# Join Order Selection — Good Enough is Easy

Florian Waas and Arjan Pellenkoft

CWI, P.O.Box 94079, 1090 GB Amsterdam, The Netherlands
{Florian.Waas,Arjan.Pellenkoft}@cwi.nl

**Abstract.** Uniform sampling of join orders is known to be a competitive alternative to transformation-based optimization techniques. However, uniformity of the sampling process is difficult to establish and only for a restricted class of join queries techniques are known.
In this paper, we investigate non-uniform sampling devising a simple yet powerful algorithm that is generally applicable. The key element of the algorithm is a mapping of randomly generated sequences of join predicates to query plans. We take advantage of the bottom-up constructing of query plans by simultaneously computing the costs and discarding partial plans as soon as they exceed the best costs found so far, which implements a highly effective cost-bound pruning component.
Sampling does not produce *the* optimal plan but a near-optimal solution which is fully sufficient as the cost function grows more and more inaccurate with increasing query size. In return, our algorithm establishes a well-balanced trade-off between result quality and time invested in the optimization process.

## 1 Introduction

Join-ordering is one of the most persistent problems in query optimization. Over the last decade, special attention has been devoted to probabilistic techniques that proofed superior to heuristics [SMK97]. Galindo-Legaria *et al.* made out a good case for using uniform random sampling of plans rather than transformation-based algorithms like Simulated Annealing [GLPK94]. They showed that sampling matches randomized algorithms in quality but outruns them in terms of convergence, i.e. finds high quality solutions earlier. The nucleus of this work is the one-to-one mapping between plans and ordinal numbers. Generating random numbers and un-ranking the associated query plan then establishes a mechanism to sample plans with uniform probability. However, the algorithm devised is a complex construction and the deployment is limited to acyclic graphs only [GLPK95]. This limitation—though popular with previous work—is a distinct restriction. Queries as for instance in the standard data warehouse benchmark suite of TPC-H/R contains indeed cyclic queries [Tra98]. But this algorithm does show the way how to exploit the characteristic features of the search space successfully.

In this paper, we investigate how to overcome this restriction without loss of performance. And more general, we address the question whether uniformity of the sampling is a necessary prerequisite.

QuickPick, the algorithm we develop in this paper, performs biased sampling by selecting edges from the join graph and adding the respective joins to the query plan. To cut down on the running time we add a cost-bound pruning strategy: We simultaneously compute the costs while building up the plan and partial plans that exceed the costs of the currently best plan are discarded as early as possible. The algorithm is distinguished by its high result quality and short running times. Additionally, QuickPick is of low complexity both in time complexity and implementation, and is applicable to any query graph overcoming the restriction for uniform sampling.

To analyze the algorithm and explain its superior performance, we scrutinize cost distributions, i.e. the frequencies of cost values in the entire search space. Reviewing previous work and complementing it with own experiments, we abstract cost distributions making them accessible to formal reasoning. This way we can derive accurate approximations of the quality of the results of sampling. Our investigations indicate that uniform sampling provides upper bounds for our new algorithm.

*Road-Map.* In Section 2, we briefly outline the model for the problem and discuss cost distributions and quality measures. In Section 3, we introduce the sampling algorithm and give a quantitative assessment of it in Section 4. We review related work in Section 5. Section 6 contains our conclusions and outlook to further research.

## 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 [SAC+79,SG88,IK91,SM97,SMK97].

A join query is given by a *join* or *query graph* whose nodes correspond to the base tables used in the query. Its edges are annotated with the predicates of the query, and denote which tables are to be joined. 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 function* or *model*. Both components together make up the join ordering problem of finding the query plan with the least costs.

Since cost models have to reflect the query engine which will execute the plan, cost models differ in general from one system to another, yet there are properties all cost models have in common as we will point out later.

### 2.1 Cost Distributions

The term *cost distribution* refers to the frequencies of all possible cost values occurring in the entire search space. They reflect the ratio of high to low quality solutions. Cost distributions are closely interconnected with the object function. As a consequence, cost distributions are characteristic for a given combinatorial optimization problem, i.e. all instances of a problem display cost distributions with similar properties. The degree of variation is problem specific but limited in its extent; apart from pathological cases. For example, the Symmetric Traveling Salesman Problem is characterized by normal distributions with heavy tails [Waa99]. The cost distributions in query optimization differ substantially from that due to the different nature of the cost function. They display a strong asymmetry with a distinct concentration of cost values close to the optimum.

The first to notice this problem intrinsic property was Swami who reported that, surprisingly, lesser sophisticated optimization strategies like Iterative Improvement are often superior to more complex methods like Simulated Annealing. Swami not only discovered the skew of the cost distribution but also noted its stability across different cost functions: He observed that in experiments with different cost models, an I/O-based one and a Main-memory cost model, performance of optimization algorithms were comparable, suspecting underlying structures common to different cost models [SG88,Swa89].

Ioannidis and Kang for the first time investigated the shape of the cost distribution explicitly finding curves that are best described as Gamma distributions [IK91]. However, they experimented only with one single I/O-based model and therefore attributed their findings to the specific cost model used.

