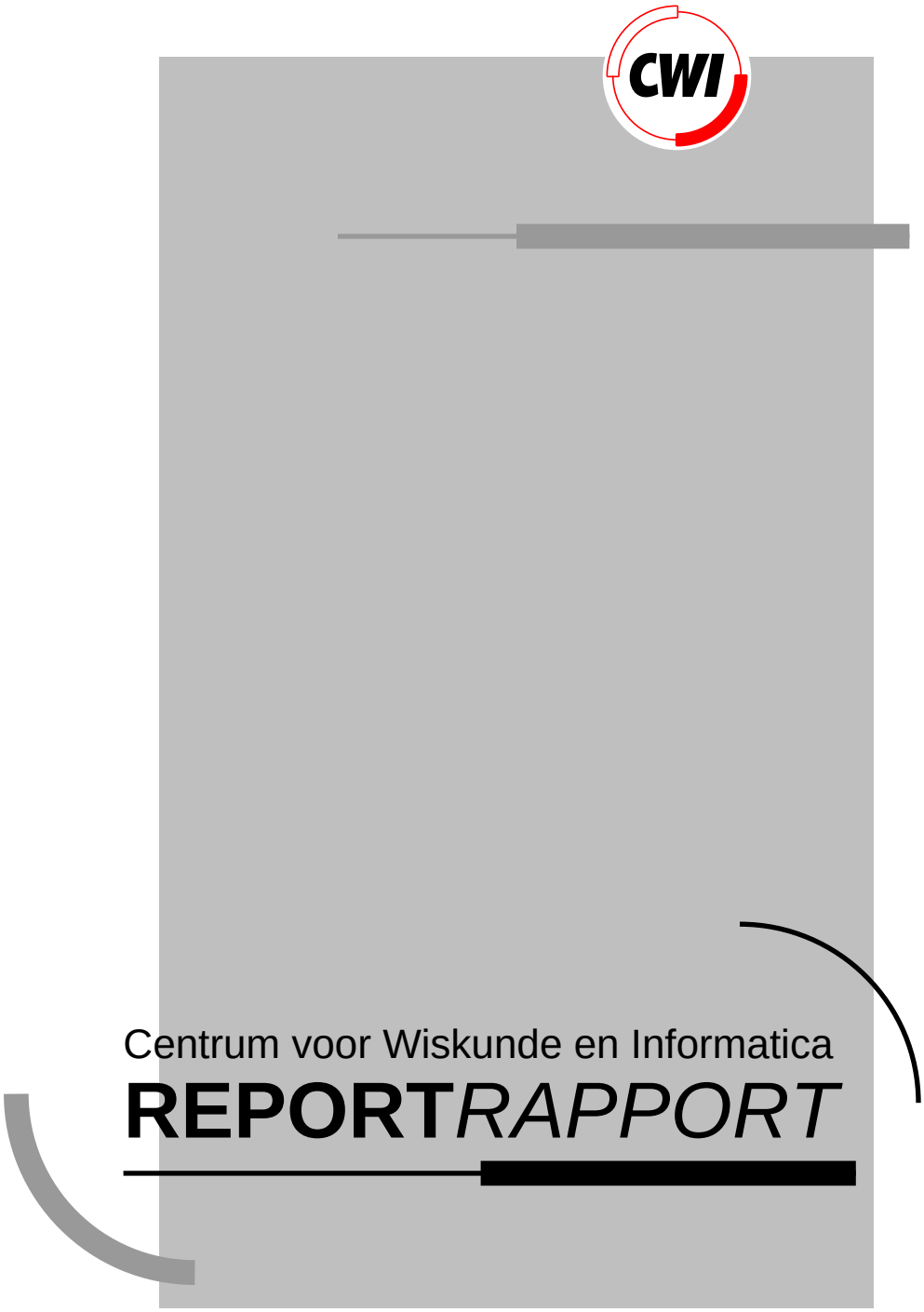

Centrum voor Wiskunde en Informatica

REPORT*RAPPORT*

The Impact of Catalogs and Join Algorithms on Probabilistic Query Optimization

J.Pellenkoft, C.A. Galindo-Legaria, M.L. Kersten

Computer Science/Department of Algorithmics and Architecture

CS-R9459 1994

The Impact of Catalogs and Join Algorithms on Probabilistic Query Optimization

Arjan Pellenkoft César Galindo-Legaria Martin Kersten

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

Abstract

Most of the work on randomized query optimization has relied heavily on the use of transformations rules for the generation of execution plans. Recently, however, we gave evidence that for the problem of choosing a join evaluation order, generating alternatives uniformly at random from the space yields solutions comparable to those obtained with transformation-intensive methods, and requires generating fewer candidate plans.

This paper presents a thorough empirical study of the impact of catalogs and join methods on the relative performance of transformation-free and transformation-based randomized optimization. Basically, our previous results remain valid for a wide variety of catalogs and relational profiles. But in contrast with the problem of selecting a join order, selecting join algorithms (e. g. hash, merge, nested-loops) seems better handled by transformations than random picking.

We then propose a two-phase approach that combines the speed of random picking with the quality of solutions of transformation-based optimization, and verify experimentally its superiority over the other algorithms, in all the search spaces considered.

CR Subject Classification (1991): [H.2.4] Database Systems, Query Processing; [H.3.3] Information Search and Retrieval; [G.2.2] Graph Theory; [F.2.2] Non-numerical algorithms and problems.
Keywords and Phrases: Randomized query optimization, transformation-based optimization, transformation-free optimization, random generation of query evaluation plans, counting of query evaluation plans.
Note: This work has been supported in part by ESPRIT-III project 7091, "Pythagoras". C. Galindo-Legaria was supported by an ERCIM fellowship.

