spotted by Ioannidis and Kang in [8] but also reported on by Steinbrunn et. al. in [11], 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 [2] and [10] 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. The following experiment provides more clarity by abandoning even queries and catalogs, the last remaining inponderabilities.

2.2 Beyond Queries and Catalogs

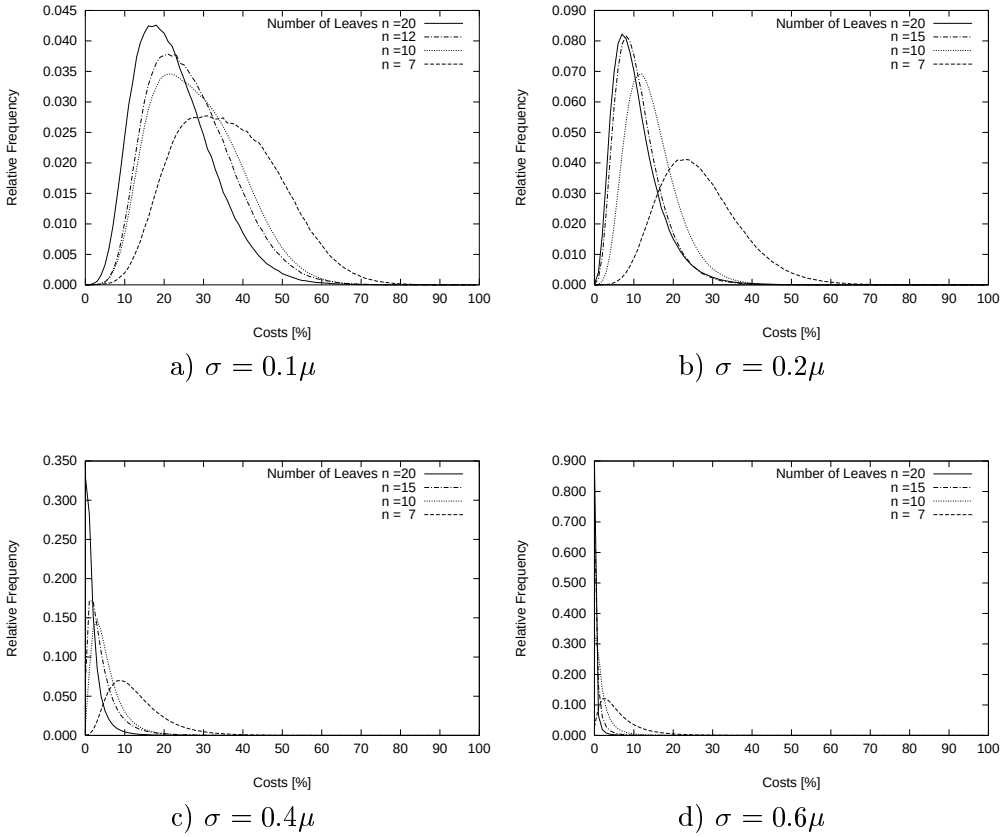
To examine the nature of cost computation along tree structures we devise a cost model for mere binary trees: Given a binary tree t the costs of an inner node v are determined as $C(v) = C(v_l) \cdot C(v_r)$ where v_l and v_r denote its left and right son, respectively. In case v is a leaf, the costs 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 here. Further experiments can be found in Appendix 1.

Figure 1 shows series of 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



a) $\sigma = 0.1\mu$ b) $\sigma = 0.2\mu$

c) $\sigma = 0.4\mu$ d) $\sigma = 0.6\mu$

Figure 1: Cost distributions for the binary tree cost model

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.

3. Classifying Query Plans

To judge the quality of a query plan, a classification based on the cost values is needed. Swami proposed such a classification consisting of three groups [12]: *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. We would expect two queries that have identic cost distributions—just with a different c_{min}—, to have exactly the same ratio of good, acceptable and bad plans. Swami's classification, however, falls short of this invariance as the following example illustrates.

