Centrum voor Wiskunde en Informatica

REPORTRAPPORT

Fast, randomized join-order selection - Why use transformations?

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

Computer Science/Department of Algorithmics and Architecture

CS-R9416 1994

Fast, Randomized Join-Order Selection —Why Use Transformations?

César Galindo-Legaria   Arjan Pellenkoft   Martin Kersten

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

Abstract

We study the effectiveness of probabilistic selection of join-query evaluation plans *without* reliance on tree transformation rules. Instead, each candidate plan is chosen uniformly at random from the space of valid evaluation orders. This leads to a transformation-free strategy where a sequence of random plans is generated and the plans are compared on their estimated costs. The success of this strategy depends on the ratio of "good" evaluation plans in the space of alternatives, the efficient generation of random candidates, and an accurate estimation of their cost.

To avoid a biased exploration of the space, we solved the open problem of efficiently generating random, uniformly-distributed evaluation orders, for queries with acyclic graphs. This benefits any optimization or sampling scheme in which a random choice of (initial) query plans is required.

A direct comparison with iterative improvement and simulated annealing, using a proven cost-evaluator, shows that our transformation-free strategy converges faster and yields solutions of comparable cost.

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: C. A. Galindo-Legaria had an ERCIM postdoctoral fellowship while conducting this work.

1 Introduction

Query optimizers must find an "optimal" execution plan from a space of many semantically equivalent alternatives. For join queries, the space of feasible evaluation orders grows very quickly, and to find an optimal plan, known deterministic search algorithms take exponential time on the number of relations of the query [OL90]. This combinatorial explosion make heuristics and probabilistic algorithms the prime vehicle for query optimization. *Simulated Annealing* (SA) and *Iterative Improvement* (II) are commonly used as reference points for research in this area [IW87, SG88, Swa89b, Swa89a, IK90, IK91, LVZ93].

The probabilistic search algorithms SA, II, and their variations rely heavily on transformation rules to generate candidate execution plans. These transformations are based on properties of the underlying algebra, such as commutativity and associativity of the relational join. The performance of these algorithms depends, in addition to the cost distribution in the search space, on the set of transformations being used. In particular, a *complete* set of transformations —i. e. one that is sufficient to transform a starting plan into any other plan in the space— does not guarantee good behavior, and it is sometimes necessary to add redundant transformations to improve the performance of algorithms [IK90].

Several sets of transformation rules have been studied, but the extent to which they allow rigorous analysis and prediction of the behavior of transformation-based algorithms is somewhat limited —rather, they serve to provide qualitative insight [IK91]. A question that motivates the present work is the following: if we are allowed to explore only a limited, fixed number of plans, then what is more likely to produce good plans, the application of transformations or a random selection from the complete space?

The distribution of cost in the search-space of join queries is the focus of [Swa91], which concludes that the proportion of "good" plans decreases quickly as the number of relations in the query increases. But even if "good" plans were only 1% of the space, random selection of 70 plans will produce a "good" one with 50% probability —i. e. a coin toss. The results in [IK91] show that the proportion of "good" plans also depends on the relative size of relations, and they give evidence that this proportion is not insignificant.

Then, since there is evidence of a considerable number of "good" plans in the space, and given the difficulties in analyzing how transformations lead to them, we study the behaviour of a *transformation-free optimization algorithm.* This algorithm generates a sequence of random plans and applies a calibrated cost-evaluator to estimate their cost. Then the plan with minimal cost becomes the preferred plan of execution.

We start our experiments by exploring exhaustively the search space of small queries, to check the ratio of "good" plans —i. e. those within a given factor times the optimum plan. This CPU-intensive exercise shows the cost distribution over the search space. The results coincide with those of other, similar studies [Swa91].

In order to explore the search space, we solve the problem of efficiently generating random, uniformly-distributed execution plans, for acyclic queries. In the process, we also determine the exact number of valid join-orders for a given query graph. Results on counting and efficient uniform generation of random plans were previously known only for query graphs with a very "uniform" topology, such as star, chain, and clique [IK91, LVZ93]; for other graph topologies, unbiased random generation of a single plan could take several minutes of CPU time for a query of 20 relations [Swa89b, Swa91].

Results of our experiments favor a transformation-free optimization algorithm in a direct comparison with the transformation-based SA and II, for the problem of join-order selection. The surprising observation is that our algorithm converges after exploring fewer execution plans than the others, finding plans of comparable cost.

Road map. Section 2 presents basic definitions, and describes the queries and cost estimation functions used in our experiments. Section 3 presents results on exhaustive exploration of search spaces for small queries. Section 4 describes algorithms for random generation of plans. Section 5 studies the quality of several random sampling methods. Section 6 compares query optimization strategies, transformation-free with SA and II. Section 7 presents our conclusions, a comparison with related work, and some directions for future research.

2 Our framework

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 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 also include annotations on the join-algorithm to use —e. g. nested loops, hash, merge, etc.— when several are available. The results presented in this paper are only for the *hash-join* algorithm. [1]

Not every binary tree on the relations of the query is an appropriate QEP, because some may require the use of Cartesian products. Those that do not require products are called *valid* [OL90], and their topology is captured as follows. Given a connected query graph G, an unordered binary tree T is called an *association tree* of G, when it satisfies the following recursive definition: The leaves of T correspond one-to-one with the nodes of G, and every subtree of T is an association tree of a connected subgraph of G.

