RANDOMIZED ALGORITHMS FOR OPTIMIZING LARGE JOIN QUERIES †

Yannis E. Ioannidis
Younkyung Cha Kang
Computer Sciences Department
University of Wisconsin
Madison, WI 53706

Abstract

Query optimization for relational database systems is a combinatorial optimization problem, which makes exhaustive search unacceptable as the query size grows Randomized algorithms, such as Simulated Annealing (SA) and Iterative Improvement (II), are viable alternatives to exhaustive search We have adapted these algorithms to the optimization of project-select-join queries We have tested them on large queries of various types with different databases, concluding that in most cases SA identifies a lower cost access plan than II To explain this result, we have studied the shape of the cost function over the solution space associated with such queries and we have conjectured that it resembles a 'cup' with relatively small variations at the bottom This has inspired a new Two Phase Optimization algorithm, which is a combination of Simulated Annealing and Iterative Improvement Experimental results show that Two Phase Optimization outperforms the original algorithms in terms of both output quality and running time

1. INTRODUCTION

Query optimization is an expensive process, primarily because the number of alternative access plans for a query grows at least exponentially with the number of relations participating in the query The application of several useful heuristics eliminates some alternatives that are likely to be suboptimal [Seli79] , but it does not change the combinatorial nature of the problem Future database systems will need to optimize queries of much higher complexity than current ones This increase in complexity may be caused by an increase in the number of relations in a query [Kris86], by an increase in the number of queries that are optimized collectively (*global optimization*) [Gran81, Sell88], or by the emergence of recursive queries The heuristically pruning, almost exhaustive search algorithms used by current optimizers are inadequate for queries of the expected complexity Thus, the need to develop new query optimization algorithms becomes apparent

Randomized algorithms have been successfully applied to various combinatorial optimization problems Two such algorithms, *Simulated Annealing* [Kirk83] and *Iterative Improvement* [Naha86], are the best known such algorithms and have been proposed for query optimization of large queries as well Ioannidis and Wong applied Simulated Annealing to the optimization of some recursive queries [Ioan87] Swami and Gupta applied both Simulated Annealing and Iterative Improvement on optimization of select-project-join queries [Swam88]

In this paper, we address the problem of using randomized optimization algorithms for select-project-join queries The major contributions of the paper are the following First, we compare the performance of Simulated Annealing and Iterative Improvement by conducting several experiments with a variety of queries and databases We show that the former almost always produces a better output, which is different from the results of the study of Swami and Gupta [Swam88] Second, we map the shape of the cost function over the space of alternative access plans by a series of experiments on that space We conclude that the shape of the cost function resembles a 'cup', which leads to an explanation of the behavior of Simulated Annealing and Iterative Improvement and the difference with the results of Swami and Gupta Third, inspired by the above analysis, we propose a new query optimization algorithm that exhibits superior performance in terms of both output quality and running time

This paper is organized as follows In Section 2, we describe the specifics of our adaptation of Simulated Annealing and Iterative Improvement to query optimization and introduce the Two Phase Optimization algorithm In section 3, we present the results of an extensive experimental performance evaluation of the three algorithms In section 4, we describe an analysis of the shape of the cost function over the space of alternative access plans for a query, based on which we explain the behavior of the algorithms Section 5 describes the results of a limited set of experiments with the three algorithms on an enhanced set of query processing alternatives Finally, Section 6 contains some related work and Section 7 concludes and discusses future work

2. RANDOMIZED ALGORITHMS FOR QUERY OPTIMIZATION

Each solution to a combinatorial optimization problem can be thought of as a *state* in a space, i e , a node in the graph, that includes all such solutions Each state has a cost associated with it, which is given by some problem-specific cost function The goal of an optimization algorithm is to find a state with the globally minimum cost. Randomized algorithms usually perform *random walks* in the state space via a series of *moves* The states that can be reached in one move from a state S are called the *neighbors* of S A move is called *uphill* (*downhill*) if the cost of the source state is lower (higher) than the cost of the destination state A state is a *local minimum* if in all paths starting at that state any downhill move comes after at least one uphill move A state is a *global minimum* if it has the lowest cost among all states A state is on a *plateau* if it has no lower cost neighbor and yet it can reach lower cost states without uphill moves Using the above terminology we describe three randomized optimization algorithms We also discuss how we adapted these algorithms to query optimization

† Partially supported by NSF under Grant IRI-8703592

2 1. Generic Algorithms

In the descriptions below, we make use of a fictitious state S_∞ whose cost is ∞ Also, cost(S) is the cost of state S, and neighbors(S) is the set of neighbors of state S

2.1.1. Iterative Improvement (II)

The generic Iterative Improvement (II) algorithm is shown in Figure 2 1 The inner loop of II is called a *local optimization* A local optimization starts at a random state and improves the solution by repeatedly accepting random downhill moves until it reaches a local minimum II repeats these local optimizations until a *stopping_condition* is met, at which point it returns the local minimum with the lowest cost found As time approaches ∞, the probability that II will visit the global minimum approaches 1 [Naha86] However, given a finite amount of time, the algorithm's performance depends on the characteristics of the cost function over the state space and the connectivity of the latter as determined by the neighbors of each state

```
procedure II() {
  minS = S∞,
  while not (stopping_condition) do {
      S = random state,
      while not (local_minimum(S)) do {
          S' = random state in neighbors(S),
          if cost(S') < cost(S) then S = S',
      }
      if cost(S) < cost(minS) then minS = S,
  }
  return(minS),
}
```

Figure 2.1· Iterative Improvement

2.1.2. Simulated Annealing (SA)

A local optimization in II performs only downhill moves In contrast, Simulated Annealing (SA) does accept uphill moves with some probability, trying to avoid being caught in a high cost local minimum The generic algorithm† is shown in Figure 2 2 SA was originally derived by analogy to the process of annealing of crystals We use the same terminology for the algorithm parameters as in the original proposal (The terminology was adopted from the analogous physical process) The inner loop of SA is called a *stage* Each stage is performed under a fixed value of a parameter T, called *temperature*, which controls the probability of accepting uphill moves This probability is equal to $e^{-\Delta C/T}$, where ΔC is the difference between the cost of the new state and that of the original one Thus, the probability of accepting an uphill move is a monotonically increasing function of the temperature and a monotonically decreasing function of the cost difference Each stage ends when the algorithm is considered to have reached an *equilibrium* Then, the temperature is reduced according to some function and another stage begins, i e , the temperature is lowered as time passes The algorithm stops when it is considered to be *frozen*, i e , when the temperature is equal to zero It has been shown theoretically that, under certain conditions satisfied by some parameters of the algorithm, as temperature approaches zero, the algorithm converges to the global minimum [Rome85] Again, given a finite amount of time to reduce the temperature, the algorithm's performance depends on the characteristics of the cost function over the state space and the connectivity of the latter

```
procedure SA() {
  S = S0,
  T = T0,
  minS = S,
  while not (frozen) do {
      while not (equilibrium) do {
          S' = random state in neighbors(S),
          ΔC = cost(S') − cost(S),
          if (ΔC ≤ 0) then S = S',
          if (ΔC > 0) then S = S' with probability e^(−ΔC/T),
          if cost(S) < cost(minS) then minS = S,
      }
      T = reduce(T),
  }
  return(minS),
}
```