Consider a cost distribution ϕ the shape of an exponential distribution, simi-

lar to Figure 1d. Moreover, we assume the mean to be $c_\mu = 2c_{min}$, for simplicity. The ratio of good plans, i.e. plans with costs below $c_{good} = 2c_{min} = c_\mu$ computes to

$$\int_{c_{min}}^{c_\mu} \phi(t)dt = \int_0^1 e^{-t}dt \approx 0.63$$

Translating this distribution by $2c_{min}$ yields $c'_{min} = 3c_{min}$ and $c'_{good} = 2c'_{min} = 6c_{min}$. Consequently, the ratio of good plans increases and totals

$$\int_{c'_{min}}^{c'_{good}} \phi(t)dt = \int_{3c_{min}}^{6c_{min}} \phi(t)dt = \int_0^3 e^{-t}dt \approx 0.95$$

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 2, both situations are depicted. The costs are normalized to fit the bare exponential distribution without scaling. 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

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

For the optimum, the normalized costs equals 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 3, the areas of plans with $\lambda(q) \leq 0.15$ and $\lambda(q) \leq 1.0$ are shown for the same distribution as before.

Finally, an important point often neglected when discussing query optimization techniques is the accuracy of the costing. As pointed out in [6], result size estimate errors propagate exponentially through the query plan, rendering comparisons of plans whose λ differ less than a few percent, meaningless. Therefore, a sound demand for comparing plans is to respect the *resolution* of the cost computation.

In the remainder of this paper we will use $\lambda = 0.15$ as a threshold of resolution, although there is evidence justifying even greater values with respect to large join queries. Hence, the optimization goal we are aiming at is to find a plan with λ below 0.15 (cf. Fig. 3).

4. Query Plan Selection

The considerations of the previous section pointed out that cost distributions have distinct characteristics in common. In this section, we develop a simple yet efficient technique to exploit these characteristics for query optimization by

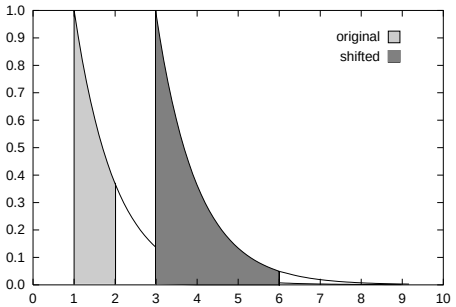


Figure 2: c_{min}–Classification

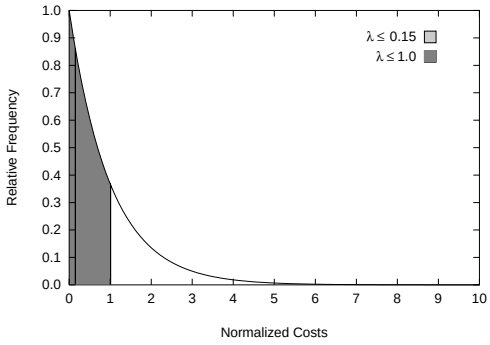


Figure 3: λ–Classification

random selection of query plans.

4.1 Uniform Selection of Query Plans
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[1] 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

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

Apparently, the following holds:

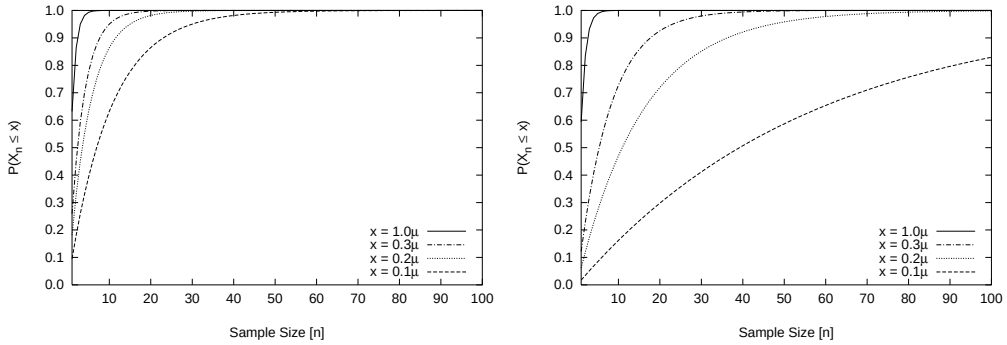
$$P(X_n \leq x) = 1 - \big[P(X > x)\big]^n = 1 - \big[\int_x^\infty \xi(t)dt\big]^n$$

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

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

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

[1] 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. 2.2).



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

Figure 4: Probability to select plan with costs below certain threshold

In Figure 4, 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. 4a). As Figure 4b 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.