In [WGL00], authors present a sampling mechanism implemented in a commercial database system and extract cost distributions for TPC-H queries using an industrial quality cost function. Furthermore full-blown query optimization and not only join ordering as in previous work was concerned. These finding bear strong resemblance with Ioannidis and Kang's results lending strong support to the abstraction with Gamma distributions.

In contrast to experiments with cost models of increasing complexity we want to complement these observations with experiments using a rudimentary cost function that handles joins only like cartesian products. Such a simplified model is of particular interest for two reasons: Firstly, joins may degenerate to cartesian products, thus the evaluation of cartesian products forms an upper bound of $O(n^2)$ for any join. Secondly, the problem of optimizing the order of cartesian products—though appearing less difficult on first sight—is of the same complexity as join ordering as Scheufele and Moerkotte showed [SM97].

In the cartesian model, costs of each operator compute recursively to

$$c(v) = c(v_l) \cdot c(v_r)$$

where $v$ denotes an operator, i.e., inner node in the query plan, and $v_l$ and $v_r$ its left and right children respectively. If $v$ is leaf, $c(v)$ is the size of the associated base table. The total cost of the plan is the sum of costs per operator.

To extract cost distributions for the cartesian model, we enumerate all non-isomorphic trees of given size, i.e. trees that are not isomorphic under commutative exchange of subtrees. For each tree we generate 1000 configurations of base table sizes according to a given distribution and compute the costs. The parameters of the experiment are size of the problem, i.e. number of base ta-

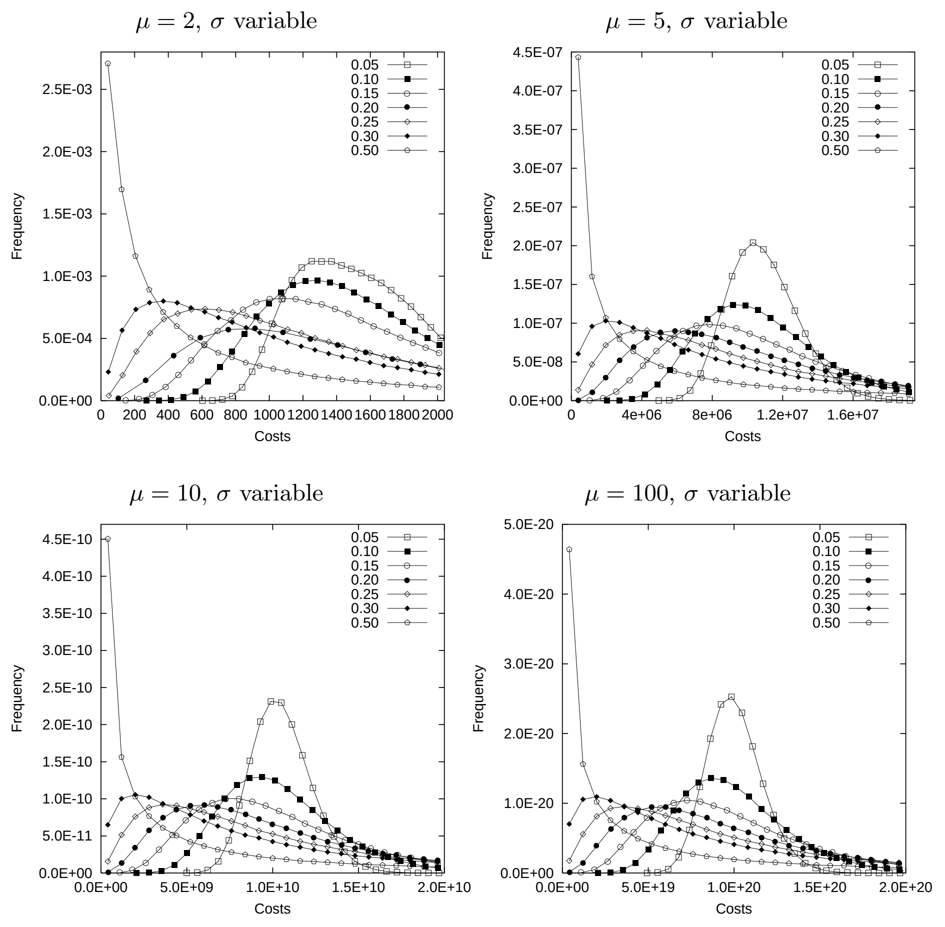


**Fig. 1.** Cost distributions for cartesian model for $\mu = 2, 5, 10, 100$ and $\sigma$ as fraction of $\mu$.

bles which translates to the number of leaves in the processing tree, and mean and deviation of the distribution of base tables. Figure 1 shows the resulting cost distributions for trees of size 10. The deviation is given as a fraction of the mean. In this experiment, we used normal distributions for the base tables but experiments with other distributions showed similar results [WP98]. We observe only little skew for a small deviation (0.05) and increasing skew for larger deviation (0.50). The whole range of shapes observed can be abstracted with Gamma distribution of shape parameter $\nu$ between 1 and 2 as suggested by Ioannidis and Kang. Moreover, this experiment also confirms the connection between the skew of the cost distribution and the deviation. The higher the deviation—i.e., the variance of the database catalog—the stronger the skew.

