RANDOMIZED ALGORITHMS FOR QUERY OPTIMIZATION

by

Younkyung Cha Kang

Computer Sciences Technical Report #1053

October 1991

RANDOMIZED ALGORITHMS FOR QUERY OPTIMIZATION

by

YOUNKYUNG CHA KANG

A thesis submitted in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy
(Computer Sciences)

at the
UNIVERSITY OF WISCONSIN - MADISON
1991

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 Iterative Improvement and Simulated Annealing, are viable alternatives to exhaustive search.

We present several analytical and experimental results that shed some light on the shape of the cost function of the access plan spaces with which the randomized query optimization algorithms must deal. These are the space that includes only left-deep trees, and the space that includes both deep and bushy trees. We conclude that the shape of both spaces essentially forms a 'well', but of a distinctly different quality. This has inspired a new algorithm, Two Phase Optimization, which is a combination of Simulated Annealing and Iterative Improvement.

We show how Iterative Improvement, Simulated Annealing, and Two Phase Optimization perform on the two spaces of interest and explain the results based on the above analysis on the shape of their cost function. In particular, the results show that Two Phase Optimization outperforms the original algorithms in terms of both output quality and running time. Additional experimentation shows that Two Phase

Optimization is also very effective on small queries, having the traditional algorithm of System R as the basis for comparison. Thus, it emerges as a strong candidate for query optimization in future database systems.

Finally, a comparison between the two spaces of interest in their form used in this work leads to the rather surprising conclusion that the space of both deep and bushy trees is to be preferred over the space of left-deep trees for query optimization. The former has a more definite shape of a 'well' than the latter and also includes more efficient alternatives in most cases. Hence, for the specific choices of connections of alternative access plans in the two spaces, the space of deep and bushy trees is superior with respect to both optimization time and output quality of the algorithms that use it as their search space.

ACKNOWLEDGMENTS

I am deeply indebted to my advisor, Professor Yannis Ioannidis, for the motivation and enthusiastic guidance he provided in the course of this research. He has been extraordinarily patient and supportive.

I am very grateful to Professors Mike Carey and Raghu Ramakrishnan for their extremely helpful comments and suggestions. Their comments and suggestions concerning this dissertation have greatly improved its quality. I would also like to thank the other members of my committee, Professors David DeWitt and Douglas Bates, for their generousity and encouragement.

My special thanks go to all my friends, who have helped me through hard times, for their help and encouragement.

I would like to express my gratitude to my parents for their unfailing love, care and support. No words can describe my love and gratitude for my parents. I would also like to thank my aunt Myong, sisters, brother, and other family members for their continuous love and support over the years.

Finally, my love and thanks go to my husband Gunseog and my daughter Yujeong for their understanding, patience, and support, without which I could not have

finished this work.

This research was partially supported by Professor Yannis Ioannidis, through h National Science Foundation grant (contract IRI-8703592).

TABLE OF CONTENTS

Chapter 1

INTRODUCTION

In a relational Data Base Management System (DBMS), based on the relational data model [Codd70], queries are expressed in a high level non-procedural language [Astr76, Ston76]. The query optimizer translates the submitted query into an efficient evaluation plan whose execution will provide the query answer. Identifying the most efficient evaluation plan is an expensive process, primarily because the number of available alternatives grows at least exponentially with the number of relations participating in the query. Several heuristics have been proposed to exclude evaluation plans that are likely to be suboptimal. Examples of such heuristics are performing selections and projections as early as possible and excluding unnecessary cross products. Even with all these heuristics, however, the search space size (i.e., the number of evaluation plans to be considered) in query optimization grows exponentially with the number of relations participating in a query.

Query optimization in systems that are geared towards some of the newest database application domains, e.g., deductive, object-oriented, and parallel database systems is even harder. The space of evaluation plans that the query optimizer has to face in future database systems is larger by several orders of magnitude than the one currently faced in conventional systems. This is because queries are much more complex, both in the number of operands in them and in the diversity and complexity of operators. In particular, 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. Since the number of alternative strategies for a query is most often exponential in the number of operators in it, the tremendous increase in the size of the strategy space is obvious. The currently used heuristics to reduce the size of the strategy space, which we mentioned above, are not as effective when applied on spaces of the expected size. This greatly exacerbates the difficulty of exploring all the alternative evaluation plans.

It is worth mentioning that these complex queries are not expected to be ad-hoc, interactive queries, although we expect some of the techniques developed in our research to be useful in that area as well. Rather, these complex queries will be presented to the system to be compiled for subsequent execution; the compiled code will then be executed as many times as queries of similar structure are presented to the system. Even if in some cases the optimization time is disproportionally large compared to the gains in query execution time, costly optimization is still beneficial. It

will be offset by the accumulated gains of the multiple executions of the optimized strategy.

The above problems are certainly posed by complex queries in relational systems. Current technology of relational query optimizers is inadequate to even support many of today's applications and is not easily extensible. DB2 imposes a limit of 15 *base* relations per query; any submitted query with more relations is rejected by the system. The primary reason for that is that the system has difficulties in optimizing it. Users can easily submit queries that hit against that limit, especially when views are involved that are translated into multiple base relations. Similar limits are imposed by other relational systems as well. In addition, we are experiencing a significant increase in the complexity of schemas designed to capture the data of many organizations, which is accompanied by even more complex applications and queries with many more joins than in the past.

Complex queries in several new types of database systems present similar challenging problems in query optimization. These new database systems include object-oriented database systems [Bane87, Care88, Dada86, Fish87, Lecl88], deductive database systems [Morr86, Tsur86], and extensible database systems of both kinds, i.e., those that are based on a specific data model [Schw86, Ston86] and those that are tool-kits that facilitate the building of database systems based on arbitrary models whose characteristics are unknown *a priori* [Bato88, Care86]. Internally, query processing in many of these advanced database systems is very similar to the processing of large join

queries in relational systems. For example, in deductive database systems, given a query on a derived relation, the relevant rules are composed together to eventually generate a query on base relations[†], which is then sent for optimization and execution. This style of operation can easily generate queries with a large number of relations. As another example, many queries in object-oriented database systems can also be looked at as join queries, only that they are pointer-based instead of value-based [Shek90]. This is especially true for such systems that are implemented on top of a relational object store, e.g., Iris [Fish87].

From the above, it should be evident that optimizing queries with many relations is an important emerging problem for both current and future database systems. There is a clear need to develop new specific optimization techniques, but also to obtain a clearer understanding of the fundamental problems underlying such techniques, so that they can be applied in several environments. These have been the motivation for the work whose results are described in this thesis.

Randomized algorithms have been successfully applied in various hard combinatorial optimization problems. Two such algorithms, Simulated Annealing [Kirk83] and Iterative Improvement [Naha86], have already been proposed for query optimization of large queries. Ioannidis and Wong applied Simulated Annealing to the optimization of some form of recursive queries [Ioan87]. Swami and Gupta applied

[†] This is only possible with nonrecursive rules. In the presence of recursion, the optimization problem is much more difficult.

both Simulated Annealing and Iterative Improvement to the optimization of select-project-join queries [Swam88]. They studied the performance of these two algorithms for the optimization of large join queries exploring left-deep trees only. In this thesis, we describe an adaptation of these randomized algorithms for query optimization of select-project-join queries so that the space of all alternatives which includes deep as well as bushy trees is explored. We also study the shape of the cost function of both spaces of alternatives and compare all three algorithms on optimizing queries in them.

The performance and appropriateness of any of these algorithms depends heavily on the shape of the cost function of the space of alternatives that the randomized algorithms search. We identify several factors that affect the shape of the cost function of arbitrary spaces. These include the structure of the space represented as a graph (node degree and node distance), the cost distribution of all the alternatives in the space, and specialized properties of the cost functions. Then we study the two primary spaces with which relational query optimizers deal, that of left-deep trees and that of deep and bushy trees, and present results on their properties that affect the above mentioned factors. Hence, we are able to obtain strong indications for the shape of the cost function for these spaces, which are also verified by a series of experiments. The main conclusion is that in both spaces of interest, the shape of the cost function resembles a 'well', although of a distinctly different nature. Inspired by this analysis, we propose the Two Phase Optimization algorithm, which is a combination of Iterative Improvement and Simulated Annealing and takes advantage of the 'well' shape of the cost function.

Next, we present the results of our performance study of all three randomized optimization algorithms for both spaces, and explain their behavior based on the study of the shape of the cost function of those spaces. Finally, we present the results of a performance evaluation of the randomized algorithms when optimizing small join queries and compare them to the traditional dynamic programming approach of System R [Seli79].

The rest of this thesis is organized as follows. In Chapter 2, we describe the two spaces of interest. Chapter 3 classifies the different shapes of cost functions that affect the behavior of randomized algorithms and discusses the factors that determine the shape of the cost function of arbitrary spaces. In Chapters 4 and 5, we present a combination of analytical and experimental results that shed some light on the properties of each of the two primary spaces of interest of query optimization. We also use these results to draw conclusions about the shape of the cost functions of the two spaces. In Chapter 6, we describe the details of the above mentioned randomized algorithms and present the results of a performance evaluation of them, explaining them based on the analysis of the cost function shape of the appropriate search spaces. We also compare the spaces themselves and suggest one of them as the most appropriate for query optimization. Chapter 7 presents the results of the performance evaluation of randomized algorithms for optimizing small join queries and compares them with the corresponding results of the System R algorithm. In chapter 8, we describe previous work that is related to this thesis. Our conclusions and future research directions are contained in Chapter 9.

Chapter 2

STRATEGY SPACES

Each solution to a combinatorial optimization problem can be thought of as a state in the state space that includes all possible solutions for the problem. 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 that has the globally minimum cost.

Randomized algorithms usually start at a random state and travel 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 the state S. The number of neighbors for a state is called the *degree* of the state. A move is called *uphill* (*downhill*) if it is from a state of low (high) cost to a state of high (low) cost. 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. A

state of low (high) cost is a *low* (*high*) *state.*

When general 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. These are the search space to be explored, the transformation rules to be used for generating the neighbors of a given state, and the cost function. In this chapter, we describe the specifics of these parameters in our study of query optimization and the testbed used for our various experiments.

2.1. Search Space

Each state in query optimization corresponds to a *strategy (evaluation plan)* of the query to be optimized. Hence, in the sequel, we use the terms strategy and state indistinguishably. An evaluation plan of a query can be represented as an *operator tree* [Smit75]. Instead of considering the space of all possible evaluation plans, using some commonly known heuristics, we can decrease the size of the search space to increase the efficiency of the optimization. We must ensure, however, that the search space is general enough to include all the interesting evaluation plans. For query optimization, we use the heuristics of performing selections and projections as early as possible and excluding unnecessary cross products. By applying these heuristics, we reduce the goal of the query optimizer to that of finding the best join order, together with the best join method for each join. Thus, each strategy can be represented as a *join processing tree.*

Usually, for every join in a strategy, there is a choice among several join methods with which it can be executed. In our work, this choice is explicit, i.e., different choices of join methods on the same join processing tree represent different strategies. Also, for every relation access, there is usually a choice among multiple access paths that lead to the relation contents. In our work this choice is implicit. That is, each strategy represents the most efficient choice of access paths for the corresponding join processing tree and join methods.

A join processing tree is an operator tree, in which leaves represent base relations, internal nodes represent 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 *deep (linear)*. Otherwise, it is called *bushy*. Most join methods distinguish the two operands, one being the *outer* relation and the other being the *inner* relation. A *left-deep tree (outer linear join processing tree)* is a deep join processing tree whose inner relations of all joins are base relations. In this study, we deal with two strategy spaces: one that includes both deep and bushy trees, which is denoted by *A*, and one that includes only left-deep trees, which is denoted by *L*.

2.2. Transformation Rules

The neighbors of a strategy, which is a join processing tree, are determined by a set of transformation rules. Each rule is applied to some internal nodes of the join processing tree, replaces them by some new nodes, and usually leaves the rest of the nodes of the strategy unchanged. There are several sets of transformation rules any

one of which could be adopted for query optimization, and the options are different for spaces *A* and *L*. The ones used in this study are described below. With a, b, and c being join processing formulas, the set of transformation rules that we used for the *A* space 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. It is the only rule that does not affect the structure of a strategy as a tree. Rules (2) and (3) make use of the algebraic properties of join commutativity and join associativity [Ullm82]. We adopted them based on past experience with similar problems [Ioan87] and because they offered a mathematical basis for studying properties of the *A* space. One can easily show that these rules alone are enough to connect the entire strategy space of interest. Nevertheless, we added two more rules to the set, for *join exchange*. Each of these two transformation rules is equivalent to applying rules (2), (3), and (2) in that order. The advantage of these transformations is that they avoid the use of join commutativity. Applying join commutativity does not change the strategy cost for some join methods, e.g., merge-scan, which tends to create

plateaux in the strategy space. Usually, the algorithms presented in this study do not 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 strategy space degrades the performance of randomized algorithms. Adding the join exchange rules helps to reduce the number of plateaux by adding direct paths that bypass them. Moreover, it is the addition of these join exchange rules that helps the structure of the strategy space to become more regular (Chapter 4).

Strategies in L can be represented as a sequence of relations, which signifies the order in which the relations are joined in the strategy. Let a, b, c, and d be arbitrary relation sequences, and R, S, and T be single relations. The set of transformation rules in L is given below:

(1) *Join method choice:* $a \bowtie_{method_i} R \rightarrow a \bowtie_{method_j} R$

(2) *Swap:* $a \bowtie R \bowtie b \bowtie S \bowtie c \rightarrow a \bowtie S \bowtie b \bowtie R \bowtie c$

(3) *3-cycle:* $a \bowtie R \bowtie b \bowtie S \bowtie c \bowtie T \bowtie d \rightarrow a \bowtie S \bowtie b \bowtie T \bowtie c \bowtie R \bowtie d$

The rules that change the structure of a strategy as a tree, i.e., rules (2) and (3), were adopted from the previous study of Swami and Gupta [Swam88]. The primary reason for our choice has been our desire to study the characteristics of L spaces as used in their study. Independent of that, however, they also form a reasonable set that connects the entire space of interest. Again, this latter property is achievable by rule

(2) alone, but rule (3) introduces more neighbors to each strategy with a mixture of desirable and undesirable consequences to be discussed in future chapters.

In both spaces, rule (1) is applicable only when multiple join methods are available in the system. Also note that for any strategy of A or L, not all applicable transformation rules lead to valid strategies in the space, since they may create cross products. To make a move from a strategy, randomized algorithms attempt to use any available transformation rule, but if a strategy with a cross product is generated, the whole effort is abandoned and repeated. The degree of a strategy is the number of its valid neighbors alone.

2.3. Cost Function

In our study, we used three models for the cost functions of database operations to observe their effect on our findings. Model CM1 accounts only for the I/O required by each strategy. It makes the following assumptions: (a) nested-loops and merge-scan are the available join methods, (b) there is no pipelining for nested-loops, i.e., temporary relations are created for the intermediate results, and (c) there is minimal buffering for all operations. Model CM2 also accounts only for the I/O required by each strategy, but removes most other restrictions: (a) nested-loops, merge-scan, and hash-join are all available, (b) pipelining is used for nested-loops, and (c) large buffers can be used to avoid excessive I/O. Model CM3 assumes that the entire database is memory resident and therefore accounts only for the cpu time required by each strategy and assumes that hash-join is the only available method. In all models, the

assumptions of on-the-fly execution of projections and no duplicate elimination on projections are made. We also make the usual assumptions about the uniformity of the distributions of values and the independence of values in the join columns [Seli79, Whan85] to estimate the join selectivities. The specifics of the cost formulas for each cost model are summarized in the Appendix.

2.4. Experimental Testbed

The specific form of the A or L space depends strongly on the query with which it is associated. In this subsection, we describe the types of queries that were used in the analytical and/or experimental parts of this study so that spaces are fully specified. We also describe the details of the physical structure of the assumed database so that the strategy costs are fully specified.

Given a query, its *join graph* is an undirected graph that has a node for every relation in the query, and an edge between nodes R_i and R_j for every join predicate of type $R_i.a\ \theta\ R_j.b$, where $\theta \in \{=, \leq, \geq, <, >\}$. A query is called a *tree query (cyclic query)* if its corresponding join graph is acyclic (cyclic). A *star query* is a special type of tree query such that its corresponding join graph has one node that is connected with an edge to all other nodes. A *string query* is another special type of query such that its corresponding join graph has two nodes with degree 1 and all remaining nodes with degree 2.

In this study we concentrated on tree queries containing only equality joins. In many ways, star and string queries represent the two different ends of the spectrum, so

specific attention was given to those occasionally.

In the experimental part of this work, tree and star queries were generated randomly. The particular focus on the latter was due to known difficulties in their optimization [Ono90]. The query size ranged from 5 to 100 joins. Queries were tested in conjunction with four different types of relation catalogs, i.e., different relation cardinalities and join selectivities. Due to the uniformity assumptions mentioned in the previous section, we used the number of unique values in the join columns to control the selectivity of a join. In Table 2.1, we summarize the relation cardinalities and the number of unique values for the join columns for the four relation catalogs. The notation used for the top three catalogs is better understood with an example. In the 'relcat3' relation catalog, the relation cardinalities are randomly chosen between 1000 and 100000 tuples and the number of unique values in the join columns is randomly

Catalog	Relation cardinality (tuples)	Number of unique values in join column (fraction of the relation cardinality)
relcat1	1000	[0.9, 1.0]
relcat2	[1000, 100000]	[0.9, 1.0]
relcat3	[1000, 100000]	[0.1, 1.0]
relcat4	[10,100) - 20% [100,1000) - 64% [1000,10000) - 16% * (0.001,0.01,0.1,0.2,0.34, 0.34,0.34,0.34,0.34,0.5, 0.5,0.5,0.67,0.8,1.0)	(0,0.2] - 75% (0.2,1) - 5% 1.0 - 20%

Table 2.1: Relation catalogs

chosen between 10% and 100% of the number of tuples of the corresponding relation. Catalog 'relcat4' is a very close approximation to the one used by Swami and Gupta in their work [Swam88]. The relation cardinalities are chosen between 10 and 10000 tuples as follows: first, one of the three ranges is chosen according to the percentage given in the table and a number in that range is chosen randomly; second, that number is multiplied by a randomly chosen fraction among those in the parenthesis to approximate the effects of the selections. The number of unique values in the join columns is chosen similarly to the first step above. The above specific relation 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 strategy space. This degree of variance increases as we move from 'relcat1' to 'relcat4'.

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, for the CM2 cost model the buffer size was assumed to be equal to 100 pages.

Chapter 3

SHAPES OF COST FUNCTIONS

In chapter 6, we describe three algorithms that we used for query optimization, Iterative Improvement (II), Simulated Annealing (SA), and Two Phase Optimization (2PO). To better motivate the discussion in this chapter and the following two on characteristics of strategy spaces, we briefly describe these algorithms and their behavior on optimizing large join queries.

II tries to find a low cost state by visiting a large number of random local minima. This is done by repeatedly generating random states and then following random downhill paths until a local minimum is found. By the nature of the algorithm, its performance is largely affected by the cost distribution of local minima. SA performs one random walk from a random state allowing moves towards both higher and lower cost states. The mobility towards higher cost states is higher in the beginning and decreases with time. The performance of the algorithm depends on the ability to overcome high cost boundaries between low cost states and is affected by the precise

cost of such boundaries. 2PO combines the other two algorithms, executing II in its first phase and SA in its second phase. Hence, its behavior is affected by both the cost distribution of local minima and the cost of boundaries between low cost states. In general, these two aspects of the costs of states determine the shape of the cost function over the search space.

Preliminary experiments with query optimization verified the importance of the cost function shape for the performance of the algorithms. Moreover, several differences were observed among state spaces with respect to their shape which was affected by many characteristics of the specific space, query, and relation catalog.

Because of the above, in this chapter, we classify the different shapes of cost functions that affect the behavior of randomized algorithms and discuss the factors that determine the shape of the cost function of arbitrary spaces.

3.1. Classification of Shapes of Cost Functions

Before we can proceed with a classification of shapes of cost functions, we need the following definition. Consider two local minima S_1 and S_2 in some space, and let P be the set of paths that connect them. For each path $p \in P$, let N_p be the set of strategies in the path excluding S_1 and S_2. The *connection cost* of S_1 and S_2 is defined to be

$$\min_{p \in P} \max_{s \in N_p} \{c(s) \mid s \in N_p\} . \tag{3.1}$$

As discussed above, there are two parameters of the shape of some strategy space that determine its basic form as it pertains to the behavior of the randomized optimization algorithms:

(A) the cost distribution of local minima, and

(B) the connection cost of low local minima.

The question of interest for (A) is what percentage of the local minima is close to the cost of the global minimum. The question of interest for (B) is whether or not, for the low local minima, their connection costs are much higher than their costs.

Based on the cost distribution of local minima, shapes can be placed in three categories that are defined qualitatively as follows.

A1 Most local minima are low.

A2 A nontrivial percentage of local minima are low, but most of them are high.

A3 Local minima are scattered at all costs.

Based on the connection cost of low local minima, shapes can be placed in three categories as well.

B1 The connection cost is very similar to the cost of the local minima, i.e., the area of low cost strategies is smooth.

B2 The connection cost is somewhat distant from the cost of most low local minima, but still relatively low compared to the cost range in the whole space, i.e., the area

of low cost strategies is bumpy.

B3 The connection cost is much higher than the cost of the local minima, i.e., there are multiple areas of low cost strategies which are separated by high 'hills'.

The above classification leads to the following *qualitative* definition of a '*well*'. Any cost function whose shape is of type Ai and Bj, with $i,j \in \{1,2\}$, forms a 'well'. The closer the shape is to A1 and/or B1, the more of a 'well' it becomes. In later sections, it is shown that the behavior of the algorithms depends not only on whether the cost function forms a 'well' or not, but also on the specific class of a 'well' to which it belongs. The spaces with which we deal are multidimensional, so it is difficult to visualize what it means for a 'well' to be formed. For a one-dimensional

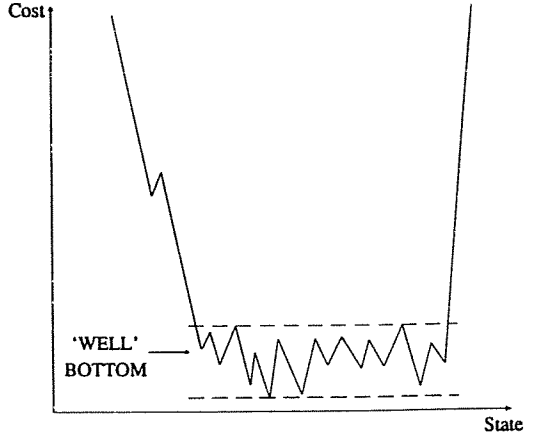


Figure 3.1: Shape of cost function that forms a 'well'

cost function, however, a 'well' of type A1-B2 is shown in Figure 3.1.

3.2. Cost Distribution of Local Minima

This section identifies three aspects of a strategy space that affect the cost distribution of its local minima: the degree of the strategies in the space, the range of the cost difference between neighbors, and the cost distribution of all strategies. The results presented below hold for spaces that are generated randomly. Clearly, such spaces are not accurate models of A or L spaces. Nevertheless, although these results cannot serve as proofs of what is happening in the spaces of interest, they are still valuable in providing some intuition that can help in explaining the experimental observations.

3.2.1. Effect of the degree of strategies

In what follows, in order to compare cost distributions, we need the following definitions. For a random variable X, its distribution function is denoted by F_X and is defined as $F_X(x) = P[X \leq x]$. Consider two random variables X and Y on the same range $[m, M]$ of possible values. Then, the distribution of Y is *shifted to the left relative to* the distribution of X if $F_Y(x)/F_X(x) > 1$ for all $x < M$. The ratio $F_Y(x)/F_X(x)$ is the *relative shift* of the distribution of Y over the distribution of X at point x. If the random variables do not have the same range, then the relative shift of their distributions can be defined after scaling them so that their ranges become equal.

Consider an arbitrarily large space such that the strategy costs are assigned based on some probability distribution. Each strategy has d neighbors, which are chosen randomly among all other strategies. Let X be the random variable for strategy costs and Y be the random variable for local minimum costs. The following series of results derive a relationship between the distribution functions of X and Y.

