Principles of probabilistic query optimization
Waas, F.

Citation for published version (APA):

General rights
It is not permitted to download or to forward/distribute the text or part of it without the consent of the author(s) and/or copyright holder(s), other than for strictly personal, individual use, unless the work is under an open content license (like Creative Commons).

Disclaimer/Complaints regulations
If you believe that digital publication of certain material infringes any of your rights or (privacy) interests, please let the Library know, stating your reasons. In case of a legitimate complaint, the Library will make the material inaccessible and/or remove it from the website. Please Ask the Library: http://uba.uva.nl/en/contact, or a letter to: Library of the University of Amsterdam, Secretariat, Singel 425, 1012 WP Amsterdam, The Netherlands. You will be contacted as soon as possible.
Probabilistic or randomized optimization algorithms are a field of research which attracted enormous attention during the last decade in all fields of combinatorial optimization. Especially with complex optimization tasks that involve a large number of parameters and where heuristics achieved only mediocre results, these algorithms helped tackling problem sizes that defy being solved to optimality due to combinatorial explosion. Major application areas include scheduling problems and graph partitioning as for instance in standard-cell placement in VLSI layout [VK83, LD88, SR91].

These algorithms are distinguished by their black-box approach, that is, no knowledge of the problem's particular structure and properties, but only a set of manipulators is required. Manipulators alter a given solution and transform it into a different, but semantically equivalent—i.e., correct—new solution. Manipulators are often also referred to as rules or transformations; we will use these notions synonymously in the following. The changes are usually of local nature, that is, the new solution is "similar" to the original one. Manipulators imply a topology on the search space as they define neighborhoods. Given a single solution, the transitive application of manipulators defines a subspace of the original search space. An important prerequisite is the irreducibility of this subspace, i.e., the transformations should be defined in such a way that any two solutions can be transformed into each other by a sequence of transformations.

Based on this setup various optimization strategies have been proposed: starting with an arbitrary solution, an optimization algorithm applies the manipulators according to a strategy to generate a set of alternative solutions. One of the alternative solutions is accepted—again according to a strategy—and the neighborhood of this solution is explored in the same way implementing an transitive exploration of neighborhoods. The algorithm records the best solution found during the process. In Section 7.3 we develop a general framework and present the most important representatives of this class of algorithms.
Given the success in other fields of combinatorial optimization and the simple way to adapt these techniques they have been first applied to query optimization by Ioannidis and Wong [IW87] and specifically to join ordering by Swami and Gupta [SG88]. Following these initial studies, Ioannidis and Kang [IK90, IK91] established an in-depth analysis of some of the algorithms for $JOPT$ and presented first theoretical results concerning the basic properties of the search space. In particular they investigated differences between the general search space containing processing trees of arbitrary shape and the restricted one constituted by linear trees only.

Despite the efforts to provide a more comprehensive understanding a commonly accepted result could not be reached owing chiefly to the fact that some of the results were contradicting, i.e., different parties proved different search strategies to be superior [SG88, IK90, IK91, Pel97, SMK97]. The differences have been attributed mainly to the different search spaces, different cost models, and different parameter settings used. Oddly enough, the transformation rules which are responsible for the topology have not been subject to discussion, but were treated as a given rather than a variable. The only concerns with the design of transformations were the irreducibility of the subspace and that all results of a transformation are again valid solutions.

In this chapter, we pursue an analysis based on the techniques and insights developed in the previous parts. First we discuss the question of linear and bushy search spaces from the cost distribution point of view on the lines of Chapter 4. In order to investigate the potential of randomized algorithms we have to liberate the algorithms from the strong interplay of different transformation rules. To that end we devise an abstract search space model consisting only of the two invariable components: solutions and their costs. The solutions become annotated nodes in a graph where edges—representing neighborhood relations—are no longer a result imposed by transformation rules, but can be freely chosen to test for various topologies. More general, the abstract search space allows us to experiment with arbitrary parameter combinations that would be hard to achieve in an actual query optimization setting. This way, we can fathom the actual potential of an optimization technique. For the analysis of different probabilistic optimization techniques we break the algorithms into the single algorithmic principles, as seen with genetic algorithms, and analyze these building blocks and subsequently discuss the complete, compound algorithm.

### 7.1 Linear vs. Bushy

In their first approach to randomized join ordering, Swami and Gupta confined their algorithms to linear processing trees only. The space of linear trees is in so far prominent as it has been used in previous work with exhaustive search techniques as the linear space is considerably smaller than
7.1. LINEAR VS. BUSHY

its bushy counterpart. Commercial database systems like Microsoft SQL Server use linear search spaces enriched with nested queries, i.e., each operator can have a linear tree and a base table, or a linear tree and a sub-query—which in turn is a linear tree—as inputs, multiple nesting allowed for.

The question we are interested in here is whether the obvious differences in shapes of trees have immediate consequences on the optimization.

As we have pointed out in Section 2.2, the possible shapes of a processing tree are limited by the join graph, i.e., the query itself. There always is a valid linear processing tree, no matter the shape of the join graph. For star shaped join graphs, linear trees are even the only possible processing trees. The other extreme is the clique graph where any shape of tree is possible. Unfortunately there exists no concise meaningful measure to classify the wealth of join graphs in between star and cliques. Notions like chain, grid, or more general, acyclic graphs etc. are too broad and without further additional parameter do not define a specific subset of shapes.