1 Introduction

A major task of relational query optimizers is to select a suitable join evaluation order for which the estimated evaluation cost is minimum [Ull82, CP85, KRB85]. For small queries, exhaustive search is often feasible, but the number of join orders increases very fast as the number of relations grow. Heuristics and/or probabilistic algorithms are then a viable alternative. Research on probabilistic algorithms has focused on *Simulated Annealing* (SA) and *Iterative Improvement* (II), and their variations [IW87, SG88, Swa89b, Swa89a, IK90, IK91, LVZ93]. Those optimization algorithms rely heavily on transformation rules

to generate alternative join evaluation orders. The transformation rules are usually based on algebraic properties of the join evaluation orders, like *commutativity* and *associativity*, and they impose a particular topology on the search space —namely, evaluation plans are adjacent if they differ by a single application of a transformation rule. But the effect of a given topology on the behavior of search algorithms remains difficult to quantify. This prompted us to examine a transformation free (TF) optimization scheme that generates plans uniformly at random and keeps the best solution generated [GLPK94]. Our finding was that transformations tend to improve solutions "slowly," and the TF scheme converges faster and finds plans comparable to those found by transformation based optimizers.

The study in [GLPK94] was based on a calibrated cost model for the DBS3 system [ACV91] —a main memory database whose cost model accounts for CPU only— and considered execution plans with only hash-joins. In this paper we extend our previous experiments to assess the stability of the phenomenon observed. We use the same I/O-dominated cost model used at the University of Wisconsin in their randomized optimization work [IK90, Kan91]. We examine the impact of indices, changes on the statistical profiles of the catalogs, and the use of different join algorithms.

For the problem of selecting a join-order, the size of the space is exponential in the number of relations (see [GLPK95] for the exact size). When, in addition, a join algorithm is selected ($n-1$ m-ary decisions for a query on n relations with m algorithms available), the resulting search space is the product of two exponentially large spaces. So, including the selection of join algorithms has a different effect on the problem than changing the cost model or the catalog profiles. In fact, our current experiments show a qualitative difference in the relative performance of optimization algorithms when different join algorithms are allowed. The "high proportion" of good solutions in the space of evaluation orders is for the most part preserved on different catalogs and cost models, but it decreases in the product space of evaluation orders with method selection. At the same time, the transformations used in this product space seem particularly appropriate and lead to good solutions.

We then study a two-phase approach similar to those of [IK90, LVZ93], using TF in the first phase and then transformations. The behavior of this algorithm combines the fast converge of random picking with the high quality of solutions of transformation-based search, and it is superior to the other algorithms in all the spaces we considered. From the behavior of this hybrid algorithm, it appears that the neighborhood structure around a given plan, from the point of view of the transformation-induced topology, depends mostly on the cost of such plan. That is, a transformation-based search behaves roughly the same way when started on any two randomly-selected plans of the same cost.

Road map. This paper is organized as follows. In Section 2 we give definitions, details on the cost model, and the three basic search algorithms. The testbed for the experiments is described in Section 3. Section 4 describes experiments with various catalogs, and Section 5 examines multiple join algorithms. Finally, Section 6 contains experimental results on hybrid algorithm. Conclusions are given in Section 7.

2 Definitions

This section defines the search space, the basic probabilistic search algorithms used on that space, and the performance measure used for comparing the algorithms.

2.1 Search Space

Query evaluation plans. We represent a query by means of a *query graph*. Nodes of such graph are labeled by relation names, and edges are labeled by predicates. An edge labeled p exists between the nodes of two relations, say R_i, R_j, if predicate p references attributes of R_i, R_j. The *result* of a query graph $G = (V, E)$ is defined as a Cartesian product followed by relational selection: $\sigma_{p_1 \wedge \cdots \wedge p_n}(R_1 \times \cdots \times R_m)$, where $\{p_1, \ldots, p_n\}$ are the labels of edges E and $\{R_1, \ldots, R_m\}$ are the labels of nodes V.

Query evaluation plans (*QEPs*) are used to evaluate queries, instead of the straight definition of product followed by selection given above. A QEP is an operator tree whose inner nodes are labeled by a join operator and whose leaves are labeled by relations. The *result* of a QEP is computed bottom-up in the usual way. QEPs include annotations on the join-algorithm to use —e. g. nested loops, hash, merge, etc.— when several are available.

Not every binary tree on the relations of the query is an appropriate QEP, because some may require the use of Cartesian products. We restrict the search space to those QEPs that do not require products, called *valid* in [SG88]. Some systems restrict the topology of QEPs further, so that each join operates on at least one base relation. Such restriction leads to the space of *linear* QEPs. We do not impose such restriction here, so we work on the more general *bushy* space.

Tree transformations. The transformations used to generate new QEPs, where applicable, are the following [IK90, IK91]: Commutativity, $A \bowtie B \leftrightarrow B \bowtie A$; associativity, $(A \bowtie B) \bowtie C \leftrightarrow A \bowtie (B \bowtie C)$; left join exchange, $(A \bowtie B) \bowtie C \leftrightarrow (A \bowtie C) \bowtie B$; right join exchange, $A \bowtie (B \bowtie C) \leftrightarrow B \bowtie (A \bowtie C)$ and join method selection, $A \bowtie_{method_i} B \leftrightarrow A \bowtie_{method_j} B$.

2.2 Search Algorithms

We experiment with three basic search algorithms, the transformation-based Iterative Improvement and Simulated Annealing, and a transformation free algorithm. We summarize them here for completeness. More details on the transformation-based optimizers can be found in a number of references, including [KCV82, NSS86, SG88, IK90, LVZ93].