Lemma 3.1: Assume that X is continuous with the density function $f_X(x)$ and let $p(x)$ be the conditional probability $P[X \text{ is a local minimum cost} \mid X=x]$. Then,

$$P[X \text{ is a local minimum cost}] = \int_0^\infty p(x) f_X(x)\, dx . \tag{3.2}$$

Proof:

Let

$$I(X) = \begin{cases} 1 & \text{if } X \text{ is a local minimum cost} \\ 0 & \text{otherwise} \end{cases}$$

Based on the above function, the following can be derived. (As is traditional, we use the notation $E[g(x)]$ to denote the expected value of a function g(x).)

$$\begin{aligned} P[X \text{ is a local minimum cost}] &= E[I(X)] \\ &= E[E[I(X) \mid X=x]] \\ &= E[P[X \text{ is a local minimum cost} \mid X=x]] \\ &= E[p(x)] \\ &= \int_0^\infty p(x) f_X(x)\, dx \end{aligned}$$

□

Lemma 3.2: Under the same assumptions as in Lemma 3.1, the following holds:

$$P[X \le y \text{ and } X \text{ is a local minimum cost}] = \int_0^y p(x) f_X(x)\, dx . \tag{3.3}$$

Proof: Similar to Lemma 3.1.

□

Theorem 3.1: Let F_X be the distribution function of X and F_Y be the distribution function of Y. Then, $F_Y(y) = 1-(1-F_X(y))^{d+1}$.

Proof: The following holds for $F_Y(y)$:

$$\begin{aligned} F_Y(y) &= P[Y \le y] \\ &= P[X \le y \mid X \text{ is a local minimum cost}] \\ &= \frac{P[X \le y \text{ and } X \text{ is a local minimum cost}]}{P[X \text{ is a local minimum cost}]} \end{aligned} \tag{3.4}$$

Using continuous approximations for the distributions of X and Y, we substitute (3.2) and (3.3) to equation (3.4), and obtain

$$F_Y(y) \approx \frac{\int_0^y f_X(x)\, p(x)\, dx}{\int_0^\infty f_X(x)\, p(x)\, dx} . \tag{3.5}$$

A formula for $p(x)$ can be obtained as follows. Ignoring plateaux, in order for a state with cost $X=x$ to be a local minimum, all the d neighbors of it should have cost higher than x. Hence, if we denote the cost of the i-th neighbor by X_i, then $p(x)$ is approximately equal to

$$\begin{aligned} p(x) &= P[X \text{ is a local minimum cost} \mid X=x] \\ &\approx P[X_1>x, X_2>x, \ldots, X_d>x] \end{aligned}$$

$$= \prod_{i=1}^{d} P[X_i > x]$$

$$= (1-F_X(x))^d .$$

The above holds because the X_i's are all independently chosen and identically distributed (distribution function F_X). In addition, the following holds:

$$\int_0^\infty f_X(x)\,(1-F_X(x))^d\,dx = -\frac{1}{d+1}\,(1-F_X(x))^{d+1}\Big|_0^\infty = \frac{1}{(d+1)} .$$

Therefore,

$$F_Y(y) = \frac{\int_0^y f_X(x)\,(1-F_X(x))^d\,dx}{\int_0^\infty f_X(x)\,(1-F_X(x))^d\,dx}$$

$$= (d+1)\int_0^y f_X(x)\,(1-F_X(x))^d\,dx$$

$$= (d+1)\left[\frac{1}{d+1} - \frac{1}{d+1}\,(1-F_X(y))^{d+1}\right]$$

$$= 1-(1-F_X(y))^{d+1} .$$

□

We note that the distribution function of Y is approximated by that of the smallest random variable among $(d+1)$ independent and identically distributed random variables with the distribution function F_X. This result is expected from the definition

of the local minimum state.

The above theorem holds for arbitrary cost distributions in the strategy space. Due to the exponential form of the formula in the theorem, we see that with high enough degree d, the local minimum cost distribution is very much shifted to the left relative to the cost distribution of all strategies, i.e., most local minima can be found among the lowest cost strategies. Also, the relative shift increases as the number of neighbors increases. An example is shown in Figure 3.2, where $F_X(x)$ is drawn as a

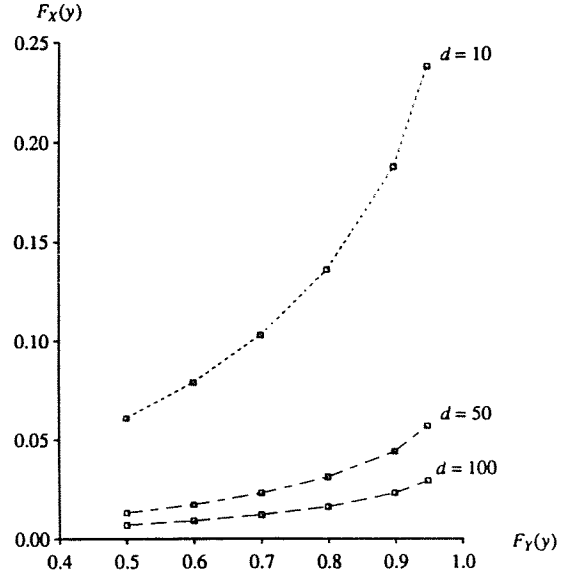


Figure 3.2: Relationship of the strategy cost and local minimum cost distributions

function of $F_Y(x)$ for various values of d. Intuitively in such a diagram, a point (a,b) indicates that the lowest (cost) a % of the local minima (x-axis) are found among the lowest b % of all strategies (y-axis). The results of Figure 3.2 are rather impressive. Even with degree d=10, 50% of the local minima are among the lowest 7% of all strategies, and this drops to the lowest 2% of all strategies for d=50, a small fraction indeed.

3.2.2. Effect of the cost difference between neighbors

One important distinction between the A and L spaces is the range of cost differences between neighbors. In A, it tends to be relatively small, since most neighbors represent very similar operations, being different in only one or two internal nodes (joins). In L, on the other hand, there are neighbors whose cost difference is quite large, having many of their internal nodes representing different operations.

The ranges of cost differences between neighbors seem to affect the local minima cost distribution. This is demonstrated as follows. Consider the random graph model that we have used above, with the additional constraint that the d neighbors of a strategy with cost c are chosen randomly among all strategies whose cost is between $c-\delta$ and $c+\delta$, for some threshold δ. In this case, $p(x)$, the probability that X of value x is a local minimum cost is equal to

$$p(x) = \left[\frac{F_X(x+\delta) - F_X(x)}{F_X(x+\delta) - F_X(x-\delta)}\right]^d .$$

Hence, (3.5) yields

$$F_Y(y) = \frac{\int_0^y f_X(x)\left[\frac{F_X(x+\delta) - F_X(x)}{F_X(x+\delta) - F_X(x-\delta)}\right]^d dx}{\int_0^\infty f_X(x)\left[\frac{F_X(x+\delta) - F_X(x)}{F_X(x+\delta) - F_X(x-\delta)}\right]^d dx} .$$

Unfortunately, neither does this equation provide a simple relationship between $F_X(x)$ and $F_Y(x)$ as in the case with no limit on the difference between neighbor costs, nor can it be analytically manipulated to derive one. Thus, to obtain an understanding of the effect of δ, we have experimented with several values of δ and with several probability distribution functions F_X. Specifically for the latter, we have experimented with several forms of the gamma (Γ) distribution. The choice of the Γ distribution was motivated by experimental results that showed that it resembles the distribution followed by the cost of strategy spaces in query optimization (Section 4.3). All experiments, however, showed no effect of the specific distribution on the general results, with occasional differences appearing only in extreme cases. So, we believe that the validity of our conclusions extends well beyond the specific distribution choices.

We show the relationship between $F_Y(x)$ and $F_X(x)$ for several values of δ. Some representative examples of the experimental results are shown in Figures 3.3 and 3.4 for d=10 and d=50 respectively. Specifically, each figure contains the results for three forms of the Γ distribution identified by the values of the parameters α and β [Roth86] . Roughly, when α=1.0, Γ is similar to negative exponential, when α=3.0, Γ is similar to normal, and when α=2.0, Γ is similar to a distribution in between. In each case, the values of δ are expressed as a fraction of the standard deviation σ of the Γ distribution

used.

There are two major conclusions that can be drawn from the results.

(a) In general, the shift to the left of the local minimum cost distribution relative to the cost distribution of all strategies monotonically increases as δ decreases, i.e., local minima tend to be of lower cost. This is an expected result and is observed in most places in Figures 3.3 and 3.4. It is mainly because the probability of a state being a local minimum, p(x), quickly becomes extremely small as its cost increases. This trend is intensified with smaller δ and causes the cost distribution of local minima to shift even further to the left. An example of the above fact is shown in Figure 3.5, where we compare p(x) as a function of absolute state cost for the case of $\delta=\infty$ with that for the case of $\delta=\sigma/2$.

(b) With low degree (d=10) the effect of δ on the relative shift is not always monotonic but presents a maximum. Beyond a certain point, as δ decreases, the relative shift decreases as well. For the most part, this phenomenon occurs only at high values of F_Y, while the relative shift remains monotonic with δ for lower values of F_Y. This is one aspect whose specifics depend on the characteristics of the distribution. The more the distribution is shifted to the left, the less this nonmonotonicity occurs. This is clearly seen in Figure 3.3, where for the range of the experiment, this phenomenon was most evident for the Γ distribution with α=3.0 (normal), while it is nonexistent for the Γ distribution with α=1.0 (negative exponential).

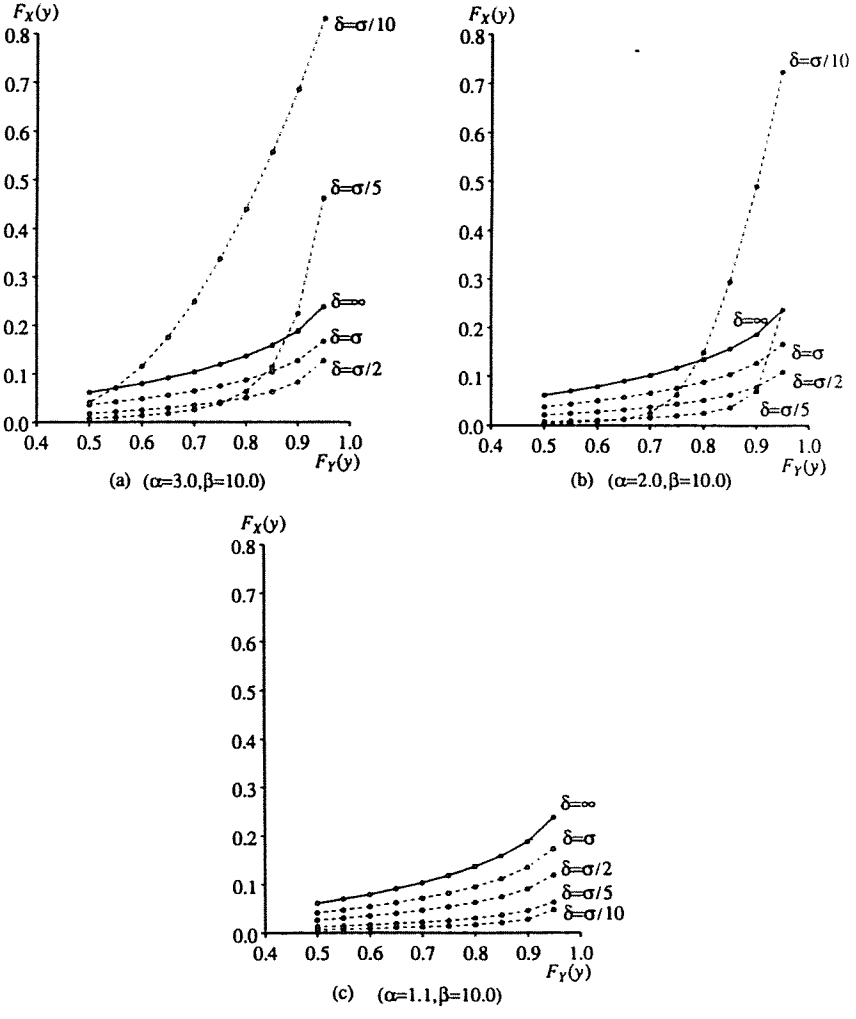


Figure 3.3: Relationship of the strategy cost and local minimum cost distributions: d=10

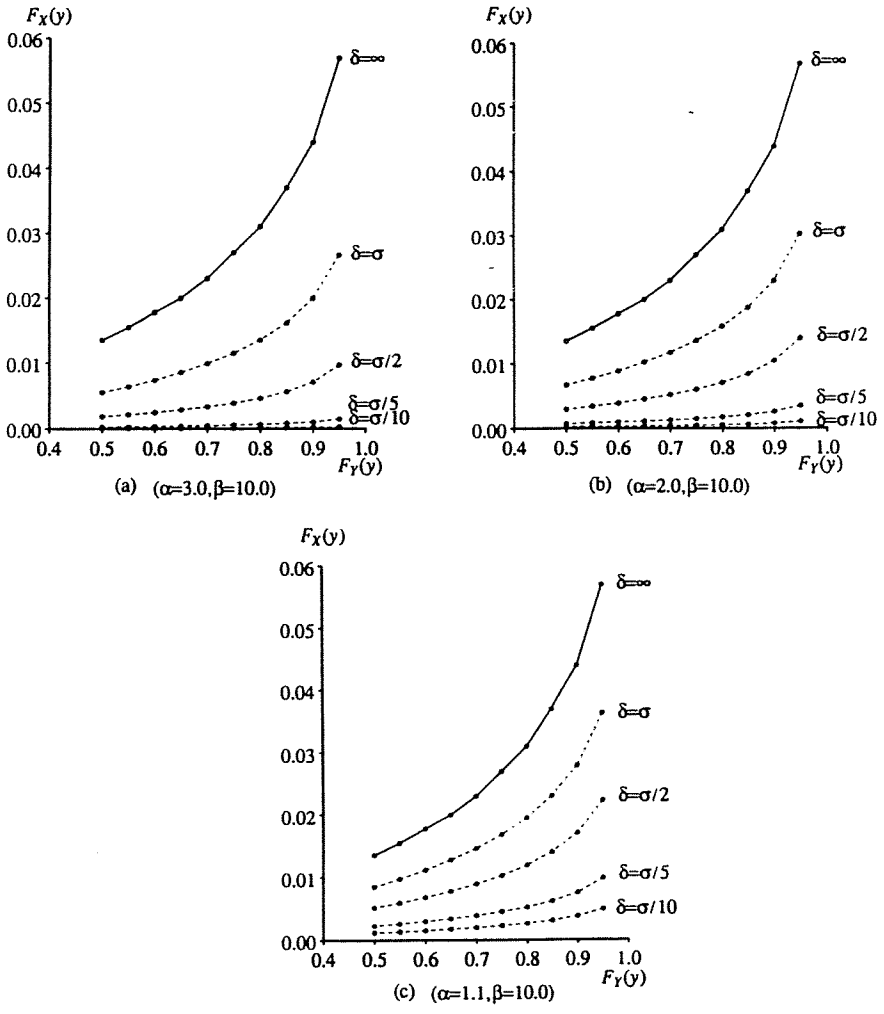


Figure 3.4: Relationship of the strategy cost and local minimum cost distributions: d=50

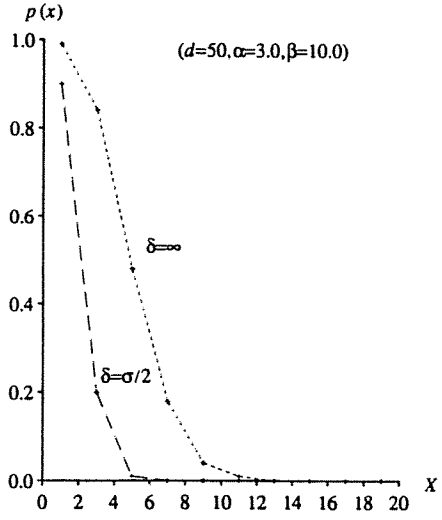


Figure 3.5: Probability of a state being a local minimum

We should note that the nonmonotonicity of the relative shift with δ is not of concern for the strategy spaces of our interest, since the typical values of δ are rather high and the degree d is not very small either, i.e., in the range of operation, the relative shift is monotonic. Hence, the above results indicate that, in our case, more abrupt changes in cost between neighbors result in more local minima among higher cost strategies.

3.2.3. Effect of the cost distribution of all strategies

As we have mentioned already, the results of Sections 3.2.1 and 3.2.2 on the relative shift of local minimum cost distribution to the left were mostly insensitive to the specific cost distribution of the strategies in the space. Hence, if that distribution changes, the local minimum cost distribution follows along. The important implication of this fact is that if the shift of f_X to the left increases, so does the corresponding shift of f_Y. This captures the rather intuitive fact that as more strategies move towards lower costs, more local minima do the same as well.

3.3. Connection Cost between Low Local Minima

This section identifies three aspects of a strategy space that affect the connection cost of low local minima: the distance between the local minima and the number of paths connecting them, the cost distribution of all strategies, and some special properties of the functions used to compute the cost of each strategy. In fact, the presented results are more general and hold for states of arbitrary costs, but our primary interest is in low local minima.

3.3.1. Effect of the distance and the number of paths between strategies

The effect of the first factor is rather intuitive although difficult to express formally. As the distance of two local minima increases and/or the number of paths connecting them decreases, their connection cost most likely increases as well. The

intuition behind the above statement is based on equation (3.1). Longer paths contain more strategies (N_p is larger) and therefore tend to be able to reach higher costs (there are more choices for the maximum of the costs of strategies in N_p). By the same token, fewer paths (smaller P) implies that there are fewer choices for the minimum among those maxima. Although there is no concrete relationship between distance/number of paths and connection cost, there is a rather intuitive trend. As two extreme examples, consider a fully connected graph with N nodes and a string of N nodes (a cycle with one of its edges removed). In the first graph, every pair of strategies is connected with a path of length 1, and the total number of paths is approximately $e\ (N-2)!$, where e is the Napierian number. The connection cost for all pairs of strategies is zero, since they are all neighbors. In the second graph, for every pair of strategies, there is a unique path that connects them. Moreover, there exist strategies whose connecting path contains all nodes in the graph and therefore their connection cost is the highest possible. Most graphs fall between these two extremes and behave analogously.

Note that the degree of a graph affects both the distance between two strategies and the number of paths between them. A higher degree decreases the former and increases the latter, thus decreasing the connection cost of strategies.

3.3.2. Effect of the cost distribution of all strategies

The effect of the second factor is also straightforward and intuitive. With a larger percentage of strategies having low cost, the connection cost of low local minima

decreases. The intuition behind the above statement is again based on equation (3.1). The further to the left the strategy cost distribution is shifted, the lower the costs will be in the set $\{c(s) \mid s \in N_p\}$.

3.3.3. Effect of special properties of the cost function

The effect of the third factor is much more complex and is based on previous work by Monma and Sidney [Monm79], Ibaraki and Kameda [Ibar84], and Krishnamurthy, Boral, and Zaniolo [Kris86]. Consider a strategy space whose nodes can be mapped to the distinct permutations of a set of symbols Σ. In addition, for every strategy S, its neighbors include all strategies obtained from S by interchanging two adjacent subsequences of S. For example, if S corresponds to the permutation *abcdef*, the strategy that corresponds to *deabcf* is a neighbor of S, since it can be obtained from S by swapping *abc* and *de*. Let A, B, U, and V be sequences of symbols in Σ, with U and V not being the empty sequence. The strategy cost function satisfies the *Adjacent Sequence Interchange* (ASI) property if the following holds: $c(AUVB) \leq c(AVUB)$ iff $rank(U) \leq rank(V)$, for some function *rank*. The importance of the ASI property for the more traditional approaches to query optimization has been discussed before [Ibar84, Kris86]. Its importance for randomized algorithms becomes evident with the following theorem.

Theorem 3.2: Consider a strategy space as described above, whose cost function satisfies the ASI property. Then, there is a unique local minimum area, i.e., all local minima in the space have the same cost and are mutually connected without any uphill

moves in-between. Therefore, the whole area corresponds to the global minimum.

Proof: Let S be a permutation and A, B $\in \Sigma$ such that rank(A) $\leq$ rank(B) implies A precedes B in the permutation. To prove the above theorem, it suffices to show that any permutation S' can be transformed into S by a series of *adjacent* symbol interchanges none of which increases the cost. That is, we show that S has the global minimum cost and can be reached by any state without an uphill move. Consider a permutation $S' \neq S$. Then, there exist symbols A, B $\in \Sigma$ such that S' = UBAV, for some sequences U and V, and such that A precedes B in S. By the choice of S, rank(A) $\leq$ rank(B). Therefore, the permutation obtained from S' by interchanging A and B has no greater cost than S' and has one fewer disagreement of symbol order with S. Repeated applications of the above adjacent symbol interchange are bound to eventually produce S. Since the original permutation S' was arbitrary and all transformations corresponded to non-uphill moves, the global minimality of the cost of S is also proved.

□

Whenever the premises of the above theorem are satisfied by a strategy space, its conclusions suggest an extremely efficient optimization algorithm: from an arbitrary strategy take a downhill path until a local minimum is reached, which is the answer. Although neither A nor L satisfies the premises of Theorem 3.2, they contain many subspaces that do satisfy them, so its conclusions are quite important in understanding the shape of their cost function.

3.4. Summary

The results of the previous sections can be briefly summarized as follows. As the degree of strategies increases, local minima are pushed towards the lower costs. Also, the interstrategy distance decreases and the number of paths between strategies increases, so the connection cost of local minima is pushed towards lower values as well. Hence, the following conclusion can be drawn:

C1: High strategy degree helps the formation of a 'well' in a space.

We also showed that as the cost difference between neighbors decreases, in general, local minima are pushed further towards the lower costs. Hence, the second conclusion is that

C2: Low cost difference between neighbors helps the formation of a 'well' in a space.

Another sets of results showed that as the cost distribution of all strategies is shifted to the left, both local minima and their connections are pushed towards lower costs. Hence, with respect to both characteristics of a space shape, the following conclusion can be drawn:

C3: A strategy cost distribution that is shifted left helps the formation of a 'well' in a space.

Finally, we showed that the ASI property generates a single local minimum area, which leads to the conclusion that

C4: The ASI property helps the formation of a 'well' in a space.

The above conclusions can serve as criteria for determining whether a space forms a 'well' or not. It should be clear that in extreme cases, one of the four identified factors dominates all others and determines the space shape. For example, when the strategy degree is equal to the space size minus 1, or when the neighbor cost difference is zero, or when the strategy cost distribution is a δ-function, or when the ASI property holds for the full space, the space shape is a 'well' with a unique local minimum area. Similarly, when the strategy degree is 1 or when the strategy cost distribution is extremely shifted to the right, the space shape is far from a 'well'. Away from the extremes, however, all factors have some significant contribution to the space shape. The relative importance of them is unknown for the moment. Part of our future work includes some extensive experimentations to address this issue. Fortunately, for this study, whenever we applied the above criteria, all conclusions were consistent with the experimental observations. Hence, identifying their relative importance was not crucial.

Chapter 4

SPACE OF DEEP AND BUSHY STRATEGIES

In the previous chapter, several results were presented for the properties of a strategy space that determine the shape of its cost function by affecting the (A) and (B) parameters (see Chapter 3). This chapter contains results that characterize the *A* space with respect to these properties and identifies the shape of the cost function of the *A* space using these results.

4.1. Structure of the Space

In this section, we present results on the degree of strategies and the interstrategy distance in the *A* space, which affect both parameters of interest. We should point out that these parameters are examined with respect to the structure of the strategies alone, independent of other additional characteristics with which the spaces can be enriched, i.e., different join methods. Incorporating these is straightforward.

4.1.1. Degree of strategies

Theorem 4.1: Consider the *A* space for a query with *J* joins. Then, the degree of any strategy in that space is equal to $2J-1$.

Proof: Consider a strategy corresponding to a query with *J* joins. (Recall that a strategy is a join processing tree.) For each join node, the join commutativity rule can be applied. Thus, a total of *J* neighbors are generated by this rule.

Next, we show that for each pair of a join node with its parent, either one of the join associativity rules or one of the join exchange rules can be applied without resulting in a cross product. Consider a join node and its parent. If the child node is the left child of its parent, then for some subtrees A, B, and C, the two joins can be represented by the formula (A ⋈ B) ⋈ C. In that case, there are the following two transformation rules that can be applied to this expression:

(A ⋈ B) ⋈ C → A ⋈ (B ⋈ C)

(A ⋈ B) ⋈ C → (A ⋈ C) ⋈ B

If the parent join involves one of the relations in B, then the first transformation, join associativity, can be applied without creating a cross product. On the other hand, if the parent join involves one of the relations in A, then the second transformation, join exchange, can be applied without creating a cross product. Thus, exactly one of the applicable transformation rules generates a valid neighbor. The above can similarly be shown for the case where the child node is the right child of its parent. There are $J-1$ such parent/child pairs in the join processing tree (since only the root cannot be a

child). Thus, for every strategy, a total of $J-1$ neighbors are generated by the join associativity and join exchange rules.