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

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

4.2 Random Edge Selection

For the class of acyclic queries Galindo-Legaria et. al. developed counting, ranking and un-ranking techniques that can be accommodated easily to random plan generation [3, 4].

Typical applications and also benchmark suites like the TPC-D however, exceed the limitations of this technique demanding also processing of queries on the basis of arbitrary join graphs. Unfortunately, the aforementioned techniques do not appear to extend to the general case.

To escape this dilemma we give up the uniformity using an algorithm that can generate any plan at random though. The one we found most appropriate is *Random Edge Selection*, short RANDOMEDGE. A largely self-explanatory outline of the algorithm is given Figure 5. For each retry a query plan q is initialized

```
Algorithm  RandomEdge
Input      G(V, E) join graph, n sample size
Output     q_best best query plan found

r ← ∞                                  // initialize lowest costs so far
while n > 0 do
      E' ← E
      q ← G'(V, ∅)                     // initialize query plan
      while E' ≠ ∅ do
            choose e ∈ E'              // random edge selection
            E' ← E' \ {e}
            AddJoin(q, e)
      done                             // plan completed
      if C(q) < r do                   // check for new best plan
            q_best ← q
            r ← C(q)
      done
      n ← n − 1
done
return q_best
```

Figure 5: Algorithm RandomEdge

to consist of the nodes V only. The subroutine AddJoin adds successively the join operators that correspond to the randomly selected edge—where sufficient, only the predicate is added to a previous join.

RandomEdge generates query plans without distinguishing left and right input of joins. This distinction, however, can be handled efficiently by the costing selecting the cheaper of the two alternatives. Moreover, RandomEdge can generate any query plan with positive probability. Accordingly,

$$\lim_{n\to\infty} P(X'_n = c_{min}) = 1$$

holds, i.e. with increasing size of the sample the probability to obtain better results increases.

On the other hand, RandomEdge suffers from a distinct drawback. Once a good plan is found the algorithms wastes most of the time to complete query plans whose costs will exceed the record. Due to the additivity of the costing and the bottom-up proceeding of the algorithm a plan's costs can be computed incrementally. Figure 6 shows the outline of an improved variant, called QuickPick, where costs are checked against the current record after every insertion of a predicate or join. Partly built-up plans are discarded as soon as the sum of costs of all sub-plans, is greater than the lowest found so far. In QuickPick, the limit of retries is not the number of trees generated as in RandomEdge but the number of join predicates added.

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

```
r ← ∞                                  // initialize lowest costs so far
E' ← E
q ← G'(V, ∅)                           // initialize query plan
while s > 0 do
    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
    s ← s − 1
done
return q_best
```

Figure 6: Algorithm QuickPick

5. Quantitative Assessment

The cost model we used in the experiments was an I/O-based one comparable to the techniques described in [10, 11]. The catalogs were generated at random in analogy to [8]. Our settings match those of previous work as the coincidence of cost distributions indicates.

In Figure 7a) and b) the distribution generated by RandomEdge is compared to the original one. The queries used were of size 100—i.e. involved 100 base relations—and given by an acyclic random graph as join graph. The original distribution was computed with the techniques described in [4]. The two figures show typical situations we encountered in large series of experiments. In Figure a), both distributions are almost the same, whereas in b) the distribution of RandomEdge has a significantly lower mean. The divergence depends not only on the catalog parameters but also on the shape of the join graph (see below).

To capture the extent of the shift, we computed the correlation coefficient for pairs of samples with 500 elements. The query varied from 7 to 200. Though 200 seems beyond any realistic size, it helps detect tendencies concerning the size of the search space. For each query size the coefficients of 50 pairs were computed (see Fig. 8). Though very close to 1 for small queries, we observe a declining tendency with increasing query size. However, the distribution of RandomEdge shifts in almost every case to the left increasing the number of plans with costs lower than the mean. In Figure 9, the ratio of plans below