Iterative Improvement (II) performs a large number of *local optimizations*. A local optimization starts at a random QEP, called the *current* QEP. By applying a randomly selected transformation rule to the current QEP a new one is generated. If this is cheaper then it is accepted as current QEP, otherwise it is rejected. A local optimization stops when a local minimum has been reached. The II algorithm stops as soon as a predefined number of plans has been generated. The plan found with the lowest cost is returned as the result. Figure 1 shows the pseudo-code of the II algorithm.

To detect a local minimum the neighbors are not searched exhaustively but a *r-local minimum* is used [Kan91], i.e. a plan is a local minimum if none of r randomly selected neighbors has a lower cost. Since the plans are selected at random, and repetitions are possible, a r-local minimum is not guaranteed to test all neighbors. In the experiments r is set to the number of neighbors of a node.

```
PROCEDURE II() {
  minS = infinite; // with cost(infinite) = infinite
  WHILE not (stopping_condition) DO {
    S = random state;
    WHILE not (local_minima(S)) DO {
      S' = random state in neighbors(S);
      if cost(S') < cost(S) THEN S = S';}
    IF cost(S)<cost(minS) then minS = S;}
  return(minS);}
```

Figure 1: Iterative Improvement

```
PROCEDURE SA(){
   S = S0;
   T = T0;
   minS = S;
   WHILE not(frozen) DO {
     WHILE not(equilibrium) DO {
       S' = random state in neighbors(S);
       deltaC = cost(C') - cost(S);
       IF (deltaC <=0) THEN S = S';
       IF (deltaC > 0) THEN S = S' with probability e^(-deltaC/T);
       IF cost(S)<cost(minS) THEN  minS = S;}
     T = reduce(T);}
   return(minS)}
```

Figure 2: Simulated Annealing

Simulated Annealing (SA). Sometimes the II algorithm fails to find good plans because it gets stuck in high cost local minima. SA attempts to solve this problem by also accepting new QEPs with a higher cost, with some probability. The SA algorithm starts at a random QEP and randomly generates next QEPs. The probability of accepting QEPs with higher cost decreases as time progresses. When a predefined number of plans has been generated or a "stable condition" has been reached the SA algorithm stops.

Figure 2 shows the pseudo-code of the SA algorithm. The *frozen* and *equilibrium* conditions used in our experiments are those given in [Kan91].

If time is infinite both transformation based search algorithms will find the global minimum, but in practice the resource available for optimization are limited and must be used as efficiently as possible.

Transformation Free (TF). To remove the reliance on transformation rules, and a potentially slow quality improvement, the TF algorithm was investigated in detail in [GLPK94]. This algorithm generates QEPs uniformly at random, and keeps track of the one

```
PROCEDURE TF(){
  minS = random state;
  WHILE not(stop_condition) DO {
    S = random state;
    IF cost(S)<cost(minS) THEN minS = S;}
  return(minS)}
```

Figure 3: Transformation Free

with the lowest cost. The algorithm terminates after it has generated a predefined number of QEPs. The QEP with the lowest cost is returned as preferred plan for execution. Like II and SA, if TF is given infinite time it will find the global minimum. But unlike SA and II, if time is finite TFs performance only depends on the cost distribution over the search space and not on the topology imposed on the space by the transformation rules. Figure 3 shows the pseudo-code of the TF algorithm. Note that the random states are chosen *uniformly* from the space. See [GLPK95] for details on how this is achieved.

2.3 Performance Measure

The behaviour of an optimization strategy can be represented by a function mapping the number n of plans explored to the estimated cost of the best plan found. For a given algorithm A, we call this cost the *solution* after n, and denote it by S_n^A. Formally, using U_n^A as the set of the first n plans visited by A, the solution after n is:

$$S_n^A = min\{cost(p) \mid p \in U_n^A\}.$$

For transformation-base algorithm, every valid plan generated by the algorithm is counted as explored, even if it is not accepted by the algorithm (e. g. because its cost is higher than the current plan in II).

Since the algorithms are probabilistic, U_n^A is a random subset of size n from the search space, and therefore S_n^A is a random variable. Based on this, we measure the success of these algorithms using the mean and standard deviation of the solution. As n increases, the mean of S_n^A should approach the minimum cost in the search space; while at the same time the standard deviation of S_n^A approaches zero. The second condition ensures that the algorithm, though probabilistic, behaves in a stable way. Although the number of plans explored does not account for all the resources required by an algorithm, we use this solution after n as an *implementation-independent* measure of algorithm performance.

3 Testbed

To assess the stability of the TF search algorithm we conducted a large number of experiments with the I/O-based cost model of [Kan91] and queries and catalogs that were also used in our earlier work with the DBS3 cost model [ACV91].

The new cost model is used exhaustive to study the impact of the catalogs and the available join methods on the performance of TF, II, and SA. The queries used in the

Catalog	Cardinality	Percentage of unique values in attribute
catalog 1	1000	[0.9,1.0]
catalog 2	[1000,100000]	[0.9,1.0]
catalog 3	[1000,100000]	[0.1,1.0]

Figure 4: Sizes and selectivities of the *original* catalogs

experiments are randomly generated and acyclic. They range form 4 to 20 joins and all join predicates are equality joins. These queries were optimized for three catalogs with different variance in attribute values and relation size. The queries and catalogs used in [IK91] constitute our starting point and in the sequel of this paper these catalogs will be referred to as the *original* catalogs.