Combining the above results for all transformation rules, we can conclude that there are $2J-1$ neighbors for each strategy.

□

The above theorem shows that all strategies in A have the same degree, which is reasonably large. Note that if the join exchange rules were not used, not all strategies would have the same degree, which would vary between J and $2J-1$.

4.1.2. Interstrategy distance

Before we can proceed with the theorem on interstrategy distance in the A space, we need the following definition. Consider a query with J joins and let $\{a_i \mid 1 \le i \le J\}$ be the set of those joins. Let T denote the set of strategies corresponding to the query and Q denote the set of sequences of length J that correspond to permutations of the a_i's. Define the relation tq between T and Q, such that for a strategy S and a join sequence s, the relationship $tq(S,s)$ holds if for every join a_j, all its descendant joins in the join processing tree that corresponds to S appear before a_j in s. Note that tq is neither a function from T to Q nor a function from Q to T. It is not the former because a strategy is related to all sequences with different orderings of joins that do not have an ancestor-descendent relationship in the strategy. It is not the latter because, due to join commutativity, multiple strategies can be related to the same sequence. As an example, consider the strategies S_1 and S_2 shown in Figure 4.1.

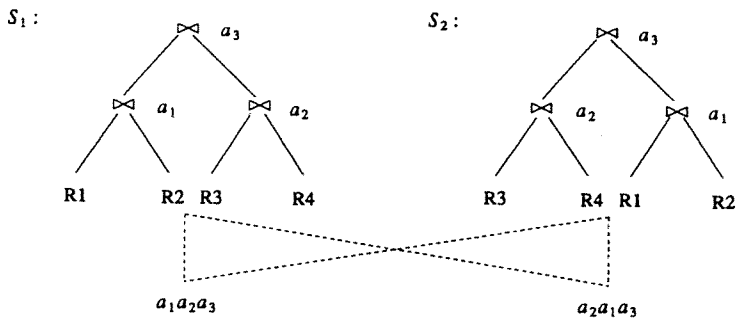


Figure 4.1: Examples of the tq relation

Then, S_1 is related to both sequences $a_1a_2a_3$ and $a_2a_1a_3$. Also, both strategies S_1 and S_2 are related to the first sequence.

Theorem 4.2: Consider the A space for a query with J joins. For every strategy pair in that space, there exists a path that connects them whose length is less than or equal to $J(J+1)/2$.

Proof: Let T, Q, and tq be defined as above. Let $S_1, S_2 \in T$ and $s_1, s_2 \in Q$, such that $tq(S_i, s_i)$, for $i=1,2$. Consider a series of join sequences $\{t_i \mid 0 \le i \le N \text{ and } t_i \in Q\}$, such that $t_0 = s_1$, $t_N = s_2$, and t_{i+1} is constructed from t_i by switching the order of two adjacent joins in t_i that appear in the opposite order in s_2. We claim that this series of join sequences corresponds via tq to a path from S_1 to S_2' in the A space of the query whose length is less than or equal to N, where S_2' is a strategy for which $tq(S_2', s_2)$. We prove the claim by induction on i.

Basis: Let i=0. Then t_0=s_1, which by definition is tq-related to S_1 and is vacuously connected to itself.

Induction Step: Assume that the claim is true for some $0 \leq i < N$. We show it for $i+1$. Let $t_i = a_1, \cdots, a_k, a_{k+1}, \cdots, a_J$ and $t_{i+1} = a_1, \cdots, a_{k+1}, a_k, \cdots, a_J$, for some $1 \leq k < J$. By the induction hypothesis, there is a valid strategy S in the A space of the query such that $tq(S, t_i)$ and S is connected to S_1 by a path of length less than or equal to i. We distinguish two cases:

(a) If a_k and a_{k+1} do not have an ancestor-descendent relationship in the processing tree that corresponds to S, then by definition, $tq(S, t_{i+1})$ as well.

(b) If a_k is a descendent of a_{k+1} in the processing tree that corresponds to S, then a_k must be a child of a_{k+1}, because otherwise their intermediate joins should appear between a_k and a_{k+1} in t_i. In that case, we have already shown (in the proof of Theorem 4.1) that exactly one associativity or join exchange rule is applicable on S, which in fact changes the order of a_k and a_{k+1}. Moreover, the properties of these transformations are such that the result strategy S' is tq-related to t_{k+1}.

In both cases, we are able to show that t_{k+1} is tq-related to some strategy that is at most one step further away from S. Hence, that strategy is connected to S_1 by a path whose length is at most $i+1$, which concludes the induction.

By the above induction, since there is a finite number of order disagreements among joins in s_1 and s_2, we conclude that there is a path from S_1 to some strategy S_2' such that $tq(S_2', s_2)$. Every edge in that path corresponds to a transformation step

in the series of sequences $\{t_i\}$. Even if all corresponding transformation steps are of type (b) above, there are at most $J(J-1)/2$ join order disagreements between s_1 and s_2. Hence, $N \leq J(J-1)/2$ and the path from S_1 and S_2' is of length at most $J(J-1)/2$.

What remains to be shown is that there exists a path from S_2' to S_2 of the appropriate length. Since both $tq(S_2, s_2)$ and $tq(S_2', s_2)$, the two strategies must be identical with respect to the operands of each join, but must be different only with respect to which operand is the left child and which one is the right child of the join. Hence, there exists a path from S_2' to S_2 in the A space of the query that corresponds to applying the join commutativity rule on the joins of S_2' so that it can be transformed to S_2. Since there are J joins, the length of that path can be at most J.

Adding the upper bounds on the lengths of the paths from S_1 to S_2' to S_2 yields that there is a path between S_1 and S_2 whose length is at most $\frac{J(J-1)}{2} + J$, which is equal to $\frac{J(J+1)}{2}$.

□

The implication of the above theorem is that the distance between strategies is rather short, quadratic in the number of joins in the worst case. In general, the distance will be much shorter than in the worst case, especially between low cost strategies, because we expect that they will tend to share much common substructure.

4.2. Effect of Ranked Join Methods

A join method is *ranked* if its cost formula is of the form $n_1 * g(n_2)$, where n_1 and n_2 are the sizes of the outer and inner relations, respectively, and $g(\cdot)$ is some appropriate function [Kris86]. Nested-loops is a representative example of a ranked join method, while merge-scan is the most well known join method that is not ranked.

In what follows we take advantage of the fact that the set of strategies in the L space for a query is a subset of the set of strategies in the A space for that query. Suppose that R is a relation in some tree query. Consider the subspace of the L space that corresponds to the query whose nodes are the strategies having R as their first (leftmost) relation. It is well known that if all available join methods are ranked, the strategy cost function for that subspace satisfies the ASI property [Kris86]. Hence, if exchanging adjacent sequences of relations in a strategy was a legal transformation to generate neighbors, by Theorem 3.2, that subspace would have a unique local minimum (or all the local minima would have the same cost and be connected by themselves). Although this is not generally the case in L, at least for star queries, exchanging any pair of adjacent relations is legal (assuming that the relation in the center of the star remains the first relation in the strategy). This fact can be used to prove several interesting properties regarding the connection cost of strategies in both A and L for star queries.

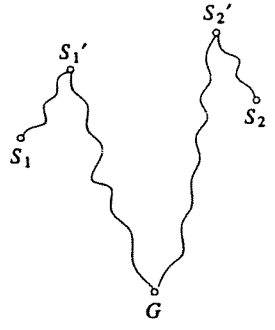
Theorem 4.3: Consider a star query with J joins and a pair of strategies S_1 and S_2 in the corresponding A space. Then the following hold:

(a) If all available join methods are ranked, there exists a path connecting S_1 and S_2 whose most expensive strategy is at most J steps apart from S_1 or S_2.

(b) If non-ranked join methods are available, there exists a path connecting S_1 and S_2 whose most expensive strategy is at most $2J$ steps apart from S_1 or S_2.

(c) If merge-scan is the only available non-ranked join method, there exists a path connecting S_1 and S_2 whose most expensive strategy is at most J steps apart from some strategy that is connected to S_1 or S_2 with an equal cost path.

Proof: For the proofs of each case of the above theorem, it suffices to show that S_1 and S_2 can be transformed to S_1' and S_2', respectively, in at most the given number of steps, where S_1' and S_2' are left-deep trees with the center of the star as their first relation and with all their join methods ranked. This is because, by Theorem 3.2, S_1' and S_2' are connected through a path of two downhill pieces, from S_1' and S_2' to the global minimum G of the subspace of all left-deep trees with T as their first relation. This is shown in the following figure.



Recall that for all strategies corresponding to a star query the center of the star has to be one of the operands of the first join. To prove the theorem, we treat the three cases (a)-(c) separately.

(a) In this case, we can obtain $S_1'(S_2')$ from $S_1(S_2)$ by applying join commutativity on each join whose right operand is not a base relation and on the first join if its left operand is not the center relation of the star join graph. This takes at most J steps.

(b) In this case, we can obtain $S_1'(S_2')$ from $S_1(S_2)$ by applying the join commutativity rule as in (a) to transform it into a left-deep tree first, and then changing the join method into a ranked one for each join whose join method is not ranked. This takes at most $2J$ steps.

(c) In this case, we can obtain $S_1'(S_2')$ from $S_1(S_2)$ as follows. First, apply join commutativity on each merge-scan join whose right operand is not a base relation. This does not increase the cost of the strategy due to the symmetry of the merge-scan cost formula. Then, change the join method of each merge-scan join into a ranked join method and apply join commutativity on the other joins if it is necessary to transform the plan into a left-deep tree. This second part takes at most J steps.

□

Considering the huge size of the strategy space, one realizes that the above linear bound is extremely small. Hence, by Theorem 4.3, the connection cost of low local minima for a star query should in general be low.

Unfortunately, the above result does not hold for arbitrary tree queries. In that case, there is no large subspace that can be guaranteed to have a unique local minimum. Nevertheless, depending on the form of the query and the specific characteristics of the relations, one can analytically identify enough smaller subspaces with few local minima, all of which are at low cost. Hence, the expectation is that the connection cost between low local minima will be relatively low in the general case as well.

4.3. Cost Distribution of All Strategies

Clearly, the cost distribution in a strategy space depends on the strategies alone and not on their connections. In this section, we present some experimental results for the typical shape of the cost distribution of strategies in the A space for different cost models and catalogs.

We have experimented with all twelve possible combinations of four types of catalogs and three different cost models described in Sections 2.3 and 2.4. For several 20-join queries, we generated 500,000 random strategies in the space of each query to obtain an approximation of its distribution. Because of the presence of strategies with cross-products, generating random strategies in spaces of random tree queries is extremely time consuming. Hence, we have confined ourselves to experimenting with star queries alone, which do not present similar difficulties. We believe, however, that our findings regarding general trends are not affected by this specific aspect of the queries. The results for all tested queries and for all cost models have been very

consistent. As we move from 'relcat1' to 'relcat4', two important parameters of the distribution change. First, the cost range increases dramatically. Second, the distribution becomes more shifted to the left. Example distributions for the four catalogs are shown in Figure 4.2. The x-axis represents the ratio of the strategy cost over the least such cost among the sampled strategies. The specific case is for a 20-join query with cost model CM1, although we observed similar trends in all other experiments. Note the strong resemblance with the Γ distributions with parameters α = 1, 2, and 3 [Roth86] for catalogs 'relcat4'/'relcat3', 'relcat2', and 'relcat1', respectively. This justifies our experimentation with these distributions in Section 3.2.2.

The observed trends were to be expected. Clearly, an increase in the variance in the characteristics of the database relations results in opening the gap between the low and high cost strategies. For example, if all relations have the same characteristics, all strategies will have the same cost. On the other hand, as the difference in characteristics grows, choosing the wrong order of joining the relations will be of much higher cost than choosing the correct order. In Figure 4.2(d), for catalog 'relcat4', only a small part of the cost range is shown, so that the details of the distribution in the low cost area can be seen. The ratio of the complete range over the lowest cost strategy was the largest in this case among all catalogs.

The above fact partly explains the shift of the distribution to the left. Consider a set of strategies that are identical up to a certain point when considered in a bottom-up

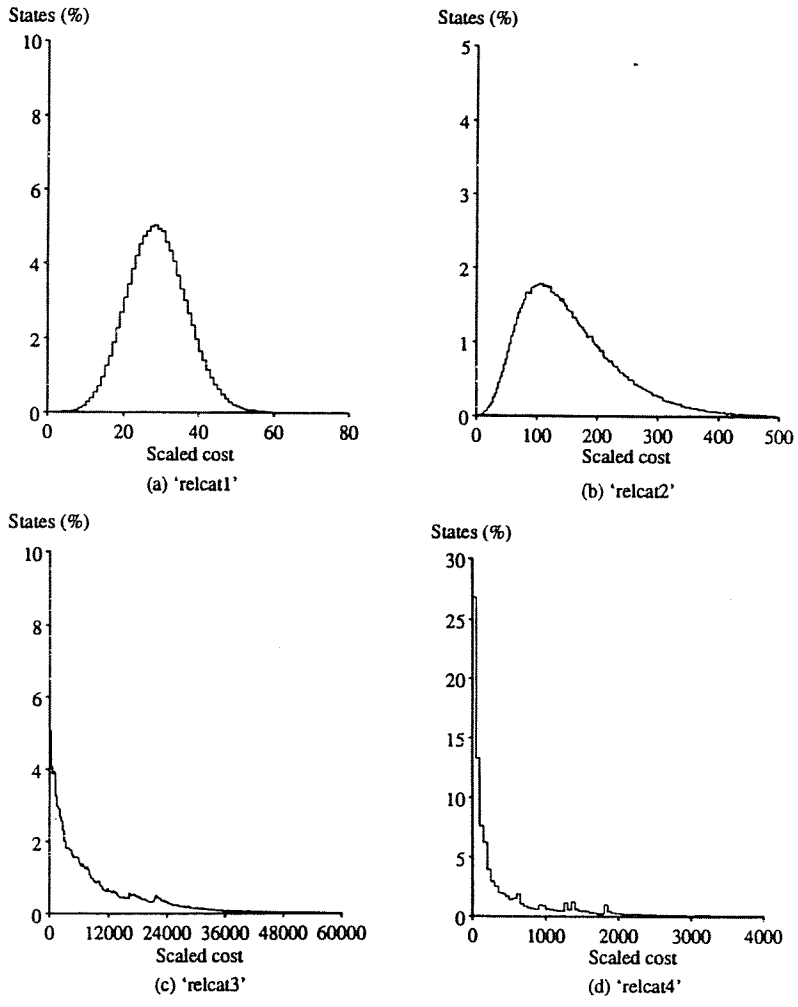


Figure 4.2: The strategy cost distribution of a star query

fashion. Their cost difference is determined by the cost difference in the remaining joins, but is affected by the size of the results of the common part. For low cost strategies, the common bottom part usually produces small relations, whereas for high cost strategies, it produces large ones. Small relations allow a relatively small range in the cost of the subsequent operations, whereas large ones allow larger differences. This has the effect that the strategy cost distribution is in general shifted to the left. As the variance in the characteristics of the relations increases, the overall cost range increases as well, and the distribution is shifted even further to the left.

4.4. Shape of the Cost Function: Expectations from Analytical Results

In this section, we identify the shape of the cost function of *A* spaces. We want to emphasize again that, except in a very few special cases, no definite results can be proved for this problem. There is enough analytical and experimental evidence, however, to provide a basis for useful conclusions. In what follows, we argue about the type of the shape of the cost function of *A* spaces based on the results reported in the previous sections. In the next section, we verify our claims by presenting the results of independent experiments.

The major conclusions drawn in Chapter 3 can be summarized as follows.

(A) The shape of the cost function changes from type A3 to A1 as the degree of strategies increases, the cost difference between neighbors decreases, and the shift to the left of the cost distribution of all strategies increases.

(B) The shape of the cost function changes from type B3 to B1 as the interstrategy distance decreases (and the degree increases), the shift to the left of the cost distribution of all strategies increases, and the ASI property holds for more subspaces of the space of interest.

Clearly, some of these factors are mutually independent and others are not. Also, as we mentioned earlier, in some cases a single one of these factors determines the shape type, and in other cases a combination of them is needed. Based on the above results, *A* spaces are characterized as follows. All strategies in the space have exactly the same degree, which is relatively high (Theorem 4.1), so the local minimum distribution tends to be shifted to the left relative to the cost distribution of all strategies. Also, the cost difference between neighbors is relatively low. Hence, from our analysis in Section 3.2, and for most strategy cost distributions, the shape of *A* spaces should be of type A1, i.e., *A* spaces have a strong tendency to have most local minima at low cost.

Again, due to the relatively high degree of all strategies and the well regulated connectivity of *A* spaces, the interstrategy distance is in general short, being $O(J^2)$ for a J join query, and the number of paths between strategies is in general large. Therefore, the connection cost of low local minima should not be very high, especially considering the small cost difference of neighbor states. The shape should be of type B2 or B1. This is further strengthened when some ranked join methods are used. Moreover, as the shift to the left of the strategy cost distribution increases, the shape

should change from B2 to B1.

The above observations lead to the overall conclusion that the shape is of type A1-B2 or A1-B1.

4.5. Shape of the Cost Function: Experimental Results

In this section, we present some experimental results that we have obtained to verify our assessment of the shape of the cost function of A spaces. First, we show results from experiments on small queries, for which an exhaustive search of the whole space was feasible. Then we present results from experiments on large queries, for which we had to rely on random sampling.

4.5.1. Exhaustive search of spaces of small queries

We have been able to obtain a complete picture on the strategy spaces of small queries by enumerating all strategies and determining the connectivity among them. Enumerating strategies in the A space is more difficult than in the L space because of the presence of bushy trees. To understand the problem, we use again the notation introduced in Section 4.1.2, which is repeated below for convenience. Consider a query with J joins and let $\{a_i \mid 1 \le i \le J\}$ be the set of those joins. Let T denote the set of strategies corresponding to the query when join methods are ignored (i.e., when concentrating on the structure of the processing trees only) and Q denote the set of sequences of length J that correspond to permutations of the a_i's. The relation tq between T and Q is defined so that for a strategy S and a join sequence s, the

relationship $tq(S,s)$ holds if for every join a_j, all its descendant joins in the join processing tree that corresponds to S appear before a_j in s.

Generating each member of T exactly once, which is our primary task, is rather difficult, whereas generating all members of Q is straightforward. The problem that bushy trees introduce is that, as mentioned above, tq is neither a function from T to Q nor a function from Q to T. Hence, exploring Q does not give us the ability to easily explore T.

To overcome this problem, we introduce two functions that are subsets of tq. In particular, we define the function $t: Q \rightarrow T$ such that, for a sequence $s \in Q$, $t(s) \in \{R \mid tq(R,s)\}$, which is called the *representative strategy of* s. Similarly, we define the function $q: T \rightarrow Q$ such that, for a strategy $S \in T$, $q(S) \in \{p \mid tq(S,p)\}$, which is called the *representative sequence of* S. These two functions give the ability to move between Q and T in deterministic ways. There are many alternatives for functions q and t that have the above characteristics and satisfy our needs. For our experiments, we made the following choices, taking advantage of the natural total order that exists on relation names. For a sequence $s \in Q$, $t(s)$ is the strategy that is tq-related to s whose every join chooses which one of its operands is its left (outer) child and which one is its right (inner) child as follows. Each one of its operands contain one of the two relations that are directly involved in the join. Based on the order of relation names, the operand that contains the relation that comes first in the name order is the left operand of the join. For a strategy $S \in T$, $q(S)$ is the sequence that is tq-related to S

and is generated from a post-order traversal of the join processing tree that corresponds to S.

Using the above two functions, we can generate all members of Q and use only the representative ones so that the corresponding strategies are uniquely generated. In particular, on every sequence we apply the composition of q and t, and only if the result is s again do we use that sequence for generating strategies. In that case, all strategies that are tq-related to s are generated and for each one of them all possible assignments of join methods to joins are enumerated. The proof that this process uniquely generates all strategies in the A space of a query is rather straightforward and is omitted.

One final aspect of the algorithm is the internal representation of each valid strategy. To avoid storing a complete specification for each strategy, we mapped all strategies to numbers by some one-to-one function. Each strategy can be identified by its representative join sequence (there are at most $J!$ of those), its specific choice of inner/outer operands for all joins (there are exactly 2^J of those), and its specific choice of join methods for all joins (there are exactly M^J of those for M join methods). Let the functions f_1, f_2, and f_3 be from join sequences to integers in the range $[1,J!]$, from inner/outer join operand combinations to integers in the range $[1,2^J]$, and from join method combinations to integers in the range $[1,M^J]$, respectively. Then, a strategy S can be mapped to a unique number by the function

$$f(S) = (f_1(S)-1)M^J 2^J + (f_2(S)-1)M^J + f_3(S) .$$

Given all of the above, we are able to generate all strategies in the A space of a query exactly once and then identify their neighbors. Pseudocode for this algorithm is presented in Figure 4.3.

```
procedure generateA() {
  for each s ∈ Q {
    if (s = q(t(s)))
      for each S such that tq(S,s)
        for each join method combination of the joins of S {
          write f(S);
          write cost(S);
          for each neighbor S' of S
            write f(S');
        }
  }
}
```

Figure 4.3: Generation of the A space of a query.

Once the complete space has been generated and stored, all of its local minima can be identified. Because there is a significant cost involved in verifying the truth of the precise definition of a local minimum, we have used an approximation, called *p−local minimum*, which is defined as a state none of whose neighbors has a lower cost. Note that states on plateaux can be mistaken as local minima with this definition. Unless otherwise noted, any reference to a local minimum in the experiments investigating the space shape refers to a p-local minimum.

We have experimented with spaces for tree queries of five and seven joins. We have tested ten queries of each join size, each query with three different cost models and two different catalogs. Having multiple join methods tends to make the space

shape more of a 'well', especially for small queries. This is due to the fact that for any given processing tree, the optimal join method for each join can be decided mostly independently of all other joins. In other words, the subspace of all strategies with the same processing tree has a unique area of local minima that are connected among themselves. Thus, in general, it intensifies the 'well' form of the space shape. We have therefore used cost models with a single join method so that the space shape can be observed without the effect of multiple join methods. A side advantage is that by avoiding multiple join methods the space size becomes much more manageable, because otherwise the exhaustive search for the same size queries is much more time consuming. The cost models that we have used are CM3 and two variants of CM1, one with the nested-loops join method only and the other with the merge-scan join method only. We call these two variants of CM1, CM1-N and CM1-M, respectively. The catalogs that we have used are 'relcat2' and 'relcat4'.

We summarize the results of the above experiments in Figures 4.4 and 4.5, for 'relcat2' and 'relcat4' respectively. Queries 1 to 10 have five joins and queries 11 to 20 have seven joins. Queries are numbered in increasing order of average strategy cost for each case. For each query, we show the average strategy cost in the space and the average and the highest of the local minimum costs. All of these costs are scaled over the global minimum cost. Note that due to differences in the cost range, in the graphs for 'relcat2' with CM1-M and CM3 the scale of the y-axis is linear, whereas in the remaining graphs it is logarithmic.