Figure 2.2: Simulated Annealing

2.1.3. Two Phase Optimization (2PO)

In this subsection, we introduce the *Two Phase Optimization* (2PO) algorithm, which is a combination of II and SA As the name suggests, 2PO can be divided into two phases In phase 1, II is run for a small period of time, i e , a few local optimizations are performed The output of that phase, which is the best local minimum found, is the initial state of the next phase In phase 2, SA is run with a low initial temperature Intuitively, the algorithm chooses a local minimum and then searches the area around it, still being able to move in and out of local minima, but practically unable to climb up very high hills Thus, 2PO is appropriate when such an ability is not necessary for proper optimization, which is the case for select-project-join query optimization as we demonstrate in the following sections

2.2. Problem Specific Parameters

When generic randomized optimization algorithms are applied to a particular problem, there are several parameters that need to be specified based on the specific characteristics of the problem For II, SA, and 2PO, they are the state space, the neighbors function, and the cost function

2.2.1. State Space

Each state in query optimization corresponds to an access plan (*strategy*) of the query to be optimized Hence, in the sequel, we use the terms state and strategy indistinguishably Using the heuristics of performing selections and projections as early as possible and excluding unnecessary cartesian products [Seli79], we can eliminate certain suboptimal strategies to increase the efficiency of the optimization Thus, we reduce the goal of the query optimizer to finding the best join order, together with the best join method for each join In this case, each strategy can be represented as a *join processing tree*, i e , a tree whose leaves are base relations, internal nodes are join operators, and edges indicate the flow of data. If all internal nodes of such a tree have at least one leaf as a child, then the tree is called *linear* Otherwise, it is called *bushy* Most join methods distinguish the two join

† In Figure 2 2, we keep track of the minimum cost state found (minS) In the end, it is minS that is reported as the answer, whereas a pure version of SA would report the state to which the algorithm has converged The version in Figure 2 2 can only improve on the results of the pure version and is the one that we use in this study

operands, one being the *outer* relation and the other being the *inner* relation An *outer linear join processing tree (left-deep tree)* is a linear join processing tree whose inner relations of all joins are base relations In our study, the strategy space includes all possible join processing trees, i e , both linear and bushy ones

2.2.2. Neighbors Function

The neighbors of a state, which is a join processing tree (i e , a strategy), are determined by a set of transformation rules Each rule is applied to one or two internal nodes of the state, replaces them by one or two new nodes, and usually leaves the rest of the nodes of the state unchanged With A, B, and C being arbitrary join processing formulas, the set of transformation rules that we used in our study is given below

(1) *Join method choice* $A \bowtie_{method_i} B \rightarrow A \bowtie_{method_j} B$

(2) *Join commutativity* $A \bowtie B \rightarrow B \bowtie A$

(3) *Join associativity* $(A \bowtie B) \bowtie C \leftrightarrow A \bowtie (B \bowtie C)$

(4) *Left join exchange* $(A \bowtie B) \bowtie C \rightarrow (A \bowtie C) \bowtie B$

(5) *Right join exchange* $A \bowtie (B \bowtie C) \rightarrow B \bowtie (A \bowtie C)$

Rule (1) changes the join method of a join operator, e g , from nested-loops to merge-scan Together with the algebraic rules (2) and (3), it is enough for the space of all interesting strategies to be connected Each of the additional exchange rules is equivalent to applying rules (2), (3), and (2) in that order Their advantage is that they avoid the use of join commutativity Applying join commutativity does not change the state cost for some join methods, e g , merge-scan, which tends to create plateaux in the state space The algorithms do not usually use the precise definition of a local minimum to recognize one but use approximations, and plateaux can be mistaken for local minima. Having many such false local minima in the state space degrades the output quality of randomized algorithms, especially II The join exchange rules reduce the number of plateaux by adding direct paths that bypass them

Most of the time, applying one of the above rules on some join nodes of a state does not affect the cost of the remaining nodes of the state Thus, the cost of a neighbor can be evaluated from the cost of the original state in constant time, taking into account the local changes performed by the transformations An unavoidable exception occurs when interesting sort order changes [Seli79], and the change must be propagated to the ancestors of the transformed nodes

2.2.3. Cost Function

The cost function that we used in our study only accounts for the I/O required by each strategy The precise formulas are not presented here due to lack of space They are based on the following assumptions (a) no pipelining, i e , temporary relations are created for the intermediate results, (b) minimal buffering for all operations, (c) on-the-fly execution of projections, and (d) no duplicate elimination on projections

2.3. Implementation Specific Parameters

Several parameters of randomized optimization algorithms are implementation specific These can be tuned to improve performance and/or output quality The following tables summarize our choices for the parameters of II, SA, and 2PO We arrived at them after some experimentation with various alternatives, and also based on past experience with the algorithms in query optimization [Ioan87] and other fields [John87]

parameter	value
stopping_condition	equal time to SA or 2PO
local_minimum	r-local minimum
next state	random neighbor

Table 2.1: Implementation specific parameters for II

parameter	value
initial state S_0	random
initial temperature T_0	$2*cost(S_0)$
frozen	$T<1$ and minS unchanged for 4 stages
equilibrium	16*(number of joins in query)
next state	random neighbor
temperature reduction	$T_{new} = 0.95*T_{old}$

Table 2.2: Implementation specific parameters for SA

parameter	value
stopping_condition (II phase)	10 local optimizations
initial state S_0 (SA phase)	minS of II phase
initial temperature T_0 (SA phase)	$0.1*cost(S_0)$

Table 2.3 Implementation specific parameters for 2PO

The only parameter that needs some explanation in the above tables is the definition of a local minimum for II Because there is a significant cost involved in exhaustively searching all neighbors of a strategy (let alone in verifying the truth of the precise definition of a local minimum), we use an approximation to identify a local minimum In particular, a state is considered to be a local minimum after *n* randomly chosen neighbors of it are tested (with repetition), where *n* is the actual number of its neighbors, none of which has lower cost Note that this does not guarantee that all neighbors are tested, since some may be chosen multiple times A state that satisfies the above operational definition is called an *r-local minimum*, to distinguish it from an actual local minimum Clearly, every local minimum is an r-local minimum, but the converse is not true Using the identification of an r-local minimum as the stopping criterion for a local optimization implies that some downhill moves may be occasionally missed, and a state may be falsely considered as a local minimum We claim, however, that the savings in execution time by using this approximation far outweigh the potential misses of real local minima This claim was verified in a limited number of experiments that we performed