However, we can apply a similar analysis as in Chapter 4 and differentiate linear and bushy trees in XOPT. Let us define the sub-problem L-XOPT where only linear trees are considered a valid solution. Note, that L-XOPT is a real subset of XOPT. We denote the cost distributions as \( \phi \) and \( \phi_L \) for XOPT and L-XOPT respectively. In Figure 7.1 the cost distribution of L-XOPT is contrasted with the one of XOPT for low and high variance of the relation sizes. The distribution of L-XOPT is highlighted. The plots of Figure 7.1a depict the situation for relation sizes of low variance, 7.1b for a high variance scenario. The left plot shows the total number of solutions as function of the costs, the right one displays the resulting cost distributions, i.e., relative instead of absolute frequencies.

In case of low variances, the distributions differ significantly. Linear trees span a part of the search space with substantially more costly minimum, whereas the maxima of both spaces are comparable. In the event of higher variances, the characteristic of L-XOPT'S space changes and reaches eventually those of the general space (see Fig. 7.1 a) and b) right plots).

To capture this shift of the distributions more formally, we use a characterization introduced by Ioannidis and Kang [IK91]: Given two densities \( \psi_1 \) and \( \psi_2 \), the ratio

\[
s(x) = \frac{\int \psi_1(x)}{\int \psi_2(x)}
\]

is called relative shift. If \( s \geq 1 \), we say \( \psi_1 \) is shifted to the left relative to \( \psi_2 \).

The major observations with our experiment are:

1. the mean of \( \phi_L \) is higher than that of \( \phi \), and \( \phi \) is relatively shifted to the left;

2. with larger variance, the number of high-quality solutions in \( \phi_L \) increases, i.e., for large variances, the best solution in both distributions are of comparable quality;
The immediate consequence of fact (1) is a different probabilistic difficulty as Figure 7.2 shows. For the low variance case the bushy space is significantly easier, the linear more difficult—for higher variance the linear space becomes more and more less difficult matching the bushy eventually. The analogous observation for jopt has been reported by Bennet et al. in [BFI91] who found better results in bushy spaces using a genetic algorithm.
than exact methods could find in the linear space for the same query.

Together with fact (2) that both spaces may contain comparable optima the space of bushy trees appears preferable.

Ioannidis et al. reached to the same conclusion, however, on totally different grounds [IK91]: They argue that bushy spaces offer a better topology for transformation based search algorithms which puts their entire analysis in direct dependency to the implementation of manipulators. Hence, their analysis would be of very limited generality only. Moreover their investigation applies only to transformation based optimization algorithms, but not to multi-start algorithms like Transformation-free [GLPK94]. As opposed to that, our analysis is not restricted to any optimization technique but reflects solely the search space and its character.

In the discussion of optimization algorithms later this chapter, we will come back to this point when analyzing the direct effects of the choice of a search space on the algorithms.

### 7.2 Abstract Search Space Model

Repeatedly, we pointed out that transformation-based optimization is often abstracted by a guided graph traversal. In order to discuss the effects reported in previous work and to contrast them with our own findings we use a graph model that allows us to study the single parameters and their
impact. This is of particular interest as authors of previous work discussed effects observed often with respect to this abstraction, i.e., they explained behavior of optimization algorithms with properties the underlying graph was assumed to have. To check for these properties we generate fully synthetic graphs called abstract search spaces described by the parameters given in table 7.1. The graphs are distinguished by a number of features. All edges are directed. The degree describes only the number of outgoing edges; the number of incoming edges may be arbitrary. A directed graph matches reality better than an undirected as transformations used in query optimization are usually not symmetric.

In Figure 7.3, the algorithm to construct the graph is outlined. First, the set of nodes is generated and each node is assigned costs according to a gamma distribution as discussed before. To facilitate and steer the generation of edges we sort the nodes according to their costs. This enables us to perform efficient binary search for nodes of given costs. In the next step we construct neighborhoods by first generating a random number from a normal distribution with deviation $\sigma$ and the costs of the current node as mean, and search for a partner node with the resulting costs. At the fringes of the cost range, we truncate the normal distribution so that only valid cost values are generated. We denote the resulting graph by $G^{(\alpha)}_{\sigma}(n, d)$.

We omit indexes if they are apparent from the context. To match the practical case, the degree should be chosen greater than 10 (see also next section). Though there is a positive probability to construct a disconnected graph in principle, with the parameter settings we use for our experiments this case can be ruled out almost certainly. Moreover we modified the binary search for a partner in such a way that not an exact match of the cost value is required but rather the nearest neighbor to that cost value is chosen. Furthermore, we exclude loops.

### 7.2.1 Experimental Setup

To simulate real world scenarios the algorithm must be able to handle very large graphs. Hence, a parallelization of the graph generation is necessary to maintain practicable running times. All steps of the algorithm offer simple yet enormously effective means for parallelization on a CC-NUMA

<table>
<thead>
<tr>
<th>$n$</th>
<th>size of the graph</th>
</tr>
</thead>
<tbody>
<tr>
<td>$d$</td>
<td>degree, i.e., number of outgoing edges, equal for all nodes</td>
</tr>
<tr>
<td>$\alpha$</td>
<td>shape parameter for gamma distributions used for cost values</td>
</tr>
<tr>
<td>$\sigma$</td>
<td>deviation of cost between neighbors</td>
</tr>
</tbody>
</table>

**Table 7.1. Parameters of abstract search space model**
architecture. For both distributing costs and constructing the edges the graph can be divided in \( k \) independent parts, \( k \) being the number of available CPUs. The operations can be carried out on those parts in isolation. In the actual implementation the granularity is chosen as 1000 nodes per sub-graph instead of \( n/k \), \( n > k \cdot 1000 \) to achieve better load balancing as the target architecture consisted of 32 CPUs of 2 different types, 8 MIPS R12K300MHz and 24 MIPS R10K250MHz. With \( k \) parts, some of the threads would become idle earlier than others due to different CPU speeds. Having smaller parts, however, reduces this effect substantially.