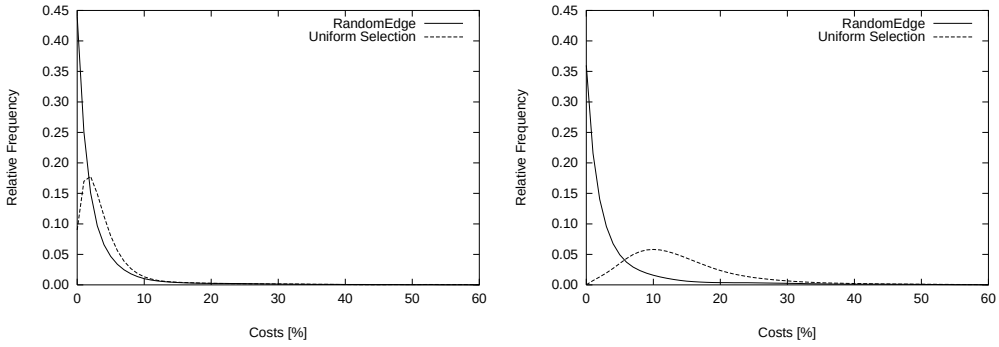


Figure 7: Distributions of Random Edge Selection compared to original

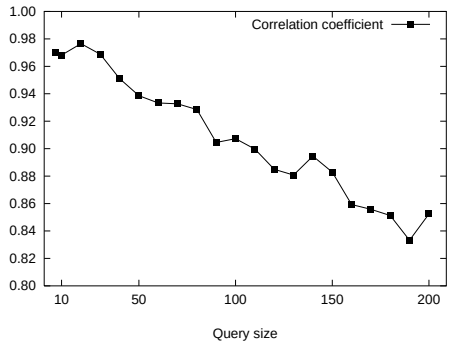


Figure 8: Correlation coefficient

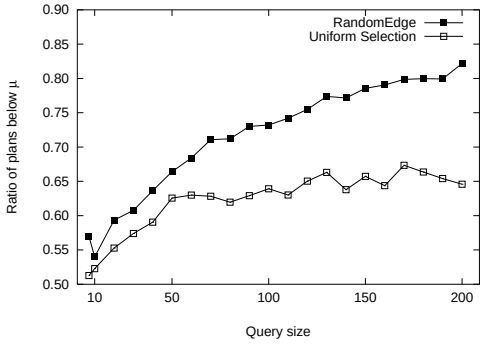


Figure 9: Ratio of plans below c_μ

the respective mean are depicted for both distributions. Accordingly, RANDOMEDGE finds plans of high quality within less steps than a uniform selection would require.

In Figure 10, the convergence behavior of RANDOMEDGE and QUICKPICK is shown in comparison to uniform plan selection for a query of size 100. On the y-axis, the normalized costs are given according to Section 2. Since it is not possible to compute c_{min} and c_μ exactly, we used approximations taken from a RANDOMEDGE sample of size 100000. Hence, the mean found is only a conservative estimate, usually lower than the actual one. On the x-axis the number of steps, i.e. the number joins or predicates added, is given. The comparison on the basis of steps is preferable to elapsed time since the time is subject to the programmer's implementation skills. For completeness, our prototype, ran on an Origin2000/250MHz achieved a ratio of approximately 16500 steps per second for RANDOMEDGE and QUICKPICK. Uniform selection is significantly slower. As the graph shows, all three algorithms find plans of high quality. QUICKPICK converges much quicker than any of the others, finding its best plan before step 1500.

Figures 11 through 14 show comprehensive experiments for different types of join graphs. The impact of the catalog deviation turned out to be not significant

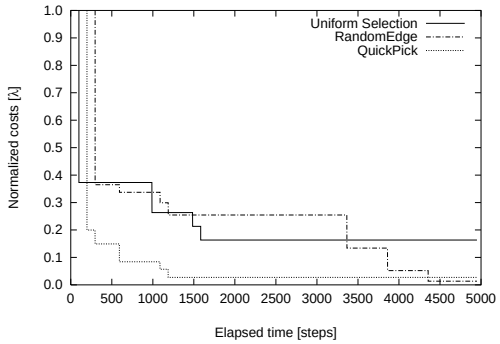


Figure 10: Convergence

since RandomEdge's distribution is in almost every case further left than the actual one and is does not change severely. In all the comparisons only the interval of normalized costs $[0, 0.15]$ is displayed. Every query was optimized with a limit of 5000 steps and repeated 50 times to also assess the stability of the results.

For star-shaped join graphs, RandomEdge generates plans uniformly, i.e. the distributions obtained are identical. As Figure 11 shows QuickPick delivers plans of the same quality than uniform selection does. For all other kinds of join graphs, RandomEdge's distribution differs giving it a better chance to find low costly plans. This effect can be already observed for the other extreme, linear join graphs. RandomEdge achieves better results though not significantly. In case of arbitrary acyclic queries, we find already distinct differences of the quality achieved but also in the stability. The explanation for that are more frequent changes in the original distributions, while RandomEdge's distributions vary only little.