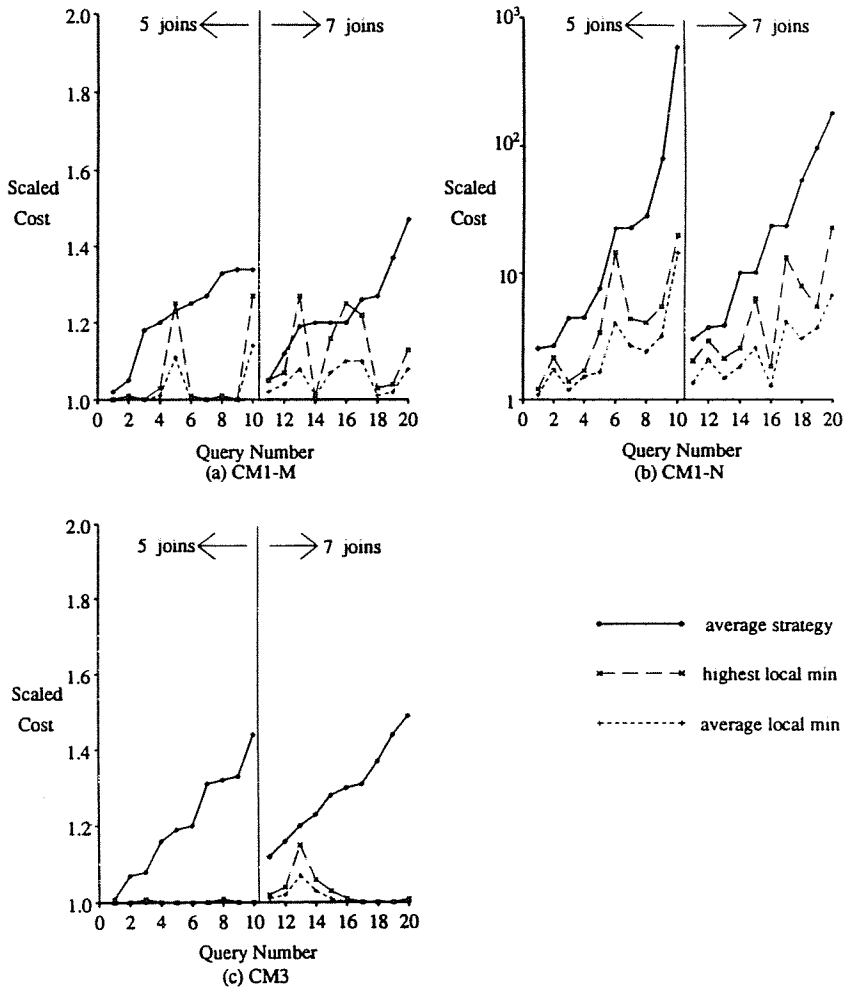


Figure 4.4: Aggregate characteristics of the cost distribution of all strategies and local minimum strategies for 'relcat2'

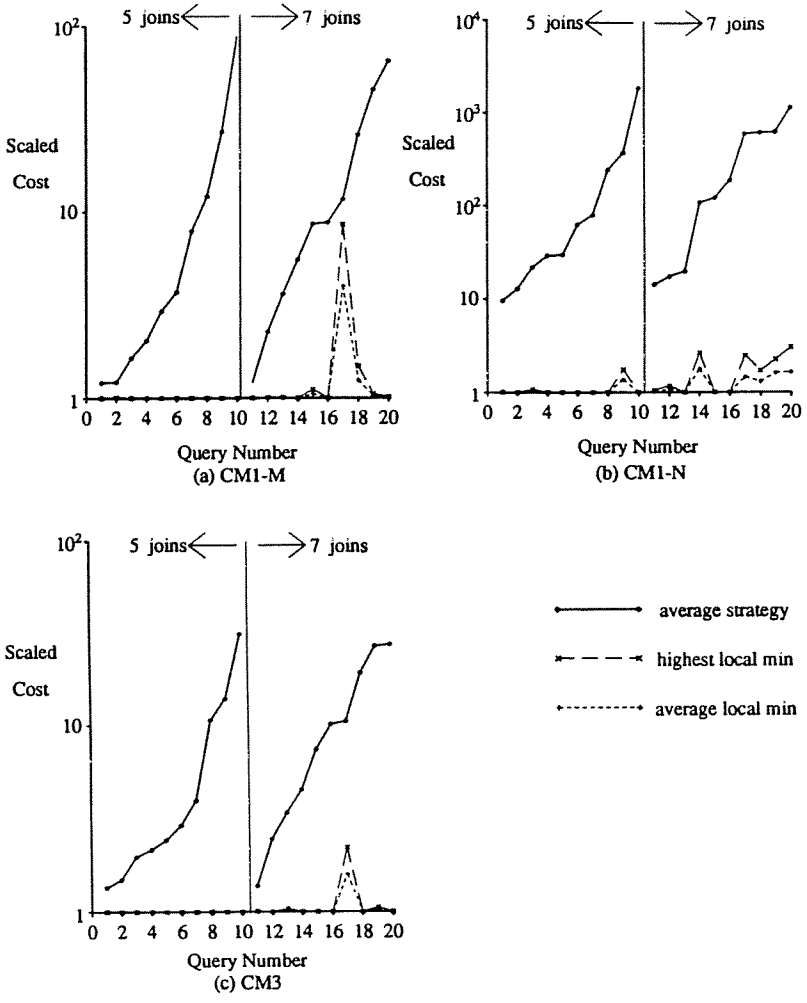


Figure 4.5: Aggregate characteristics of the cost distribution of all strategies and local minimum strategies for 'relcat4'

In general, the entire distribution of local minima is located very close to the global minimum cost relative to the average cost of all strategies. There have been a few exceptions in the case of CM1-M with 'relcat2' (Figure 4.4(a)), where the highest local minimum was higher than the average strategy. However, in all such instances, the overall cost range is extremely small, so fluctuations of this form are to be expected. It is interesting to compare Figures 4.4 and 4.5. As predicted in Section 4.3, when moving from 'relcat2' to 'relcat4' the overall cost range increases dramatically. At the same time, local minimum costs are pushed very close to the global minimum. Another interesting comparison is that among cost models, especially CM1-M and CM1-N. We observe that by being a quadratic algorithm, nested-loops is usually much more costly in the sense that it gives rise to much larger cost ranges than merge-scan. Choosing the wrong order when using nested-loops can be much more disastrous than when using merge-scan. In conclusion, the results of Figures 4.4 and 4.5 show that for all interesting cases and for all cost models and catalogs used, the shape of the cost function with respect to the local minimum cost distribution is of type A1. Thus our experimental results are consistent with our assessment of Section 4.4, which was based on analytical evidence.

To verify our assessment on the connection cost of low local minima, we have concentrated on those whose cost was below the average local minimum cost. Their connection cost was measured by a simple variation of a shortest path algorithm [Sedg83]. This experiment involved the same queries discussed above with the same three cost models and catalogs. Because of the similarities of most relevant data to

Figures 4.4 and 4.5, we do not show the results of these experiments on a per-query basis, but we summarize them in Tables 4.1 and 4.2. For each cost model and join size, we show the average over ten queries of the highest cost among the local minima that are lower than the average local minimum and the connection cost of those local minima. We also show the average cost of all local minima and that of all strategies. All the costs are scaled over the global minimum cost of each instance.

We clearly observe that in all cases the connection cost is very close to the highest low local minimum cost. This is again consistent with our assessment that the connection cost in *A* spaces is low and therefore that the overall space is of type B1.

cost model	CM1-M		CM1-N		CM3	
#joins	5	7	5	7	5	7
average strategy cost	1.22	1.23	77.18	41.07	1.21	1.29
average local min cost	1.03	1.05	3.38	2.80	1.00	1.01
connection cost	1.02	1.05	4.15	5.91	1.00	1.01
highest low local min cost	1.01	1.04	1.23	2.48	1.00	1.01

Table 4.1: Connection cost of local minima for 'relcat2'

cost model	CM1-M		CM1-N		CM3	
#joins	5	7	5	7	5	7
average strategy cost	14.79	17.77	268.49	342.84	7.22	11.37
average local min cost	1.00	1.33	1.04	1.29	1.00	1.06
connection cost	1.00	1.44	1.00	1.23	1.00	1.01
highest low local min cost	1.00	1.29	1.00	1.11	1.00	1.00

Table 4.2: Connection cost of local minima for 'relcat4'

4.5.2. Random sampling in spaces of large queries †

In this section, we present the results of experiments that we have performed to verify our assessment on the shape of *A* spaces for large queries. To get a feeling for the shape of the entire cost distribution, we first show a typical example of both the local minimum cost and random strategy cost distributions in Figure 4.6. The two distributions are drawn on the same x-axis but with different y-axis scales to show their relationship more clearly. The specific example is for a 40-join random tree query and the 'relcat1' catalog. The sample size has been 50 million for random strategies and 10000 for random local minima. It is evident that the local minimum cost distribution is very much shifted to the left relative to the cost distribution of all strategies, and that most local minima are located within a small area of low cost strategies. Similar experiments were conducted with many queries and different configurations with similar results. In the experiments with large queries, in addition to randomly generated tree queries, we have also looked into star queries specifically, to investigate whether any peculiarities arise from this especially hard to optimize query type. We have experimented with ten 20-join and ten 40-join random tree queries, each query being used with CM1 and three relation catalogs 'relcat1', 'relcat2', and 'relcat3'. The above has also been repeated for the same number of star queries. The previous results

† The experiments reported in this section for the strategy space analysis for large queries 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 have been very difficult to collect all necessary data in a reasonable time.

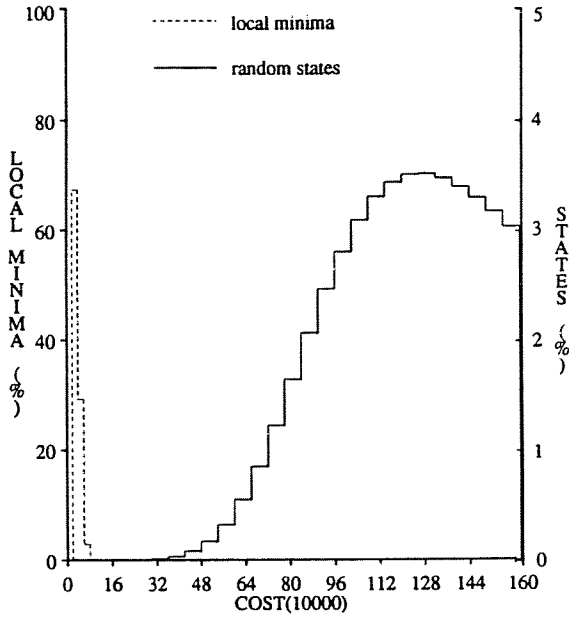


Figure 4.6: Cost distribution of random states and local minimum states

have shown that for *A* spaces, most of the local minimum costs are relatively low. To verify this, we randomly sampled 10000 local minima for each space. Random local minima were obtained by starting at random strategies and then following random downhill paths until a local minimum was reached. Thus, we were also able to obtain the cost distribution of all states in the space from the costs of the initial states.

As mentioned above, local minima were verified based on the p-local minimum definition. Even so, enumerating all neighbors of a strategy to check if it is a local

minimum is expensive, especially for large queries. For example, to give an idea for the computational requirements of these experiments, it takes more than ten days of cpu time on a μVax 3200 to obtain 10000 local minima for a 40-join query.

The results of the above experiments are summarized in Figures 4.7 and 4.8 for random tree and star queries respectively. For each query, we show the average strategy cost in the space and the average and the lowest local minimum costs found in the experiment. All costs are again represented by their ratio over the lowest local minimum cost found in the experiment. Queries 1 to 10 are 20-join queries and 11 to 20 are 40-join queries.

The general conclusions from the results are again consistent with what has been presented before, and show no essential difference between random tree and star queries. The average local minimum cost is several orders of magnitude lower than the average strategy cost. 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, consistent with our observations of increasing cost range, 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 lowest local minimum cost. The specific ratio of average vs. lowest 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,

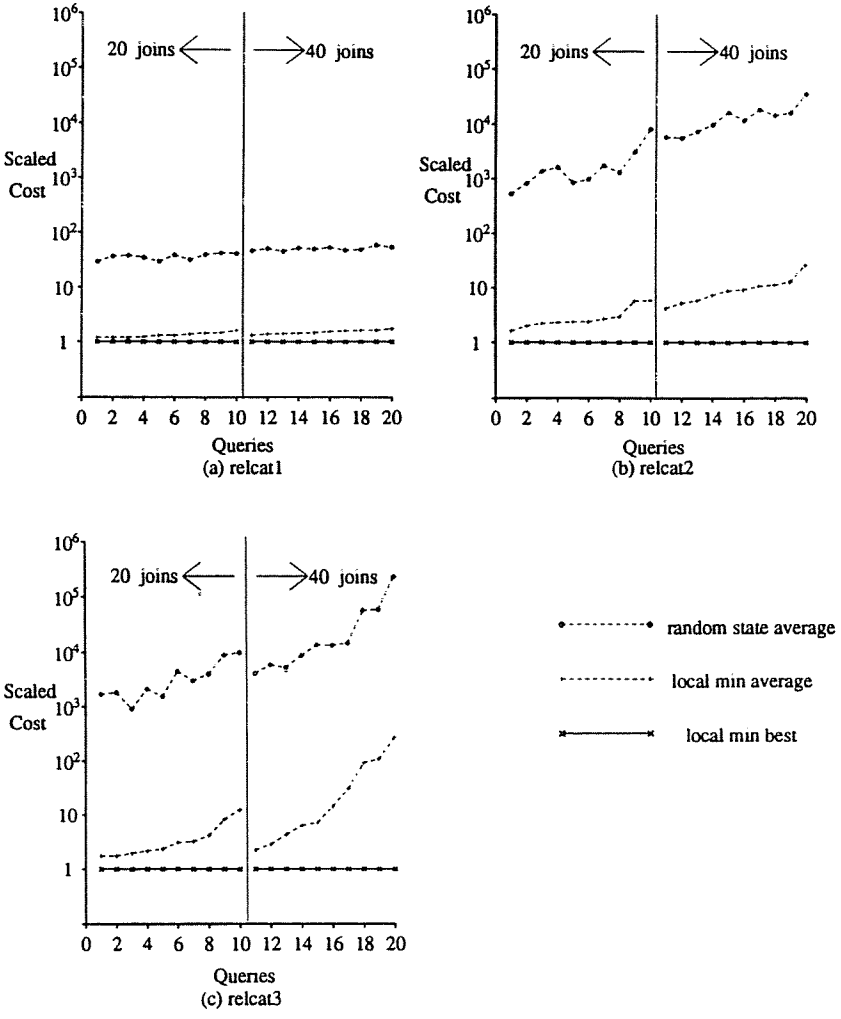


Figure 4.7: Cost distribution of random states and local minimum states for tree queries

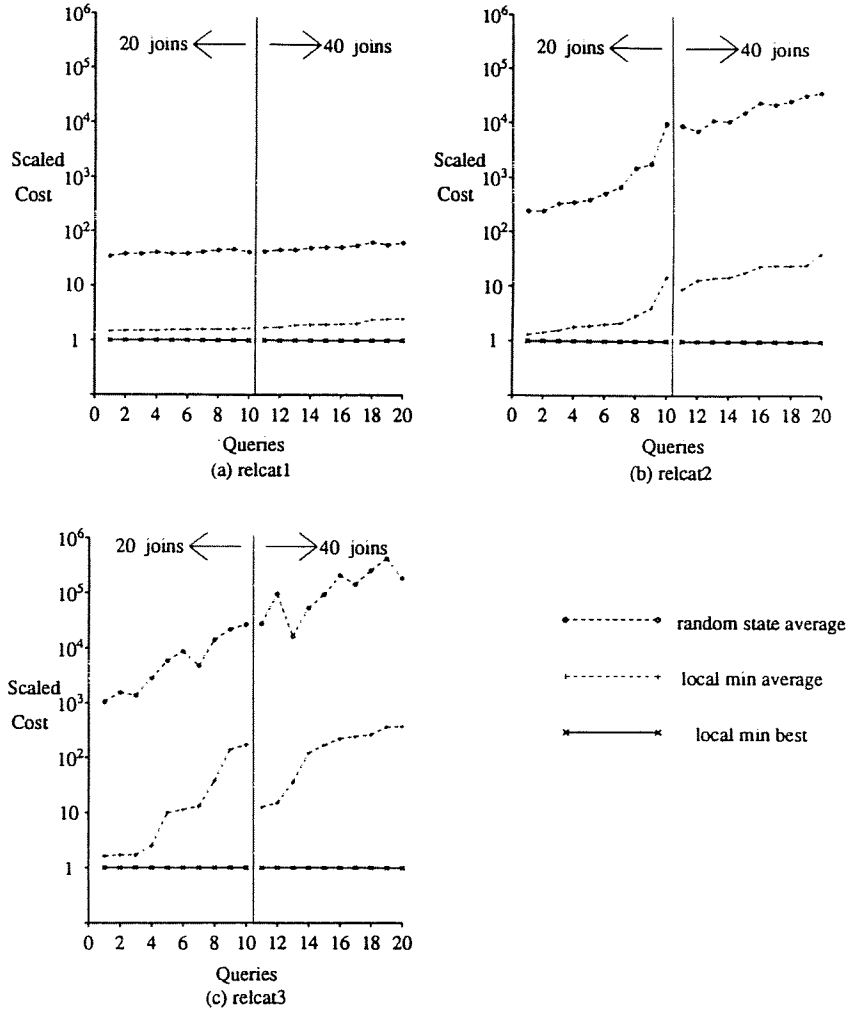
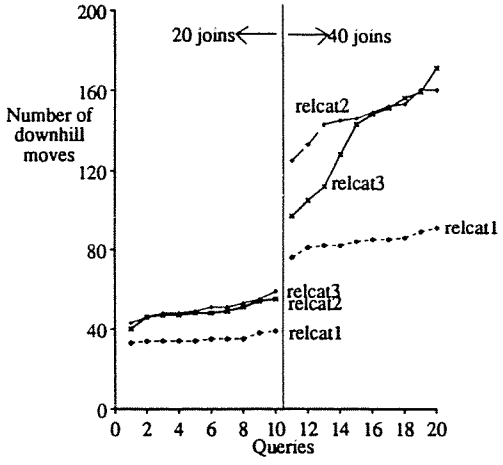


Figure 4.8: Cost distribution of random states and local minimum states for star queries

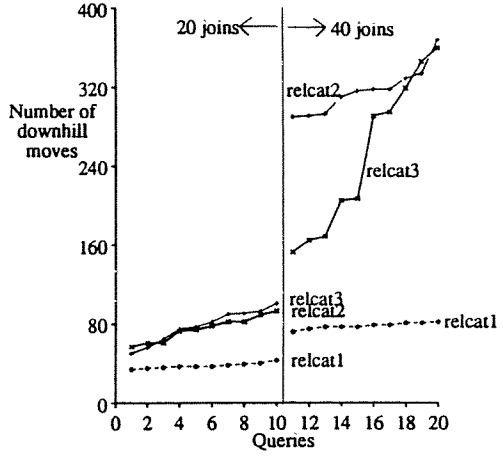
however, that cost difference is insignificant compared to the difference between the average strategy cost and the best local minimum cost, which is higher than five orders of magnitude. Hence, we can conclude that most local minima are not only far better than the average random state, but also that there is a relatively small variance in their costs.

During the above experiment, we also measured the number of downhill moves taken before local minima are reached. The average number of downhill moves for each random tree and star query is shown in Figure 4.9. This number of downhill moves is higher for star queries than for random tree queries, which is to be expected due to the larger size of the space for star queries. Moreover, it increases as the query size grows, while as the catalog changes from 'relcat1' to 'relcat3', it tends to present a maximum for 'relcat2'. The latter observation is again consistent with the expected increase in cost range and shifting to the left of the strategy cost distribution as the catalog changes from 'relcat1' to 'relcat3'. Especially from 'relcat2' to 'relcat3', the difference in shifting to the left is very significant and this decreases the average distance between random strategies and local minima. The general conclusion from the results in Figure 4.9 is that starting at a random state, many downhill moves are needed to reach a local minimum, which serves as yet another indication of the high range of strategy costs and the A1-type of the space shape.

To estimate the connection cost of low local minima in *A* spaces, we performed an experiment of 'random' walks in areas of low cost strategies. The purpose was to



(a) tree queries



(b) star queries

Figure 4.9: Number of downhill moves

obtain an understanding of the number of low local minima that are mutually connected with paths of relatively low costs and their mutual distance. For each query tested, we performed five 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 five times the average of the 10000 local minimum costs that we had from the previous experiments. Each series of downhill moves ended in a local minimum. Thus, a total of 10000 local minima were also visited in this experiment. For the same set of tree and star queries as in the previous experiment, we counted the number of local minima visited that had distinct costs. This provides only 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. The results showed no significant difference among the relation catalogs and the query types (random tree or star). So, they are summarized in Table 4.3, where we show the range of values for both measured quantities averaged over all queries of each size for both query types and all three catalogs. The general conclusion from the results in Table 4.3 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

	#distinct local minima (lower bound)		distance (upper bound)	
#joins	20	40	20	40
minimum	470	1187	6	13
average	3847	5447	16	48
maximum	8306	9903	37	221

Table 4.3: Local minima in the low cost area.

relatively close to each other (the size of the state space is several orders of magnitude higher than any distance reported in Table 4.3). Hence, the above results indicate that the shape of the *A* space of large queries tends to be of type B1 or B2.

All the above observations for the *A* spaces for large queries can be summarized as follows:

(1) The average local minimum has relatively low cost compared to the average state.

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

(3) The number of local minima is large.

(4) Many local minima are connected through low cost states within a short distance.

The above points (1)-(4) lead to a conclusion consistent with all other results, i.e., that the shape of the cost function over the strategy space resembles a 'well' of type A1-B1 or A1-B2. In other words, there is a small area of strategies with low costs, the 'well' bottom, surrounded by the remaining strategies with increasingly higher costs.

Chapter 5

SPACE OF LEFT-DEEP STRATEGIES

In this chapter, we present a combination of analytic and experimental results that characterize L spaces with respect to the shape of the cost function over them.

5.1. Structure of the Space

Similarly to A spaces, in this section L spaces are examined with respect to the structure of the strategies alone, ignoring any other features, e.g., join methods, whose role is straightforward. In addition, only the Swap transformation rule (rule (2) in Section 2.2.2) is considered. Any neighbor of a strategy that is produced by applying the 3-cycle transformation rule is reachable with two consecutive applications of the Swap rule as well. Hence, extending the forthcoming results to the complete space is straightforward.

5.1.1. Degree of strategies

Unlike A spaces, the L strategy space for an arbitrary query is rather difficult to analyze. In particular, studying the degree of strategies is rather difficult, since that parameter varies among different queries of the same join size and even among different strategies for the same query. Thus, we focus our attention on queries of some special form, which are relatively easier to analyze. Specifically, we study queries whose query trees have all their non-leaf nodes with the same degree, i.e., queries in which each relation participates in either 1 join or g joins, for some constant g. Let L_g denote the space that corresponds to such a query. It is straightforward to show that in the query graph (tree) of such a query with J joins, there are $(\frac{J-g}{g-1}+1)$ internal nodes and $(J-\frac{J-g}{g-1})$ leaves. For such queries, the following theorems hold.

Theorem 5.1: Consider a strategy in L_g for a query with J joins. If d is the degree of the strategy, then

$$d \le \begin{cases} \frac{J-1}{2} * \left[J - \frac{J-g}{g-1} \right] + g & \text{if } g < J \\ \frac{J-1}{2} * \left[J - \frac{J-g}{g-1} \right] + 1 & \text{if } g = J \end{cases}$$

Proof: Let s be the *relation sequence*[†] representing a strategy S in L_g. Let s_i, $0 \le i \le J$, be the i-th element of s. Let d_i, $0 \le i \le J$, be the degree of relation s_i in the

[†] Whenever we use the relation sequence notation to represent a strategy in L space, relations in the sequence are in the order in which they appear in the join processing tree, the first element being the outer operand of the first join.

query graph (tree) of the corresponding query. (Recall that d_i=1 if s_i is a leaf node in the query graph and d_i=g if s_i is not a leaf.) For each relation s_i, $0 \le i < J$, we obtain an upper bound on the number u_i of relations from $\{s_j \mid j>i\}$ that can be swapped with s_i (Swap rule) without creating a cross product. The sum of these upper bounds serves as an upper bound on the degree of S in L_g. The derivation is based on the fact that for a relation s_j, $j>i$, to be exchangeable with s_i it is necessary for s_j to be a neighbor of one of the relations $s_0, \ldots, s_{i-1}$ in the query graph. Note that the above condition is not sufficient, which is why we only obtain an upper bound and not the accurate value of the strategy degree.

By induction on $0 \le i \le J$, we show that $u_0 \le d_1$, and $u_i \le \sum_{j=0}^{i-1} d_j - (2i-1)$, for $1 \le i \le J$.

Basis: Clearly, the first relation s_0 in the relation sequence can be exchangeable only with neighbors of the second relation s_1 in the query graph (excluding itself, of course) and also with s_1 itself. Hence $u_0 \le d_1$. Similarly, s_1 is exchangeable only with neighbors of s_0 (excluding s_1), so $u_1 \le d_0 - 1$.