The critical element is the sorting, which is done in a hierarchical decomposition where first each thread sorts the range of nodes it is assigned to with e.g. quick-sort. In the next phase, threads with even id merge the segment of their predecessor with their own. Threads with odd id are terminated. After renumbering the threads, this step is repeated until no thread is active anymore. By then the sorting is complete. Since sorting is only of little running time compared to the other steps mainly because of the substantial share of floating point arithmetic of the random number generators we achieve almost ideal speedup for sufficiently large graphs making graphs the size of several millions of nodes a feasible target.

### 7.2.2 Topologies

The topology of an abstract search space of given size is essentially determined by the degree and the distribution of costs among neighbors, i.e., the "length" of the edges. While the impact of the degree is evident, the impact of the distribution of costs among neighbors is somewhat more subtle. As a simple example consider a deviation of \( 5 \cdot 10^{-5} \) with arbitrary but sufficiently large degree, say 50. The resulting graph is the linear graph. Recall, that we do not allow loops but do allow multiple edges. The very small deviation is only used to suggest a direction, i.e., since we forbid loops we are forced to take the next possible node in the given direction. The high degree in this example ensures probabilistically that every node has some neighbors of higher and some of lower costs. Moreover, as seen in this example, \( \sigma \) is only used to search for a neighbor, but the resulting deviation among neighbors may differ.

Another detail deserving attention is the fact that we use a constant degree for all nodes. However this poses no significant restriction. For the transformation sets proposed for the \( \text{JOPT} \), this is always true; for the general case of query optimization the degree can differ. However, these differences are usually small compared to the total number of neighbors.

For a comparison with previous work we extend the above schema and allow also to choose neighbors completely at random without the demand to obey to any distribution of neighbor costs other than the global cost distribution. We use the notation

\[
G^{(\alpha)}(n, d)
\]
**Algorithm**  Generate Abstract Search Space

**Input**  
- $n$ size, $d$ degree,  
- $\alpha$ shape parameter, $\sigma$ neighbor deviation

**Output**  
$G(V, E)$ abstract search space

$V \leftarrow \emptyset$

$E \leftarrow \emptyset$

generate set of nodes $V$

**foreach** $v \in V$

- $c \leftarrow$ random number from Gamma$(\alpha)$
- annotate $v$ with $c$

**done**

sort $V$ according to costs

**foreach** $v \in V$

- for $i = 1$ to $d$ do
  - $c \leftarrow$ random number from $N(0, \sigma)$
  - $v' \leftarrow$ search$(V, c)$
  - $E \leftarrow E \cup (v, v')$

**done**

**done**

**return** $G(V, E)$

**Figure 7.3.** Generate Abstract Search Space

...to refer to this variant.

### 7.2.3 Local Minima

Local minima are nodes that have only edges to nodes of higher costs. The fact that we consider directed edges is of special importance. A node can be a local minimum although a node of lower costs has an edge pointing to it. This situation has its equivalent in query optimization, as pointed out above, where transformations are not symmetric in general.

In an optimization based on the traversal of the graph as we will discuss in the next section, the notion of local minima seems not only intuitive but also of strong influence on the behavior of the optimization algorithm. At least it might seem so on first sight.

Let us first analyze the distribution of local minima in the search space. For the case $G_{\alpha}^{(\gamma)}(n, d)$ where arbitrary directed edges—i.e., without respecting a cost distributions among neighbors are allowed—the distribu-
The situation for $G^{(N)}_\sigma$ differs substantially—particularly for very small values of $\sigma$. As we choose mainly nodes from a small symmetric range of costs the overall cost distribution is of little influence when choosing neighbors for nodes of costs greater than $2\sigma$. In this case, we can approximate the probability for a local minimum by $\left(\frac{1}{2}\right)^d$. This changes for costs less than $2\sigma$ as the normal distribution used to compute cost values for neighbors gets more and more truncated, up to the point where it degenerates to the right half of the original bell only. In order to maintain the property of being a distribution we multiply the truncated density with the appropriate factor, i.e., we norm it.
In Figure 7.5 experimentally obtained densities are shown. The size of the space was $20 \times 10^6$, the degree is varied between 10 and 100, and $\sigma$ is of values from 0.05 through 0.5. Above $2\sigma$ the distribution of local minima is of the quality of the shape of the overall cost distribution as expected. For costs less than $2\sigma$ the truncation of the distribution used for selecting neighbors grows more and more asymmetric which results in a larger number of local minima. For higher degrees, the probability for a local minimum above $2\sigma$ converges to 0. Local minima occur only close to the global minimum.

In Figure 7.6, an overview over the number of local minima is given for the case where $\alpha = 2$ and both degree, i.e., size of the neighborhood, and deviation of costs among the neighbors is varied.

Figure 7.7 shows the absolute frequency of local minima as close-up for a cost distribution with $\alpha = 2$ and $\sigma = 0.1$. For small degrees the aforementioned effect that local minima are scattered over the whole cost range is clearly visible. With increasing degree, this effect vanishes.