The above experiment pinpoints the cause of the skew that has been observed with richer cost models. The tree structure of the plans with its multiplicative costing is responsible for the shape of the cost distributions. When moving from the cartesian model to join ordering the cost distributions become less smooth as not all non-isomorphic trees are valid tree-shapes. Moreover the additional selectivities of the join predicates add further distortions. However, as the sum of experiments—previous work and own ones presented in this paper—suggests further additions to the cost model, including extensions to cover a large variety of operator implementation like different join implementations but also other kinds of operators than joins or cartesian products, do not alter the major characteristic of the cost distributions. Thus, the abstraction proposed by Ioannidis and Kang appears more general than authors first thought.

Before we discuss how to exploit the distributions for optimization purposes, we present some considerations on how to measure the quality of the optimization results.

## 2.2 Quality Measures

The cost computation in a database system uses statistical data about the state of the database to estimate the execution costs. As natural consequence, the estimates contain errors. While rather precise for small queries, the accuracy of the cost estimation deteriorates with increasing size of the query as estimate errors propagate exponentially through the query plan [IC91]. Plans whose costs differ only by a few percent cannot be distinguished any more reliably; near-optimal results are as good as the optimum itself.

To reach a trade-off between time spent on the optimization and the result quality, Swami proposed a classification according to which query plans can be divided into three groups: *good*, *acceptable* and *bad* 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. In the following we refer to this classification as *scaling-based classification.*

Note, quality measures of this kind are only of theoretic value in general. When optimizing queries in ad-hoc manner, neither the costs of the optimum nor an approximation is usually available. The same holds for the measure we present below.

Scaling-based classification suffers from the severe drawback to be not invariant under additive translation as the following example shows. Consider a cost distribution of the quality of an exponential distribution

$$\phi_t(x) = \begin{cases} e^{-x+t}, & \text{if } x \geq t \\ 0, & \text{else} \end{cases}$$

$t$ is the additive shift of the distribution. Figure 2 shows $\phi_1$ and $\phi_3$. The optima are of costs 1 and 3 respectively. We should expect both distributions to have the same ratio of good, acceptable and bad plans as the distribution are of exactly the same shape, only shifted by 2. For $\phi_1$ the ratio of good plans computes to

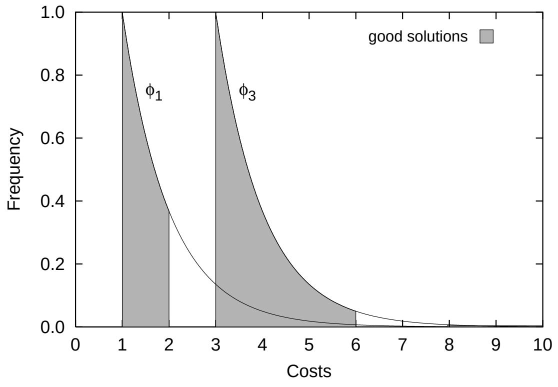


**Fig. 2.** Classification according to Swami

0.63 (interval $[1, 2]$ in Fig. 2). However, for $\phi_3$ the ratio of good plans is 0.95 (interval $[3, 6]$). Though the distribution actually stayed the same, the ratio of good plans is up 50% at 0.95. The cause for the instability is that only a single reference point, namely $c_{min}$, is taken into account.

To overcome this drawback, we classify plans with respect to the cost distribution. We denote the quality of a plan by the *relative quantile* $Q_x$ of the cost distribution its costs are in. We say a plan $q$ is of quality $Q_x$ if the following holds for its costs

$$\frac{c(q) - c_{min}}{c_\mu - c_{min}} \leq x.$$

Figure 3 shows the exponential distribution of the example above (cf. Fig. 2). For example, $Q_1$ denotes the quantile from $c_{min}$ up to $c_\mu$. The quantiles $Q_1$ and $Q_{0.1}$ are highlighted. As the figure shows, quantile-based classification is independent of any translation.

In the following we will use $Q_{0.1}$ as target quantile for the optimization, i.e., we try to find a plan whose costs are in $Q_{0.1}$. Larger quantiles may be justified for larger join queries though.

## 3 Probabilistic Bottom-up Join Order Selection

In order to implement an unrestricted sampling mechanism we use a mapping of join predicates to query plans. It might be helpful to outline the idea behind this mapping first: given a sequence of join predicates, we add the corresponding join operators of the query plan one after another. If the predicate involves a base table that is not yet leaf of the query plan, we add a join operator whose children are the new base table and the partial plan that contains the other table. If both tables are already part of the tree, we add only the predicate to

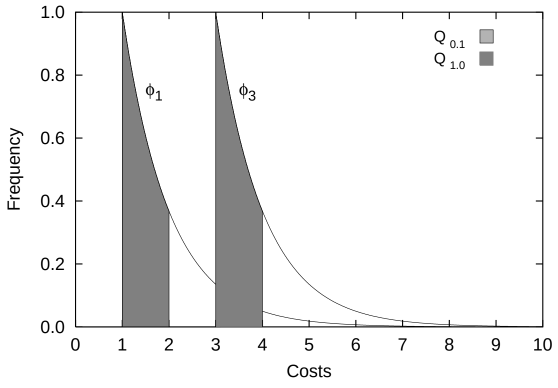


**Fig. 3.** Quantile-based classification

the deepest possible join. To generate now query plans at random we simply generate random sequences of predicates.