3.1 Cost Model

The cost model called CM2 in [Kan91] is the basis for our experiments. This cost model assumes a disk-based database system. Since the cost of evaluating a QEP is dominated by the I/O, the number of pages that are read or written during the evaluation of a QEP is used as cost metric. A large buffer is assumed in the cost model. The major difference with the DBS3 cost model used in our previous work, is that the DBS3 model assumes a main memory database system, in which the CPU cost is the predominant factor.

The CM2 cost model is able to handle three join algorithms, namely *nested-loop, merge-scan* and *hash-join*. The cost functions for the nested-loops algorithm are *page-level* nested-loops join and *index-scan* nested-loop. The cost of the cheapest alternative is returned as cost for a nested-loop join. The cost of the merge-scan join consist of sorting the inputs, if they are not already sorted, and by merging the two input streams. The hash-join also has two alternatives of which the one with the cheapest cost is returned. These two alternatives are *simple hash-join* and *hybrid hash-join*. In the computation of the cost for the hash-join it is assumed that the hash table is build on the smallest input.

When an index is available for a join attribute it can be used to reduce the loading cost. The usual assumption is made that the attribute values are uniformly distributed and that the columns values are independent.

3.2 Factors Considered

The factors considered in our study are the following:

- Catalog variance (the difference in relation size and join selectivity).
- Relation sizes (original catalogs or enlarged catalogs).
- Indices (present or not).
- Join algorithms (nested-loop, hash-join, merge-scan).

The catalogs used are randomly generated from a profile that specifies an allowed range for relation sizes and uniqueness of attributes. Figure 4 gives the profiles for the three types of catalogs used. For example, a catalog of type 2 (or simply catalog 2) uses relation sizes

ranging from 1,000 to 100,000 tuples and the uniqueness of the attribute values range from 90% to 100%. This percentage of unique values is used for the computation of the join selectivity in the cost estimation. The ranges are chosen such that the *variance* in catalog 1 is small, and it is increased in catalogs 2 and 3.

The *enlarged* catalogs are constructed by multiplying the relation sizes in the original catalog (Figure 4) by a hundred. These enlarged catalogs were used to study the impact of the large I/O buffer in the cost model and possible non-linear behaviour of the cost functions. For both the original and the enlarged catalogs we used two variants; one with many indices and one without indices. They are used to check the hypothesis that indices have a strong effect on the shape of the search space and, therefore, affect the performance of the search algorithms.

In the experiments discussed in Section 4 there is only one join method available for a single QEP. So all of join operators in a QEP are either nested-loop, merge-scan or hash-join. Section 5 and 6 describe experiments in which the plans considered combine different join algorithms.

3.3 Performance Characteristics

These graphs showing our results present the average of solutions found by the various algorithms after exploring a given number of plans. As is usual in the work on this subject, the y-axis is a linear measure of *scaled cost*, with a scaled cost of 1 for the cheapest individual plan found by any algorithm, in the given search space.

These graphs have some properties useful for the comparison of search algorithms. A general description of the graph of TF and II is as follows. Up to a *crossover point* the TF algorithm generates better plans, and after that the II algorithm finds better plans. This crossover point marks the solution that is found by both algorithms after exploring the same number of plans.

After exploring a many plans, the cost of solutions found by probabilistic algorithms improves very slowly. We could say that at some point the optimizer becomes *stable* and call the quality of the plan at that point the *final cost*. The difference in final cost is used to compare algorithms.

Another important characteristic of the graph is the *cost range*. If the difference between the best solution and the worst solution in the search space is small, the optimization has a relatively smaller impact on the execution time of the query. If, on the other hand, the cost range is large, the optimizer can produce a dramatic improvement on query performance.

These three aspects —*crossover point, final cost (difference)* and *cost range*— of a performance graph are helpful in analysing the performance of the search strategies. In Figure 5 these aspects are marked in a skeleton performance graph.

4 Effect of Variance, Indices, and Relation Sizes

This section discusses the experiments done to determine the circumstances for which the random generation of plans is comparable to the transformation based approach. The sequel of this section describes the behaviour of TF, II and SA for various catalogs and join methods. All graphs shown are averages over a large number of runs.

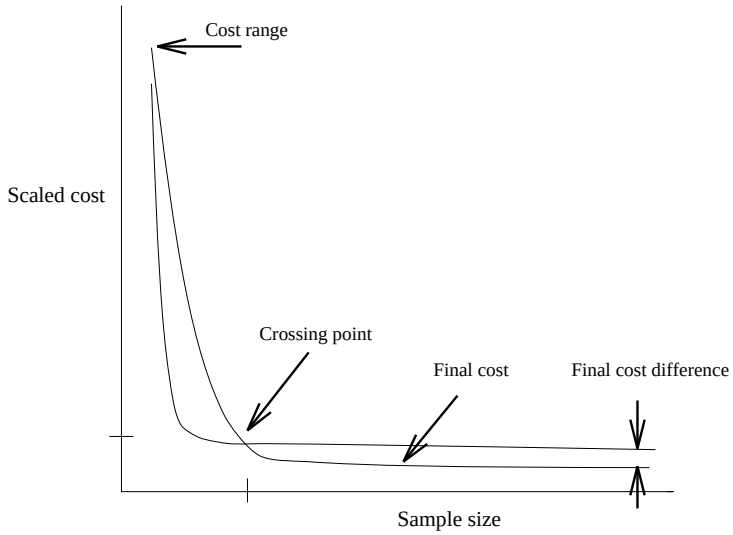

Figure 5: Skeleton performance graph

4.1 Catalogs with Indices

The original catalogs with indices are used for our first experiment. As mentioned in Section 3 the optimizers only consider QEPs in which all join algorithms are either nested-loop, merge-scan or hash-join.