Association trees are unordered —i. e. do not distinguish left from right subtree— because not all join-algorithm distinguish a left and right argument [Gra93]. For those who do, we consider ordering the tree as part of the join algorithm selection. Ordering a tree of n leaves requires a binary choice in each of the $n - 1$ internal nodes, so there are 2^{n-1} ordered trees for each unordered tree of n relations.

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, so we work on the more general *bushy* space. Studies presented in [IK91] suggest that the space of bushy plans has a higher number of good plans.

Tree transformations. Our implementation of the traditional transformation-based algorithms uses the tree transformations of [IK90, IK91] for bushy trees, except for algorithm selection, because we use only hash-join. The transformation rules are: Commutativity, $A \bowtie B \leftrightarrow B \bowtie A$; associativity, $(A \bowtie B) \bowtie C \leftrightarrow A \bowtie (B \bowtie C)$; left join exchange, $(A \bowtie B) \bowtie C \leftrightarrow (A \bowtie C) \bowtie B$; and right join exchange, $A \bowtie (B \bowtie C) \leftrightarrow B \bowtie (A \bowtie C)$. Only those transformations that lead to a valid QEP can actually be used on a given tree.

Queries used. The experiments foreseen require care in the design and population of the test databases. Traditional benchmark databases, such as Wisconsin and AS3AP, are primarily geared towards performance assessment of the algorithms in relation to the architecture. Moreover, their database statistics do not necessarily reflect real-world applications,

[1] We did experiments with *nested-loop* join as well, and the results are similar to those for hash-join.

which make them less suitable for assessing the quality of an optimizer. On the other hand, designing a new benchmark database complicates comparison with published results.

As in [LVZ93], we run our experiments against the Portfolio Club Experimental Model (PEM). This model was designed to provide a realistic experimental application base for complex query definition, evaluation, and benchmarking in the EDS project [Val92]. Within this database, we consider several acyclic queries containing from 4 to 12 relations, and 3 catalogs with different database profiles. Both the queries and the catalogs used in [LVZ93] constitute our starting point.

Cost model. We used the Analytical Performance Evaluator (APE) developed and tuned for DBS3 [ACV91]. The APE tool provides accurate costs for query plan execution on the DBS3 prototype. In particular, the cost model has been calibrated towards this system architecture for nested loop joins [AKK93]. The cost model for hash-join is similar to that used for main-memory databases in [Kan91].

Systems used. Our experiments were performed on a Silicon Graphics Challenge Series machine, with 6 processors running at 150MHz. Optimization algorithms were programmed in SWI Prolog [Wie92].

3 Cost distribution in search spaces

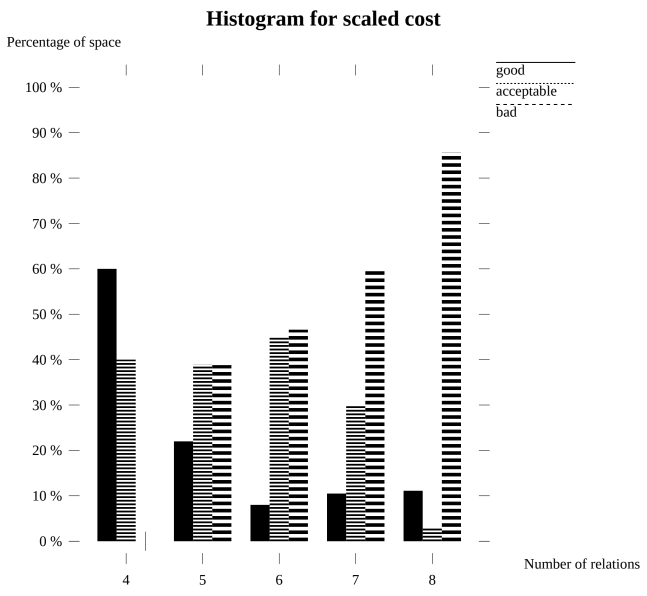
This section presents our results on the cost distribution of small spaces, obtained by exhaustive search. Of particular interest is the ratio of "good" plans, since it is an important factor for a transformation-free optimization strategy.

What is a good plan? A *good* QEP is one whose estimated cost is within a given factor times the optimum plan. Here we follow the classification of [Swa91], where plans are considered *good* when their cost is up to twice the optimum; they are *acceptable* when their cost is more than twice, but up to ten times the optimum; and *bad* otherwise. We also use his format in the presentation of our histogram of good plans.

Exhaustive generation of QEPs. Since we consider queries with connected, acyclic query graphs, removal of any edge disconnects the graph, leaving two connected graphs. The set of valid QEPs can be enumerated by recursively splitting a query graph G as follows. If the graph has one node, then the only QEP is the relation that labels such node; otherwise remove an edge, say labeled p, to disconnect the graph, then find QEPs Q_1, Q_2 for the two connected graphs that remain, and finally return $(Q_1 \overset{p}{\bowtie} Q_2)$ as a QEP for G.

If the selection of edges is done such that at each recursion level all possible splittings of a graph are considered, then all valid QEPs are generated. Using the backtracking facility of Prolog, the algorithm takes only a few lines of code.