In Figure 4, the algorithm, called QUICKPICK, is outlined in pseudo code. After initializing the variable $r$ that records the cheapest plan found so far with $\infty$, the candidate set $E'$ is initialized with the set of edges of the join graph, and $q$ with the base relations. Throughout the random bottom-up construction of a tree $q$ holds all partial trees, i.e. $q$ is actually a forest. Generally, only at the very end—earlier only for cyclic join graphs—, $q$ is completed to a single processing tree.

Until the stopping criterion, say a time limit, is fulfilled $q$ is incrementally built-up by choosing and removing an edge $e$ from the candidate set and adding the corresponding join to the tree (ADDJOIN). In doing so, the subtrees that contain the two endpoints of $e$, i.e. the base relations joined by this edges, are connected with a join operator. If both relations are already leaves to the same sub-plan, only the predicate of $e$ is added to the tree at the deepest possible point. After each such insertion, the costs of the subtrees are computed and summed up. Recall, that $q$ is generally a forest consisting of several disjoint processing trees. If the costs exceed $r$, the costs of the best plan found so far, we discard $q$ and initialize $E'$ and $q$ again and start assembling a new tree. If the set of candidate edges is empty—i.e. we have completed the processing tree—we check for a new record and in this case copy the plan to $q_{best}$. After initializing $E'$ and $q$ we start building a new tree.[1]

[1] The basic principles of QUICKPICK—without cost-bound pruning—have been described already by Pellenkoft [Pel97]. There, this algorithm is called Random Edge Selection and proofed to be incapable of achieving uniform sampling. However, no further performance analysis is conducted. Others might have probably used similar algorithms to generate initial solutions. However, they also did not evaluate the potential of this elementary technique.