3. PERFORMANCE EVALUATION

In this section, we report on an experimental evaluation of the performance and behavior of SA, II, and 2PO on query optimization First, we describe the testbed that we used for our experiments, and then we discuss the obtained results

3.1. Testbed

We experimented only with tree queries [Ullm82] containing only equality joins Due to known difficulties in their optimization [Ono88], specific attention was given to star queries Tree and star queries were generated randomly The query size ranged from 5 to 100 joins Each query was tested in conjunction with three different types of relation catalogs, i e , different relation cardinalities and join selectivities We made the usual assumptions about uniform distribution of values and independence of

values in the join attributes [Seli79, Whan85] Because of these assumptions, we used the number of unique values in the join attributes to control join selectivities The catalogs that we used are summarized in Table 3 1

Catalog Name	Relation Cardinality (tuples)	Number of unique values in join column (% relation cardinality)
relcat1	1000	[90, 100]
relcat2	[1000, 100000]	[90, 100]
relcat3	[1000, 100000]	[10, 100]

Table 3.1: Relation catalogs

As an example for the meaning of the entries in Table 3 1, in the 'relcat3' catalog, relation cardinalities were randomly chosen between 1000 and 100000 tuples, and the number of unique values in the join columns was randomly chosen between 10% and 100% of the cardinality of the corresponding relation The catalogs were selected so that we could test queries with different degrees of variance in the relation cardinalities and join selectivities, and thus, with different cost distributions in the state space This degree of variance increases as we move from 'relcat1' to 'relcat3'

Each relation page contained 16 tuples All relations had four attributes and were clustered on one of them There was a B^+-tree or hashing primary index on the clustered attribute, or the relation was physically sorted on it These alternatives were equiprobable The other attributes had a secondary index with probability 1/2, and again there was a random choice between a B^+-tree and hashing secondary index

Finally, the join methods that we considered were 'nested-loops' and 'merge-scan'

3.2. Experiment Profile

We implemented all algorithms in C, and tested them on a Sun-4 workstation when no-one else was using the machine We allowed the query size to grow up to 100 joins Twenty different queries were tested for each size up to 40 joins, and five were tested for larger sizes For each query and relation catalog, SA and 2PO were run five times, except for the cases where each run of SA would require more than two hours, for which no experiments were conducted with SA Thus, we have no results on SA for both types of queries with more than 60 joins for catalogs 'relcat2' and 'relcat3', and for star queries with more than 40 joins for catalog 'relcat3' This decision was based on the expectation that the behavior of SA compared to II and 2PO for the more expensive queries will be similar to that for the less expensive ones For each problem instance, II was also run five times, each run having as much time as the average time taken by a SA or 2PO run on the same query for the same catalog, depending on whether there were SA runs or not respectively

3.3. Behavior as a Function of Time

As part of the experiments, we recorded how the minimum cost found changed over time during the course of the algorithms' execution The typical behavior is shown in Figure 3 1 The particular example is for a 40-join tree query with the 'relcat1' catalog The y-axis represents the ratio of the strategy cost over the minimum strategy cost found for the query among all runs of all algorithms Clearly, there are significant differences between SA and II On the one hand, after a few local optimizations, II reaches a state of cost that is close to the minimum cost found by a complete run of SA The improvement that this cost represents over the initial random state cost is several orders of magnitude, in general After that, II makes only small improvements On the other hand, in the early stages, SA wanders around states of very high cost During the later stages, however, it reaches states of costs similar to those found by II after a few local optimizations, and most often, it eventually finds a better state This observation indicates that SA is performing useful work only after it reaches low cost states and the temperature is low, since at high temperature, it only visits high cost states This fact is what motivated the introduction of 2PO The first phase of 2PO produces a low cost state from which the second phase can start with low temperature Indeed, we observe that initially 2PO improves as quickly as II, but soon it surpasses it, and eventually converges to its final solution much more rapidly than SA

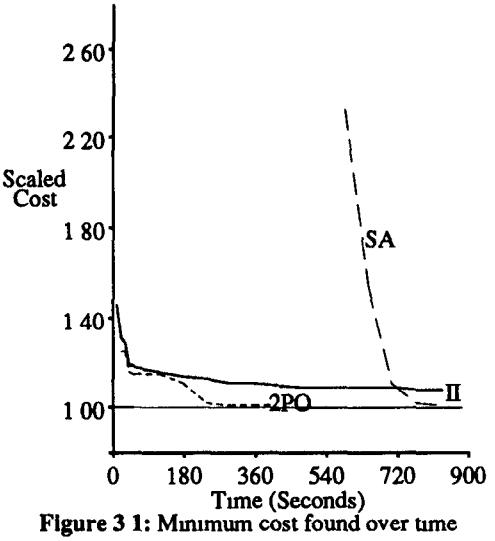

Figure 3 1: Minimum cost found over time

3.4. Output Quality

In general, we observed no significant qualitative difference in the relative output of the algorithms between tree and star queries Because of this and the lack of space, we only present the results for 'relcat2' for tree queries, whereas we present the

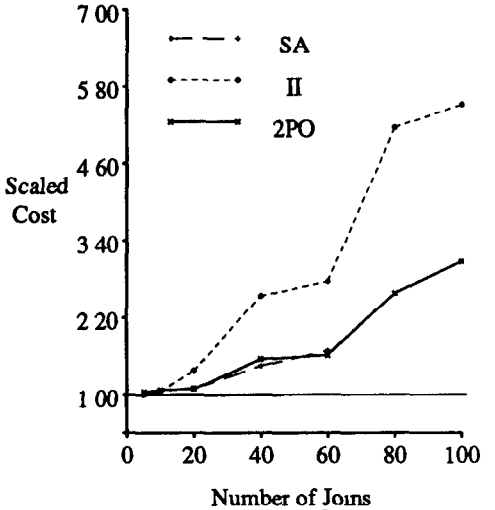

Figure 3.2: Average scaled cost of the output strategy for tree queries and the 'relcat2' catalog