Induction step: Assume that the claim is true for some $0 \le i < J$. We prove it for $i+1$. Based on the discussion in the beginning of the proof, s_{i+1} is exchangeable only with relations of $\{s_j \mid j>i+1\}$ that are neighbors of some relation in $\{s_j \mid j \le i\}$, i.e., of some relation that appears in the sequence before s_{i+1}. Since strategies contain no cross products, the subgraph of the query graph induced by the nodes that correspond to the relations in $\{s_j \mid j \le i\}$ is connected, i.e., it is a tree itself. By the induction hypothesis, the number of neighbors of the relations in $\{s_j \mid j<i\}$ is less than or equal

to $\sum_{j=0}^{i-1} d_j - (2i-1)$. Adding s_i to this set removes one neighbor from the above number and adds $d_i - 1$ new neighbors. (The subtraction of 1 captures the fact that the connection of s_i to $\{s_j \mid j<i\}$ is consumed as well.) Hence,

$$u_i \le \sum_{j=0}^{i-1} d_j - (2i-1) + (d_i - 1) - 1 = \sum_{j=0}^{i} d_j - (2(i+1)-1) .$$

This concludes the induction, which proves the claim on the upper bound of u_i.

The sum U of the above upper bounds on u_i, $0 \le i \le J$, gives an upper bound on the number of neighbors for the strategy S.

$$U = \sum_{i=0}^{J} u_i \le \sum_{i=1}^{J} \sum_{j=0}^{i-1} d_j - \sum_{i=1}^{J} (2i-1) + d_1$$

Note in the above formula that the lower i is, the more times d_i contributes to the derived upper bound. Hence, this upper bound is maximized when all leaf relations in the query tree follow all non-leaves in the relation sequence, or more formally when d_i=1 implies d_j=1 for all $j>i$. For the special type of queries that we assume, this further implies that d_i=g, for $0 \le i \le \frac{J-g}{g-1}$, and d_i=1 for $i > \frac{J-g}{g-1}$. Using the above, the following series of derivations are possible.

$$\begin{aligned} d &\le \sum_{i=1}^{J} \sum_{j=0}^{i-1} d_j - \sum_{i=1}^{J} (2i-1) + d_1 \\ &= \sum_{i=1}^{\frac{J-g}{g-1}+1} ig + \sum_{i=\frac{J-g}{g-1}+2}^{J} \left[(\frac{J-g}{g-1}+1)g + (i-(\frac{J-g}{g-1}+1)) \right] - \sum_{i=1}^{J} (2i-1) + d_1 \end{aligned}$$

$$= g\frac{(\frac{J-g}{g-1}+1)(\frac{J-g}{g-1}+2)}{2} + g(\frac{J-g}{g-1}+1)(J-\frac{J-g}{g-1}-1) + \frac{J(J+1)}{2}$$

$$- \frac{(\frac{J-g}{g-1}+1)(\frac{J-g}{g-1}+2)}{2} - (\frac{J-g}{g-1}+1)(J-\frac{J-g}{g-1}-1) - 2\frac{J(J+1)}{2} + J + d_1$$

$$= \frac{J-1}{2}(J - \frac{J-g}{g-1}) + d_1$$

The theorem is proved by noticing that if $g=J$ then $d_1=1$ (to maximize U), while otherwise $d_1=g$.

□

Theorem 5.2: Consider the L_g space for a query with J joins whose query tree has a root node whose distance from all leaves is Δ. Then, there exist strategies in that space whose degree d is equal to

$$d = \frac{(g-1)^{\Delta-1}(g-2)(\Delta g-1)}{2} + g,$$

i.e., for a given g, $d=O(J\log_g J)$.

Proof: Let s and s_i be defined as in the proof of Theorem 5.1 for a strategy S. Consider a strategy S such that s_0 is a leaf node in the query graph (tree) and in general has leaf nodes as early as possible in s. For example, in the following query tree with $g=3$ and $\Delta=3$, we number the relations based on their order in such a sequence s. Relation 7 is the root.

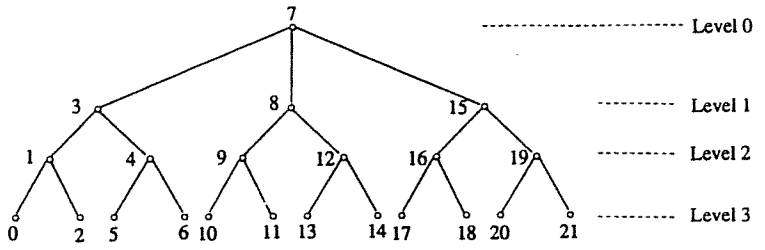


Figure 5.1: Query tree and relation sequence numbering that corresponds to a strategy of minimum degree in L_g

To obtain the number of neighbors in L_g of such a strategy, for each relation s_i in the sequence, we count the number u_i of relations from $\{s_j \mid j<i\}$ that can be swapped with s_i. It should be evident from the structure of this strategy that s_i can only be swapped with leaves in $\{s_j \mid j<i\}$ in order for a cross product not to be produced. Specifically, let R be the set of relations in the path between s_0 and the root of the query tree. For the above example, $R=\{0,1,3,7\}$. Then, every relation $s_i \notin R$ can be swapped exactly with all leaves s_j, $j<i$, of the subtrees rooted at the siblings of s_i that are not members of R. As an exception, the siblings of s_0 can be swapped with s_0 as well. Also, every relation $s_i \in R$ can be swapped with all leaves s_j of the subtrees rooted at the grandchildren of s_i that are not members of R. Again as an exception, the grandfather and father of s_0 can be swapped with s_0 as well.

We group all relations into *levels* based on their distance from the root in the query tree, and calculate the u_i's for relations in the same level together based on the above analysis. All calculations are of the form of counting the number of leaves of

certain subtrees and are rather straightforward. The cumulative results for each layer are shown below. We always show three terms in the sum which correspond to the node in R, its siblings, and the remaining nodes of the level concerned. Of course, some terms may be omitted if they are meaningless for some level.

Level 0: $\sum_{s_i \in l_0} u_i = (g-2)(g-1)^{\Delta-2}$

Level 1: $\sum_{s_i \in l_1} u_i = (g-2)(g-1)^{\Delta-3} + (g-1)^{\Delta-1}\frac{(g-2)(g-1)}{2}$

Level k, k=2,...,Δ−3:

$$\sum_{s_i \in l_k} u_i = (g-2)(g-1)^{\Delta-k-2} + (g-1)^{\Delta-k}\frac{(g-3)(g-2)}{2}$$

$$+ (g-1)^{\Delta-k}\frac{(g-2)(g-1)}{2}(g(g-1)^{k-2}-1)$$

Level Δ−2: $\sum_{s_i \in l_{\Delta-2}} u_i = (g-1) + (g-1)^2\frac{(g-3)(g-2)}{2}$

$$+ (g-1)^2\frac{(g-2)(g-1)}{2}(g(g-1)^{\Delta-4}-1)$$

Level Δ−1: $\sum_{s_i \in l_{\Delta-1}} u_i = 1 + (g-1)\frac{(g-3)(g-2)}{2} + (g-1)\frac{(g-2)(g-1)}{2}(g(g-1)^{\Delta-3}-1)$

Level Δ: $\sum_{s_i \in l_\Delta} u_i = \frac{(g-2)(g-1)}{2} + \frac{(g-2)(g-1)}{2}(g(g-1)^{\Delta-2} - 1)$

$$= \frac{(g-2)(g-1)}{2}g(g-1)^{\Delta-2}$$

The sum of the above numbers for all levels is equal to the degree d of the given strategy. Hence,

$$d = \sum_{k=0}^{\Delta}\sum_{s_i \in l_k} u_i = (g-2)\sum_{k=0}^{\Delta-2}(g-1)^k + 2 + \frac{(g-3)(g-2)}{2}\sum_{k=1}^{\Delta-2}(g-1)^k$$

$$+ \sum_{k=2}^{\Delta-1}\left[(g-1)^{\Delta-k}\frac{(g-2)(g-1)}{2}(g(g-1)^{k-2}-1)\right] + \frac{(g-1)^{\Delta-1}(g-2)(2g-1)}{2}$$

$$= (g-1)^{\Delta-1} + 1 + \frac{(g-3)((g-1)^{\Delta-1}-(g-1))}{2}$$

$$+ \frac{g(g-2)(g-1)^{\Delta-1}(\Delta-2)}{2} - \frac{(g-1)^{\Delta}-(g-1)^2}{2} + \frac{(g-1)^{\Delta-1}(g-2)(2g-1)}{2}$$

$$= \frac{(g-1)^{\Delta-1}(g-2)(\Delta g-1)}{2} + g$$

□

Example 5.1: As an example of the above theorems, consider star queries, which correspond to the space L_J. By Theorem 5.1, all strategies in L_J are of degree less than or equal to $J(J-1)/2+1$. In fact, for that space, all strategies have the same degree, which is equal to the given upper bound. As another example, consider string queries, which correspond to the space L_2. By Theorem 5.1, all strategies in L_2 are of degree that is less than or equal to $J+1$. By Theorem 5.2, there exist strategies of degree 2, which in fact is the lower bound on the degree as well.

□

Although the formulas derived in the above theorems hold for the L space of tree queries of a specific form only, nevertheless, they provide good indications for the general case and their qualitative implications should hold for any query. The conclusion from the above theorems is that in the L space, the degree of strategies is not always quadratic in the number of joins. Although high for many strategies in L,

the degree of many other strategies in the same space can be rather low.

Theorems 5.1 and 5.2 come in interesting contrast to Theorem 4.1. A comparison of the two show that in A, all strategies have the same degree, whereas in L, strategies have varying degrees, which can be higher or lower than the degree of the corresponding A space.

5.1.2. Interstrategy distance

Theorem 5.3: Consider the space L for a query with J joins. All strategies in that space are connected with a path of length less than or equal to $J(J+1)/2$.

Proof: Let S_1 and S_2 be two strategies for a query with J joins and let a and b be the relation sequences representing the two strategies, respectively. We denote the ith element of $a(b)$ by $a_i(b_i)$, $0 \le i \le J$. The sequence a can be transformed to b as follows.

```
procedure traverse(a, b) {
   s = a;
   for i=1 to J {
      j=i-1;
      while s_j, s_{j+1} appear in the opposite order in b {
         swap (s_j, s_{j+1});
         j=j-1;
      }
   }
}
```

Clearly, the above procedure transforms a to b, and takes at most $J(J+1)/2$ steps (application of swap routines). Each such step corresponds to an application of the Swap rule to the appropriate relations. To complete the proof of this theorem, we need

to show that no such step creates a cross product. Assume to the contrary that swapping $s_j=r$ and $s_{j+1}=t$, when $s_0=q$ creates a cross product for the first time. This implies that in the query tree, r has to be in the path between q and t with r and t being neighbors. However, b is either of the form $...q...t...r...$ or of the form $...t...q...r...$, and corresponds to a strategy without a cross product. Hence, r cannot be in the path between q and t in the query tree, which contradicts the assumption. Therefore, all transformations of the given procedure create no cross products.

□

Theorem 5.4: Consider two strategies in the L space for a query with J joins. Let l be the length of the longest path in the query tree whose relations appear in reverse order in the two strategies. Then, any path between the two strategies is longer than $l(l+1)/2 - 2$.

Proof: Let p be the path of the statement of the theorem, where $p=R_0R_1 \cdots R_l$. Let S_1 and S_2 be two strategies such that the relations on the path p appear in reverse order in the two strategies. To obtain a lower bound on the number of transformation steps to reach from S_1 to S_2, we concentrate on the transformations reversing the order of the relations in p. Considering the subsequence of the relations of p in any cross-product-free strategy, any swapping of two of these relations that does not create a cross product must be between two relations that are next to each other in the subsequence. Hence, the order of the relations in p must be reversed a pair-at-a-time. Thus, the whole transformation requires at least $l(l+1)/2$ steps. With respect to the first swap of R_0, the above accounting is not tight, since two steps can be saved by

swapping R_0 with R_2 directly instead of R_1. Hence, the lower bound on the length of any path from S_1 to S_2 can be tightened to $l(l+1)/2 - 2$.

□

Example 5.2: Consider star queries again. In the corresponding strategy space, any pair of strategies is connected with a path of length less than or equal to J. This space is the one with the closest connections between strategies. For string queries, by Theorem 5.4, there exists a pair of strategies for which the shortest path connecting them is of length $J(J+1)/2 - 2$.

□

The implications of the above theorems are that the distance between strategies is rather short, quadratic in the number of joins in the worst case. Moreover, there do exist strategies whose distance has a quadratic lower bound, so the results of these theorems are tight.

Comparing Theorems 4.2 and 5.3, we see that the maximum interstrategy distance in the two spaces for all queries is the same, although the size of the A space is much greater than that of the corresponding L spaces.

5.2. Effect of Ranked Join Methods

As mentioned in Section 4.2, if exchanging adjacent sequences of relations in a strategy was a legal transformation to generate neighbors, that subspace would have a unique local minimum area. Although this is not true in all L spaces, at least for star queries, exchanging any pair of adjacent relations is legal (assuming that the relation in the center of the star remains the first relation in the strategy). This fact can be used

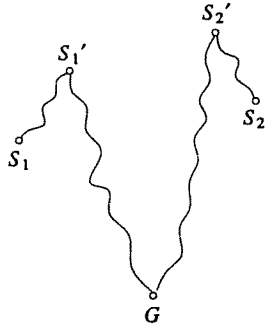
again to prove several interesting properties regarding the connection cost of strategies in L spaces for star queries.

Theorem 5.5: Consider a star query and a pair of strategies S_1 and S_2 in the corresponding L space. Then the following hold:

(a) If all available join methods are ranked, there exists a path connecting S_1 and S_2 whose most expensive strategy is either S_1 or S_2 or one of their neighbors.

(b) If non-ranked join methods are available, there exists a path connecting S_1 and S_2 whose most expensive strategy is at most $J+1$ steps apart from S_1 or S_2.

Proof: The proof is similar to that of Theorem 4.3. For each case, it suffices to show that S_1 and S_2 can be transformed to S_1' and S_2', respectively, in at most the given number of steps, where S_1' and S_2' are left-deep trees, with the center of the query tree as their first relation and with all join methods ranked.



To prove the theorem, we treat the two cases separately.

(a) In this case, we can obtain $S_1'(S_2')$ from $S_1(S_2)$ by applying join commutativity on the first join if its left operand is not the relation in the center of the star. This takes at most one step.

(b) In this case, we can obtain $S_1'(S_2')$ from $S_1(S_2)$ by the transformation in (a), if necessary, and then by changing the join method into a ranked one for each join whose join method is not ranked. This takes at most $J+1$ steps. □

Again, considering the huge size of the strategy space, one realizes that the above linear bound is extremely small. The implications of Theorem 5.5 for L are very similar to those of Theorem 4.3 for A. Essentially, the general conclusion is that for both spaces the connection cost among low local minima is also relatively low.

5.3. Cost Distribution of All Strategies

With respect to the cost distributions of all strategies, A and L spaces are very similar. We have performed several experiments, for which we obtained approximations of the L space cost distributions for the same queries with the same cost models and catalogs that we used for the study of the A space. The results have been very similar to those reported in Section 4.3, so we do not present them here as well. Again, the primary issue that seems to affect these distributions is variance in the contents of the relation catalogs; when it increases, the cost range of strategies increases as well and the distribution is shifted further to the left.

5.4. Shape of the Cost Function: Expectations from Analytical Results

In this section, we identify the shape of the cost function of L spaces. We argue on the shape type based on the results of the previous sections.

The characteristics of L spaces are as follows. Most strategies in the space have a high degree (Theorem 5.1), so the local minimum distribution is shifted to the left relative to the cost distribution of all strategies. On the other hand, for random tree queries, there are strategies with relatively few neighbors (Theorem 5.2) and the cost difference between neighbors can be high as well. Hence, from our analysis in Chapter 3, most likely the shape is of type A2, compared to the shape of A spaces which was of type A1.

With respect to the connection cost, L has very similar characteristics with A, and therefore the same arguments apply to both. The interstrategy distance is small compared to the size of the strategy space, so one can conclude that the connection cost of low local minima cannot be much higher than their cost. Hence, the shape should be of type B2 or B1. The effect of using some ranked join methods and that of the specific strategy cost distribution is the same in the two spaces as well, so for example when the distribution is further shifted to the left, the shape changes from type B2 to B1.

The overall conclusion from the above observations is that the shape is of type A2-B2 or A2-B1.

5.5. Shape of the Cost Function: Experimental Results

In this section, we present the results of several experiments that we performed, which verify our conjectures from the previous section on the shape of L spaces. Again, we first show results for small queries and then for large ones.

5.5.1. Exhaustive search of spaces of small queries

Unlike strategies in A, strategies in L can be enumerated in a straightforward way, since each strategy has a natural representation as a relation sequence. The only concern is that relation sequences that correspond to strategies with cross products must be excluded. This is achieved by a variation of depth first search on the query tree, by which all relation sequences starting from a given relation can be generated. Repetition of this process for each relation in the role of the first relation in the sequence accounts for all relation sequences.

Again, we mapped all strategies to unique numbers for efficient internal representation. Each strategy in L can be identified by the corresponding relation sequence (there are at most $(J+1)!$ of those) and its specific choice of join methods for all joins (there are exactly M^J of those for M join methods). Let the functions f_1 and f_2 be from relation sequences to integers in the range $[1,(J+1)!]$ and from join method combination to integers in the range $[1,M^J]$, respectively. Then, strategy S can be mapped to a unique number by the function

$$f(S) = (f_1(S)-1)M^J + f_2(S) .$$

Figure 5.2 shows the pseudocode for the enumeration algorithm of the strategies in the

L space of a query together with the identification of their neighbors.

```
procedure generateL() {
   for each valid relation sequence {
      for each join method combination of S {
         write f (S);
         write cost (S);
         for each neighbor S' of S
            write f(S' );
      }
   }
}
```

Figure 5.2: Enumeration of L strategy space

Using the above space construction process, we performed the same experiments for the same queries of 5 and 7 joins with the same cost models and catalogs as in the study of A spaces. The results with respect to the local minimum distribution are summarized in Figures 5.3 and 5.4 for 'relcat2' and 'relcat4', respectively. Again, queries 1 to 10 have five joins and queries 11 to 20 have seven joins. For each query, we show the average strategy cost as well as the average and highest local minimum cost. All these costs are scaled over the global minimum cost.

In contrast to the A spaces, we can identify several queries with no strong shifting to the left of the local minima distribution relative to the cost distribution of all strategies, even with catalogs that produce reasonably large cost range. We should emphasize that unlike our analytical work, for these experiments, the complete set of transformation rules was used to construct the L space, i.e., both the Swap and 3-cycle rules were used. This implies that strategies have the potential of having $O(J^3)$

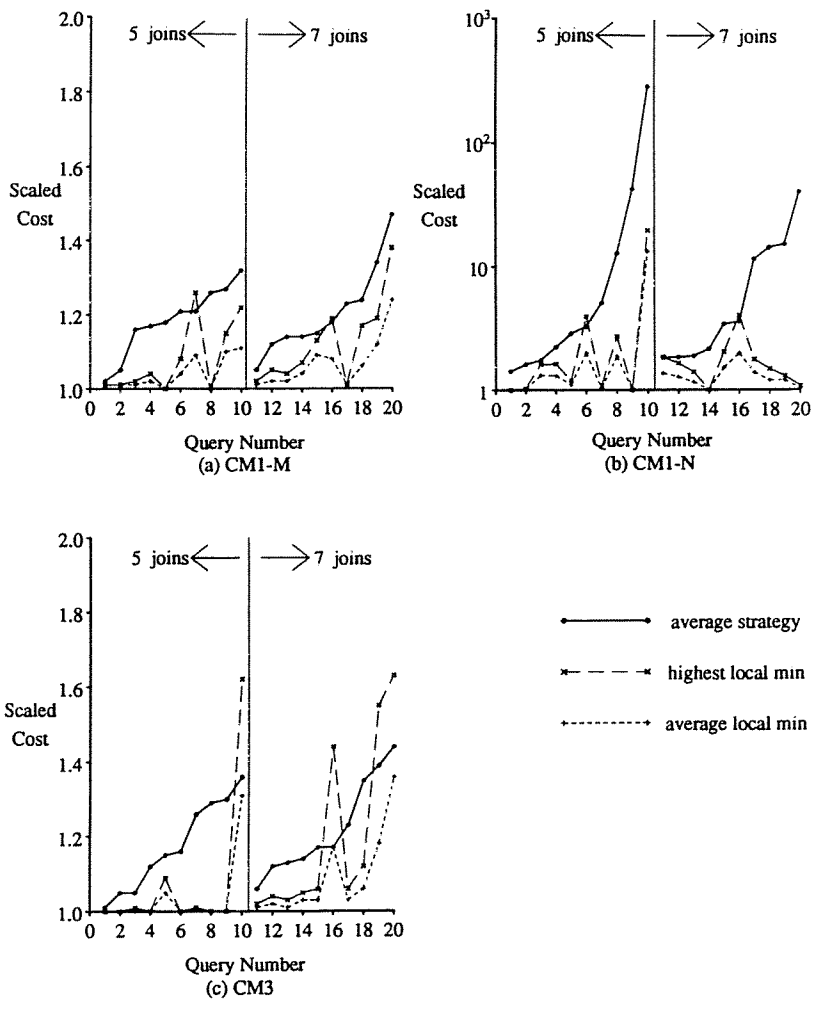


Figure 5.3: Aggregate characteristics of the cost distribution of all strategies and local minimum strategies for 'relcat2'

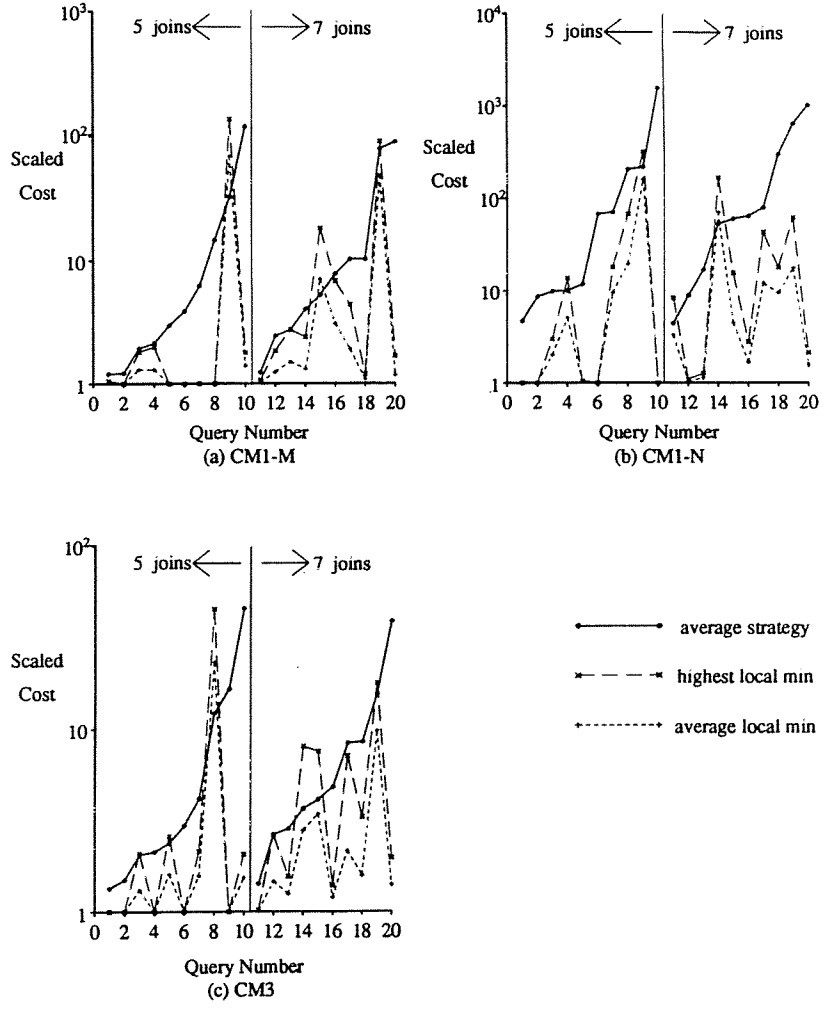


Figure 5.4: Aggregate characteristics of the cost distribution of all strategies and local minimum strategies for 'relcat4'

neighbors. Nevertheless, the effect of low degree strategies is clear. Hence, this verifies the claim that the shape of L spaces, with respect to the local minimum cost distribution, can in general be characterized as type A2.