Results. Figure 1 shows the cost distribution in the space of valid QEPs for queries having from 4 to 8 relations, all on the same catalog. It is representative of the distributions we found on all our catalogs. For these spaces, the number of good plans is sufficiently large that a run of several tens of randomly selected plans will hit a good one with high probability.



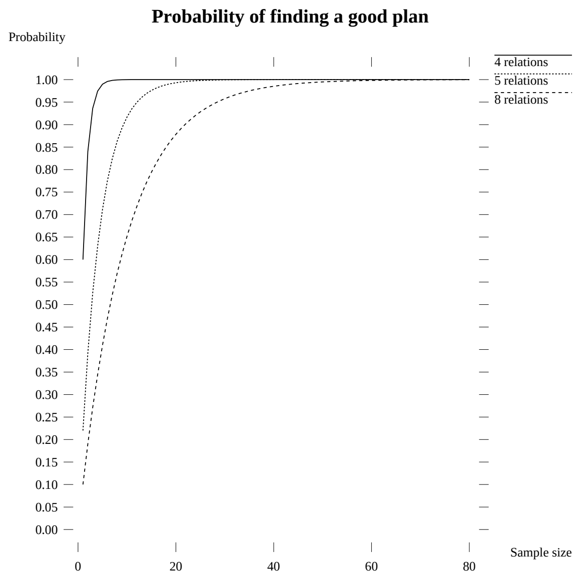


Figure 1: Histograms and probability of finding a good plan in a random sample.

Figure 1 shows this probability as a function of the sequence length, for the queries of 4, 5 and 8 relations.

We coincide with [Swa89b] in observing that, in general, the percentage of good plans decreases as the number of relations increases —although this percentage also depends on the particular query, as shown in the graph for query 6, in which the ration of good queries is slightly lower than that of queries 7 and 8. We do point out that this percentage is still significant to apply a transformation-free optimization approach.

4 Random generation of QEPs

For large queries it becomes impossible to explore the search space exhaustively, so we are forced to use sampling. In this section we describe *quasi*-random techniques as well as uniformly-distributed generation of random plans.

Quasi random QEPs. The following two procedures generate *quasi*-random QEPs. They are easy to implement, but either do not guarantee uniform probability over the space, or else take a very long time.

Random-walk. This procedure is based on transformation rules to move from one valid plan to another. If we start at some plan in the space and successfully apply transformation rules at random, we get a sample from the space being explored.

first choice	second choice	plan	probability of generation
(b, c)	-	$(a \bowtie b) \bowtie (c \bowtie d)$	1/3
(a, b)	(c, d)	$(a \bowtie (d \bowtie (b \bowtie c)))$	1/6
	(b, c)	$(a \bowtie (b \bowtie (c \bowtie d)))$	1/6
(c, d)	(a, b)	$(d \bowtie (a \bowtie (b \bowtie c)))$	1/6
	(b, c)	$(d \bowtie (c \bowtie (a \bowtie b)))$	1/6

Figure 2: Generation of quasi random QEPs by edge selection.

Random-edge selection. This procedure is similar to the technique used in section 3 to generate plans exhaustively. Here, instead of considering all possible graph splittings, we split the graphs by randomly selecting an edge at each step, which results in a random plan.

Random walks in graphs have been widely studied —see, for example, [Rag90]. In particular, if all nodes in a graph have equal degree, as is the case here [Kan91], then we are equally likely to be at any node of the graph after n steps, for a sufficiently large n, regardless of the starting point. In practice, however, the length of the walk seems too large to be used to generate a single uniformly-distributed plan. Instead, we consider all plans visited in a random walk as a random sample of the space.

To see why the random-edge method does not produce equi-probable QEPs, consider the query graph $\{(a, b), (b, c), (c, d)\}$. If we select (b, c) as the first edge to split the graph, then the plan is already completely specified —remember that left and right children are not distinguished. If, instead, the first edge selected is either (a, b) or (c, d) we must make a second choice. If choices are made uniformly from the available options, the table in Figure 2 shows the probability of generation of each plan. In principle, it seems that the procedure can be modified to use *weighted* instead of uniform selection at each step, so that the resulting QEPs are all equi-probable. But computation of the appropriate weights is difficult, and we have not found a way to do it efficiently.

Uniformly distributed random QEPs. Uniformly distributed generation of random plans is difficult, because there is no one-to-one mapping between valid association trees and simple combinatorial structures —e. g. a permutation of graph edges— except for query graphs with a very "uniform" topology such as star, chain, or clique.[2]

Next, we sketch a new construction that allows efficient generation of uniformly distributed random QEPs, for arbitrary acyclic query graphs. In the process, we also count the exact number of existing plans. Due to lack of space, we do not present formal proofs here. The interested reader is referred to [GLPK94] for detailed proofs, as well as corresponding results for the restricted space of linear QEPs.

[2]For arbitrary graphs, an approach to uniform generation of random QEPs is to generate random binary trees on the relations, check them to see if they require Cartesian products, and stop as soon as a valid QEP is found. This procedure is not feasible in practice, as shown in [Swa91], since a vast majority of binary trees do require Cartesian products, so it takes a long time to generate a single valid QEP.