We observed that as the catalogs changed, from low variance to high variance, the *cost range* of the graphs increased and the *crossover point* shifts to the left. The *final cost* of the II and TF algorithm are similar for catalogs 2 and 3. Only for the low-variance catalog 1 the II algorithm is consistently better. For the high-variance catalogs the relative performance TF algorithm is best. Figure 6 illustrates shows the results for a query of 20 joins when only hash-joins are considered (the results for nested-loops and merge-join are similar).

To our surprise the QEPs with only nested-loops join were the cheapest in absolute cost. A closer examination of the QEPs generated showed that the large buffer size, relative to the size of the relations involved, caused this effect. Most of the processing can be done such that the inputs to the join operator fit in the buffer, so the nested loops algorithm does not require any reloads. Due to the overhead cost of the other two algorithms they resulted in more expensive QEPs.

4.2 Catalogs without Indices

We drop all indices in the next round of experiments, to test the assumption that indices reduce the *cost range* and make the search space *smoother*. That is, the *cost difference* with neighbors becomes smaller.

Surprisingly, for the experiments conducted, the performance difference between search algorithms in spaces with indices or without is small. But an interesting change can be observed for the high variance catalogs in Figure 7. Compared to the indexed catalogs the *crossover points* shifted slightly to the right for all join methods and the quality of the plans at the *crossover points* is better. The *cost range* of the graphs and the *final costs* are similar to those of the indexed catalogs.

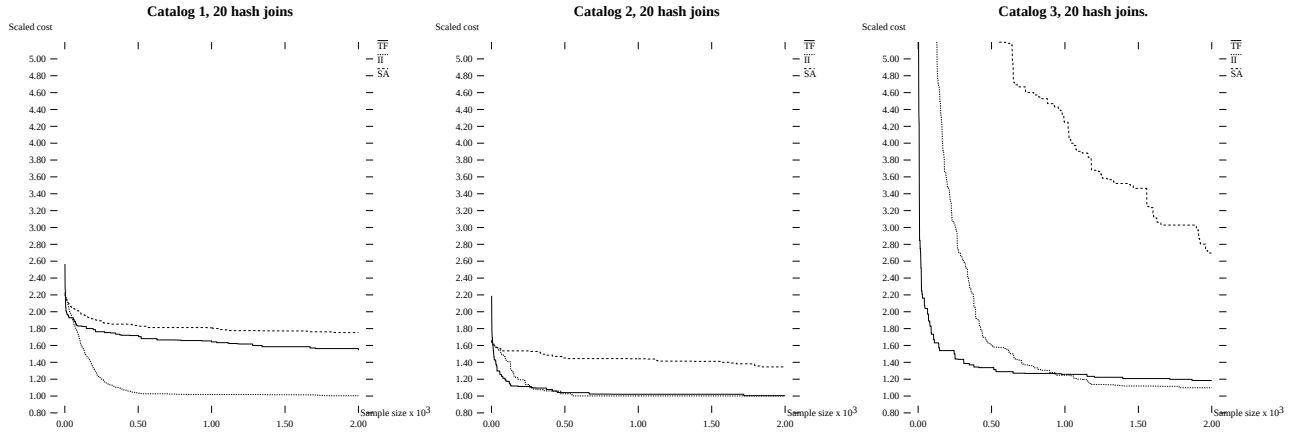

Figure 6: Space of hash QEPs for the original catalog with indices

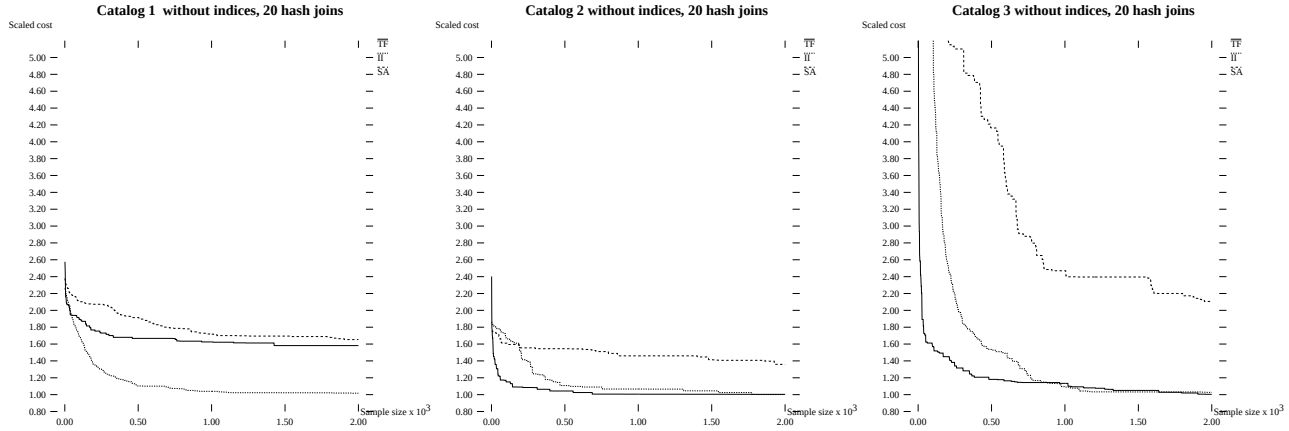

Figure 7: Space of hash QEPs for the original catalogs without indices

Our experiments, then, lead to the conclusions that although indices have a noticeable impact on optimization performance, it is relatively small compared to the impact of the catalog variances or variance in join selectivity.