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

$r \leftarrow \infty$                        // initialize lowest costs so far
$E' \leftarrow E$
$q \leftarrow G'(V,\emptyset)$                 // initialize query plan
repeat
    choose $e \in E'$                     // random edge selection
    $E' \leftarrow E' \setminus \{e\}$
    AddJoin$(q,e)$                        // add join or predicate
    if $E' = \emptyset$ or $c(q) > r$ do       // either plan complete or costs exceed
                                          // best costs so far
        if $c(q) < r$ do                    // check for new best plan
            $q_{best} \leftarrow q$
            $r \leftarrow c(q)$
        done
        $E' \leftarrow E$
        $q \leftarrow G'(V,\emptyset)$             // reset query plan
    done
until stopping criterion fulfilled
return $q_{best}$
```

**Fig. 4.** Algorithm QuickPick

Essential for the cost bound pruning is the cost computation along the structure in the making. We assume a monotonic cost formula where operators do not influence the costs of their predecessors other than monotonically increasing, i.e. adding an operator later cannot *reduce* the costs of any subtree.

## 4 Assessment

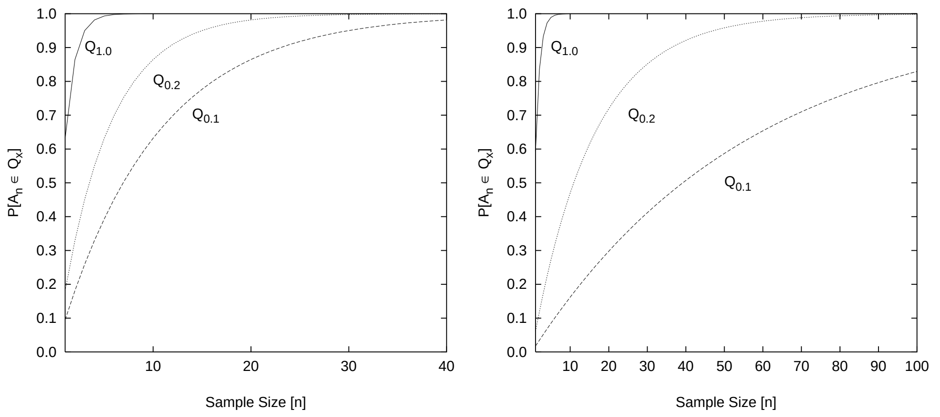
Before presenting figures on QuickPick we investigate the potential of uniform sampling using abstractions for the cost distributions. Afterwards, we compare it to non-uniform sampling and point out the differences and their impact.

### 4.1 Uniform Sampling

We can put random sampling on solid formal grounds and compute probabilities for a successful search in dependency of the running time invested.

Let $A$ be the random variable

$A :=$ *costs of a query plan chosen at uniform probability.*



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

**Fig. 5.** Probability to select plan with costs below certain threshold

The probability to obtain a plan in $Q_x$ under a cost distribution $\xi$ is

$$P[A \in Q_x] = \int_{c_{min}}^{c_{min}+\frac{x}{c_\mu - c_{min}}} \xi(t)dt.$$

For the random variable $A_n$

$A_n :=$ *lowest costs in a sample of n plans chosen at uniform probability*

the following holds:

$$P[A_n \in Q_x] = 1 - \big(P[A \notin Q_x]\big)^n = 1 - \big(1 - P[A \in Q_x]\big)^n$$

In Figure 5, the probability $P[A_n \in Q_x]$ is shown for various $x$. As cost distribution we use Gamma distributions with shape parameter $\nu = 1, 2$. The sample size is given on the x-axis and the probability is plotted against the ordinate. As both diagrams show, finding a plan better than average ($Q_{1.0}$) is almost certainly achieved by a sample of only as little as 10 plans. For $\nu = 1$ the probability to obtain a plan in $Q_{0.2}$ within a sample of size 20 is already beyond 0.95. For a sample larger than 47 plans, the probability for plans in $Q_{0.1}$ is higher than 0.99 (cf. Fig. 5a). As Figure 5b shows, larger sample sizes are needed to achieve the same quality in case of $\nu = 2$. In particular, to reach into $Q_{0.1}$ with a probability greater than 0.99 requires $n$ to be at least 982. Table 1 shows the necessary values of $n$ to achieve $P[A_n \in Q_x] \geq 0.99$. Note, those figures are by far smaller than the widely accepted limits used for transformation-based probabilistic optimization or even genetic algorithms.

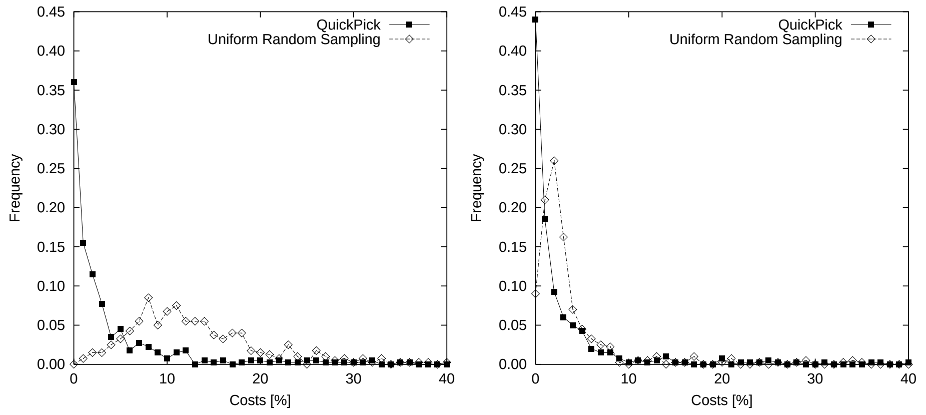


**Fig. 6.** Comparison of cost distributions obtained with biased and uniform sampling

## 4.2 Cost Distribution

In order to determine the cost distribution under QuickPick, we implemented a cost model comparable to those proposed in [EN94,KS91,Ste96]. Clearly, to be successful, the cost distribution $\phi_B$ under QuickPick must be at least as favorable as the original, i.e. shifted to the left relative to $\phi$.

In the following we compare $\phi_B$ and $\phi$ under three aspects: (1) selective samples, (2) the correlation coefficient between a larger set of cost distributions, and (3) the shift of $\phi_B$ relative to $\phi$.