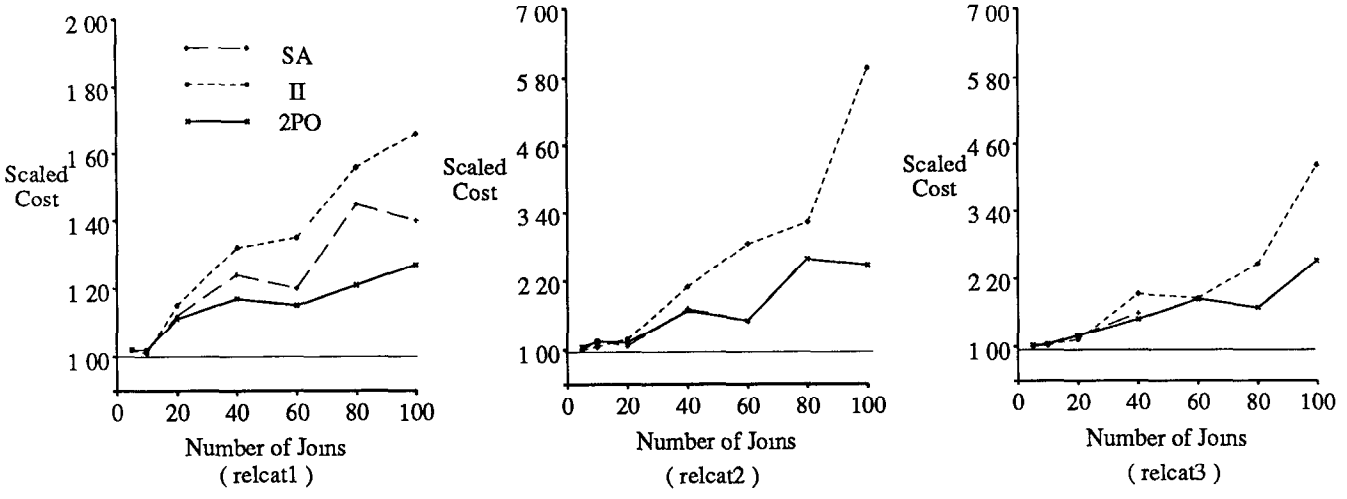

Figure 3.3 Average scaled cost of the output strategy of SA, II, and 2PO for star queries

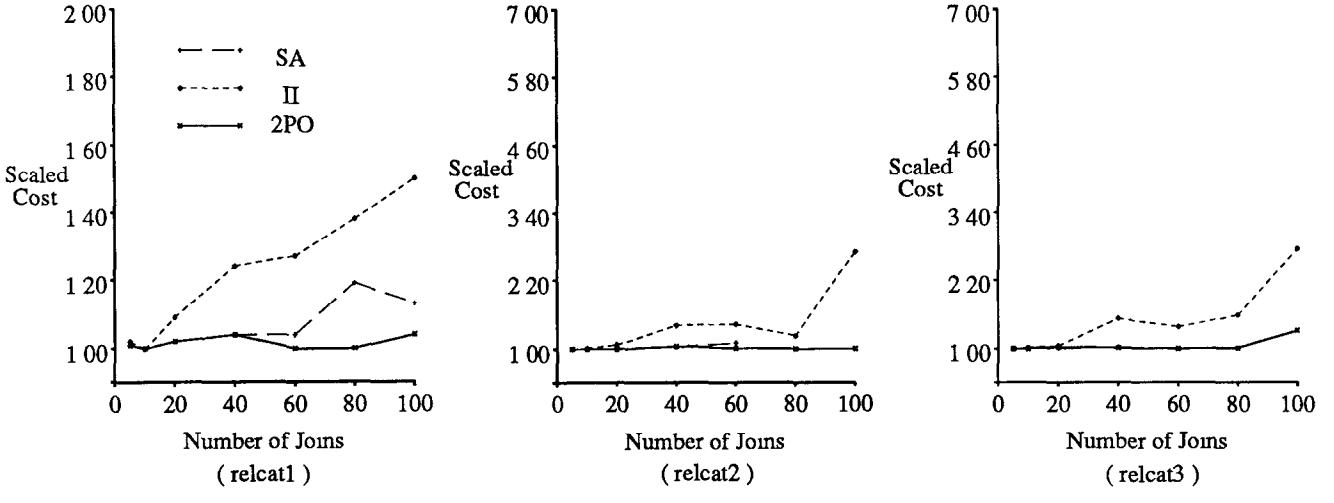

Figure 3 4 Best scaled cost of the output strategy of SA, II, and 2PO for star queries

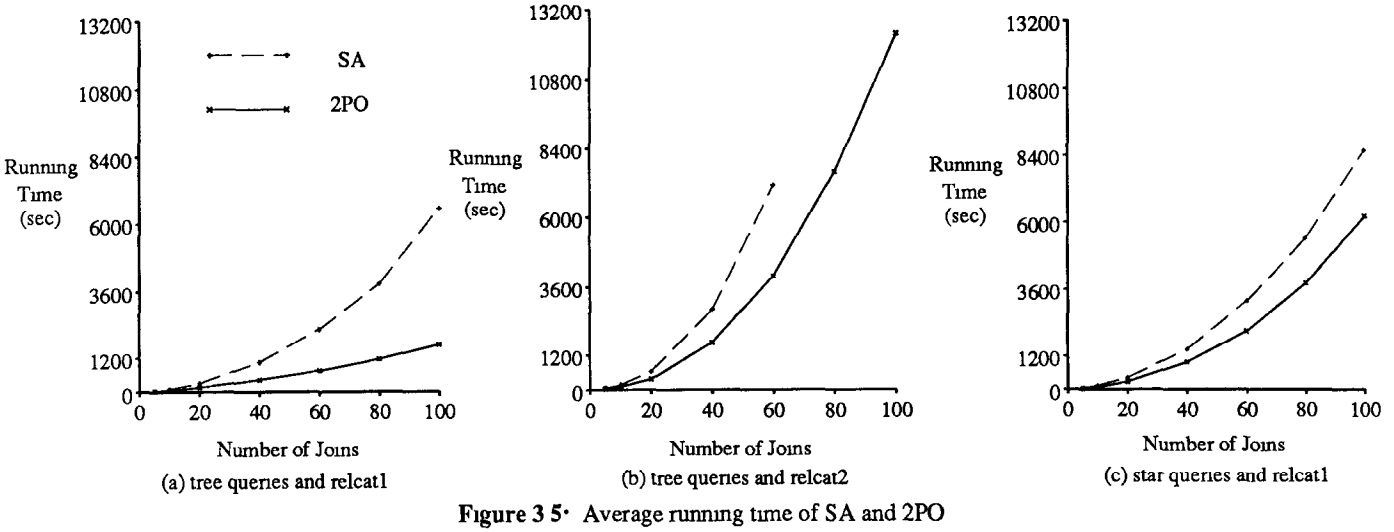

Figure 3 5 Average running time of SA and 2PO

complete set of results for star queries, since they are the hardest tree queries to optimize [Ono88] The cost of the output strategies produced by the algorithms for the various relation catalogs as a function of query size is shown in Figures 3 2 for tree queries and Figure 3 3 for star queries Again, the y-axis represents scaled cost, i e , the ratio of the output strategy cost over the minimum cost found for the query among all runs of all algorithms For each size, the average over all queries of that size, of the average scaled cost over all five runs of each query is shown