4.3 Large Catalogs with Indices

To examine the impact of the large buffer on the performance of the search algorithms, we enlarged the relation sizes of the original catalogs. For these big relations the QEPs with only hash-joins were consistently cheaper than QEPs with only merge scan or nested loop. This search space of QEPs, with only hash joins, also showed the biggest change in performance. For catalog 2 TF finds plans much faster than II and also the distance between the graphs has grown in comparison to the original catalog 2. For catalog 3 the TF algorithm improves faster before the *crossover point*, but this *crossover point* has a high cost.

With the enlarged catalogs both the cost and the difference between final costs has grown. To make the performance graphs of the search algorithms visible, the scale of the

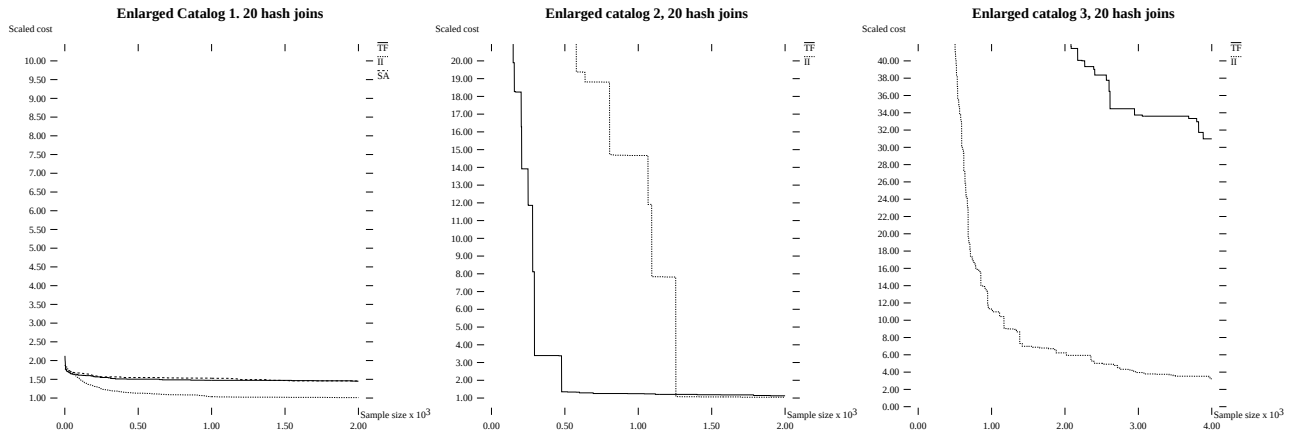

Figure 8: Space of hash QEPs for enlarged catalogs with indices

y-axis was enlarged by a factor of ten. In Figure 8 the performance graphs of the search algorithms are given for the tree catalogs when only hash-joins are used.

We also ran experiments for enlarged catalogs *without* indices. The results are basically the same as those presented for the spaces with indices, so they are not shown here.

5 Effect of Multiple Join Algorithms

We now consider the use of multiple join algorithms in QEPs. To deal with this case in transformation based strategies, a rule is added that changes the algorithm at a specific join operator. Such addition leads to a dramatic growth of the search space. If m join algorithms are considered and the QEPs joins n relations, each QEP in the original search space is mapped to m^{n-1} QEPs with join selection. This big search space seems to contain cheaper QEPs —e. g. a hash-join whose inputs are sorted can be replaced by a merge-scan— but it also introduces many QEPs with higher cost. Important for the performance of all three search algorithms is how the cost distribution changes, and for transformation based optimizers also the modified connectivity of the search space.

Uniformly random generation of elements from the enlarged space is easy —modulo the uniform generation of evaluation orders. Simply select independently and uniformly a join algorithm for each join in the QEP.

Figure 9 shows the performance graphs for the three search methods using all join algorithms. For reference, we also show the result of II and TF on the restricted space of plans that use hash-joins only. The effect of the enlarged space is clear from this graph. Initially, both TF and II progress about as quickly in the space restricted to hash-joins as in the more general space. But then TF becomes stable in more costly solutions when it has to select a join algorithm, while II finds better solutions when selecting a join algorithm.

We can conclude that the reduced percentage of good plans in the bigger space has a negative effect on the performance of the TF algorithm. However, the topology imposed by the change-join-algorithm transformations seems particularly appropriate for a transformation-based search.

In the following section we show experiments in which random generation and the use

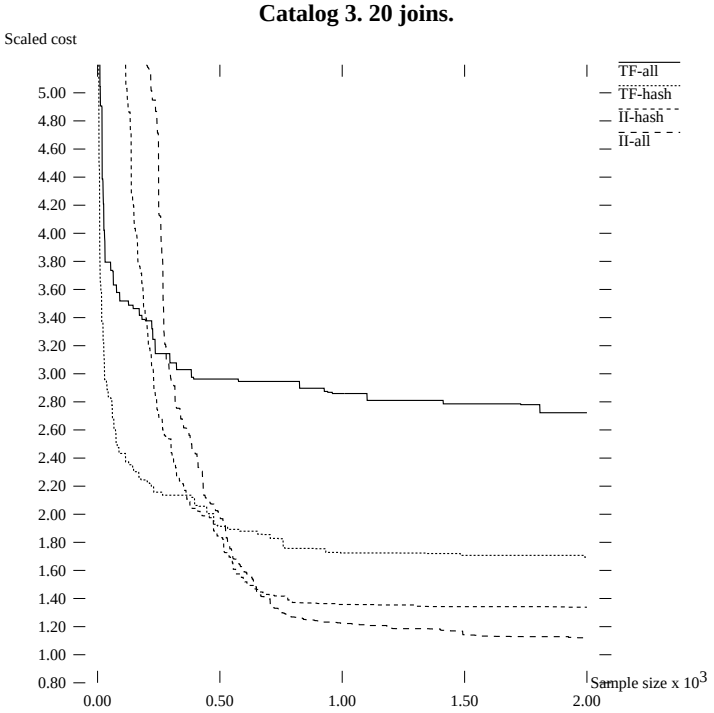

Figure 9: Multiple join methods

of transformation rules are mixed. Ideally these methods should incorporate the good behaviour of both the TF and II algorithm, fast convergence and good final plans.