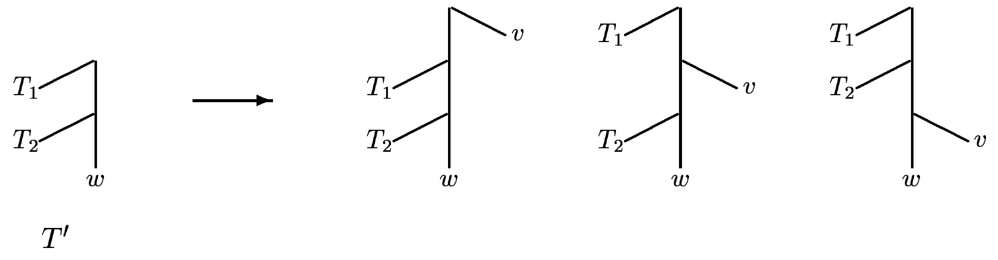


Figure 3: Adding a new leaf to a tree.

For convenience, we sometimes represent trees using *lists anchored on a leaf* w, which contain the subtrees found when traversing the tree from the root to w. For example, the lists anchored on w of the trees in Figure 3 are (T_1, T_2), (v, T_1, T_2), (T_1, v, T_2), and (T_1, T_2, v), respectively.

Given a query graph G on n relations, and a relation v used in G, we start by considering those association trees in which v is at some level k or, equivalently, the trees whose list anchored on v has length k. This set of trees is denoted by $\mathcal{T}_G^{v(k)}$. The set of all association trees for the graph is denoted $\mathcal{T}_G$. Some elementary observations are the following:

- If the graph has only one node, then there is exactly one association tree, and v is at level 0; that is, $|\mathcal{T}_G| = \left|\mathcal{T}_G^{v(0)}\right| = 1$, for $n = 1$.
- If the graph has more than one node, then there is no association tree in which v is at level 0; that is, $\left|\mathcal{T}_G^{v(0)}\right| = 0$, for $n > 1$.
- There is no association tree in which v is at level greater than or equal to n; that is, $\left|\mathcal{T}_G^{v(i)}\right| = 0$, for $i \geq n$.
- Since v appears at some unique level in any association tree of G, the total number of association trees is
$$|\mathcal{T}_G| = \sum_i \left|\mathcal{T}_G^{v(i)}\right| \ .$$

The previous observations provide base cases for the computation of $\left|\mathcal{T}_G^{v(i)}\right|$, and establish their relation with $|\mathcal{T}_G|$. The next lemmas give recurrence equations to compute the number of association trees of a graph, based on the number of trees of some of its subgraphs. The first case *extends* a subgraph by adding one more node; and the second case takes the *union* of two subgraphs whose intersection is a single node.

Lemma 1. *Let $G = (V, E)$ be an acyclic query graph. Let v be a node in V such that $G' = G|_{V-\{v\}}$ is connected, and let $(v, w) \in E$. Then*

$$\left|\mathcal{T}_G^{v(k)}\right| = \sum_{i \geq k-1} \left|\mathcal{T}_{G'}^{w(i)}\right| \ .$$

Proof sketch. There is a one-to-one mapping between trees in $\mathcal{T}_G$ and pairs of the form (T', k), where $T' = (T_1, \ldots, T_i)$ is the list anchored on w of a tree in $\mathcal{T}_{G'}$, and $1 \leq k \leq i + 1$. To obtain the tree $T \in \mathcal{T}_G$ corresponding to such a pair, insert a new leaf v in position k of the anchored list of T'. Figure 3 shows an example of T', and the trees

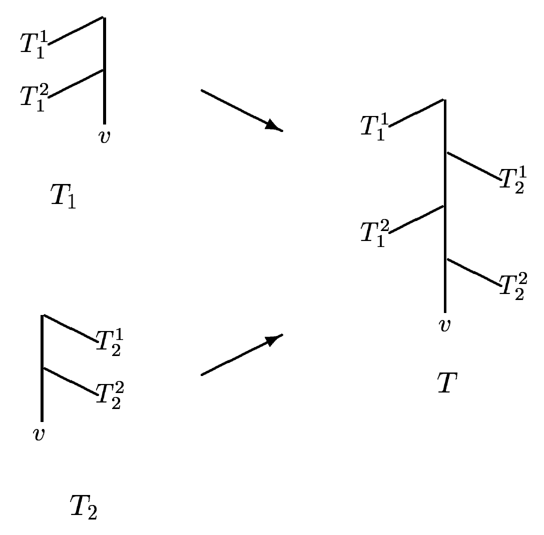


Figure 4: Merging two trees that share a leaf.

obtained from pairs $(T',1)$, $(T',2)$, $(T',3)$. The equation follows from the fact that the tree obtained from a pair (T',k) is in $\mathcal{T}_G^{v(k)}$, and the length of the list of T' is at least $k-1$. ■

Lemma 2. *Let $G=(V,E)$ be an acyclic query graph. Let V_1, V_2 be sets of nodes such that $V = V_1 \cup V_2$, $V_1 \cap V_2 = \{v\}$, and the graphs $G_1 = G|_{V_1}$ and $G_2 = G|_{V_2}$ are connected. Then*

$$\left|\mathcal{T}_G^{v(k)}\right| = \sum_i \left|\mathcal{T}_{G_1}^{v(i)}\right| \cdot \left|\mathcal{T}_{G_2}^{v(k-i)}\right| \cdot \binom{k}{i} .$$