We discuss how the results change as we move along two dimensions of interest query size and variance in catalog parameters (i) *Query size* For small queries, with 5 or 10 joins, there is no difference among the three algorithms, regardless of the catalog type Almost all runs of the three algorithms find states with the same cost In general, as query size grows, the output of 2PO improves compared to that of SA, which improves compared to that of II At the same time, however, the average output strategy cost of all algorithms becomes less stable, i e , the average scaled cost moves farther from 1 This means that there are more cases in which algorithms miss the optimum Of the three algorithms, however, 2PO is the relatively most stable one (ii) *Variance in catalog parameters* The output difference between the algorithms increases with higher variance in the relation cardinalities and the join selectivities, i e , as we move from 'relcat1' to 'relcat3' Interestingly, there is not much difference between 'relcat2' and 'relcat3' In fact, occasionally, 'relcat2' gives rise to higher differences between 2PO and II/SA than 'relcat3' This shows that variance in relation size has a more significant impact on the output of the two algorithms than variance in selectivity factors †

It is also interesting to compare the best output found among the five runs of each algorithm for each problem instance We show the average of that over all star queries of a given size in Figure 3 4 Clearly, when the best of five runs for each algorithm is considered, 2PO not only outperforms the other algorithms in all cases, but it also becomes very stable, i e , the best scaled cost of five runs of 2PO is very close, if not equal, to 1 This suggests that 2PO is the algorithm of choice for large queries, particularly if it is run a small number of times for stability We should also observe, however, that the performance of SA becomes very stable as well To the contrary, II is still not very stable, rarely outperforming SA The effect of query size and variance in catalog parameters on the relative output quality of SA, II, and 2PO in this case is similar to that for the average of five runs, and we elaborate on it no more

3.5. Query Optimization Time

The average running time of 2PO and SA as a function of query size for various interesting cases is shown in Figure 3 5 (Recall that II was given the same amount of time as SA or 2PO, depending on whether SA was run or not.) Again due to lack of space, we only show some of the relevant graphs Those that correspond to the remaining cases are similar Below, we discuss the effect of query size, variance in catalog parameters, and query type on query optimization time (i) *Query size* Clearly, 2PO needs less time than SA in all cases As expected, the absolute difference in running time increases with query size To the contrary, the relative difference increases with query size for tree queries (it reaches a factor of 4 for 100-join tree queries with 'relcat1'), but decreases with query size for star queries The latter tend to be less regular than tree queries, and therefore, tend to need more time in phase 2 of 2PO (ii) *Variance in catalog parameters* The cost steadily increases from 'relcat1' to 'relcat3', as expected This can be seen, for example, in Figures 3 5 (a) and 3 5 (b), which correspond to tree queries for 'relcat1' and 'relcat2' (iii) *Query type* As expected [Ono88], star queries took significantly more optimization time than random tree queries In addition, the improvement of 2PO over SA for star queries was much smaller than for star queries This is shown in its more dramatic instantiation in Figures 3 5 (a) and 3 5 (c), which correspond to the two query types for 'relcat1'

† Note that we used a different scale for the y-axis for 'relcat1' than for the other catalogs

4. STATE SPACE ANALYSIS ‡

To understand the results of the performance evaluation and the cause for the behavior of SA, II, and 2PO that was presented in the previous section, we studied the shape of the cost function over the state space that the three algorithms had to search The outcome of this study is reported in this section

4.1. Shape of Cost Function over the Strategy Space

The size of the strategy space is prohibitive of any attempt of an exhaustive search of it Hence, in order to study the cost function shape, randomization was employed again, and the following types of experiments were performed

(i) Random generation of 10000 strategies and calculation of their costs

(ii) "Random" generation of 10000 local minima and calculation of their costs This was achieved by performing a local optimization from each strategy generated in experiment (i)

(iii) "Random" walks in areas of low cost strategies Our purpose was to get a feeling for the number of good local minima that exist and their mutual distance For each query tested, we performed 5 random walks, each one of which started from a low cost local minimum Each walk was a sequence of 2000 smaller parts Each part consisted of a series of uphill moves followed by a series of downhill moves Each series of uphill moves ended when the strategy cost exceeded a prespecified limit, which ensured that the search remained in areas of low cost strategies (The limit was equal to 5 times the average local minimum cost found in experiment (ii)) Each series of downhill moves ended in a local minimum Thus, a total of 10000 local minima were visited in experiment (iii) also

We tested ten 20-join and ten 40-join tree queries, and an equal number of the same sizes of star queries Each query was tested with all three relation catalogs For experiments (ii) and (iii), we used a better approximation for a local minimum than the r-local minimum that we used in the implementation of II, to improve the accuracy of the analysis In particular, we used the approximation of the *p-local minimum*, which is defined as a

‡ The experiments reported in this section for the state space analysis were conducted using Condor [Litz88] Condor is a facility for executing UNIX jobs on a pool of cooperating workstations Jobs are queued and executed remotely on workstations at times when those workstations would otherwise be idle Our experiments are very time-consuming Without Condor it would be very difficult to collect all the necessary data in a reasonable time

strategy none of whose neighbors has a lower cost Note that plateaux can still be mistaken as local minima with this definition Unless otherwise noted, any reference to a local minimum in this subsection refers to a p-local minimum

The results of experiments (i) and (ii) are summarized in Figure 4 1 for star queries We do not show the results of the random tree queries tested, because they are very similar to those of Figure 4 1 Queries 1 to 10 have 20 joins and queries 11 to 20 have 40 joins There is no significance in the order of placement of the various queries on the x-axis, except that they are grouped into those with 20 joins and those with 40 joins For each query, we show the scaled cost of the average strategy (experiment (i)) and the scaled cost of the average local minimum (experiment (ii)) that corresponds to the query Again, the scaled cost is the ratio of a strategy cost over the lowest local minimum cost found in experiment (ii) for the corresponding query The average local minima costs are several orders of magnitude lower than the average state costs As the query size grows, the difference remains relatively stable for queries with the same catalog, although the absolute scaled costs increase On the other hand, the difference seems to increase as the catalog changes from 'relcat1' to 'relcat3' Finally, compared to the average cost of random strategies, the average cost of local minima is relatively close to the best local minimum cost The specific ratio of average vs best local minimum cost is affected by the variance in the catalog parameters and by the particular query itself In some cases, it represents cost differences as high as two orders of magnitude (e g , 40-join queries with 'relcat3') Even in these cases, however, that cost difference is insignificant compared to the difference between the average strategy cost and the best local minimum cost, which is higher than five orders of magnitude Thus, we can conclude that most local minima are not only far better than the average random state, but there is also relatively small variance in their costs