In addition to the costs of the local minima, we also measured the highest connection cost among the local minima that were lower than the average cost of all local minima. These experiments involved the strategy spaces of the same queries of 5 and 7 joins with the same catalogs and three cost models as above. We summarize the results in Tables 5.1 and 5.2. Similar to A spaces, the general observation is that the connection cost is relatively close to the highest local minimum cost, which is consistent with our assessment that the space shape is of type B2 or B1. There were a few exceptional spaces where the connection cost would seem relatively high, but even then, it was lower than the average strategy cost.

cost model	CM1-M		CM1-N		CM3	
#joins	5	7	5	7	5	7
average strategy cost	1.19	1.21	36.04	9.69	1.21	1.22
average local min cost	1.04	1.07	2.49	1.30	1.00	1.09
connection cost	1.00	1.06	3.08	1.30	1.00	1.02
highest low local min cost	1.00	1.02	1.10	1.14	1.00	1.01

Table 5.1: Connection cost of local minima for 'relcat2'

cost model	CM1-M		CM1-N		CM3	
#joins	5	7	5	7	5	7
average strategy cost	18.54	21.33	215.27	223.13	9.07	9.10
average local min cost	7.88	6.70	20.00	12.34	3.41	2.61
connection cost	1.07	18.39	3.54	7.42	1.07	4.20
highest low local min cost	1.03	1.98	1.98	3.40	1.04	1.55

Table 5.2: Connection cost of local minima for 'relcat4'

5.5.2. Random sampling in spaces of large queries

We have applied the experiments of Section 4.5.2 on L spaces of large queries to a very limited extent. The reason for not experimenting extensively with the L space as with the A space is that for the same queries, the experiments were much more time consuming with the former than with the latter where they were already quite expensive. The increased cost of the experiment is due to the potentially very high degree of strategies. In particular, visiting low cost strategies with degree that is $O(J^3)$ takes a tremendous amount of time, since such a large number of neighbors must be checked to discover possible lower cost neighbors or to identify the strategy as a local minimum.

We have experimented with ten 20 join queries with 'relcat4' and cost model CM3, which corresponds to the testbed that Swami and Gupta used in their work [Swam88]. Given the limited scope of experimentation, we concentrated on this case, since an understanding of its characteristics was of much interest to us. The experiments were conducted exactly as described in Section 4.5.2. The results

presented below address the local minimum cost distribution only. The behavior of the various randomized algorithms, which is discussed in the following chapter, provides additional evidence that verifies our assessment with respect to the connection cost for the L spaces of large queries as well.

Table 5.3 contains the lowest, average, and highest local minimum cost, as well as the average strategy cost found. All costs are scaled to the lowest local minimum found. Seven of the queries used in the experiments produced very similar results, which are therefore shown together. For all of them, it is clear that the corresponding L space is of type A1, with most local minima being of very low cost. The fact that local minima are so low can be attributed in part to the characteristics of catalog 'relcat4' that we mentioned in Section 4.3. That is, the strategy cost distribution is shifted extremely to the left, i.e., a large percentage of all strategies are of low cost. This shifts most local minima to low cost as well. The remaining three queries have spaces that seem much more irregular and should probably be classified as type A2. Interestingly, additional experimentation showed that despite the average characteristics of the local minima, a large percentage of them (about 10%) had cost within a factor of 2 from the lowest local minimum. This can again be attributed to the characteristics of catalog 'relcat4' and the resulting shift to the left of the strategy cost distribution.

	query1-7	query8	query9	query10
average strategy	1×10^4	7×10^7	286.91	2×10^3
highest local minimum	32.17	8×10^5	652.74	270.38
average local minimum	1.39	7×10^3	11.17	34.36
lowest local minimum	1.00	1.00	1.00	1.00

Table 5.3: Characteristics of local minimum cost distribution for the L space.

In conclusion, at least within the limited scope of these experiments, the general trends have been consistent with our overall assessment on the characteristics of the local minimum cost distribution in L spaces that it is of type A2 but moving to type A1 when the strategy cost distribution is shifted extremely to the left. Overall, the space can be characterized as a 'well' of type A2-B1 or A2-B2, changing towards A1 on certain cases.

Chapter 6

RANDOMIZED ALGORITHMS

In the previous chapters, we presented the results of our study of the characteristics of both *A* and *L* spaces, which randomized query optimization algorithms must search for a global minimum. We concluded that the shapes of the cost functions of both spaces resemble a 'well', although of a different quality. In this chapter, we describe three randomized optimization algorithms with which we have experimented on searching these spaces for query optimization. These are Iterative Improvement (II) and Simulated Annealing (SA), which are known to be effective for many combinatorial optimization problems, including query optimization, and Two Phase Optimization (2PO), which we proposed in order to take advantage of the 'well' shape of the cost function, and which is a combination of II and SA. We present the results of a performance evaluation of these algorithms on query optimization within the *A* and *L* spaces. Based on these results, we also compare the two spaces, and suggest one of them as the most appropriate for query optimization.

6.1. 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.