Proof sketch. There is a one-to-one mapping between trees in $\mathcal{T}_G^{v(k)}$ and triplets of the form (T_1, T_2, c), where $T_1 = (T_1^1, \ldots, T_1^i)$ is the list anchored on v of a tree in $\mathcal{T}_{G_1}^{v(i)}$; $T_2 = (T_2^1, \ldots, T_2^{k-i})$ is the list anchored on v of a tree in $\mathcal{T}_{G_2}^{v(k-i)}$; and $c = (\alpha_1, \ldots, \alpha_{k-i+1})$ is a *composition of i in $k-i+1$* [NW78]. To obtain the tree $T \in \mathcal{T}_G^{v(k)}$ corresponding to such a triplet, merge the lists for T_1 and T_2 as specified by c —each α_j indicates how many subtrees of the list of T_1 go between subtrees T_2^{j-1} and T_2^j in the merged list. For example, Figure 4 shows how to find the T of a triplet $((T_1^1, T_1^2), (T_2^1, T_2^2), (1,1,0))$. The equation results from the fact that there are $\binom{k}{i}$ compositions of i in $k-i+1$. ■

Figure 5 shows how the above lemmas are used to compute the number of association trees for a graph $Qg = (V,E)$ of five nodes; $V = \{a,b,c,d,e\}$ and $E = \{(a,b),(b,c),(c,d),(c,e)\}$. The graph is neither a star nor a chain. Each column of the table shows the data for a subgraph of G, and it is computed based on the values of previous columns. The bottom row shows the total number of trees for the subgraph.

For example, since $Qg|_{\{abcde\}}$ extends $Qg|_{\{abcd\}}$ by one node, lemma 1 is used to compute the entries of the last column. The new node e is connected to old node c, so we need to

$Qg\vert_{\{a\}}$	$Qg\vert_{\{ab\}}$	$Qg\vert_{\{abc\}}$	$Qg\vert_{\{d\}}$	$Qg\vert_{\{cd\}}$	$Qg\vert_{\{abcd\}}$	$Qg\vert_{\{abcde\}}$
$\vert\mathcal{T}_G^{a(0)}\vert = 1$			$\vert\mathcal{T}_G^{d(0)}\vert = 1$			
	$\vert\mathcal{T}_G^{b(1)}\vert = 1$	$\vert\mathcal{T}_G^{c(1)}\vert = 1$		$\vert\mathcal{T}_G^{c(1)}\vert = 1$		$\vert\mathcal{T}_G^{e(1)}\vert = 5$
		$\vert\mathcal{T}_G^{c(2)}\vert = 1$			$\vert\mathcal{T}_G^{c(2)}\vert = 2$	$\vert\mathcal{T}_G^{e(2)}\vert = 5$
					$\vert\mathcal{T}_G^{c(3)}\vert = 3$	$\vert\mathcal{T}_G^{e(3)}\vert = 5$
						$\vert\mathcal{T}_G^{e(4)}\vert = 3$
$\vert\mathcal{T}_G\vert = 1$	$\vert\mathcal{T}_G\vert = 1$	$\vert\mathcal{T}_G\vert = 2$	$\vert\mathcal{T}_G\vert = 1$	$\vert\mathcal{T}_G\vert = 1$	$\vert\mathcal{T}_G\vert = 5$	$\vert\mathcal{T}_G\vert = 18$

In each column, G is the subgraph of Qg specified in the top row.

Figure 5: Counting the number of association trees.

know the number of trees in $Qg|_{\{abcd\}}$ in which c appears at various levels. This information is available in the second-to-last column, whose values were in turn computed using lemma 2 on the result of earlier columns.

The following theorems follow from the application of lemmas 1 and 2. The first theorem refers to the time required to compute a matrix similar to that of Figure 5.

Theorem 1. *The number of association trees for a given acyclic query graph G on n relations can be computed in polynomial time.*[3]

The second theorem is based on a natural numbering, or *ranking* of all trees of a given graph, extracted from the matrix used in the computation of the number of trees. For example, from the last column of the matrix in Figure 5, we assign the numbers 1 through 5 to the trees of Qg in which e appears at level 1; numbers 6 through 10 to those in which e is at level 2; 11 through 15 to those in which e is at 3; and finally 16 through 18 to those trees in which e is at level 4. Our unranking procedure is based on those presented in [RH77, Li86].

Theorem 2. *Association trees of a given acyclic query graph G on n relations can be unranked in polynomial time.*

Since trees are numbered, and we can reconstruct efficiently any of them given its number, the next corollary follows.

Corollary of theorem 2. *Assuming a source of random numbers is available, association trees of a given acyclic query graph G on n relations can be generated at random with uniform distribution in polynomial time.*

5 Quality of random sampling

We now compare samples obtained from a known space by our quasi-random and uniformly-random procedures. We use the accumulated cost distribution as the basis of our analysis.

[3]A loose upper bound on theorems 1 and 2, and the corollary of theorem 2 is $O(n^3)$.