During experiment (ii), we also measured the number of downhill moves taken by the local optimizations Figure 4 2 shows the average number of downhill moves for each star query This number of downhill moves is higher for star queries than for tree queries Moreover, it increases as the query size grows and as the catalog changes from 'relcat1' to 'relcat3', with a maximum for 'relcat2' The general conclusion from the results in Figure 4 2 is that starting at a random state many downhill moves are needed to reach a local minimum

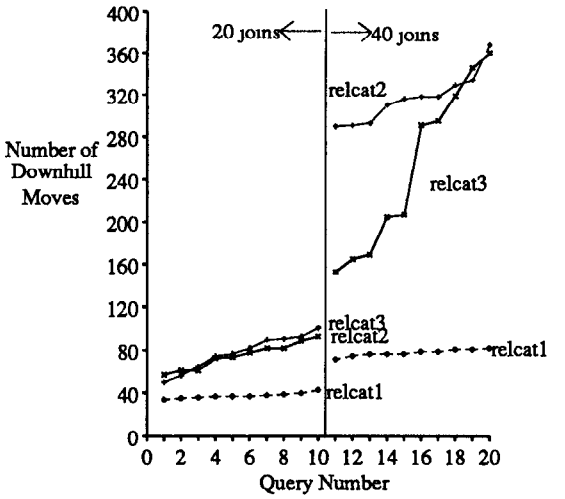

Figure 4.2: Number of downhill moves for star queries

In experiment (iii), for the same set of tree and star queries as before, we counted the number of local minima visited that had distinct costs This only provides a lower bound on the number of distinct local minima In addition, for each query, we measured the average distance between two consecutively visited local minima Again, since there could be shorter paths between them, this only provides an upper bound on their distance These results are summarized in Table 4 1 where we show the range of values for both measured quantities for all queries of both query types and all three catalogs

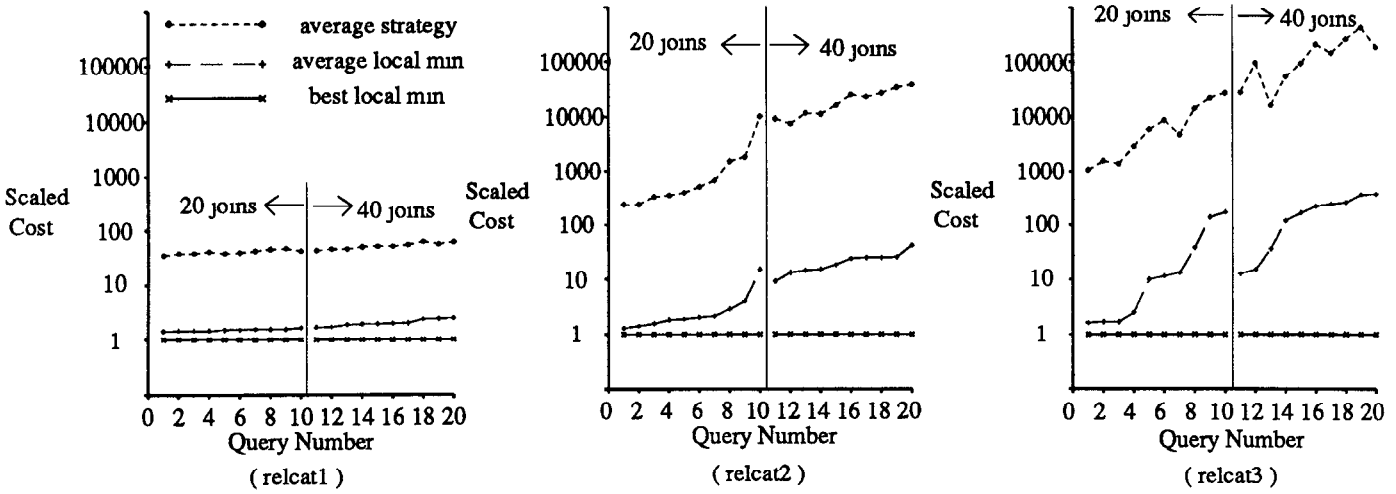

Figure 4 1: Cost distribution of random states and local minima for star queries

	#distinct local minima (lower bound)	distance (upper bound)
min	470	6
average	4681	32
max	9903	221

Table 4 1: Local minima in the low cost area

The general conclusion from the results in Table 4 1 is that, for all queries, any connected area of relatively low cost strategies (due to the prespecified limit) contains many local minima Moreover, these local minima are relatively close to each other (the size of the state space is several orders of magnitude higher than any distance reported in Table 4 1)

All the above observations can be summarized as follows

(1) The average local minimum has relatively low cost compared to the average state (from (i) and (ii))

(2) The average distance from a random state to a local minimum is long (from (ii))

(3) The number of local minima is large (from (iii))

(4) Many local minima are connected through low cost states within short distance (from (iii))

In addition to the above, the overall behavior of the three algorithms described in Section 3 is summarized below

(5) Searching among low cost states that are connected to each other produces better results than searching in multiple, (potentially) unconnected, areas of states

The above points (1)-(5) lead to the following conjecture regarding the shape of the cost function over the strategy space

The shape of the cost function resembles a 'cup', with some relatively small variations at the bottom

In other words, there is a small area of strategies with low costs, the *cup bottom*, surrounded by the remaining strategies with increasingly higher costs There is relatively small variation among the costs in the cup bottom, but enough to make exploration of that area worth while The space with which we deal is multidimensional, so it is hard to visualize For a 1-dimensional cost function, the corresponding situation is shown in Figure 4 3

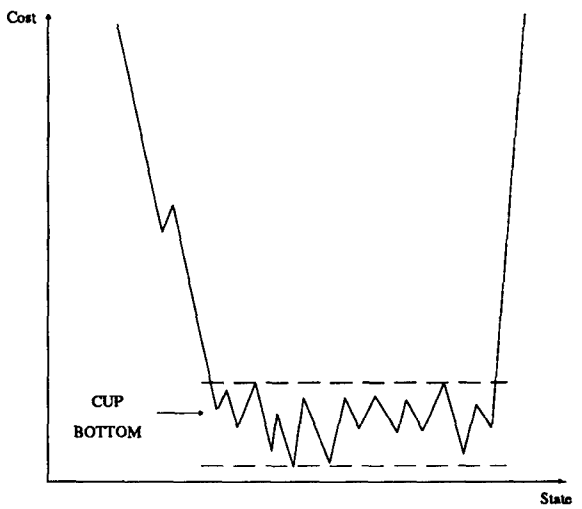

Figure 4.3 Shape of cost function

Actually, the reported experimental results do not exclude the case where the shape of the cost function is several cups whose bottoms are at similar cost levels Additional experiments, however, indicate that the existence of multiple cups is unlikely In particular, whenever the output strategies of two runs of 2PO on the same query differed significantly in cost, we ran SA with low temperature starting from one of the strategies The low temperature guaranteed that the algorithm did not move into high cost states, i e , that it remained in the same cup In all cases tested, SA did visit the other strategy, which indicates that the inability of 2PO to visit the best of the two strategies in both runs was most likely due to the randomness of the algorithm and not due to 2PO searching in two different cups in the two runs

