In the previous chapter we have explained why it is generally very difficult to distinguish randomized algorithms in performance. This phenomenon has accompanied research in this field from the begin on and led Swami to conclude:

*These results lead us to speculate that until significant new insights are obtained into the characteristics of the search space it will not be profitable to experiment with very complex methods for optimization [referring to Simulated Annealing].* [Swa89b], page 376.

Now, with knowledge about the structure and characteristics of the search space at hand it seems in order to turn this statement around: Given our assessment of cost distributions and randomized algorithms we can expect even lesser sophisticated algorithms to produce more than just acceptable results.

Galindo-Legaria *et al.* made out a good case for using uniform sampling of plans instead of transformations (see Chap. 7) [GLPK94]. The algorithm devised is a complex construction whose deployment is, however, limited to acyclic graphs. This limitation—though popular with related work—is a distinct restriction. Queries as for instance in the standard data warehouse benchmark suite of TPC-H contain indeed cyclic queries. But this algorithm shows the way how to exploit the shape of cost distributions successfully. The question we tackle here is whether we can overcome the limitations without performances losses, and furthermore, whether uniformity of the sampling is a necessary prerequisite.

Using the principles discussed in Section 3.2 we devise a simple technique that performs biased rather than uniform sampling, but is distinguished by its low complexity and applicability to arbitrary join graphs. It further gives room to a cost bound pruning component that discards partial query plans which cannot lead to a better plan than the currently best, as early as possible [3].
Algorithm QUICKPICK
Input $G(V, E)$ join graph
Output $s_{\text{best}}$ best query plan found

$r \leftarrow \infty$ \quad // initialize lowest costs so far
$E' \leftarrow E$
$q \leftarrow G'(V, \emptyset)$ \quad // initialize query plan

repeat
  choose $e \in E'$ \quad // random edge selection
  $E' \leftarrow E' \setminus \{e\}$
  ADDJOIN($q, e$)

  if $E' = \emptyset$ or $c(q) > r$ do \quad // either plan complete or costs exceeded
    if $c(q) < r$ do \quad // check for new best plan
      $s_{\text{best}} \leftarrow q$
      $r \leftarrow c(q)$
    done
  done
  $E' \leftarrow E$
  $q \leftarrow G'(V, \emptyset)$ \quad // reset query plan

until stopping criterion fulfilled
return $s_{\text{best}}$

Figure 8.1. Algorithm QUICKPICK

8.1 Biased Sampling

To implement a biased sampling we utilize the techniques introduced in the context of the enumeration of non-isomorphic processing trees in Section 3.2. There we presented MERGETREES a simple algorithm to turn sequences of join predicates into processing trees. We then concentrated on the enumeration of non-isomorphic sequences. Instead of enumerating sequences of edges, we generate random sequences and turn them subsequently into processing trees.

In Figure 8.1, the algorithm, called QUICKPICK, is outlined in pseudo code. After initializing the variable $r$ that records the cheapest plan found, 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 incremen-
8.2. ASSESSMENT

tally built-up by choosing and removing an edge $e$ from the candidate set and adding the corresponding join to the tree. 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 (see Fig. 3.5). 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 $s_{\text{best}}$. After initializing $E'$ and $q$ we start building a new tree.

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.

The algorithm performs a non-uniform, or biased sampling as different sequences of edges may lead to the same result (see also Section 3.2.3). But even computing and excluding redundant edges does not restore uniformity as different prefixes entail different number of possible completions, in general. For example assume there are 1000 non-redundant sequences with $e_1$ as first element and 2000 for $e_2$. Since we do not know the numbers in advance, we cannot adjust the probabilities with which they are chosen. Selecting $e_1$ or $e_2$ with equal probability clearly leads to a non-uniform sampling.

8.2 Assessment

For an assessment of QUICKPICK the abstract search space model is not useful since the only characteristic exploited by the algorithm is the cost distribution which differs now from the original due to the bias of the sampling. Instead, in order to determine the cost distribution under QUICKPICK, we implemented a cost model comparable to those proposed in [EN94, KS91, Ste96].