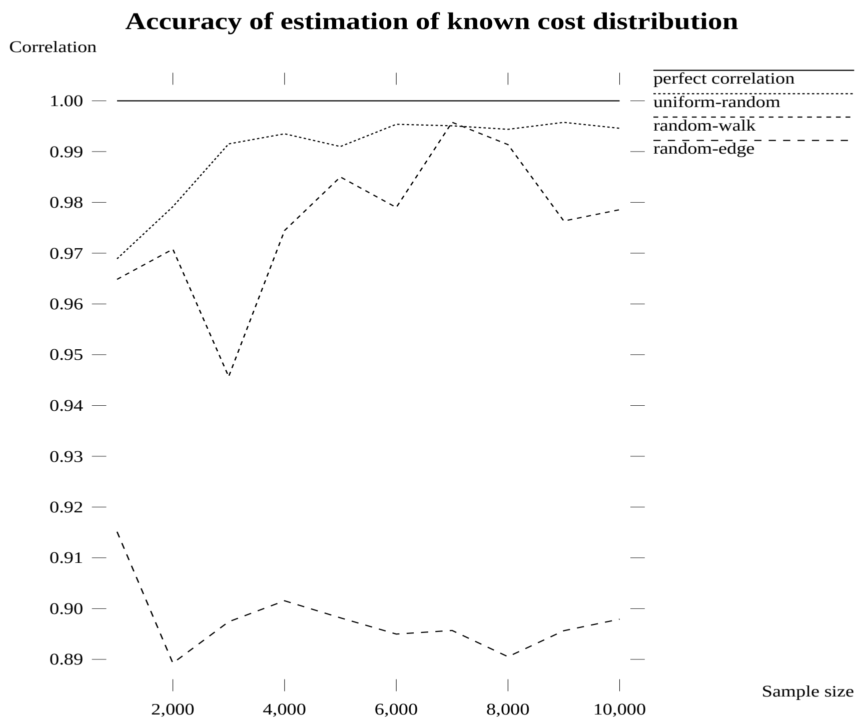


Figure 6: Correlation of approximations to cost distribution using random sampling.

This distribution can be seen as a function on cost c, which gives the percentage of plans having a cost less or equal to c. Formally, in a space S the *accumulated cost distribution* is

$$F_S(c) = \frac{|\{p \mid p \in S, cost(p) \leq c\}|}{|S|} * 100.$$

As samples become larger, they are expected to approximate the real cost distribution function of the space. For spaces of up to eight relations, we obtained the exact accumulated cost distribution in section 3, which allows us to compute the accuracy of the samples taken. Therefore, we compute the correlation coefficient of the function $F_S(c)$ with $F_{S'}(c)$, where S is the complete search space and S' is a random sample obtained by one of our methods. Figure 6 shows the correlation coefficients found for a query of eight relations, for increasing sample sizes.

The random-edge sampling method does not approach the exact cost distribution, because it favors certain plans. Since we do not know the quality of the plans that are favored, the effect of using this method in an optimization strategy is not clear.

The random-walk and uniformly-random method give a more accurate sample and improved their approximation as the sample size increased, but the uniformly-random method converges faster to the known cost distribution.

With the uniformly-random method we also sampled spaces that could not be explored exhaustively. These big spaces still showed some "good" plans but the percentage of "good" plans decreased. This is in line with the observations of [Swa89a] for the spaces of linear execution plans and with our results of section 3. The decreasing number of good plans implies an increase in the number of plans required by a transformation-free optimization strategy. But also other optimization strategies need to explore more plans as the search space increases. In the next section we present a direct comparison of optimization algorithms.

6 Comparison of optimization algorithms

We now compare the performance of an optimization method that relies completely on uniform random generation of candidates with two transformation-based optimization algorithms, *Simulated-annealing* and *Iterative-improvement.* Detailed descriptions of these algorithms can be found in [IK90, SG88]. Our implementation of SA and II is based on the pseudo-code presented in [IK90]. We also set the parameters of the algorithms to the values suggested in that paper.

Simulated Annealing (SA) starts at a random plan and randomly generates moves to other plans. If the next plan is an improvement the move is accepted; if the move leads to a plan with higher cost, then it is accepted with a certain probability. As time progresses this probability is decreased until it disallows moves to higher costs.

Iterative Improvement (II) performs a large number of *local optimizations.* Each local optimization starts at a random plan and then generates random moves to other plans, accepting a move only if it improves the current cost.

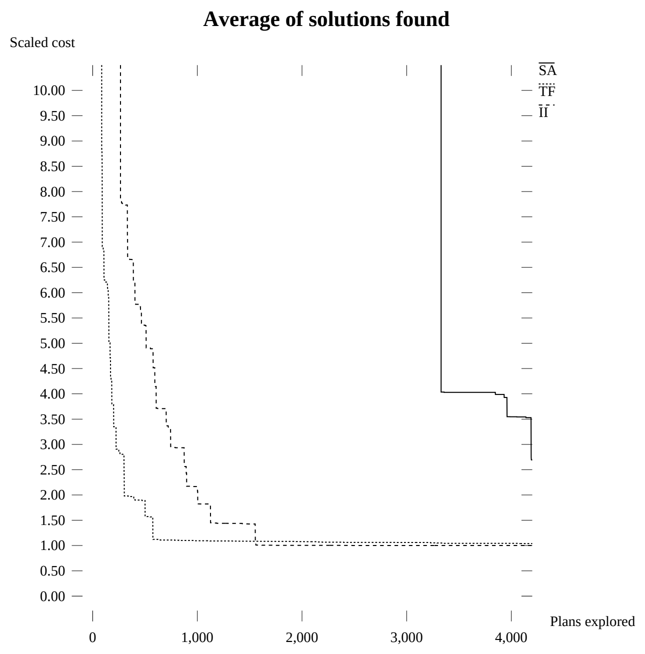
Transformation-Free (TF) generates valid plans at random in a uniform way and keeps track of the one with the lowest cost.