6.1.1. Iterative Improvement (II)

The generic Iterative Improvement (II) algorithm is shown in Figure 6.1. 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

```
procedure II( ) {
  minS = S∞;
  while not (stopping_condition) {
      S = random state;
      while not (local_minimum(S)) {
          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 6.1: Iterative Improvement

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.

6.1.2. 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 6.2. 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

[†] In Figure 6.2, 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 6.2 can only improve on the results of the pure version and is the one that we use in this study.

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) {
        while not (equilibrium) {
            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 6.2: Simulated Annealing

6.1.3. Two Phase Optimization (2PO)

The Two Phase Optimization (2PO) algorithm is a combination of II and SA. As the name suggests, 2PO can be divided into two phases. In the first phase, 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,

where 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. Hence, 2PO is appropriate when such an ability is not necessary for proper optimization.

6.2. Query Optimization in the Space of Deep and Bushy Strategies

In this section, we present the results of a performance evaluation of the above algorithms for query optimization in *A* spaces. We further explain these results based on the previous analysis of the characteristics of the spaces and the shape of their cost function.

6.2.1. Implementation specific parameters of the algorithms

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 for query optimization in *A* spaces. We arrived at them after some experimentation with various alternatives, and also on the basis of past experience with the algorithms in query optimization [Ioan87] and other fields [John87]. In what follows, J is used to denote the number of joins in a query.

The only parameter that needs some explanation in the above tables is the definition of a local minimum for II. Clearly, there is a significant cost involved verifying the truth

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

Table 6.1: 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	$16*J$ visited states in the current stage
next state	random neighbor
temperature reduction	$T_{new} = 0.95*T_{old}$

Table 6.2: 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 6.3: Implementation specific parameters for 2PO.

of the precise definition of a local minimum. Moreover, even exhaustively searching all neighbors of a strategy, to test whether it is a p-local minimum or not, is too expensive during execution of the algorithms. Hence, we use another approximation to identify a local minimum for II. 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.*

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 occasionally be missed, and a state may be falsely considered as a local minimum. In most cases, however, 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.

6.2.2. Experiment Profile

We implemented all algorithms in C, and tested them on a Sun-4 workstation when no one else was using the machine. For *A* spaces, we experimented extensively with cost mode CM1 and three relation catalogs, 'relcat1', 'relcat2', and 'relcat3', for both tree and star queries. 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 in which each run of SA would require more than two hours, for which no experiments were conducted with SA. Thus, we have no results for 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.

6.2.3. 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 6.3. 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

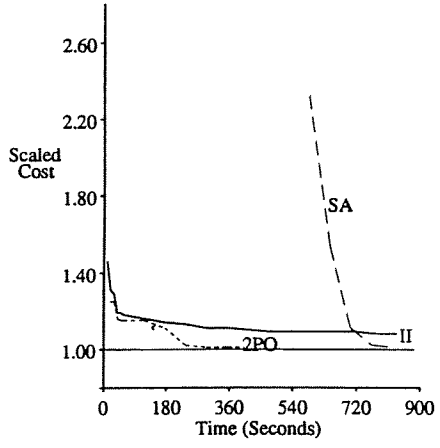


Figure 6.3: Minimum cost found over time

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, in general, several orders of magnitude. 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 directly. Indeed, we observe that initially 2PO improves as quickly as II, but soon surpasses it, and eventually converges to its final solution much more rapidly than SA.

The above typical behavior of SA, II, and 2PO over time can be easily explained by the fact that the shape of *A* spaces is of type A1-B1 or A1-B2 (Chapter 4). SA starts from a random state, which tends to be in the high cost area. While the temperature remains high, because of the large number of uphill moves from states in the high and middle cost areas, 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 'well' bottom, which it explores extensively, taking advantage of its ability to visit many local minima by accepting uphill moves. On the other hand, II can reach the 'well' 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 'well' 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.

6.2.4. Output quality

The cost of the output strategies produced by the algorithms for the various relation catalogs as a function of query size is shown in Figure 6.4 for tree queries and Figure 6.5 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 scaled cost over all queries of that size over all five runs of each query is shown.

In general, we observed no significant qualitative difference between tree and star queries with respect to the relative output of the algorithms. Hence, below we discuss how the results change as we move along the two other 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.

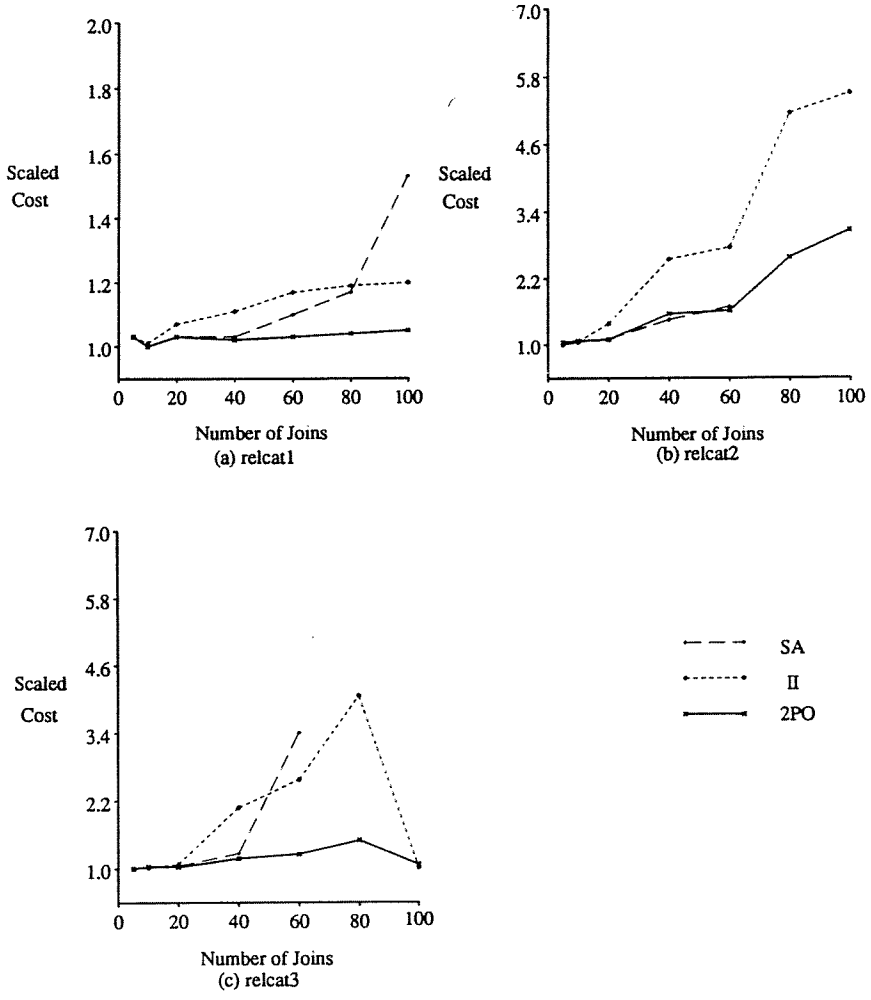


Figure 6.4: Average scaled cost of the output strategy of SA, II and 2PO for tree queries

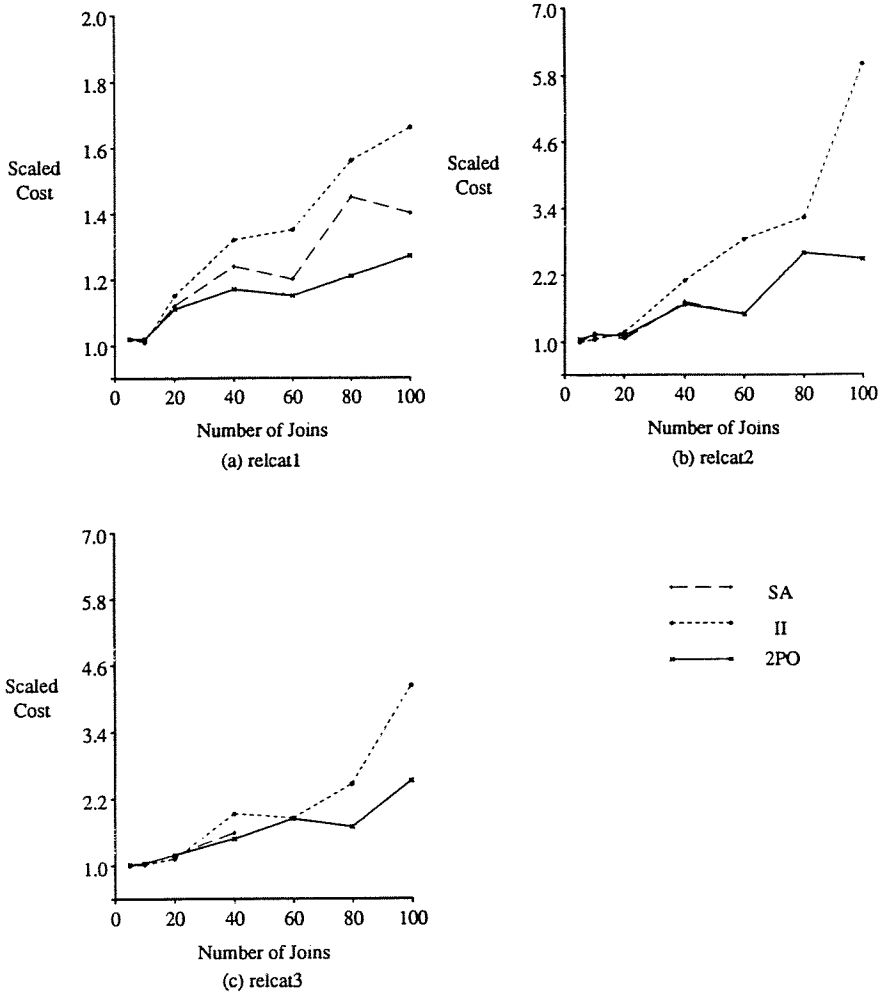


Figure 6.5: Average scaled cost of the output strategy of SA, II and 2PO for star queries

This means that there are more cases in which algorithms miss the optimum. Of the three algorithms, however, 2PO is relatively the most stable. (ii) *Variance in catalog parameters:* The output difference among the algorithms increases with higher variance in the relation cardinalities and the join selectivities, i.e., as we move from 'relcat1' to 'relcat2'/'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 does 'relcat3'. This shows that initially, as we move from 'relcat1' to 'relcat3', the increase in cost range has a stronger effect than the shifting to the left of the strategy cost distribution. Therefore, optimization becomes harder and unstable algorithms fail more often. Beyond a certain point, however, the situation is reversed and the extreme shifting to the left of the strategy cost distribution makes optimization easier with more algorithms succeeding more often.

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 this over all queries of a given size in Figure 6.6 for tree queries and in Figure 6.7 for star queries. 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

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

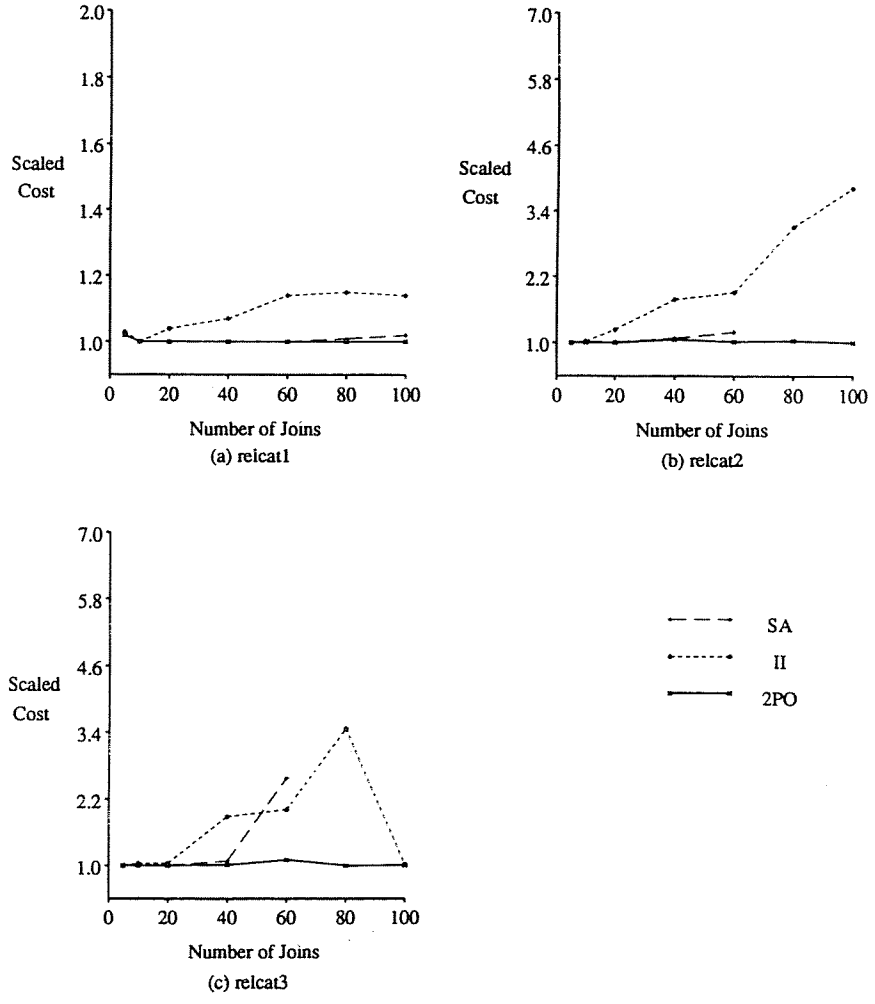


Figure 6.6: Best scaled cost of the output strategy of SA, II and 2PO for tree queries

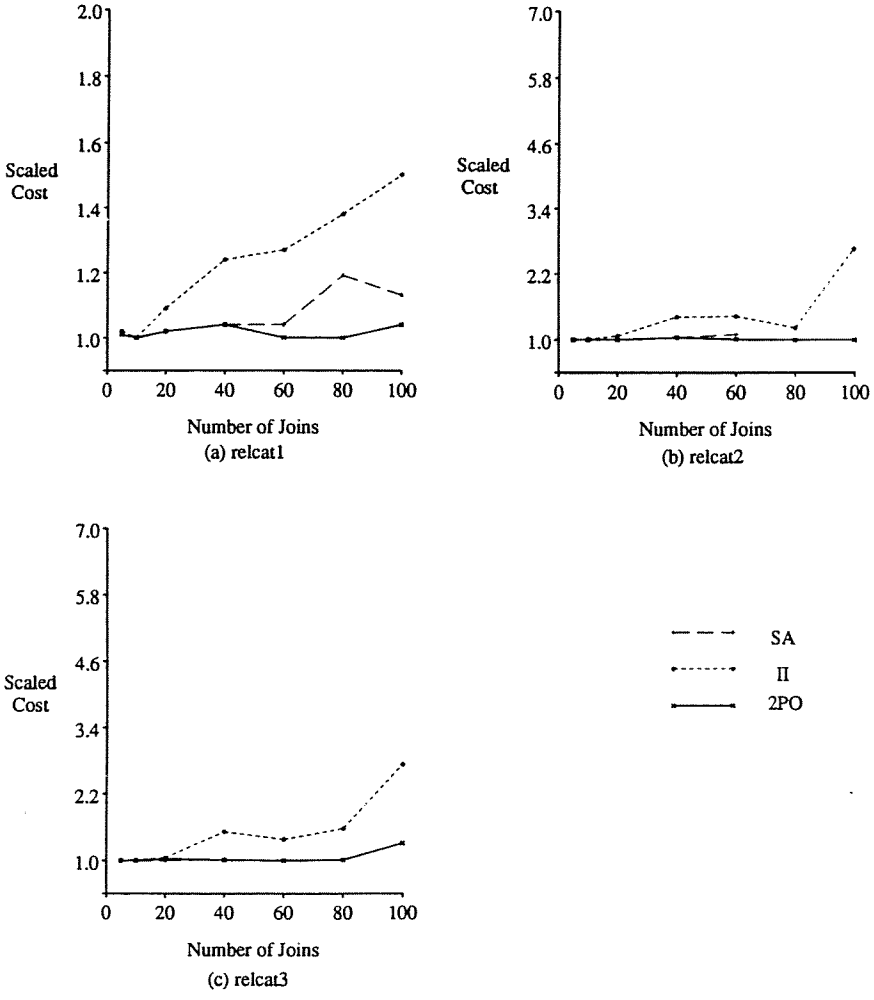


Figure 6.7: Best scaled cost of the output strategy of SA, II and 2PO for star queries

very stable as well. In contrast, 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, so we elaborate on this no more.

As an exception to the above general observation, there was one case in which 2PO and II did not exhibit any performance difference on either the average or the best of five runs (100-join tree queries with catalog 'relcat3'). In fact, we observed that 2PO hardly improves the result from the best local minimum found among the ten local optimizations of the first phase. II does not improve the result after a small number of local optimizations either. To understand this phenomenon better, we randomly sampled 1000 local minima for each query in that case. Quite surprisingly, the lowest local minimum cost found in these experiments was still very similar to the minimum cost found by the two algorithms. Moreover, a large percentage of the local minima were within 1.1 times the minimum cost found. (The actual percentages for each of the five queries were 5.3%, 2.5%, 1.8%, 1.9%, and 4.7%.) Hence, it became evident that in these cases, the local minimum cost distribution is shifted extremely to the left. With so many local minima of costs close to that of the global minimum, a small number of local optimizations was enough to hit the 'well' bottom. The reason for the above phenomenon is the combination of having a large number of relations with high variance in the catalog characteristics ('relcat3'). As discussed in Chapter 4, this creates a large number of similarly good strategies that shift the overall strategy cost distribution to the left with the above effects.

The fact that the shape of *A* spaces is of type A1-B1 or A1-B2 can be used again to 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 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 relatively long (Figure 4.8); 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 'well' 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 II visits more states than SA overall, which visits more states than 2PO, the last two visit more states of low cost than the first. A typical situation is shown in Figure 6.8. The example is for a 40-join tree query with the 'relcat1' catalog. Similar behavior was observed for other queries.

6.2.5. Query optimization time

The average running (CPU) time of 2PO and SA as a function of query size for various cases is shown in Figure 6.9 for tree queries and in Figure 6.10 for star queries. (Recall that II was given the same amount of time as SA or 2PO, depending on whether SA was run or not.) Below, we discuss the effect of query size, variance in

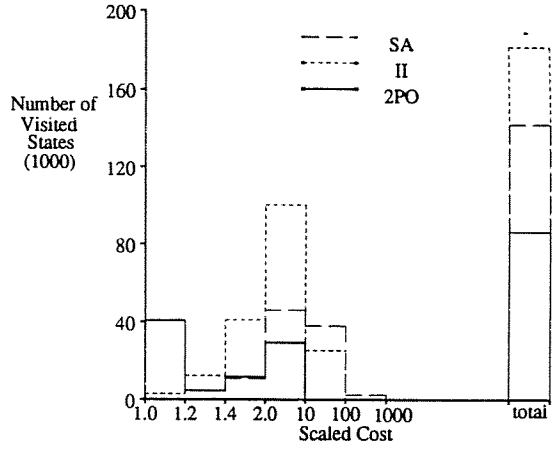


Figure 6.8: Number of visits in each cost area.

catalog parameters, and query type on query optimization time. (i) *Query size:* In general, as expected, the running time increases very steadily with query size for both SA and 2PO. The only fluctuations on the time increase for 2PO were with 'relcat3' for both random tree and star queries of many joins. This is again due to the overall shift of the strategy cost distribution to the extreme left in these cases, as discussed in previous section. This has a clear effect on processing time, since the number of stages in the SA phase depends on the initial temperature, which is a function of the initial state cost of that phase. Comparing the two algorithms, we observe that 2PO needs less time than SA in all cases. As expected, the absolute difference in running time increases with query size. (ii) *Variance in catalog parameters:* In general, the time increases from 'relcat1' to 'relcat3'. This is primarily due to the increase in the

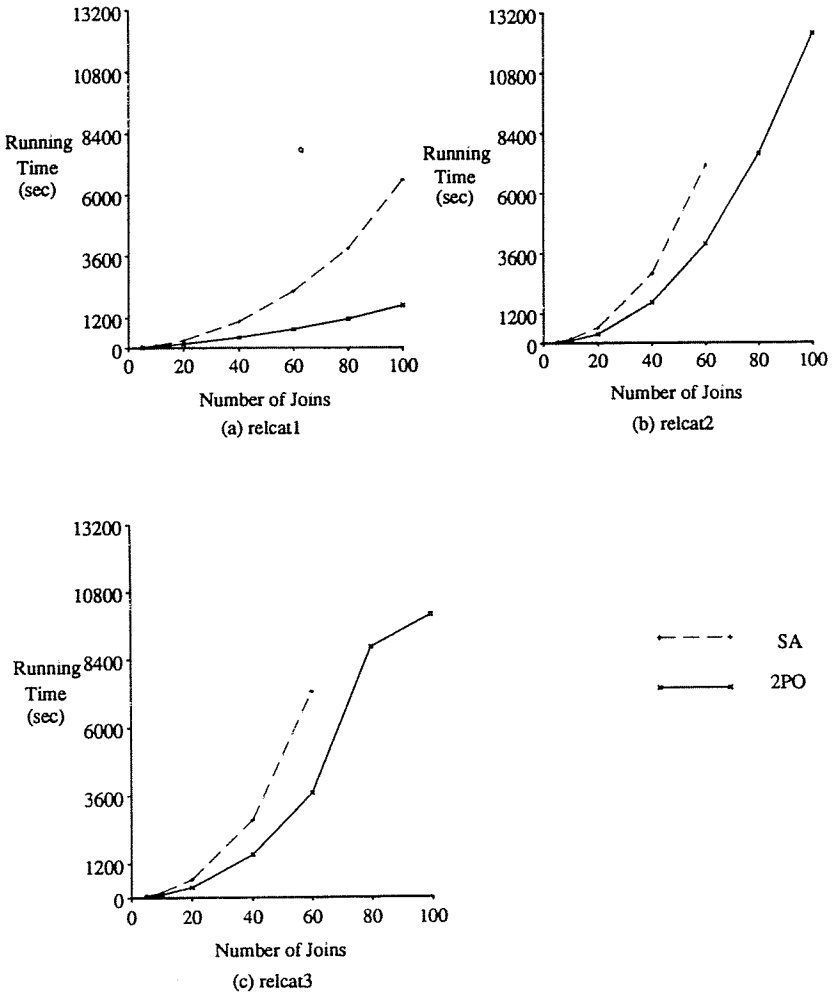


Figure 6.9: Average running time of SA and 2PO for tree queries

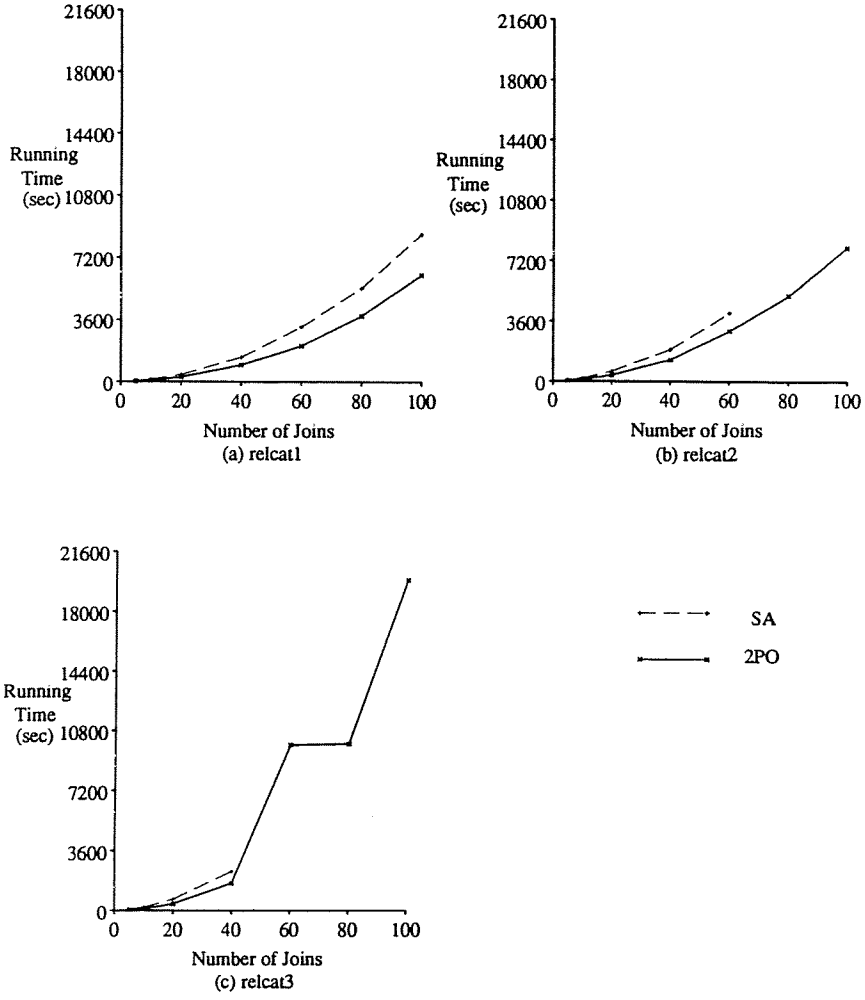


Figure 6.10: Average running time of SA and 2PO for star queries

average strategy cost. Deviations from this occur when the strategy cost distribution is shifted to the left, as discussed above. (iii) *Query type:* As expected [Ono90], star queries took significantly more optimization time than random tree queries for both algorithms. In addition, the improvement of 2PO over SA for star queries was much smaller than for tree queries. 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 correspond to much larger spaces, and therefore tend to need relatively more time in the second phase of 2PO.

6.2.6. Experiments with CM2

As an extension to the study described in the previous sections, and to verify that the specific choice of cost model did not affect the obtained results, we conducted a limited set of experiments with the CM2 cost model. In particular, we experimented with twenty 20-join and twenty 40-join tree queries with the same three catalogs. The set-up of these experiments was exactly the same as that described for the experiments with CM1. The results on the behavior of all three algorithms in the new setting are very similar to those of the CM1 experiments. 2PO always outperforms II and SA in terms of both output quality and running time. This implies that our results on CM1 were not a result of our choice of join methods or the specific restrictions on the cost function. In Tables 6.4 and 6.5, we show the average and best scaled cost of the output strategies of five runs for all three algorithms for each combination 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 are averaged over all queries with the same characteristics.

The one point that we want to emphasize in these results is that, again, optimizing queries with the 'relcat2' catalog appears to be the most difficult case. For queries with the 'relcat1' catalog the cost range is very small, so all algorithms have an easy time. For queries with 'relcat3' catalog the strategy cost distribution is shifted to the left significantly, so all algorithms perform better again than with the 'relcat2' catalog. This shift to the left and its effect on the behavior of algorithms is more evident with CM2 due to the presence of three join methods, which tends to intensify the 'well' characteristics of the space shape.

	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 6.4: 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 6.5: Best scaled cost of the output strategy.

6.3. Query Optimization in the Space of Left-deep Strategies

In this section, we describe the expected behavior of the algorithms in L spaces and present a limited set of experiments to verify those expectations. We concentrate on the issues in which L spaces are different from A spaces.

6.3.1. Implementation specific parameters of the algorithms

For query optimization in L spaces, the implementation specific parameters of II and SA were chosen to be the same as in the previous study of L spaces by Swami and Gupta [Swam88]. For II, we use again the r-local minimum approximation, where a state is considered to be a local minimum after n randomly chosen neighbors of it are tested (with repetition), none of which has lower cost. This time, however, n is equal to J, the number of joins, which can be much less than the number of neighbors for most strategies. There are two reasons for using a different definition of a r-local minimum in L spaces. First, different strategies in the L space have different degrees. Hence, given a strategy, it is virtually impossible to compute its degree without actually generating all of its neighbors, which could defy the purpose of using an approximation to local minima. Second, even if the strategy degree was known, quite often that is very large, up to $O(J^3)$ for a J-join query. Making that many attempts to identify downhill moves from a strategy is extremely time consuming and results in very poor performance. This has been verified in a limited set of experiments, where using $n=J^3$ in the definition of an r-local minimum was severely slower than using

$n=J$. Because of these two reasons, we adopted the definition of Swami and Gupta, which had the additional benefit of allowing us to study their work more closely. The remaining parameters of II are the same as in optimizing A spaces.

The following table summarizes our choices for the parameters of SA for query optimization in L spaces.

The initial temperature is chosen after some preliminary experimentation. Specifically, starting from a temperature that is equal to the cost of a random state, we double the temperature until it becomes high enough so that at least 40 % of all attempted moves are accepted with that temperature. The other parameters are self explanatory.

Finally, the parameters of 2PO remained the same as with the optimization of A spaces, except that the internal parameters of the two phases were modified based on their new values for II and SA for L spaces.

parameter	value
initial state S_0	random
initial temperature T_0	percentage of accepted moves > 40 %
frozen	percentage of accepted moves < 2 % and minS unchanged for 5 stages
equilibrium	J visited states in the current stage
next state	random neighbor
temperature reduction	$T_{new} = 0.975 * T_{old}$

Table 6.6: Implementation specific parameters for SA.

6.3.2. Algorithm behavior

First, we discuss how the II, SA, and 2PO algorithms should be expected to perform based on the shape of the cost function of spaces. Then we present several experimental results that support those expectations.

We concluded that the shape of L spaces is of type A2-B2 or A2-B1. This implies that II needs many local optimizations before it reaches one that is of low cost. In general, much effort is wasted because of local optimizations finishing at high local minima. Nevertheless, since the shape is of type A2 and not A3, after a reasonable number of local optimizations, some strategies of low cost must be visited, so the final output must be of good quality.

SA will waste much time among strategies in high cost in the beginning. As the probability for uphill moves decreases, it should eventually be able to escape the high cost local minima that it will encounter and converge to the 'well' bottom. Due to the risk of getting trapped in one of the many high cost local minima, however, its performance should not be very stable.

The behavior of 2PO depends on the length of its first phase, i.e., on how many local optimizations will be done before the simulated annealing phase starts. If the first phase involves few local optimizations, it is likely that the 'well' bottom will not be reached during that time. This implies that in the second phase, there is high chance that the algorithm will be trapped in a high cost local minimum and its final output will be unsatisfactory. On the other hand, if the first phase involves many local optimizations, the 'well' bottom will be reached during that time and will be adequately explored during the second phase. So, the final output should be better than that of II.

Below we present the results of the experiments that we conducted with the L space for ten 20-join and ten 40-join queries. Each algorithm was run five times on each query. Table 6.7 contains the average over all queries of the same size for both the best and the average result among the five runs that were performed in each case. All numbers represent scaled cost, i.e., for each query, this is the ratio of the actual cost over the cost of the best strategy found by any experiment with the query. The results are for catalog 'relcat2' and for the CM2-N cost model, which is modified CM2 with nested-loops as the only available join method. We used CM2-N for the same reasons

	best		average	
	20	40	20	40
2PO	1.00	1.22	1.67	6.25
II	1.07	1.57	1.46	2.17

(a) 10 local optimizations in the first phase of 2PO.

	best		average	
	20	40	20	40
2PO	1.00	1.07	1.17	1.42
II	1.21	1.37	1.45	1.89

(b) 50 local optimizations in the first phase of 2PO.

Table 6.7: Output quality of algorithms in the L space for catalog 'relcat2'.

that made us use CM1-M and CM1-N in Section 4.5. That is, we wanted the results not to be affected in favor of our claims by the Join method choice transformation rule (rule (1) in Section 2.2), since the more join methods there are, the more closely the shape of a space resembles a 'well'.

Based on the analysis of Chapter 5, the shape of the space searched by the algorithms is of type A2-B2 in this case. We present results for two implementations of 2PO, one in which the first phase has a small number of local optimizations (10), and another in which it has a larger number of local optimizations (50), and compare them to those of II. The first implementation is denoted by 2PO-10 and the second one by 2PO-50. II was given the same amount of time as 2PO-10 or as 2PO-50, depending on which 2PO it was compared to. SA was not run in this case because it had become obvious that it is always inferior to 2PO. The results are according to our expectations. When the best of five runs is considered, 2PO-10 is superior to II, because of the overall 'well' shape. As long as 2PO reaches the bottom in one of those runs, it gives better results. With respect to the average behavior, however, 2PO-10 is in general worse than II, indicating that its performance is not stable. This is because on the average, due to the A2-type shape, 10 local optimizations are not enough to reach the 'well' bottom. So, 2PO has poor behavior. On the other hand, when the first phase involves many local optimizations, the situation is reversed. 2PO-50 is superior to II in terms of both average and best performance, indicating that it always makes it to the 'well' bottom.

This sensitivity of the performance of 2PO when searching *L* spaces comes in real contrast to its overall domination when searching *A* spaces. The root of this difference is the fact that *A* spaces are of type A1 with most local minima having low cost, whereas *L* spaces are of type A2 in general, with many local minima having high cost.

We have also experimented with *L* spaces for the 'relcat4' catalog and the CM3 cost model. This represents the same experimental testbed used by Swami and Gupta in their study of *L* spaces. The results are rather interesting. In this case, because of the high variance in the values of the catalog parameters, the strategy cost distribution should be shifted extremely to the left (Section 5.3). Therefore, the shape should be of type A1-B1, i.e., the 'well' bottom should be very smooth. The results obtained in our experiments are again consistent with this shape type. Because of the characteristics of the strategy cost distribution, query optimization is rather easy in this case, and all three algorithms essentially produce the same output. Both 2PO and II reach the 'well' bottom very quickly, and beyond that point they improve very little. On the other hand, by its nature, SA spends enough time in the high cost region before finally converging to the 'well' bottom. An example of this behavior of the algorithms is shown in Figure 6.11. The specific profile is from a 40-join query. As usual, the y-axis represents scaled cost.

From the above discussion, the expected results based on the space shape analysis and the observed experimental results for *L* spaces with the 'relcat4' catalog and the CM3 cost model are in agreement. However, these results are slightly different with

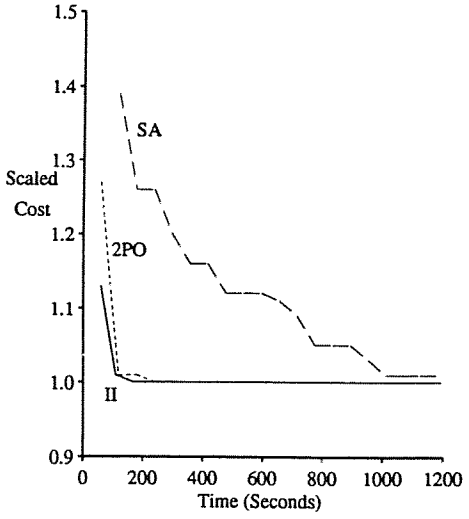


Figure 6.11: Minimum cost found over time.