6 Hybrid Algorithms

Considering all experiments performed, an improvement of transformation based optimizers seems feasible by balancing the generation of random plans with the application of transformations. Other multi-phase optimization schemes have been proposed in [Kan91, LVZ93], but they still rely mainly on transformations to generate alternatives.

It is reasonable to consider starting the search by generating a predefined number of plans (TF-phase), followed by one transformation-based local optimization. During this local optimization phase no new random starting points are generated. A generalization of this idea is what we call the *Set-based Iterative Improvement* (SII_n) algorithm. This hybrid algorithm is an II algorithm that uses the best plan of a randomly generated set as starting state for a local optimization. The n represents the size of the randomly generated start set. Figure 10 shows the pseudo-code of the algorithm.

Figure 11 shows the performance of SII_{100}, as well as TF and II for the space of join ordering, when using the enlarged catalog 3. The graph of the SII_{100} algorithm reflects the behaviour of both the TF and II algorithm. It converges as fast as the TF graph in the first part of the graph and then picks up the behaviour of the II algorithm, resulting in very good quality plans. Figure 11 is typical for the behaviour of the SII algorithm.

Figure 12 shows the performance of SII_{100} on the space of join ordering plus join-

```
PROCEDURE SII(n) {
  minS = infinite; // with cost(infinite) = infinite
  WHILE not (stopping_condition) DO {
    S = random state
    FOR i = 1 TO n - 1 DO {
      S' = random state;
      IF cost(S') < cost(S) THEN S = S';}
    WHILE not (local_minima(S)) DO {
      S' = random state in neighbors(S);
      IF cost(S') < cost(S) THEN S = S';}
    IF cost(S)<cost(minS) then minS = S;}
  return(minS);}
```

Figure 10: Set-Based Iterative Improvement

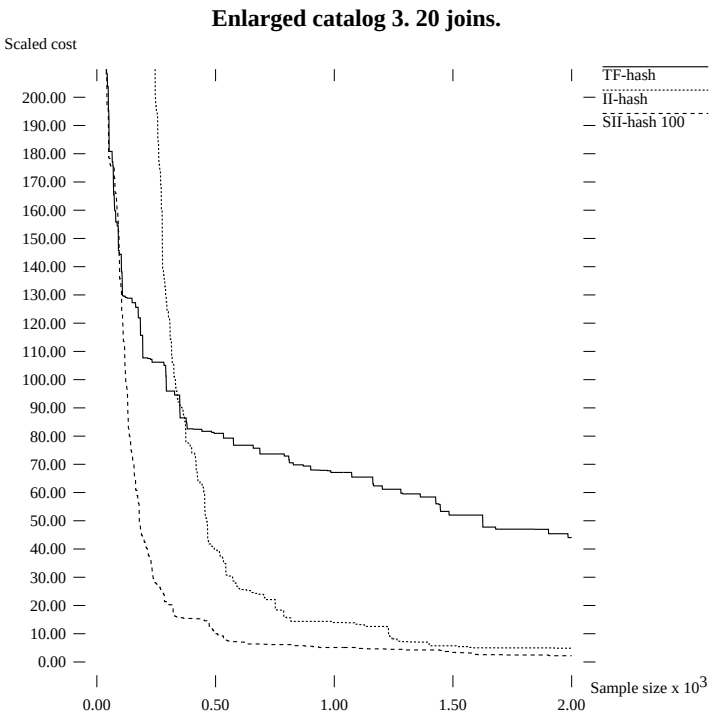

Figure 11: Hybrid search on the restricted space of hash-joins

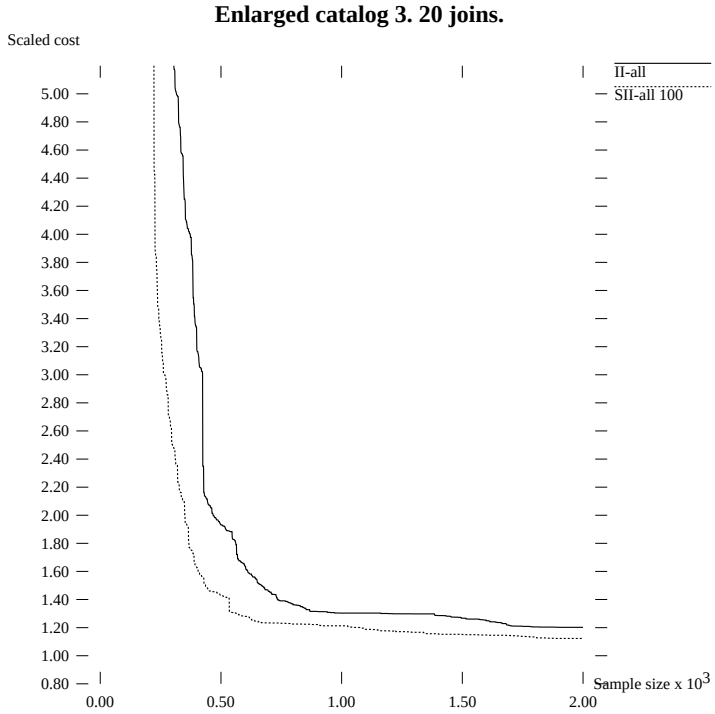

Figure 12: Hybrid search using all available join algorithms

algorithm selection, also in combination with enlarged catalog 3. Although the TF algorithm has a weak performance for this search space, the SII_{100} algorithm maintains its good behaviour.