4.2. Explanation of Behavior as a Function of Time

Using the conjecture of the cup shaped cost function, we are now in a position to explain the typical behavior of SA, II, and 2PO over time as shown in Figure 3 1 SA starts from a random state, which tends to be at the high cost area While the temperature remains high, due to the large number of uphill moves from states in the high and middle cost area, and the high probability of accepting uphill moves, SA tends to spend much time without improvement. After the temperature is reduced significantly, SA reaches the cup bottom, which it explores extensively, by taking advantage of its ability to visit many local minima by accepting uphill moves On the other hand, II can reach the cup bottom quickly by accepting only downhill moves Since most local minima are there, II can find a relatively good one within a few local optimizations This explains why II performs so well in the beginning stages, while SA performs so poorly As for 2PO, as expected, it reaches the cup bottom very quickly (II phase) and then improves further by searching in that area (SA phase), finishing in less time than either of the other algorithms

4.3. Explanation of Output Quality

We can also explain why 2PO (and usually SA also) outperforms II in terms of output quality II visits relatively few local minima, because finding one is expensive for the following reasons (a) for each local optimization, II has to generate a

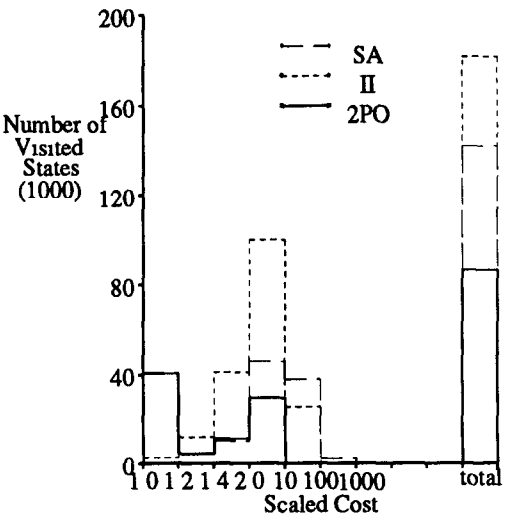

Figure 4.4: Number of visits in each cost area

random state and evaluate its cost, both of which are expensive operations, (b) starting at a random state, it takes time to find a local minimum, because the distance between the two is long (point (2) and Figure 4 2), and (c) during a local optimization, especially when in a low cost state, II tries many neighbors before it finds a downhill move On the other hand, 2PO and SA spend a reasonable amount of time at the cup bottom (with low temperature), and are able to explore it much more thoroughly than II, thus increasing the probability that they find the global minimum The above was also verified by measuring the number of states visited by each algorithm as a function of the cost of the state We observed that, although overall II visits more states than SA, which visits more states than 2PO, the last two visit more states of low costs than the first A typical situation is shown in Figure 4 4 The example is for a 40-join tree query with the 'relcat1' catalog Similar behavior was observed for other queries

5. INCLUDING HASH-JOIN, PIPELINING, AND BUFFERING

As an extension to the study described in the previous sections, we have conducted a limited set of experiments with a strategy space that included 'hash-join' as an alternative join method (in addition to 'nested-loops' and 'merge-scan') and with a cost function that partially removed assumptions (a) and (b) of Section 2 2 3, i e , it allowed pipelining for nested-loops and in several cases took advantage of arbitrary amounts of available buffer space We experimented with twenty 20-join and twenty 40-join tree queries with all three catalogs The set-up of these experiments was exactly the one described in the previous sections We studied both the behavior of all three algorithms in the new setting and the characteristics of the shape of the cost function The results are very similar to those of Section 3 and 4 2PO always outperforms II and SA in terms of both output quality and running time In addition, the conjecture of the 'cup'-shaped cost function remains valid This implies that our conclusions in Section 3 and 4 were not a result of our choice of join methods or the specific restrictions (a) and (b) on the cost function As a point of reference, in Tables 5 1 and 5 2, we show the average and best scaled cost of the output strategies of five runs for all three algorithms for all combinations of query size and catalog As before, the costs are scaled based on the lowest strategy cost found for each specific query and catalog, and averaged over all queries with the same characteristics

	relcat1		relcat2		relcat3	
	20	40	20	40	20	40
II	1 01	1 01	1 25	3 57	1 09	1 42
SA	1 00	1 00	1 12	1 36	1 02	1 11
2PO	1 00	1 00	1 07	1 29	1 01	1 06

Table 5.1. Average scaled cost of the output strategy

	relcat1		relcat2		relcat3	
	20	40	20	40	20	40
II	1 00	1 00	1 13	3 11	1 05	1 32
SA	1 00	1 00	1 05	1 15	1 00	1 06
2PO	1 00	1 00	1 00	1 02	1 00	1 00

Table 5.2. Best scaled cost of the output strategy

6. RELATED WORK

Query optimization has been a very active area of research for relational database systems The reader is referred to the survey paper by Jarke and Koch [Jark84] and the book by Kim, Reiner, and Batory [Kim86] Regarding large join queries, their optimization was specifically addressed by Krishnamurthy et al [Kris86], who proposed a quadratic algorithm that took advantage of the form of the join cost formula. In this section, we want to primarily compare our work with that of Swami and Gupta [Swam88], who conducted a performance evaluation of SA and II that was similar to our study reported in Section 3 There are several distinct differences between the two studies First, Swami and Gupta's strategy space consists of left-deep trees only, whereas we include all strategies Second, they examine only one join algorithm, namely hash-join, whereas we have experimented with two of them, namely nested-loops and merge-scan Third, they assumed a main memory database and therefore used cpu-time as the cost of a strategy, whereas we used I/O time for that Fourth, they used an approximation for a local minimum in the implementation of II that is different from the r-local minimum that we used Fifth, the neighbors of a strategy are determined by two transformation rules, namely cyclic exchange of the position of two or three relations in the join tree that corresponds to the strategy, whereas most of our transformation rules are based on algebraic properties of joins There are also several other implementation difference between the two studies that seem unnecessary to point out here