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 so-far. 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:

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

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 follow [LVZ93] in using it as an approximation of an *implementation-independent* measure for optimization cost.

Results. In our experiments, we measured the values of S_n^A for various queries and catalog, for algorithms II, SA, and TF. In each run, we let each algorithm explore 10,000 plans. The number of repetitions for each experiment was 20, each leading to a different observation of the random variables S_n^A. At the end of the experiments, costs were scaled to the best found; for example, a cost of 2 corresponds to a plan that is twice as expensive as that found by any method.



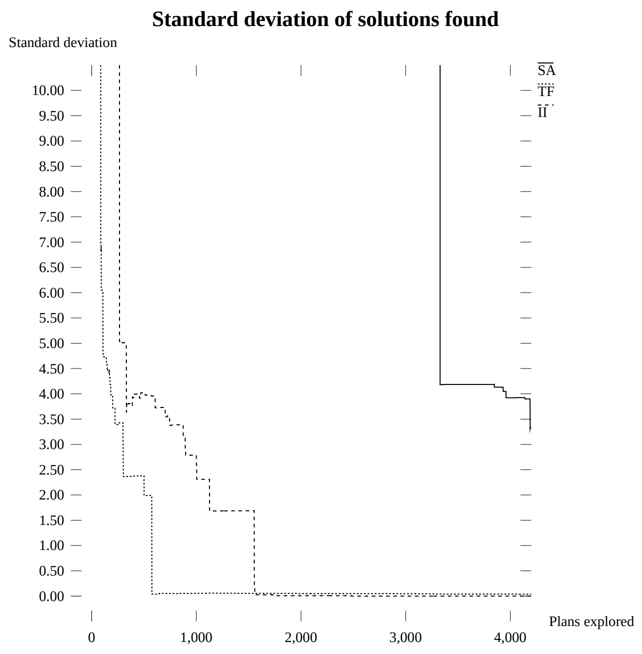


Figure 7: Average and standard deviation of cost of solutions found.

To analyze the results we computed the average and standard deviation of the solutions S_n^{II}, S_n^{SA} and S_n^{TF}, for $1 \leq n \leq 10,000$. The result of this analysis, for a query of 12 relations, is shown in figure 7. To get a more readable graph, figure 7 is limited to 4,000 plans. The graph is typical of the results we obtained on all queries and catalogs examined.

The average of the solutions found after 10,000 plans by each algorithm were avg($S_{10{,}000}^{SA}$) = 1.045, avg($S_{10{,}000}^{II}$) = 1.000, and avg($S_{10{,}000}^{TF}$) = 1.055. The standard deviations were std($S_{10{,}000}^{SA}$) = 0.055, std($S_{10{,}000}^{II}$) = 0.000, std($S_{10{,}000}^{TF}$) = 0.015.

On the average, SA is not able to find a good plan within 4000 plans; it finds these only after exploring 2000 more plans (not visible in the graph). On the average, TF finds good plans faster than II —e. g. TF finds plans with a scaled cost of 2 after exploring about 300 plans while II needs about a 1000. When II and TF keep exploring more plans II will find slightly better plans than TF. The maximum difference occures after exploring 1800 plans but is very small (1.01 v.s. 1.08).

The other graph shows that TF not only finds good plans fast, on average, but that the quality of the plans found in different runs are also close together. After 600 plans the standard deviation of TF is already 0.04 while that of II is only about 3.90. This leads to consider the *time of convergence*, defined as the number of plans required to reach a given maximum standard deviation. Setting the threshold to 0.1, SA converges after exploring 8985 plans, finding a solution of average cost 1.11 at that point; II converges after 1559 plans, finding a solution of cost 1.05; and TF converges after 575 plans, finding a solution of cost 1.12.

7 Conclusions

Why use transformations? In abstract terms, a set of transformation rules imposes a topology on the search space, but it is difficult to determine how a specific topology affects the performance of search algorithms. For the problem of join-order selection, the thorough studies of Ioannidis and Kang provide an empirical basis to understand the search space [IK90, IK91, Kan91]. Close analysis reveals that our results are not only consistent with their studies, but in fact complement them. They observe that "...starting at a random state, many downhill moves are needed [by II] to reach a local minimum" [Kan91, p. 65]. In the spaces we considered, II had to explore well over a 100 plans to reach a local minimum, on average —yet we point out that the expected length of a sequence of random plans that finds one in the best 1% is 100.

Ioannidis and Kang conclude that SA finds very good solutions but takes a long time, compared with II. Their *two-phase optimization* algorithm uses II to find several local minima that are then used as starting points for SA. A similar multi-phased approach is taken in the *toured simulated annealing* of Lanzelotte, Valduriez, and Zait [LVZ93], where starting points of SA are obtained using a greedy deterministic algorithm. In this context, our result is that *transformation-based optimizers find very good solutions but take a long time, compared with our transformation-free algorithm.* To be concrete, in a query of 12 relations II converges in 1559 steps to a solution of cost 1.05, while TF converges in 575 steps to a solution of cost 1.12.