\footnote{The basic principles of QUICKPICK—without cost-bound pruning—have been described already by Pellenkof [Pel97]. There, this algorithm is called Random Edge Selection and proved 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.}
8.2.1 Cost Distribution

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 8.2, two pairs of cost distributions for high and low variance catalogs are shown. Both samples are of size 5000, the query used is of size 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 computes to

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

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 8.3, 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 $\phi_B$. To that end, we compute the cumulative distributions for $\phi_B$ and $\phi$ on the interval $[0, \mu_\phi]$, recall the shift is defined with respect to single reference points (see Section 7.1). In Figure 8.4 the shift $s(\mu(\phi))$ is plotted as function of the query size. Again, each data point represents the average of 50 queries.

Our results pin-point a clear 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.

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

#### Result Quality

Figure 8.5 shows the quality of the results in terms of quantile-based quality (cf. Chap. 5). 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 comprises queries that can also be optimized with uniform sampling; The second group exceed these limitations. 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 lesser suitable as almost all queries in actual applications are of a connectivity no higher than one.

For acyclic graphs, \textsc{QuickPick} delivers results of a quality comparable to that of uniform sampling—for star graphs, \textsc{QuickPick} actually implements even uniform sampling. In case of cyclic query graphs, \textsc{QuickPick} finds clearly better, near-optimal solutions (see Fig. 8.5).

\section*{Convergence Behavior}

Like with uniform sampling, \textsc{QuickPick}'s strong point is its quick convergence. Figure 8.6 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, \textsc{QuickPick} converges significantly quicker. With longer running time the competitors catch up. Iterative Improvement often beats \textsc{QuickPick}, not significantly though.

To underline the differences between uniform sampling and \textsc{QuickPick}, we compute the probability to hit the quantile \( Q_{0.1} \) for both algorithms.
8.2. ASSESSMENT

Q_{0.1} refers to the respective quantile of the original distribution. In Figure 8.7, 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 within 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 heavily 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 8.8 this effect is demonstrated with low and high variance catalogs for a query of size 100. The left plot in 8.8a, 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. 8.8a 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. 8.8a right).

In Figure 8.8b 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.

In the first case savings amount to some 20%, in the second almost 60% on average.

8.3 Summary

For queries of increasing size, the accuracy of the costing techniques drops. Consequently, plans of costs in the top quantile of the distribution are as good as the optimum. This premise formed the basis for the work of Galindo-Legaria et al. who developed a mechanism to generate join orders with uniform probability. They proof uniform sampling to be competitive to other randomized algorithms such as Iterative Improvement or Simulated Annealing. Moreover, uniform sampling can be used as a building block for compound algorithms or for generating initial states for other algorithms.

In this chapter we scrutinized the potentials of biased random sampling. Using parts of the algorithms introduced in Section 3.2. The resulting algorithm QUICKPICK is distinguished by its short running time, low complexity with respect to both implementation and run time behavior, and its result quality.

Not only an interesting result in its own right, this analysis also gives an impression of the potential effectiveness and importance of choosing the initial solutions for randomized optimization algorithms as discussed in Chapter 7.

Though QUICKPICK extends the domain of sampling to the general case, achieving better results, the challenge of finding an algorithm for general, uniform sampling remains. The algorithm presented in Section 3.3 can well be used for sampling, since counting implies uniform sampling. In fact, we conducted experiments with Microsoft SQL Server backing the claims that sampling is truly a very good alternative to exhaustive strategies. However, this approach is limited by the exponential growth of its MEMO structure.

So far, no method is known for uniform generation of join orders regardless the join graph, nor seem existing techniques to extend to it. Determining the complexity of this problem is still an open problem—i.e. it is
unclear whether it is in $P$ or $\#P$. The same holds for counting, which is in general more difficult than uniform generation [JVV86, Sin92].

Another direction of future research is sampling in the context of query optimization beyond join ordering. The abovementioned approach of sampling over the fully expanded MEMO is of course not very useful as the best solution can be extracted immediately once the MEMO is constructed.

However had we a sampling technique for random generation for com-
plete arbitrary SQL92—or even SQL99—statements, the sampling could be used to get very good initial plans that can then be copied in and enumerated with the MEMO. Using only a reduced set of transformation rules ensures that only the "vicinity" of the candidates is searched. In fact, a simpler variant of this idea is for example implemented in SQL Server where a prospective join order is determined before the plan is copied into the MEMO. With a sampling phase and multiple MEMO structures this idea could be pushed way further using say 10 best sampled plans out of a set of 100 and optimizing them simultaneously with very rudimentary rule sets. The sampled plans also serve as fall-back solutions in case time limits are reached.

Additionally, the method sketched offers obvious possibilities for an immediate parallelization of the optimization phase.