7 Conclusions

In this paper we examined the impact of several factors on the performance of probabilistic query optimization algorithms, in particular the relative behavior of random picking of solutions with respect to transformation-based search. The results of random picking give a direct indication of the proportion of good solution in the search space, while the transformation-based search also depends on the topology imposed by the specific set of transformations used.

Our experiments show that the results obtained in [GLPK94] for a main-memory database remain valid, for the most part, when the I/O-based cost model of [IK90, Kan91] is used instead. A transformation-free algorithm finds good plans faster than a transformation-based approach, but the transformation-based search finds the best plans in the end. This happens because the ratio of good plans is substantial and the topology imposed by associativity/commutativity/exchange transformations does not seem to aid the search significantly, especially at the beginning of the process. We observed that the presence of indices does not reshape the search space, and affects only marginally the performance of all the search methods.

We then studied the effect of selecting a join algorithm, in addition to a join evaluation

order. In this case the search space becomes the product of two exponentially large spaces, and its properties turn out to be qualitatively different from those of selection of a join order evaluation alone. The proportion of good plans decreases in this combined space, and at the same time the topology induced by the change-algorithm rule seems to favor the transformation-based search.

Finally, we described and tested a two-phase optimization approach that starts with random picking to generate good plans quickly, and then applies transformations for further refinement. The result is a combination of the best of both search strategies: fast convergence to solutions of very high quality. We believe this hybrid approach is basically the best alternative in a *purely stochastic search* —i. e. one that does not consider heuristics— probably with an additional Simulated Annealing phase at the end as suggested in [IK90].

There are related issues that remain to be addressed. The first is how to incorporate heuristics in a robust manner. In our view, the use of heuristics in randomized search must be that of "rigging the odds" in favor of the better plans. We are in the process of formulating the necessary framework. Also, the two specific spaces identified in this paper on which the transformation-based and transformation-free schemes behave significantly differently provide a test case for the study of when and how are transformations advantageous for optimization.

Acknowledgements. To conduct the experiments reported on this paper, we coded the uniformly-distributed generation of join trees, and the TF and hybrid algorithms on top of the code for randomized query optimization developed at the University of Wisconsin [IK90, Kan91]. We are grateful to Yannis Ioannidis for kindly providing us with a copy of their software, and for allowing us to modify it for our experiments.

References

[ACV91] F. Andrès, M. Couprie, and Y. Viémont. A multi-environment cost evaluator for parallel database systems. *Procedings of the 2nd Int. DASFAA Japan*, 1991.

[CP85] S. Ceri and G. Pelagatti. *Distributed Databases: Principles and Systems.* McGraw-Hill, New York, 1985.

[GLPK94] C. A. Galindo-Legaria, A. Pellenkoft, and M. L. Kersten. Fast, randomized join-order selection —Why use transformations? In *Proceedings of the Twentieth International Conference on Very Large Databases, Santiago*, 1994. Also CWI Technical Report CS-R9416.

[GLPK95] C. A. Galindo-Legaria, A. Pellenkoft, and M. L. Kersten. Uniformly-distributed random generation of join orders. In *Proceedings of the International Conference on Database Theory, Prague*, 1995. Also CWI Technical Report CS-R9431.

[IK90] Y. E. Ioannidis and Y. C. Kang. Randomized algorithms for optimizing large join queries. *Proc. of the ACM-SIGMOD Conference on Management of Data*, pages 312–321, 1990.

[IK91] Y. E. Ioannidis and Y. C. Kang. Left-deep vs. bushy trees: An analysis of strategy spaces and its implications for query optimization. *Proc. of the ACM-SIGMOD Conference on Management of Data*, pages 168–177, 1991.

[IW87] Y. E. Ioannidis and E. Wong. Query optimization by simulated annealing. *Proc. of the ACM-SIGMOD Conference on Management of Data*, pages 9–22, 1987.

[Kan91] Y. C. Kang. *Randomized Algorithms for Query Optimization.* PhD thesis, University of Wisconsin-Madison, 1991. Technical report #1053.

[KCV82] S. Kirkpatrick, C.D. Gelatt, Jr., and M.P. Vecchi. Optimization by simulated annealing. Technical Report RC 9355, IBM Thomas J. Watson Research Center, Yorktown, 1982.

[KRB85] W. Kim, D. S. Reiner, and D. S. Batory, editors. *Query processing in database systems.* Springer, Berlin, 1985.

[LVZ93] R. S. G. Lanzelotte, P. Valduriez, and M. Zaït. On the effectiveness of optimization search strategies for parallel execution spaces. *Proc. of the 19th VLDB Conference, Dublin, Ireland*, pages 493–504, 1993.

[NSS86] S. Nahar, S. Sahni, and E. Shragowitz. Simulated annealing and combinatorial optimization. *23rd Design Automation Conference*, pages 293–299, 1986.

[SG88] A. N. Swami and A. Gupta. Optimization of large join queries. *Proc. of the ACM-SIGMOD Conference on Management of Data*, pages 8–17, 1988.

[Swa89a] A. N. Swami. *Optimization of Large Join Queries.* PhD thesis, Stanford University, 1989. Technical report STAN-CS-89-1262.

[Swa89b] A. N. Swami. Optimization of large join queries: Combining heuristics and combinatorial techniques. *Proc. of the ACM-SIGMOD Conference on Management of Data*, pages 367–376, 1989.

[Ull82] J. D. Ullman. *Principles of Database Systems.* Computer Science Press, Rockville, MD, 2nd edition, 1982.