In Figure 6, two pairs of cost distributions for high and low variance catalogs are shown. Both samples are of size 5000, the query size used is 50. To obtain cost distributions with QuickPick we disabled the cost bound pruning so that complete trees were constructed. In a larger series of test cases $\phi_B$ was without exception always left of $\phi$. Moreover, $\phi_B$ bore in all cases strong resemblance with exponential distributions.

To test for a connection of $\phi_B$ and $\phi$ we compute the correlation coefficient. For two random variables, this coefficient is defined as

$$k = \frac{E[(X - E[X])(Y - E[Y])]}{\sigma_X \sigma_Y},$$

**Table 1.** Sample size needed for $P[A_n \in Q_x] \geq 0.99$

| | $Q_{0.1}$ | $Q_{0.2}$ | $Q_{0.3}$ | $Q_{1.0}$ |
|---|---|---|---|---|
| $\nu = 1.0$ | 47 | 24 | 16 | 5 |
| $\nu = 2.0$ | 982 | 261 | 123 | 16 |

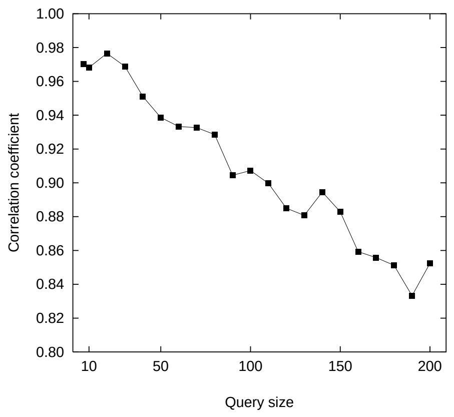


**Fig. 7.** Correlation of uniform and biased cost distribution

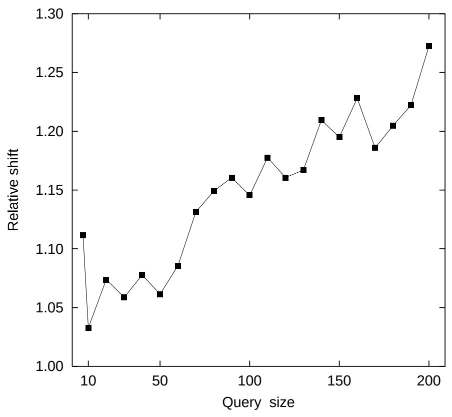


**Fig. 8.** Relative shift

where $E[X]$ denotes the mean of $X$ and $\sigma_X$ is the deviation. For fully correlated distributions, $k$ approaches 1. The more the distributions differ, the lower $k$ gets. In Figure 7, the correlation coefficient is plotted as a function of the query size. Each point comprises 50 pairs of randomly generated queries. The plot shows a clear trend of decreasing correlation with increasing query size.

Finally, we determine the relative shift of $\phi_B$ which is defined as

$$s(x) = \frac{\int_{c_{min}}^{x} \phi_B(x)}{\int_{c_{min}}^{x} \phi(x)}$$

(see e.g. [IK91]). In Figure 8 the shift $s(\mu(\phi))$ is plotted as function of the query size. Again, each data point represents the average of 50 queries. Values above 1 indicate that $\phi_B$ is relatively shifted to the *left* of $\phi$.

Our results clearly exhibit the trend that the biased cost distribution is even more favorable to sampling than the original one. With increasing query size, the difference between the two distributions becomes more distinct, showing the biased one stronger to the left of the original.

## 4.3 Quantitative Assessment

According to our analysis of the cost distribution, the results reported on by Galindo-Legaria *et al.* in [GLPK94] can immediately be transferred and serve, so to speak, as an upper bound for the result quality.

Like uniform sampling, QuickPick is unlikely to find the optimum as sampling works on the premise that all solutions in the top quantile—the size is

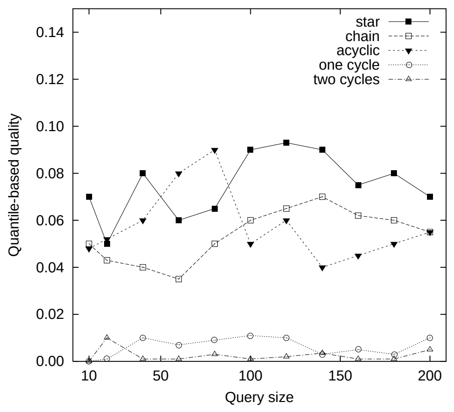


**Fig. 9.** Performance

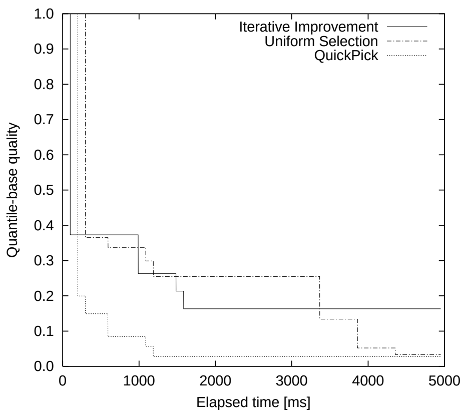


**Fig. 10.** Convergence behavior

parameter to the problem—are equally good. Thus "hitting" this quantile in the course of the sampling *is good enough*.

**Result Quality** Figure 9 shows the quality of the results in terms of quantile-based quality. For the experiments we differentiated the following shapes of query graphs: *stars*, *chains*, and *tree-shaped* on the one hand, and a type which we call *n-cycle* on the other hand. The first group of three comprises queries that can also be optimized with uniform sampling. The second group exceed the scope of uniform sampling. A graph of type $n$-cycle contains exactly $n$ cycles, as the name suggests, but the remainder of the graph is unspecified, i.e. we use randomly generated tree-shaped graph and insert $n$ additional edges. Our notion of cyclic graphs reflects real queries better than highly connected graph structures such as grids or cliques. Also the graph theoretic notion of connectivity is little suitable as almost all queries in actual applications are of a connectivity no higher than one.

For acyclic graphs, QuickPick delivers results of a quality comparable to that of uniform sampling—for star graphs, QuickPick actually implements even uniform sampling. In case of cyclic query graphs, the results are of even higher quality (see Fig. 9).