Summing up, we identify a distinct skew of the distribution of local minima in both models: local minima occur mainly at costs close to the optimum. With increasing degrees the concentration of local minima shifts further to left. Specifically, for high degree, the differences in costs between local and global minima are not significant anymore.

### 7.3 Probabilistic Optimization Strategies

In the introduction to this chapter, we already gave a flavor as to how probabilistic optimization algorithms work. Here, we now present a classification and detailed description of the main representatives.

Blind search optimization algorithms can be divided into several groups according to the optimization principles used, the categories are not exclusive though. We distinguish three major groups: transformation-based, randomized and multi-start algorithms (cf. Fig. 7.8).

The major representative of the first class are classical exhaustive search algorithms as used for example in commercial database systems (cf. Chapter 3). Besides exact methods, also heuristics belong to this class including simple rule-based approaches where a set of transformation is applied until no further application is possible. A typical example is the push-down-selection heuristic, where selection operators are moved down as far as possible in the processing tree. Tabu search is a further member of this class though more sophisticated variants of it periodically also use what is known as randomized phases [Glo89, MMS94].

The class of randomized algorithms overlaps with the previous but also contains the class of multi-start algorithms. It includes most prominently Simulated Annealing and its close relative Threshold Accepting, which are strictly navigating algorithms. Iterative Improvement—also known as It-
Iterative Local Optimization—conducts simple navigations repeated with a new starting point whenever the local termination criterion is fulfilled. Transformation-free Optimization can be viewed as the special case of Iterative Improvement where the navigation is completely omitted. We will devote special attention to the relation of these two later.

Generally, this classification identifies the major building blocks but further combinations are of course conceivable. For example iteratively repeated Simulated Annealing as proposed by Lanzelotte et al. [LVZ93]. Like
### Figure 7.6. Number of local minima (in thousands). $n = 20 \cdot 10^6$, $\alpha = 2$, degree and deviation for neighborhood distribution varied

<table>
<thead>
<tr>
<th>Degree</th>
<th>Deviation 0.05</th>
<th>Deviation 0.10</th>
<th>Deviation 0.15</th>
<th>Deviation 0.20</th>
<th>Deviation 0.25</th>
<th>Deviation 0.30</th>
<th>Deviation 0.35</th>
<th>Deviation 0.40</th>
<th>Deviation 0.45</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>32.7</td>
<td>70.9</td>
<td>132.5</td>
<td>217.1</td>
<td>320.2</td>
<td>440.7</td>
<td>577.0</td>
<td>728.8</td>
<td>891.7</td>
</tr>
<tr>
<td>20</td>
<td>2.7</td>
<td>10.3</td>
<td>23.0</td>
<td>40.7</td>
<td>63.2</td>
<td>89.9</td>
<td>120.9</td>
<td>157.6</td>
<td>197.4</td>
</tr>
<tr>
<td>30</td>
<td>1.1</td>
<td>4.3</td>
<td>9.5</td>
<td>17.0</td>
<td>26.6</td>
<td>37.8</td>
<td>51.3</td>
<td>66.8</td>
<td>84.3</td>
</tr>
<tr>
<td>40</td>
<td>0.6</td>
<td>2.3</td>
<td>5.2</td>
<td>9.3</td>
<td>14.6</td>
<td>20.9</td>
<td>28.3</td>
<td>36.9</td>
<td>46.7</td>
</tr>
<tr>
<td>50</td>
<td>0.4</td>
<td>1.5</td>
<td>3.3</td>
<td>5.9</td>
<td>9.3</td>
<td>13.1</td>
<td>17.9</td>
<td>23.4</td>
<td>29.6</td>
</tr>
<tr>
<td>60</td>
<td>0.3</td>
<td>1.1</td>
<td>2.3</td>
<td>4.0</td>
<td>6.4</td>
<td>9.0</td>
<td>12.4</td>
<td>16.1</td>
<td>20.6</td>
</tr>
<tr>
<td>70</td>
<td>0.2</td>
<td>0.8</td>
<td>1.7</td>
<td>2.9</td>
<td>4.7</td>
<td>6.6</td>
<td>9.1</td>
<td>11.7</td>
<td>15.1</td>
</tr>
<tr>
<td>80</td>
<td>0.2</td>
<td>0.6</td>
<td>1.3</td>
<td>2.2</td>
<td>3.5</td>
<td>5.1</td>
<td>6.9</td>
<td>8.9</td>
<td>11.4</td>
</tr>
<tr>
<td>90</td>
<td>0.1</td>
<td>0.5</td>
<td>1.0</td>
<td>1.8</td>
<td>2.8</td>
<td>4.0</td>
<td>5.4</td>
<td>7.0</td>
<td>9.0</td>
</tr>
<tr>
<td>100</td>
<td>0.1</td>
<td>0.4</td>
<td>0.8</td>
<td>1.5</td>
<td>2.3</td>
<td>3.3</td>
<td>4.4</td>
<td>5.6</td>
<td>7.3</td>
</tr>
</tbody>
</table>
with genetic algorithms, it is not our goal to scrutinize any technical aspect of these algorithm in the sense of tuning, enhancing, or combining single techniques. Rather, we want to check for the generality of the observations made in previous work. All authors of related work tried to explain their results and different models—often built mainly on intuition—have been proposed. We try to view them in a different light to identify strengths and weaknesses of these approaches.

After introducing the algorithms in more detail we will present experiments using the abstract search space model developed above and discuss the effects observed.