Finally, in Figure 14 experiments for query graphs containing one cycle are depicted. In contrast to the previous of acyclic queries, the distribution is even more favorable for QuickPick. We attribute the shift to a larger cost range and the stronger restriction of the additional predicate. A trend we found to continue 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.

As all experiments besides the special case of star queries show, the quality of the optimization results are as good as independent of the query size, i.e. QuickPick is able to exploit the features of the cost distribution confirming the results of Section 4.1.

6. Conclusion

Cost distributions, an integral part of the query optimization problem, were so far largely disregarded for optimization purposes. Join queries, for instance, are characterized by asymmetric, left-weighted distributions under standard costing techniques. Although, reported on independently in different previous papers, this fact was interpreted as property of the cost model rather than the query

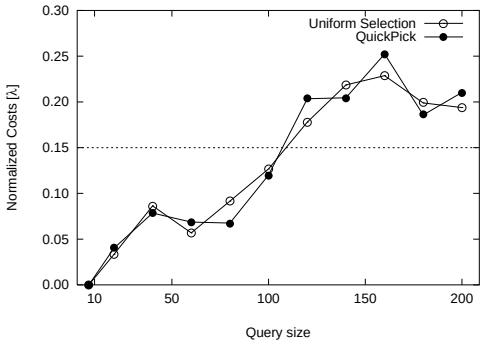


Figure 11: Star queries

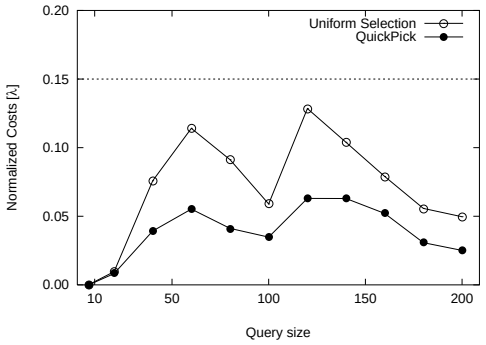


Figure 12: Chain queries

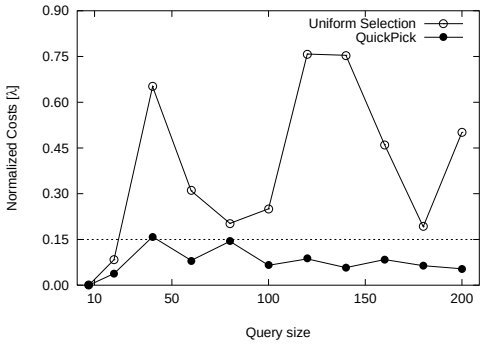


Figure 13: Acyclic queries

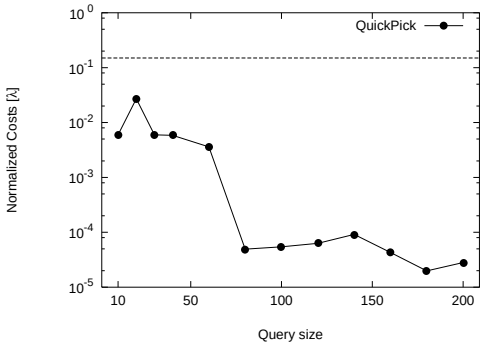


Figure 14: Cyclic queries

plan.

Since these shapes appear to be very favorable for optimization approaches on the basis of random sampling, we investigated the factors responsible for it. In order to single out influential parameters, we simplified different cost models gradually. It turned out that richness of the model and specific properties of the catalog play a subordinate role only. The actual cause is the intrinsic binary tree structure with its dependencies among the operators.

We show analytically that random sampling is a highly effective method to exploit the cost distributions found. Moreover, to give an exact assessment of the performance, we develop a classification scheme for query plans that overcomes the drawbacks of single reference point classification, used so far. Our new technique enables not only a precise definition of the optimization goal but also respects the cost model's resolution.

QuickPick, the algorithm we present, works on the basis of randomly selecting edges from the join graph and building the corresponding query plan bottom-up. Despite its stunning simplicity, it achieves results of very high quality, is distinguished by its stability and very short running times. Furthermore, our experiments confirm the analytical results and show that optimization techniques that are capable of exploiting the features of costs distributions are independent of the size of the search space. Hence, for large join queries they

appear to be superior to transformation-based or evolutionary techniques.