**Convergence Behavior** Like with uniform sampling, QuickPick's strong point is its quick convergence. Figure 10 shows the costs of the best plan found as function of the elapsed time in comparison with Iterative Improvement and uniform sampling. Due to its biased cost distribution, QuickPick converges significantly quicker. With longer running time the competitors catch up. Iterative Improvement sometimes beats QuickPick, not significantly though.

To underline the differences between uniform sampling and QuickPick, we compute the probability to hit the quantile $Q_{0.1}$ for both algorithms. $Q_{0.1}$ refers

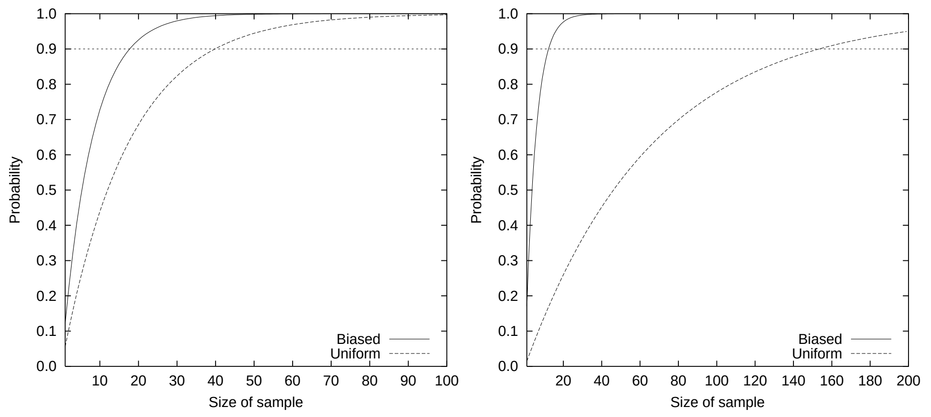


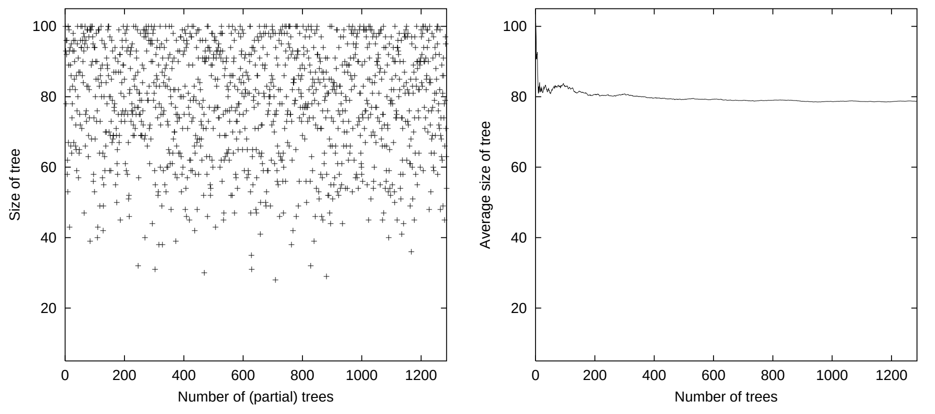
**Fig. 11.** Probability to hit quantile $Q_{0.1}$ with QuickPick and uniform sampling. Left, high variance, right low variance catalog

to the respective quantile of the original distribution. In Figure 11, these probabilities are plotted as function of the size of the sample. The left plot shows the situation for a high, the right for a low variance catalog. To hit the quantile with more than 90% probability in the high variance case requires a sample size of 18 and 40 for QuickPick and uniform sampling respectively. In case of low variance catalog, the numbers differ even more significantly: 13 and 154.

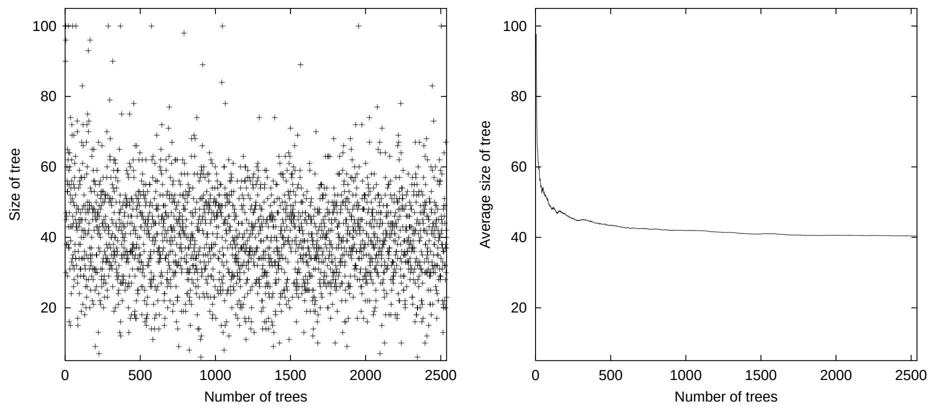
**Cost-bound Pruning** Let us finally investigate the impact of cost-bound pruning on QuickPick. We introduced the algorithm in the form that partial trees are discarded as soon as their costs exceed the currently best plan's cost.

According to our general considerations about the cost distributions the effectiveness of the pruning depends on the shape of the distributions. The further to the left the distribution is the lower the gains, i.e. the trees are built-up almost to completeness. In Figure 12 this effect is demonstrated with low and high variance catalogs for a query of size 100. The left plot in 12a, shows the number of join predicates inserted with AddJoin—referred to as size of tree in the figure. As a stopping criterion we used 100000 insertions which made in this example for 1286 explored trees in total. For each (partial) tree we indicate the size when it was discarded (see Fig. 12a, left), 100 being the maximum. Note, not every tree completed is a new record since the last join can still exceed the best costs so far, which happens specifically frequent with high variance catalogs. The plot on the right hand side shows the average tree size as function of the number of trees. Starting at 100 it drops quickly to about 80 (see Fig. 12a, right).

In Figure 12b the same analysis is done for a low variance catalog. Since there is no strong concentration of solutions as opposed to the previous case, pruning kicks in earlier. The average tree size drops to about 40. Consequently, 100000 steps make for a larger number of (partial) trees explored; 2539 in this example.



(a) High variance catalog



(b) Low variance catalog

**Fig. 12.** Effectiveness of cost-bound pruning in QuickPick

In the first case savings amount to some 20%, in the second almost 60% on average. As a practical consequence, QuickPick with cost-bound pruning inspects a larger sample of trees, 1286 and 2539 in these cases, within the same running time that would be required to build up 1000 plans completely.

## 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. [SAC$^+$79,VM96,PGLK97]), heuristics have been developed in order to tackle larger instances [KBZ86,SI93]. However, as Steinbrunn et. al. pointed out, heuristics yield only mediocre results as the queries grow in size [SMK97].

On the other hand, beginning with [IW87], 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.

In [GLPK94], Galindo-Legaria *et al.* showed that uniform sampling can achieve results of similar quality but significantly quicker.

In addition, navigating algorithms like Simulated Annealing depend on the quality of the initial query plan 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 [LVZ93,IK91].

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