Figure 7.7. Absolute frequency of local minima as function of cost; shape parameter $\alpha = 2$, degree varied
7.3.1 Transformation-based Algorithms

First of all, we give an outline of a transformation-based randomized optimization algorithm that serves as a template for all algorithms we discuss later on (cf. Fig. 7.9).

For the current solution $s_i \in S$—the initial solution $s_0$ is an input parameter to the algorithm—an alternative solution $s' \in N(S)$ is chosen randomly, i.e., a transformation is applied to $s_i$. The new solution is costed and either accepted and becomes the new current solution, i.e., $s_{i+1} = s'$, or rejected otherwise, i.e., $s_{i+1} = s_i$. The acceptance is controlled by a strategy based on a set of parameters that may include the cost difference between $s_i$ and $s'$ as well as the time elapsed so far. The parameters and their importance vary from algorithm to algorithm. A transition from $s_i$ to $s_{i+1}$ is called down-hill step in case $c(s_i) \geq c(s_{i+1})$, and up-hill step otherwise. If $s'$ has been accepted we also check for a new record: if $c(s')$ is less than the lowest cost found so far, $s'$ is also stored in $s_{\text{best}}$.

After each such step the parameters are updated and the proceeding is repeated for the new, current solution until certain stopping criteria are fulfilled. Typically, stopping criteria suggested in previous work included elapsed time, number of solutions visited, number of steps without further improvement.

Intuitively, one of the aims in controlling the navigation should be to make up for bad initial solutions and to be as independent of the start solution as possible.

**Simple Improvement**

What we call Simple Improvement here actually is the “inner loop” of Iterative Improvement. We split Iterative Improvement up into two parts that implement very different principles of randomized optimization.
Algorithm Probabilistic Optimization
Input $s_0$ initial solution, $A_0$ set of initial parameters to accept $f$
Output $s_{best}$ best solution found

$i = 0$
$s_{best} = s_0$
$c_{best} = c(s_0)$
repeat
    choose $s' \in N(s_i)$
    if acceptable($c(s'), A_i$) then
        $s_{i+1} = s'$
        if $c(s') < c_{best}$ then
            $s_{best} = s'$
            $c_{best} = c(s')$
        endif
    else
        $s_{i+1} = s_i$
    endif
    $A_{i+1} = update(A_i)$
    $i = i + 1$
until stopping criteria fulfilled

Figure 7.9. Randomized optimization algorithm - Outline

In Figure 7.10 an outline of Simple Improvement is provided using the notation introduced above: A randomly chosen neighbor of the current solution is accepted only if its cost value is less than the current solution's costs. Evidently, this algorithm gets immediately trapped as soon as a local minimum is reached. In this case, any further processing is a waste of time as no improvement can be achieved. Thus, the crucial point of this algorithm is to detect whether the current solution is a local minimum. Usually, to inspect all the current solution's neighbors is too expensive [SG88]. As an approximation, a solution can be judged by a sample of its neighbors, i.e., it is considered to be a local minimum if no cheaper neighbor can be chosen within $r$ consecutive attempts. A local minimum classified this way is referred to as $r$-local-minimum (cf. [IK90]). Consequently, local minima can be identified only with a probability $p < 1$. After $r$ unsuccessful steps the algorithm terminates and the last current solution—which is also the best found so far—is returned. In [IK90] and [SG88] $r$ is either chosen as the number of neighbors or the number of edges in the corresponding join graph.

Our previous experiments concerning local minima however showed
Algorithm  Simple Improvement
Input  \( r \)-local-minimum size, \( s_0 \) initial solution
Output  last solution (=best solution found)

\[
i = 0 \\
t = 1 \\
\text{repeat} \\
\quad \text{choose } s' \in N(s_i) \\
\quad \text{if } c(s') < c(s_i) \text{ then} \\
\quad \quad s_{i+1} \leftarrow s' \\
\quad \quad t \leftarrow t + 1 \\
\quad \text{else} \\
\quad \quad s_{i+1} \leftarrow s_i \\
\quad \quad t \leftarrow t + 1 \\
\quad \text{endif} \\
\quad i \leftarrow i + 1; \\
\text{until } t > r
\]

Figure 7.10. Simple Improvement

that getting trapped in a local minimum is actually not as disadvantageous as may seem simply because almost all local minima in our search space are close to the global optimum and the difference in costs is not significant. Thus we will direct our attention in the next section to the more difficult question whether local minima recognized with the proposed techniques are really local minima.

**Simulated Annealing**

Simulated Annealing is probably the most prominent of randomized optimization techniques. It has been deployed and studied in almost all fields of research concerned with combinatorial optimization and besides experimental assessment also mathematical models to capture its behavior have been developed.

Simulated Annealing is a dynamic variant of the *Metropolis Algorithm* derived from statistical mechanics [MRR+53]. It also starts with a randomly chosen solution \( s \) and a neighbor \( s' \) is accepted with the probability

\[
e^{-\frac{1}{T} c(s') - c(s)}.
\]

In contrast to the Metropolis Algorithm, the temperature \( t \) decreases gradually and the algorithm terminates after \( t \) falls below a given threshold. The output of the algorithm is the least costly solution visited so far [KGV83].
7.3. PROBABILISTIC OPTIMIZATION STRATEGIES 141

Algorithm Simulated Annealing
Input $s_0$ initial solution, $T_0$ initial temperature
Output $s_{\text{best}}$ best solution found

$i \leftarrow 0$
$s_{\text{best}} \leftarrow s_0$
$c_{\text{best}} \leftarrow c(s_0)$

repeat
  repeat
    choose $s' \in N(s_i)$
    choose $p \in [0, 1]$
    \[ \text{if } p \leq e^{\frac{\min}{T_i} (c(s_i) - c(s'))} \text{ then} \]
    \[ s_{i+1} \leftarrow s' \]
    \[ \text{if } c(s') < c_{\text{best}} \text{ then} \]
    \[ s_{\text{best}} \leftarrow s' \]
    \[ c_{\text{best}} \leftarrow c(s') \]
  endif
  else
    $s_{i+1} \leftarrow s$
  endif
  $i \leftarrow i + 1$
until equilibrium reached
$T_{i+1} \leftarrow \text{lower}(T_i)$
until frozen

Figure 7.11. Simulated Annealing

Simulated Annealing is superior to Simple Improvement as it can escape local minima with a time dependent probability.