References

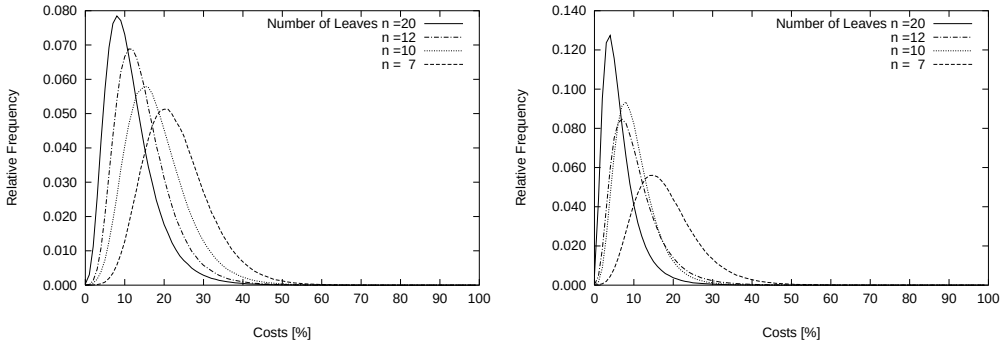
1. B. Bergsten, M. Couprie, and M. Lopez. DBS3: A Parallel Data Base System for Shared Store. In *Proc. of the Int'l. Conf. on Parallel and Distributed Information Systems*, pages 260–263, San Diego, CA, USA, January 1993.
2. E. Elmasri and S. B. Navathe. *Fundamentals of Database Sytems*. Benjamin/Cummings, Redwood City, CA, USA, 2nd edition, 1994.
3. 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.
4. 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*, volume 893 of *Lecture Notes in Computer Science*, pages 280–293, Prague, Czechia, January 1995.
5. S. Ganguly, W. Hasan, and R. Krishnamurthy. Query Optimization for Parallel Execution. In *Proc. of the ACM SIGMOD Int'l. Conf. on Management of Data*, pages 9–18, San Diego, CA, USA, June 1992.
6. 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.
7. 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.
8. 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.
9. 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.

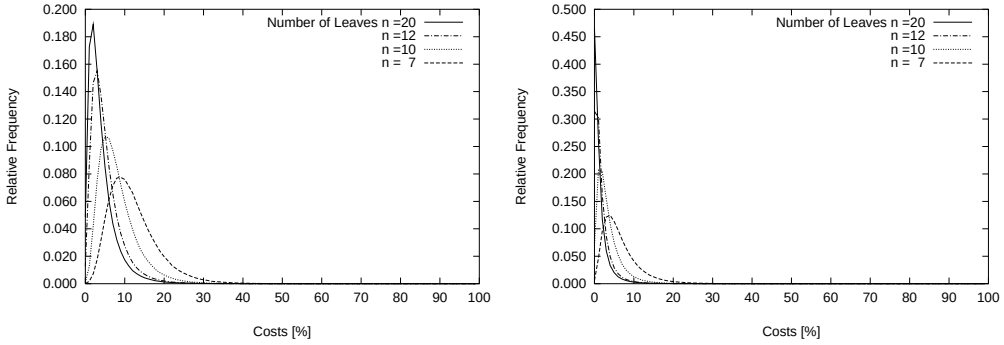
10. H. Korth and A. Silberschatz. *Database Systems Concepts*. McGraw-Hill, Inc., New York, San Francisco, Washington, DC, USA, 1991.
11. M. Steinbrunn, G. Moerkotte, and A. Kemper. Heuristic and Randomized Optimization for the Join Ordering Problem. *The VLDB Journal*, 6(3):191–208, August 1997.
12. A. Swami. Distributions of Query Plan Costs for Large Join Queries. Technical Report RJ7908, IBM Almaden Research Center, San Jose, CA, USA, January 1991.
13. 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.
14. F. Waas. Efficient Enumeration of Non-isomorphic Processing Trees. Technical Report INS-R9808, CWI, Amsterdam, The Netherlands, July 1998.

1. Appendix

The following graphs show the experiment described in Section 2.2 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 15: Cost distribution for Gamma-distributed random variables