## 6 Conclusion

Based on the observation that sampling is a competitive alternative to transformation-based optimization algorithm like Simulated Annealing, we sat out to investigate background and limitations of sampling techniques for query optimization. To date, the problem of uniform random generation of query plans is only solved for acyclic query graphs.

In this paper, we devised a randomized bottom-up join order selection that performs *biased* sampling and is not limited in its application. Our experiments suggest that results for uniform sampling form an upper bound for the new sampling technique underlining its superior performance. The algorithm presented is distinguished by (1) its low complexity of both run time behavior and implementation (2) high quality results and (3) quick convergence.

Our results show that join ordering is, due to its cost distribution, actually “easier” than its property of being NP-complete may suggest. Similar effects are known for other NP-complete problems like graph coloring [Tur88]. The algorithm we presented establishes a well-balanced trade-off between result quality and time invested in the optimization process.

*Future Research.* Results presented make our algorithm an interesting building block for optimization of more complex queries including aggregates and sub-queries. Our future research is directed to investigate ways of integrating sampling and exact methods to speed up the latter. Another direction we are eager to explore is the random generation of plans for complex queries on the lines of the technique presented in this paper.

## References

[EN94] E. Elmasri and S. B. Navathe. *Fundamentals of Database Sytems.* Benjamin/Cummings, Redwood City, CA, USA, 2nd edition, 1994.

[GLPK94] 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.

[GLPK95] 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, Czech Republic, January 1995.

[IC91] 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.

[IK91] 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.

[IW87] 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.

[KBZ86] 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.

[KS91] H. Korth and A. Silberschatz. *Database Systems Concepts.* McGraw-Hill, Inc., New York, San Francisco, Washington, DC, USA, 1991.

[LVZ93] 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.

[Pel97] A. Pellenkoft. *Probabilistic and Transformation based Query Optimization.* PhD thesis, Universiteit van Amsterdam, Amsterdam, The Netherlands, 1997.

[PGLK97] 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.

[SAC$^+$79] 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.

[SG88] 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.

[SI93] A. 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.

[SM97] 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.

[SMK97] 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.

[Ste96] M. Steinbrunn. *Heuristic and Randomised Optimisation Techniques in Object-Oriented Database.* DISDBIS. infix, Sankt Augustin, Germany, 1996.

[Swa89] A. Swami. Optimization of Large Join Queries: Combining Heuristics and Combinatorial Techniques. In *Proc. of the ACM SIGMOD Int'l. Conf. on Management of Data*, pages 367–376, Portland, OR, USA, June 1989.

[Tra98] Transaction Processing Performance Council, San Jose, CA, USA. *TPC Benchmark D (Decision Support)*, Revision 1.3.1, 1998.

[Tur88] J. S. Turner. Almost All k-Colorable Graphs are Easy to Color. *Journal of Algorithms*, 9(1):63–82, March 1988.

[VM96] 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.

[Waa99] F. Waas. Cost Distributions in Symmetric Euclidean Traveling Salesman Problems—A Supplement to TSPLIB. Technical Report INS-R9911, CWI, Amsterdam, The Netherlands, September 1999.

[WGL00] F. Waas and C. A. Galindo-Legaria. Counting, Enumerating and Sampling of Execution Plans in a Cost-Based Query Optimizer. In *Proc. of the ACM SIGMOD Int'l. Conf. on Management of Data*, Dallas, TX, USA, May 2000. Accepted for publication.

[WP98] F. Waas and A. Pellenkoft. Exploiting Cost Distributions for Query Optimization. Technical Report INS-R9811, CWI, Amsterdam, The Netherlands, October 1998.