As previous work pointed out, Simulated Annealing depends heavily on the parameter $t$ and the cooling schedule that determines its decrease. If the initial temperature is too low the process terminates early without finding low costly solutions—if it is too high, time is wasted since many expensive solutions are considered. Experiments show that mathematical parameter estimation is only of very limited use what makes finding appropriate parameters a matter of experience [Haj88]. In [Swa89b] and [IK90] different simple heuristics for a computation of the parameters are presented.

Moreover, additional termination criteria like time limits or a maximum number of generated solutions were introduced. Unfortunately, combination of different termination criteria may veil the impact of single parameters as they may impact each other negatively. For example, if too short a
Algorithm  |  Iterative Improvement  
Input    |  \( r \)  r-local-minimum size  
Output   |  \( s_{\text{best}} \)  best solution found  

\[ i = 0 \]  
\[ c_{\text{best}} = \infty \]  
repeat  
  choose \( s \in S \)  
  \( s_{i+1} \leftarrow \text{Simple Improvement}(s, r) \)  
  if \( c(s') < c_{\text{best}} \) then  
    \( s_{\text{best}} \leftarrow s' \)  
    \( c_{\text{best}} \leftarrow c(s') \)  
  endif  
until stopping criteria fulfilled

**Figure 7.12. Iterative Improvement**

Running time limit is imposed, the effects of the coding cooling parameter may vanish etc.

### 7.3.2 Multi-start Strategies

Multi-start algorithm are repetitive applications of any randomized optimization algorithm starting each run of the particular algorithm with a new initial solution—different from the previous ones if possible.

Essentially, every randomized optimization algorithm can be used for multi-start optimization, however, navigating and multi-start technique are contradicting principles as one of the aims pursued in the previous is independence from the start solution, i.e., in navigating algorithms some sophisticated acceptance strategy is deployed so that algorithms can escape unfavorable conditions such as local minima. The higher this level of sophistication, the less we should expect the impact of multi-starts. We have seen a similar effect with genetic algorithms (cf. Chapter 6) and re-starts of the optimization.

**Iterative Improvement**

Iterative Improvement is the multi-start variant of Simple Improvement. It has been investigated in a broad variety of configurations (cf. [SG88, IK90, INSS92, SHC96]). As shown in Figure 7.12, Simple Improvement is applied repeatedly on different start solutions and \( s_{\text{best}} \), the best result found so far, is returned. As a termination criterion often a time limit is used be
Algorithm Transformation-free Optimization
Output \( s_{\text{best}} \) best solution found

\[
c_{\text{best}} \leftarrow \infty
\]

repeat

\[
\text{choose } s \in S
\]

if \( c(s) < c_{\text{best}} \) then

\[
s_{\text{best}} \leftarrow s
\]

\[
c_{\text{best}} \leftarrow c(s)
\]
endif

until stopping criteria fulfilled

Figure 7.13. Transformation-free Optimization

it either explicitly given or implicitly specified by a maximum number of repetitions.

Though neglected already a long time ago because of its simplicity and all too obvious shortcomings it has been re-discovered lately in various application areas as more sophisticated algorithms including genetic algorithms could not achieve performance gains that would justify the increased running time [Boe96].

**Transformation-free Optimization**

In Transformation-free Optimization the whole optimization is narrowed down to the multi-start principle: Solutions are chosen randomly and costed [GLPK94]. The best solution found is returned as soon as the stopping criterion is fulfilled (cf. Fig. 7.13). In other words Transformation-free Optimization implements uniform random sampling. this approach is distinguished by both the techniques deployed and its results: The algorithm is in so far interesting as the authors develop uniform generation of join orders which is all but a trivial task [GLPK95]. More notably, however, is the quantitative assessment presented. Sampling apparently outperforms other optimization algorithms including Simulated Annealing and Iterative Improvement in that it finds high quality solutions quicker. The differences to solutions found with other strategies are not significant.

In our setting here, Transformation-free Optimization appears rather trivial as the uniform generation of join orders is substituted by a uniform random choice of a node in the graph.
### Algorithm

Two-Phase Optimization

**Input**
- \( r \): r-local-minimum size,
- \( T_0 \): initial temperature,
- \( n \): number of runs

**Output**
- \( S_{best} \): best solution found

```plaintext
i ← 0
\( c_{init} ← \infty \)

while \( i < n \) do
    \( s ← \text{Iterative Improvement}(r) \)
    if \( c(s) < c_{init} \) then
        \( S_{init} ← s \)
        \( c_{init} ← c(s) \)
    endif
    \( i ← i + 1 \)
endwhile

\( S_{best} ← \text{Simulated Annealing}(S_{init}, T_0) \)
```

---

**Figure 7.14. Two-Phase Optimization**

### 7.3.3 Hybrid Strategies

Recognizing the shortcomings of a particular algorithm, one might be tempted to find improvements. Especially, combinations of algorithms attracted repeatedly attention. In order to speed up Simulated Annealing which can deliver results of high quality but is extremely time consuming, variants that apply a pre-processing first have been introduced [IK90, LVZ93]. The pre-processing tries to exclude very unfavorable initial solutions: The best solution found in the first phase is being explored in detail with Simulated Annealing. Therefore, the parameter setting can be tightened and less running time is needed to obtain low costly result solutions.