We are currently investigating multi-phase, hybrid algorithms for fast-convergence/high-quality, based on our results. Nevertheless, a pure transformation-free algorithm has some specific advantages that should not be casualy ignored. In our view, a key property of transformation-free optimization is that it has no "knobs to tune." For other algorithms, the setting of parameters for optimum performance is not obvious; results on the sensitivity of the algorithm to these settings are not available; and both optimum setting and sensitivity may depend on the specific cost model and database. Also, in parallel systems, transformation-free optimization can easily take advantage of available processors, achieving nearly optimal speedup —simply replicate the original algorithm in various processors, and add a final phase to determine the best solution found. Transformation-based "walks," on the other hand, are inherently sequential.

Contributions. The prime novelty of this paper is its transformation-free query optimization scheme, which provides a cheap and effective alternative to transformation-based algorithms. The mechanism relies on both an accurate estimation of query evaluation cost and an efficient mechanism to generate query plans uniformly distributed over the search space. This leads to a strategy where a random sequence of valid plans is generated and analyzed on their perceived cost. The best plan within the run is selected for execution.

Exhaustive exploration or sampling of the search space of a class of queries provides a precise measure of the run length required to hit a good plan. Our results then provide a natural baseline against which the added value of applying transformations and heuristics can be quantified.

To realize our proposed optimization scheme, we solve the problem of efficiently generating uniformly-distributed random plans, for queries with acyclic graphs. In the process, we also count the exact number of existing plans. Generation of a uniformly-distributed random plan is a basic primitive that other randomized algorithms can now use —due to the complexity of previously known methods, only *quasi*-random selection had been used for non-star graphs.

Related work. Transformation-based optimization is a general and powerful techniques with applications beyond join-order selection; see, for example, [FMV94]. More related to our present work, research on randomized optimization of join queries has been performed by Swami and Gupta [SG88, Swa89b, Swa89a, SI92], Ioannidis and Kang [IK90, IK91, Kan91], and Lanzelotte, Valduriez, and Zaït [LVZ93]. In contrast to our work, their approach is based mostly on tree transformations. In terms of search space, Swami and Gupta, and Ioannidis and Kang study very large queries (up to 100 relations); Swami and Gupta, and Lanzelotte, Valduriez, and Zaït allow cyclic query graphs; but Swami and Gupta only explore linear QEPs. Finally, the cost model of Lanzelotte, Valduriez, and Zaït is that of a parallel database.

Future research. Our agenda for future research includes, first, the extension of our experiments to larger queries. Then, to remove our current restriction on the query graph topology, we need an efficient procedure to generate uniformly-distributed plans on cyclic graphs.

We are currently studying hybrid, multi-phase algorithms based on a "mix" of randomized choices, transformations, and heuristics. Depending on the ratio of optimization cost over query evaluation cost, and on the number of times the optimized query will be executed, some applications do need such a "mix," despite the likely necessary tuning.

Finally, we are also interested in the accuracy of the estimation of query evaluation cost —e. g. cost model calibration and size estimation of intermediate results— and how it affects the quality of the optimization output.

Acknowledgements. We are grateful to Johan van den Akker, Carel van den Berg, Josef Kolař, Fred Kwakkel, Zandra Navarro-Villicaña, Arno Siebes, and Chris Thieme for their suggestions and comments on preliminary drafts of this paper.

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.

[AKK93] F. Andrès, F. Kwakkel, and M.L. Kersten. Calibration of a DBMS cost model with the software testpilot, 1993. Manuscript.

[FMV94] J. C. Freytag, D. Maier, and G. Vossen, editors. *Query Processing for Advanced Database Systems.* Morgan Kaufmann, San Mateo, California, 1994.

[GLPK94] C. Galindo-Legaria, A. Pellenkoft, and M. Kersten. Uniformly-distributed random generation of join orders. Technical report, CWI, 1994. In preparation.

[Gra93] G. Graefe. Query evaluation techniques for large databases. *ACM Computing Surveys*, 25(2):73–170, June 1993.

[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.

[Li86] L. Li. Ranking and unranking of AVL-trees. *SIAM Journal of Computation*, 15(4):1025–1035, November 1986.

[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.

[NW78] A. Nijenhuis and H. S. Wilf. *Combinatorial algorithms.* Academic Press, New York, 2nd edition, 1978.

[OL90] K. Ono and G. M. Lohman. Measuring the complexity of join enumeration in query optimization. *Proc. of the 16th VLDB Conference, Brisbane, Australia*, pages 314–325, 1990.

[Rag90] P. Raghavan. Lecture notes on randomized algorithms. Technical Report RC 15340, IBM Research Division, T. J. Watson, 1990.

[RH77] F. Ruskey and T. C. Hu. Generating binary trees lexicographically. *SIAM journal of Computation*, 6(4):745–758, December 1977.

[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.

[SI92] A. N. Swami and B. R. Iyer. A polynomial time algorithm for optimizing join queries. Technical Report RJ 8812, IBM Research Division, Almaden, 1992.

[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.

[Swa91] A. N. Swami. Distribution of query plan costs for large join queries. Technical Report RJ 7908, IBM Research Division, Almaden, 1991.

[Val92] P. Valduriez, editor. *Parallel Processing and Data Management.* Chapman and Hall, London, 1992.

[Wie92] J. Wielemaker. SWI-Prolog 1.6: Reference Manual. Technical report, SWI, University of Amsterdam, 1992.