The results of our study presented in Section 3 contradict those of Swami and Gupta [Swam88], whose conclusion was that SA was never superior to II, independent of the amount of time that was given to it In principle, any combination of the differences mentioned above could be the source for the difference in the results Intuitively, however, we believe that the primary reason is the difference in the choice of neighbors for the strategies in the strategy space, i e , the difference in the transformation rules that were used in the two studies Swami and Gupta's transformation rules generate neighbors that have large differences in their cost, which makes the shape of the cost function much less smooth (not a cup) Therefore, SA does not have the opportunity to spend much time in a low cost area and performs poorly On the other hand, II can move down to a local minimum in a few moves and therefore visits many of them in the same amount of time Thus, the two algorithms are ordered differently in terms of output quality in the two studies Although we believe that most likely the above is the main reason for the difference in conclusions, further investigation is required to understand the issues precisely

As a closing comment, we should mention that our work on mapping the shape of the cost function over the strategy space of a query is unique, Swami and Gupta included no such study in their work. Moreover, the cup formation gave the ability to use 2PO as an optimization algorithm, which exhibits superior performance On the other hand, Swami proposed and experimented with a set of heuristics, which in general improved the performance of both II and SA [Swam89]

7. CONCLUSIONS AND FUTURE WORK

We have adapted the well-known randomized optimization algorithms of Simulated Annealing and Iterative Improvement to optimization of large join queries and studied their performance We observed that II performs better than SA initially, but if enough time is given to SA, it outperforms II We studied the shape of the cost function over the state space of a query, and experimentally verified that it resembles a cup with a non-smooth bottom Based on this observation, we explained the behavior of the two algorithms Finally, making use of the cup shape of the

cost function, we proposed the Two Phase Optimization algorithm, whose performance is superior to that of the other algorithms with respect to both output quality and running time

The work reported in this paper is only the beginning in understanding how randomized optimization algorithms perform on complex queries, and also what the shape of the cost function is over the space of equivalent strategies for queries There are several issues that are interesting and on which we plan to work in the future First, we want to investigate the sensitivity of this study's conclusions to the specific choices that we made for various parameters In particular, we would like to complete our experiments with hash-join, pipelining, and various degrees of buffer availability, which all affect the cost formulas for individual joins and the cost relationships between neighbors We would also like to experiment with non-uniformly distributed data Second, we want to identify the key properties that cause the cup formation of the cost function This will be helpful not only in understanding the behavior of the algorithms, but also in providing us with criteria for the applicability of Two Phase Optimization Such results will be very helpful in extensible query optimizers [Care86] Third, we want to compare Two Phase Optimization and the other randomized optimization algorithms with the traditional ones, e g , those of System-R [Seli79] or Starburst [Lohm88] This should lead into an understanding of the relative advantages between generation-based and transformation-based query optimization Finally, we want to experiment with other types of relational queries, i e , cyclic queries and queries that involve union

8. REFERENCES

[Care86]
M Carey and et al , The Architecture of the EXODUS Extensible DBMS, in *Proc of the 1986 International Workshop on Object-Oriented Database Systems*, edited by K Dittrich and U Dayal, Pacific Grove, CA, September 1986, pp 52-65

[Gran81]
J Grant and J Minker, Optimization in Deductive and Conventional Relational Database Systems, in *Advances in Data Base Theory, Vol I*, edited by H Gallaire, J Minker, and J M Nicolas, Plenum Press, New York, NY, 1981, pp 195-234

[Ioan87]
Y E Ioannidis and E Wong, Query Optimization by Simulated Annealing, in *Proceedings of the 1987 ACM-SIGMOD Conference*, San Francisco, CA, June 1987, pp 9-22

[Jark84]
M Jarke and J Koch, Query Optimization in Database Systems, *ACM Computing Surveys* **16** (June 1984), pp 111-152

[John87]
D S Johnson, C R Aragon, L A McGeoch, and C Schevon, *Optimization by Simulated Annealing An Experimental Evaluation (Part I)*, unpublished manuscript, June 1987

[Kim86]
W Kim, D Reiner, and D Batory, *Query Processing in Database Systems*, Springer Verlag, New York, NY, 1986

[Kirk83]
S Kirkpatrick, C D Gelatt, Jr , and M P Vecchi, Optimization by Simulated Annealing, *Science* **220**, 4598 (May 1983), pp 671-680

[Kris86]
R Krishnamurthy, H Boral, and C Zaniolo, Optimization of Nonrecursive Queries, in *Proceedings of the 12th International VLDB Conference*, Kyoto, Japan, August 1986, pp 128-137

[Litz88]
M J Litzkow, M Livny, and M W Mutka, Condor - A Hunter of Idle Workstations, in *The 8th International Conference on Distributed Computing Systems*, 1988, pp 104-111

[Lohm88]
G M Lohman, Grammar-like Functional Rules for Representing Query Optimization Alternatives, in *Proceedings of the 1988 ACM-SIGMOD Conference*, Chicago, IL, June 1988, pp 18-27

[Naha86]
S Nahar, S Sahni, and E Shragowitz, Simulated Annealing and Combinatorial Optimization, in *Proceedings of the 23rd Design Automation Conference*, 1986, pp 293-299

[Ono88]
K Ono and G M Lohman, *Extensible Enumeration of Feasible Joins for Relational Query Optimization*, IBM Research Report RJ6625, December 1988

[Rome85]
F Romeo and A Sangiovanni-Vincentelli, Probabilistic Hill Climbing Algorithms Properties and Applications, *Proceedings of IEEE Conference on VLSI*, 1985, pp 393-417

[Seli79]
P Selinger, M M Astrahan, D D Chamberlin, R A Lorie, and T G Price, Access Path Selection in a Relational Database Management System, in *Proceedings of the 1979 ACM-SIGMOD Conference*, Boston, MA, June 1979, pp 23-34

[Sell88]
T K Sellis, Multiple Query Optimization, *ACM Transactions on Database Systems* **13**, 1 (March 1988), pp 23-52

[Swam88]
A Swami and A Gupta, Optimization of Large Join Queries, in *Proceedings of the 1988 ACM-SIGMOD Conference*, Chicago, IL, June 1988, pp 8-17

[Swam89]
A Swami, Optimization of Large Join Queries Combining Heuristics and Combinatorial Techniques, in *Proceedings of the 1989 ACM-SIGMOD Conference*, Portland, OR, June 1989, pp 367-376

[Ullm82]
J D Ullman, *Principles of Database Systems*, Computer Science Press, Rockville, MD, 1982

[Whan85]
K Y Whang, *Query Optimization in Office-by-Example*, IBM Research report, RC11571, 1985