Among others the most prominent algorithm of this class is Two-Phase Optimization which applies several Iterative Improvement runs before the Simulated Annealing [IK90] (see Fig. 7.14).

### 7.4 Discussion

An assessment of randomized optimization algorithms is a difficult task because of the various degrees of freedom in setting up a test bed. In the previous chapters we have seen some fundamental properties common to all cost models and query optimization scenarios which can be used—and
in fact have been used—as a rough orientation; most notably the catalog variance. However, all algorithms presented rely on a couple of parameters which are difficult to tune in general. In [SG88] and [IK90], the parameters for Simulated Annealing e.g. have been chosen differently yet both parties claim that their choice was optimal with respect to elaborate test series. This in turn suggests that their test cases must have been of some significant structural difference. Another question totally ignored so far is whether the parameters should be adapted to the particular query type, i.e., its cost distribution. Up to date all heuristics that have been proposed to determine appropriate parameters are constant in the sense that they do not take the differences between given queries into account, e.g. one heuristic suggests to choose the initial temperature in Simulated Annealing so that a certain percentage of neighbors of the initial solution would get accepted. However, depending on the quality of the initial solution this choice may be anything from very slack to very restrictive.

As parameter ranges are unbound, at least covering large areas of "reasonable" values, an exhaustive tuning is infeasible. Moreover, the experiments in related work have been conducted with cost models of different levels of sophistication. Though we could expect basic tendencies to be found in all of them as motivated in Chapter 4, the numerical values differ in general. Most of the experiments were also presented as scaled costs, i.e., as factor of the best solution found by any algorithm. This can be a conclusive assessment when using industrial quality cost models, however, it is reflecting merely the trend with less sophisticated cost models. By this we mean that the order may very well be preserved—algorithm A is better then algorithm B—but e.g. twice the number of I/Os does not necessarily imply that B needs twice the running time. This becomes increasingly important if the cost values differ only by a few percent.

Conducting a series of experiments on the lines or related work, i.e., using a fixed set of parameters and an own cost model, would simply add yet another data set to the anyhow large collection of results available. Rather, we use the wealth of results as a basis for a discussion in order to explain some of the effects observed which could not be clarified completely yet.

To that end we will focus on two so far largely neglected aspects: The role of local minima and their classification on the one hand, and the evaluation of the influence of the multi-start principle.

### 7.4.1 Classification of Local Minima

As we have seen in the previous section navigation in the search space is the key element of most probabilistic optimization methods. Landscape models i.e., cost function of the kind \( f : R^2 \rightarrow R \), serve the intuition very well when it comes to discussing the behavior of these optimization algorithms. Landscapes subliminally suggest a continuous cost function and, what is synonymous, that the search space can be embedded sensibly into \( R^2 \). A reasonable embedding would however require the graph of the search
space to be planar, which is not the case for instances of non-trivial size. Let us however cling to this model for a moment. The navigation of an algorithm is then motivated as search along structures that have been described as "slopes", "valleys", "cups" etc. The notion of a local minimum then comes in very intuitively, almost compelling. Based on these considerations local minima have been suspected to have significant impact on the performance of an optimization strategy.

On the other hand, we found there are hardly any local minima of significant distance from the minimum, i.e., finding a local minimum is almost always as good as finding the optimum.

Here, we will investigate the accuracy of the classification of local minima. Judging a local minimum correctly eventually requires to evaluate the costs of all neighbors. However, the generation of all neighbors, i.e., applying all possible rules to a given solution, is a time consuming task. Therefore, all implementations proposed so far use sampling. For large parts of the cost range \(c > 2 \cdot \sigma\), this technique finds a better neighbor within a very small number of retries. Recall that the distributions of neighbor costs is almost symmetric. Thus in the upper cost range, the classification by sampling is sufficiently accurate. In the lower cost range where most local minima are located, the situation is very different. Assume a solution has 99 more costly and 1 less costly neighbor, i.e., it is not a local minimum. To classify the solution correctly for not being a local minimum and therefore finding the only neighbor with lower costs with an error of less than 5\% requires almost 300 retries. Keeping the error as low as possible is crucial—just assume the only better neighbor in the example above was the global optimum.

Figures 7.15 show the ratio of spurious local minima, i.e., solutions that were incorrectly classified as a local minimum, to real local minima as function of the degree. The size of the sample, i.e., the number of retries was varied between 10 and 100. In Figure 7.15a the sample is constant throughout the experiment, in Figure 7.15b, the size of the sample is a fraction of the degree. Swami et al. suggested what comes down to \(\frac{d}{3}\), Ioannidis et al. recommend \(d\), \(d\) being the degree. Even for large sample sizes, the number of spurious minima is distinctly above twice the number of real minima. Remarkably, as the experiments showed, the spurious minima are in significantly higher distance of the global minimum and scattered over a larger range of costs than real minima.

In the next experiment, we investigate the trade-off between navigation and multi-start principle. We run Simple Improvement with a limit of \(n\) retries and \(n\) steps in total and compare it to Transformation-free Optimization. The sample size of Transformation-free Optimization is also \(n\). Note, we re-start Simple Improvement if the limit is not yet exhausted. For both techniques we take the average of 1000 runs. Moreover Simple Improvement is always started in a high costly area, i.e., in a solution with roughly \(2\mu\) costs. This way we can exclude the effect of randomly choos-
7.4. DISCUSSION

(a) Fixed number of retries

(b) Number of retries as fraction of degree

Figure 7.15. Effectiveness of classification of local minima by sampling

In contradistinction, Simple Improvement needs a longer time just to navigate to the areas of interest wasting its energy along the way. This does not come as a surprise in principle. However, what is notable is the extent to which Transformation-free Optimization is able to outrun Simple Improvement. Specifically for small numbers of \( n \). The plot also suggests that in order to make effective use of multiple start a number of more than 20 better 50 restarts is advisable. In the following section we use the results obtained so far to interpret related work, i.e., setup and results and discuss their finding.

7.4.2 Analysis of Related Work

The four most prominent contributions to the debate are [SG88], [IK90, IK91], [SMK97] and [GLPK94]. Their results can be summarized as:

- [SG88]: Iterative Improvement outperforms Simulated Annealing
- [IK90, IK91]: Simulated Annealing outperforms Iterative Improvement, Two-Phase Optimization performs best
- [SMK97]: Iterative Improvement (often) outperforms Simulated Annealing. Two-Phase Optimization is better but not significantly better.

- [GLPK94]: Transformation-free Optimization converges fastest. Iterative Improvement outperforms Simulated Annealing.

Further related work like [SI93] or [LVZ93], suggest combinations of algorithms which we will not discuss here in detail.

Though most of the results independently found are coinciding largely, there is a distinct discrepancy concerning Iterative Improvement: In [IK91] authors elaborate on this fact presenting an analysis of the search space based on local minima. However, to understand the differences in result quality it is helpful to analyze the different test beds first.

Swami and Gupta use a linear search space where initial solution are not chosen with uniform probability, however, all solutions can be chosen with a probability that is distinctly greater than zero. Furthermore due to the set of rules, the probability to generate a neighbor solution with exactly the same costs is marginal.

In [IK91] things differ as authors use a bushy search space. However, they use only linear trees as initial solutions. Particularly, in the light of our findings about linear and bushy spaces together with the authors’ findings that in most cases the optimal plan was a bushy plan, this suggests that Iterative Improvement is made wasting a large portion of running time just to “escape” into the bushy space. In other words, only a subspace is used...
7.5. SUMMARY

to choose initial solutions.

To prune the search space their cost model applies commutative exchanges when necessary, i.e., the costing decides whether to use a join as given in the plan or to commutatively flip the inputs, whatever is cheaper. As a result the effect of plateaus where many neighbors can be of the same costs has higher incidence that in the previous setup. This leads to a skew of the classification of local minima as neighbors of equal costs are unacceptable in Iterative Improvement. Consequently more time is spent on classification of local minima. Also the number of spurious local minima is substantially higher. In contrast to that [SMK97] used initial solutions generated with non-uniform probability from the general space of bushy plans the same holds for the set up used in [GLPK94]. As an immediate consequence, Iterative Improvement proves a competitive algorithm in their work.

Given our analysis of Chapter 4 and the considerations about restart in Chapter 6 the excellent performance of multi-start algorithm is evident. This is further supported by Ioannidis' findings concerning Two-Phase Optimization as well as Lanzelotte, Valduriez and Zait's Toured Simulated Annealing; also Swami and Iyer use restarts showing that a very small number starts—less than 10 in their AB algorithm—makes already a difference.

Using a fair setup for the algorithms the differences in result quality are below any significance. This fact has been ignore in most studies and even for join orders of up to 100 joins, differences of less than 10% have been recorded and taken for an indication what algorithm to use.

### 7.5 Summary

Probabilistic algorithms can find acceptable solutions for even very large queries within very short running times. They are distinctly superior to deterministic heuristics, which also require only short running times, but as Steinbrunn et al. pointed out, produce increasingly low result quality for larger queries [SMK93].

In this section we first investigated the basic feature of linear and bushy search spaces concluding that the bushy space includes equally good and often better solutions than its linear counterpart. To asses number and location of local minima we devised the concept of abstract search spaces, which also served as a work bench to test algorithms. We discussed incidence and location of local minima with respect to the influence they have on optimization algorithm, in particular on local optimization also known as Iterative Improvement.

Our findings essentially suggest that there are only very few local minima that are of significant difference to the global optimum, i.e., for most of the local minima, the costs are comparable to the optimum in the sense they are acceptable optimization results.
Thus, getting trapped in a local minimum is not the primary reason for inferior performance of these algorithms as conjectured in related work. Rather the classification of local minima is a very time consuming task and too many solutions are wrongly classified as local minima. Algorithms that do not need to classify local minima like Simulated Annealing can put the saved time in further navigation. The effect can be further aggravated by the choice of a search space where the distribution of costs in neighborhoods is unduly skewed by plateaus, caused by a disadvantageous combination of rules and cost model. Another important factor is the choice of initial solutions. If they are chosen—not necessarily uniformly though—from the target space Simple Improvement or Iterative Improvement as well as sampling can find very good solutions. If the choice of initial solution is however limited to an unfavorable sub space, the multi-start principle is largely ineffective. Otherwise simple restart of an optimization can greatly improve the result quality because of the advantageous cost distribution.