those of the study of Swami and Gupta, which included II and SA alone [Swam88]. Specifically, in their experiments, the output quality of SA was worse than that of II. We believe that this difference is due to one specific aspect of their modeling of the algorithms. In particular, a time limit was used as one of the stopping criteria for SA. Because of the ability of the algorithm to make uphill moves and the high expense of each move (Proposition 6.1 and Proposition 6.3), SA needs ample time before the temperature decreases enough for the algorithm to be positioned in the 'well' bottom. This explains the difference in the results, since in the implementation of Swami and Gupta, it was very likely that the time limit was reached while the temperature was still

high, and therefore the best strategy visited at that time had high cost as well.

6.4. Comparison of the Two Strategy Spaces

In the previous sections, for a specific strategy space (A or L), we identified the algorithm that should be used for query optimization over that space. The main conclusion was that for L, the situation is not so clear-cut, with 2PO possibly being slightly better than II, especially when many local optimizations are attempted in the first phase. For A, on the other hand, 2PO is superior to the other algorithms across the board. In this section, we want to address the issue at the next higher level, i.e., given a choice, which of the two spaces should be used for optimization. The following two subsections identify some inherent properties of the two spaces that are helpful in drawing some conclusions in the third subsection.

6.4.1. Generated structure of spaces

In both A and L, the neighbors of any strategy S are determined by a set of transformation rules that can be applied to S (Section 2.2). However, among the strategies thus constructed, only those that are cross-product-free are real neighbors of S. To move from some strategy to one of its neighbors, all these algorithms described in this thesis blindly apply one of the appropriate transformation rules and then check whether it includes a cross-product or not to determine its validity as a neighbor. The set of strategies that can be generated by applying all transformation rules on a given strategy S, independent of whether they contain a cross product or not, is the set of *generated neighbors* of S, and their number is the *generated degree* of S. Clearly, the

degree of a strategy is less than or equal to its generated degree. The following two propositions shed some light on the generated degrees of strategies in the two spaces L and A. For consistency with Section 5.1, we again concentrate on the Swap transformation rule in L; including the 3-cycle rule is straightforward. We also focus on the structure of strategies ignoring join methods.

Proposition 6.1: In L, every strategy is of generated degree $\frac{J(J+1)}{2}$.

Proof: Consider a strategy in the L space of a query with J joins. For each pair of relations, the Swap rule can be applied. This accounts for $\frac{J(J+1)}{2}$ generated neighbors.

□

Proposition 6.2: In A, every strategy is of generated degree $3J-2$.

Proof: Consider a strategy in the A space of a query with J joins. For each join node, the join commutativity rule can be applied. This accounts for J neighbors. For each pair of a join node and its parent, exactly one of the join associativity rules and one of the join exchange rules are applicable. The specific rules that are applicable are determined by whether the child node is the left child of its parent or the right child. There are exactly $J-1$ such node pairs in the join processing tree, so these transformation rules account for $2(J-1)$ generated neighbors. Hence, there are $3J-2$ generated neighbors for each strategy.

□

Comparing the above results with Theorems 4.1, 5.1, and 5.2, a clear advantage of A over L emerges. For most queries, a large portion of the generated neighbors of a strategy in L are not part of the actual space. This becomes worse as we move from a star query to a string query. On the other hand, for all queries, 2/3s of the generated neighbors of a strategy in A belong to the space. The corresponding ratio in L is higher for star queries, but lower for most other queries. Moreover, it is independent of the number of joins in A, whereas in general it grows with more joins in L.

Clearly, the lower the ratio of the degree of a strategy over its generated degree is, the more time will be spent in trying to find a legal neighbor of it, and the slower any algorithm will be. Hence, one can conclude that, in general, the same amount of useful work requires more time when searching L than when searching A.

6.4.2. Incremental cost computation

When a move is attempted from a strategy S_1 to one of its neighbors S_2, the cost of the latter need not be recomputed from scratch. It is enough to subtract the cost of all operations in S_1 that were modified and add the cost of the operations that replaced them in S_2. The following two results shed some light on the cost computation of neighbors in the two spaces L and A.

Proposition 6.3: Consider a strategy in the L space for a query with J joins. The number of operations whose cost needs to be recomputed when transforming the strategy into one of its neighbors is bounded by J.

Proof: Consider the relation sequence that corresponds to a strategy. If the Swap rule is applied on the first and the last relation in the sequence, the cost of all J joins is modified and needs to be recomputed.

□

Proposition 6.4: Consider a strategy in the A space for a query with J joins. The number of operations whose cost needs to be recomputed when transforming the strategy into one of its neighbors is bounded by 2.

Proof: Any transformation rule affects at most two join nodes.

□

If merge-scan is one of the available join methods, due to the side effects of modifying the interesting order of a temporary result [Seli79], there can be transformations of a strategy in A that require a recomputation of the costs of up to J operations as well. Nevertheless, cost recomputation is in general much more time-consuming in L than in A (linear in the number of joins vs. constant). Hence, the same conclusion as in the previous section can be drawn: the same amount of useful work requires more time when searching L than when searching A.

6.4.3. Desirability of spaces

Space A has many more strategies than space L. Nevertheless, contrary to conventional wisdom, the former is easier to optimize than the latter. The shape of the cost function of A has a much more definite shape of a 'well' than that of L: type A1 vs. A2. Hence, algorithms like 2PO can take advantage of this and have a robust performance in optimization. Also, the results in the previous subsections show that searching itself is more efficient in A than in L. Finally, in most cases, the optimum

strategy in L is of higher cost than the global minimum in A. The combination of easier and more efficient optimization with the potential of a better quality output makes A the strategy space of choice. We believe that it should be the preferred one when optimizing large join queries.

Chapter 7

RANDOMIZED ALGORITHMS FOR SMALL JOIN QUERIES

Thus far, our study has concentrated on applying randomized optimization algorithms on optimizing large join queries. However, we have also observed that these algorithms perform well for small queries and are quite efficient. Since small queries have a relatively small state space, their performance has been satisfactory and very stable. In this chapter, we investigate how these algorithms (2PO especially) compare to the traditional, almost exhaustive, search algorithms that are used by most current systems for optimizing small join queries. In particular, as the main representative of the traditional algorithms, we use the one that was proposed for query optimization in System R [Astr76, Seli79]. For convenience, the algorithm is called *the System R algorithm* (SysR). Its performance is compared to that of randomized algorithms with respect to running time and quality of output.

The main trade-offs between the two types of algorithms are that, in general, randomized algorithms cannot guarantee to find the actual optimal strategy, but they can visit many states in a short time. On the other hand, an exhaustive search usually requires restricting the search space to reduce running time, which can degrade the quality of the solution. Our results shed some light into how both types of algorithms perform against each other with respect to these tradeoffs.

7.1. The System R Algorithm

SysR is based on dynamic programming and is applied on the L space only. It constructs the entire L space of a query by iteration on the number of relations joined, while carefully pruning the parts of the space that are known to be suboptimal. The general algorithm is described below.

(1) For each relation, all possible ways to access it, via all existing access paths and including the simple sequential scan, are obtained. These partial (single-relation) strategies are partitioned into groups based on the interesting order in which they produce their results. Also, one partition is formed by the partial strategies that produce their result in no interesting order. From each partition, the cheapest strategy is retained for further consideration. Moreover, if the cheapest strategy of the no-order partition is not cheaper than all other strategies, then nothing is retained from that partition.

(2) Assume that the cheapest strategies to join $i-1$ relations for each interesting order are known for all sets of $i-1$ relations that can be joined without the formation of

a cross product. On the ith iteration, for each such set, all possible ways to join one more relation with it without creating a cross product are evaluated. All generated partial strategies for each set of i relations are partitioned as in (1) and again only the cheapest of each partition is retained for further processing.

(3) The above step (2) is repeated until the $(J+1)$st iteration, when the best strategy for joining all $J+1$ relations in the query is found.

The SysR algorithm is guaranteed to find the optimal strategy for a query within the L space. It avoids enumerating all strategies in the space by being able to dynamically prune suboptimal parts of the space as partial strategies are generated. However, its memory requirements grow exponentially with query size since all viable partial strategies generated in one iteration must be stored to be used in the next one. The effect of this drawback on the performance of the algorithm is discussed in the following section. SysR was implemented in C as described above. We compare the performance of the SysR algorithm with that of the randomized algorithms in the following sections.

7.2. Query Optimization in the Space of Left-deep Strategies

In this section, we present the results of a performance evaluation of the randomized algorithms introduced in Chapter 6 and of SysR when optimizing small queries in the L space. The randomized algorithms were implemented slightly differently from the way described in Section 6.3. These differences are motivated and

explained in the following subsection.

7.2.1. Modification to the transformation rules

Initial experimentation with the randomized algorithms on the L space of small queries as defined in Section 2.2 revealed several performance problems. In particular, all randomized algorithms had relatively poor running times compared to SysR. The primary reason for this was the relatively high degree of strategies in the L space compared to the total size. For small queries, this becomes an important issue, since with high degree, much time is spent by II in finding downhill moves from states in the low cost area. Moreover, the generated degree is high as well, so all algorithms are affected negatively as discussed in Section 6.4. Again, although these issues exist in large queries as well, they become a major concern in small queries due to the tremendous decrease of the space size. To avoid the above problems and improve the performance of randomized algorithms, we used a different set of transformation rules to define the neighbors of strategies in L. Specifically, with 'a' being a join processing formula, and R and S being single relations, the set of transformation rules that we used is given below:

(1) *Join method choice:* $a \bowtie_{method_i} R \rightarrow a \bowtie_{method_j} R$

(2) *Join exchange:* $(a \bowtie R) \bowtie S \rightarrow (a \bowtie S) \bowtie R$

The join exchange rule allows to swap only adjacent relations in the strategy, i.e., in the left-deep tree. Note that the set of edges in the L space of a query formed by using

the above rules is a subset of that formed by the transformation rules defined in Section 2.2. Although the revised set of rules has a negative impact on the formation of a 'well' in the space, the forthcoming results show that this has not been significant, and that the space remains more or less a 'well'. In addition, with these transformation rules, randomized algorithms have shown significant improvement in their performance. Thus, our adoption of them has been most beneficial.

7.2.2. Parameter tuning

We have performed a set of experiments to tune the parameters of the 2PO algorithm for optimizing small queries (up to 15 joins) in the *L* space. Two parameters were varied that affect the algorithm's running time and output quality: the number of local optimizations in the II phase and the number of states visited in each stage of the SA phase. They have been the focus of our tuning effort, because their original values resulted in unnecessarily high running times. The remaining parameters were left as in the canonical implementation described in Section 6.2. For a query with *J* joins, the combinations of those two parameters that were tested are:

Algorithm	# local optimizations II phase	# states visited in each stage SA phase
2PO-*L*1	10	4**J*
2PO-*L*2	10	8**J*
2PO-*L*3	100	4**J*

We experimented with all three versions of the 2PO algorithm. Our experiments involved ten different tree queries for each size of 5, 10, and 15 joins. We used the CM1 cost model and the 'relcat1', 'relcat2', and 'relcat3' catalogs. Each version of 2PO was run ten times for each query and catalog. Figure 7.1 shows the average, over all queries of each size, of the average output strategy cost over all ten runs of each query. As usual, for each query the cost is scaled over the best cost found by all runs of all versions of 2PO for that query. Clearly 2PO-*L*3, which uses 100 local optimizations for the II phase, is superior to the other two versions. On the other hand, spending more time in each stage of SA makes no difference in the output quality (2PO-*L*1 and 2PO-*L*2 are very similar).

These observations are direct consequencies of the space shape. Despite the modified transformation rules that we used for *L* spaces of small queries, the shape remains of type A2-B2/B1. Due to local minima at high costs, performing more local optimizations in the II phase of 2PO improves performance significantly (2PO-*L*3 wins). Similar phenomena were observed with large queries as well (Section 6.3.2). On the other hand, due to the small query size (and space) the 'well' bottom is relatively smooth, and not much time needs to be spent in the SA phase (2PO-*L*1 and 2PO-*L*2 give similar cost-related performance).

The corresponding running (CPU) time of all three versions of 2PO is shown in Figure 7.2. Interestingly, 2PO-*L*2 and 2PO-*L*3 are very similar in that respect, both being about twice as expensive as 2PO-*L*1. Based on these results and those on the

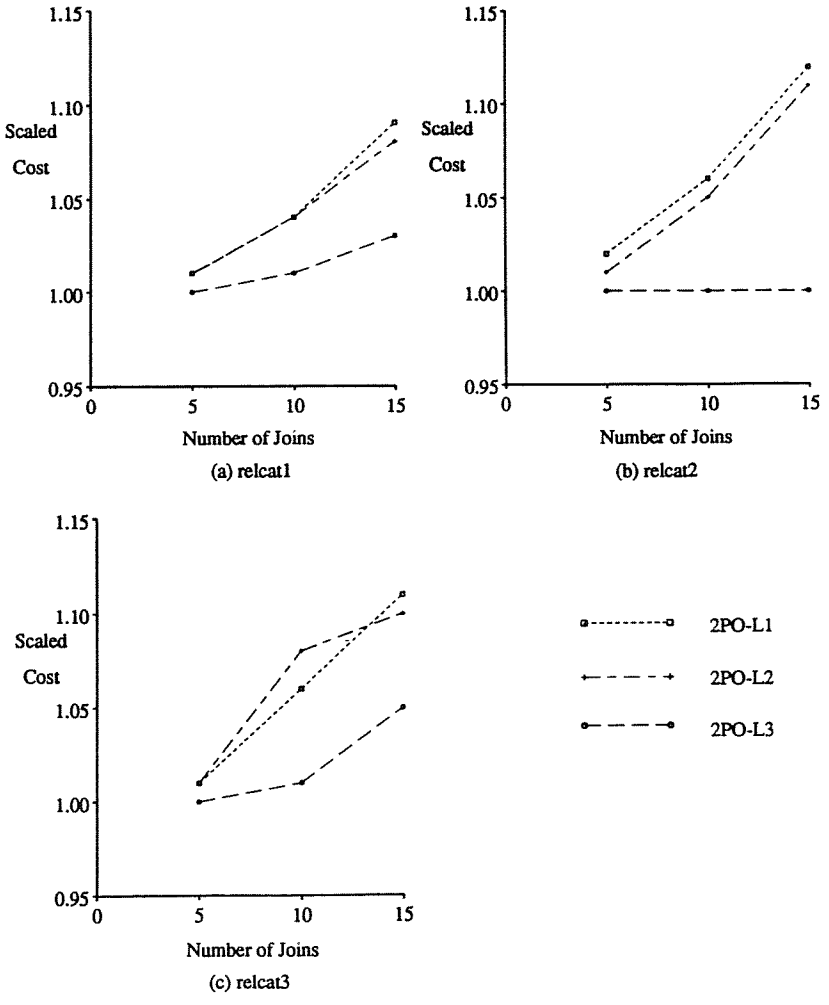


Figure 7.1: Average scaled cost of the output strategy of various versions of 2PO

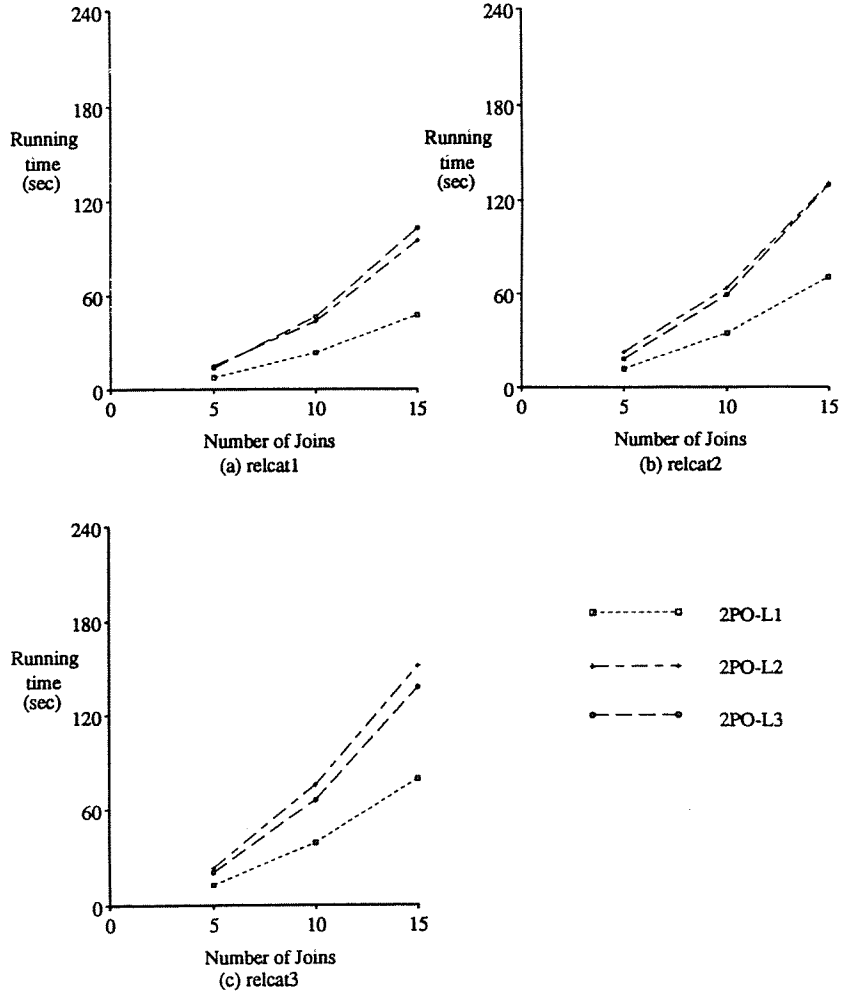


Figure 7.2: Average running time of various versions of 2PO

output quality, it is clear that 2PO-*L*3 is the version of choice, and is the one that was used in our experiments that are discussed in the following section.

7.2.3. Performance of algorithms

In this subsection, we compare 2PO, II, and SysR with respect to output quality, i.e., cost of output strategy. For each problem instance, all algorithms were run ten times. Each run of II was given as much time as the average time taken by a run of 2PO, i.e., 2PO-*L*3, for the same problem instance. We were only able to perform experiments with up to 15-join queries, because beyond that point SysR collapsed due to its memory requirements. In fact, we experimented with more than ten 15-join queries to obtain a set of ten on which SysR could finish. So, at least on the high end, our results are slightly biased in favor of SysR, since they come from relatively easier queries.

The average cost of the output strategies produced by the algorithms over all runs for each query, over all queries of each size as a function of query size, is shown in Figures 7.3 and 7.4 for the CM1 and CM2 cost models respectively. Three different diagrams are shown for the three catalogs, 'relcat1', 'relcat2', and 'relcat3'. The cost is scaled over the output cost found by SysR for each problem instance. In general, we observed no significant qualitative difference in the results between CM1 and CM2. In all cases, 2PO was better than II and the difference increased with query size. This shows that even if 100 local optimizations are performed in the II phase, 2PO is still better than II, i.e., it is still more effective to search the area of the 'well' bottom using

SA after the II phase than to perform more local optimizations. Compared to SysR, we observe that the cost of the output of 2PO is always within 5% of the global minimum, which is found by SysR. Although this difference increases with query size, being so close is rather remarkable and demonstrates the effectiveness of 2PO. Clearly, this can only be fully substantiated after comparing the running times of the algorithms, but this discussion is postponed until Section 7.3.2.

7.3. Query Optimization in the Space of Deep and Bushy Strategies

In this section, we present the results of another set of experiments that we performed on the same set of queries of the previous section but with the corresponding *A* spaces. This gave us the ability to compare algorithms and spaces together. Having demonstrated the superiority of 2PO over II, in all previous sections, we only experimented with 2PO in this case. Initially, we performed experiments to tune its parameters for optimizing small queries in the *A* space, similar to the ones that we performed for the *L* space. We again concentrated on the number of local optimizations in the II phase and the number of states visited in each stage of the SA phase. In comparison to the experiments with *L* spaces, in this case, performing more local optimizations in the II phase hardly improved the output quality. This was to be expected, since *A* spaces do not present the problem of having local minima at high costs (Section 4.4). Hence, we used 10 local optimizations for the II phase. On the other hand, due to the increased space size, more searching was beneficial, so we used

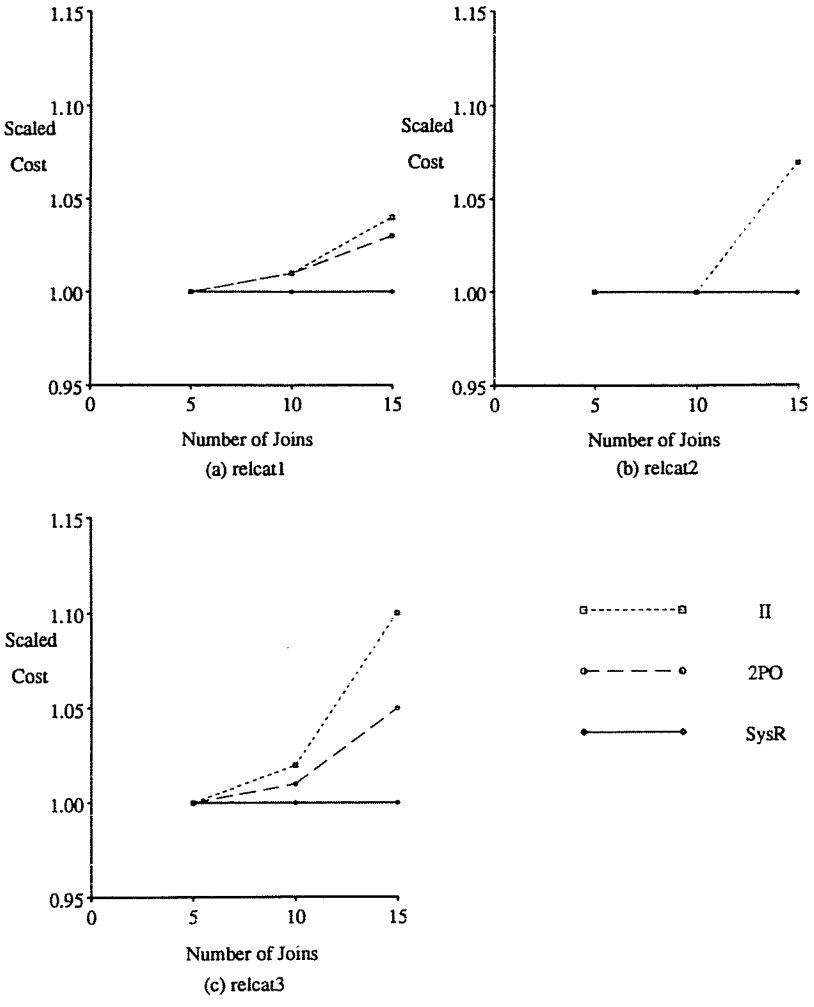


Figure 7.3: Average scaled cost of the output strategy of 2PO, II and SysR under the cost model CM1

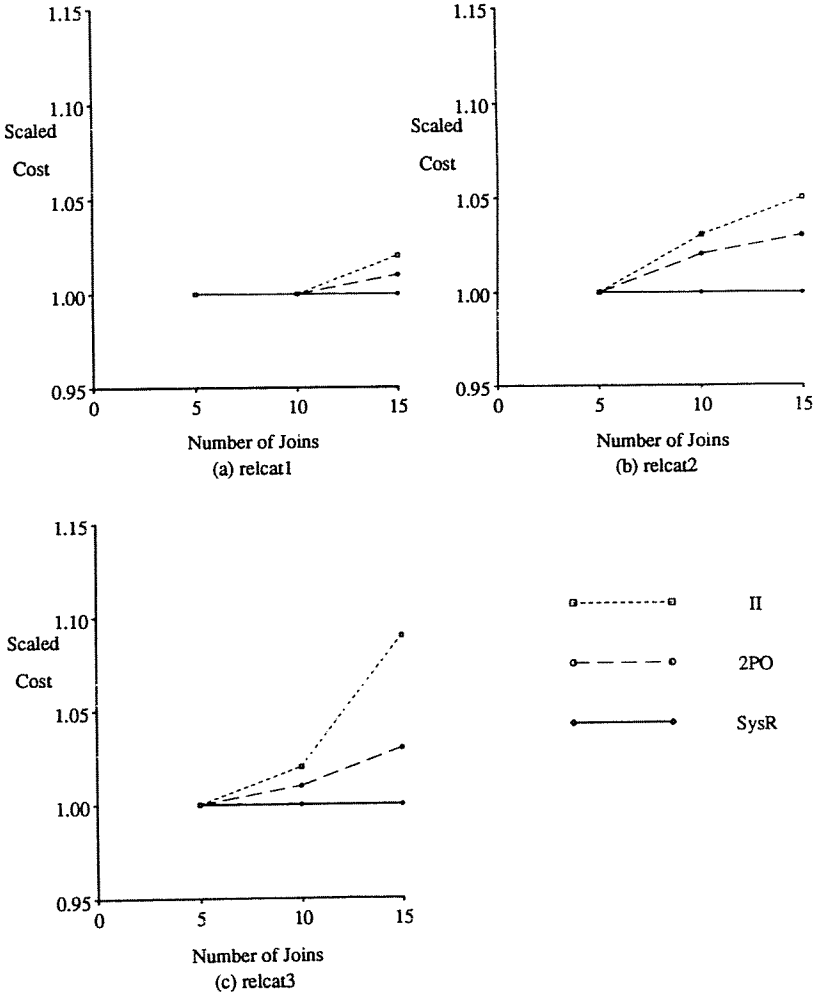


Figure 7.4: Average scaled cost of the output strategy of 2PO, II and SysR under CM2

8*J as the number of states visited in each stage of the SA phase. We use '2PO-A' to denote this version of 2PO, whereas we use '2PO-L' to denote the version of 2PO that was used in the experiments with the L space in the previous section.

7.3.1. Performance of algorithms

This subsection contains results that are helpful in comparing 2PO-A, 2PO-L, and SysR. Similarly to the other algorithms, for each problem instance, 2PO-A was run ten times. The average cost of the output strategies produced by the algorithms over all runs for each query, over all queries of each size as a function of query size is shown in Figures 7.5 and 7.6 for the CM1 and CM2 cost models respectively. The cost is again scaled over the cost of the output of SysR for each problem instance. In general, we again observed no significant qualitative difference in the results between CM1 and CM2. In most cases, the improvement in output quality of 2PO-A over SysR and 2PO-L resulting from the inclusion of bushy plans in the search space is rather clear. The improvement becomes more significant as the query size grows.

Clearly, the above improvement in output quality by 2PO-A is possible when there are bushy strategies of lower costs than the optimal strategies in the corresponding L spaces. Figures 7.5 and 7.6 have some very interesting results in that respect. Independent of the cost models, A spaces with 'relcat1' and 'relcat3' - the easier to optimize catalogs - do contain bushy strategies that are of lower cost than the optimum left-deep strategy. On the other hand, 2PO had trouble finding better strategies in A spaces with 'relcat2'. In fact, this was true for almost all queries, so

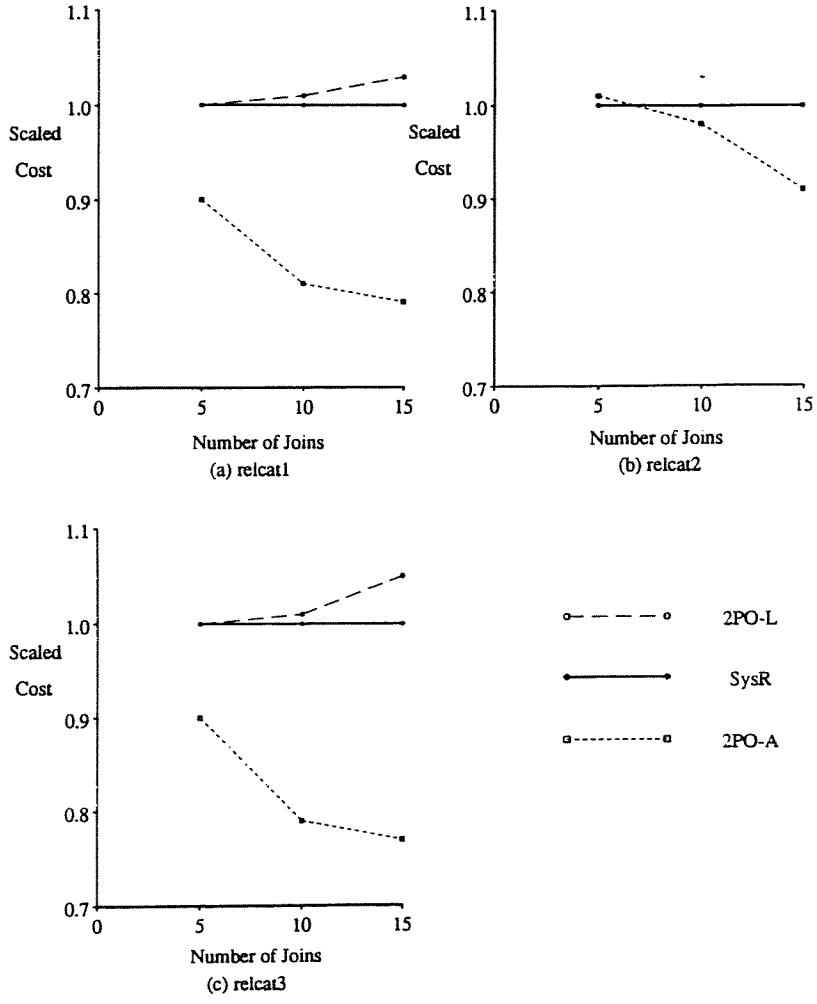


Figure 7.5: Average scaled cost of the output strategy of 2PO-L, 2PO-A, and SysR under the cost model CM1

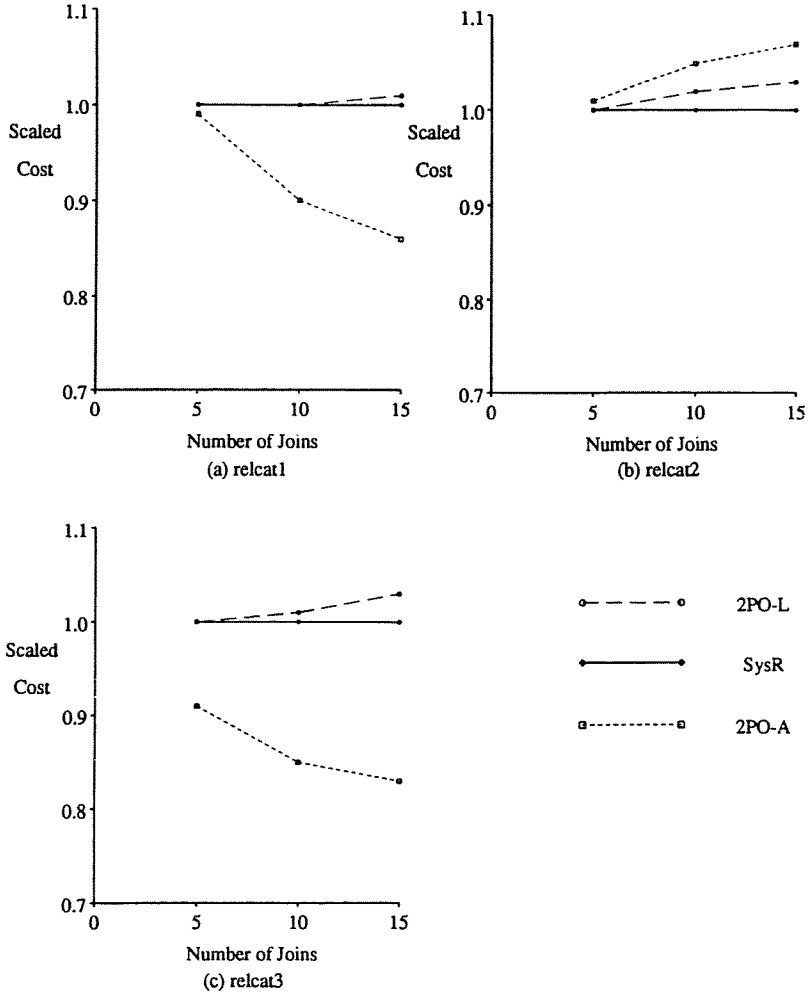


Figure 7.6: Average scaled cost of the output strategy of 2PO-*L*, 2PO-*A*, and SysR under the cost model CM2

Figure 7.6b is not an artifact of averaging but truly reflects the nonexistence of bushy strategies of low cost. Even so, however, 2PO-*A* was never more than a few percent above the optimum left-deep strategy. Hence, its overall superiority becomes rather evident from the above results.

7.3.2. Optimization time

The average running (CPU) time of 2PO-*A*, 2PO-*L*, and SysR as a function of query size is shown in Figures 7.7 and 7.8 for the CM1 and CM2 cost models respectively. Clearly, when optimizing up to 10 join queries, SysR takes much less time than the randomized algorithms for both cost models. However, for 15-join queries, there are so many partial strategies that must be generated by SysR that its running time becomes much worse than either version of 2PO. Interestingly, the relative difference among the algorithms is less favorable to SysR in CM2 than in CM1, indicating that the addition of join methods has a larger impact on SysR than 2PO. The overall conclusion from the results is that for really small queries SysR is very effective and should be the algorithm of the choice. For larger queries, however, even within the range of queries supported by current systems (at least on the high end), 2PO is much more efficient and should be used instead. Between the two versions of 2PO, the running time of 2PO-*A* is less than that of 2PO-*L* for both cost models. The difference, however, is relatively small compared to that between these two algorithms and SysR, although it increases with query size. It is also interesting to observe that the difference between 2PO-*L* and 2PO-*A* becomes very small with

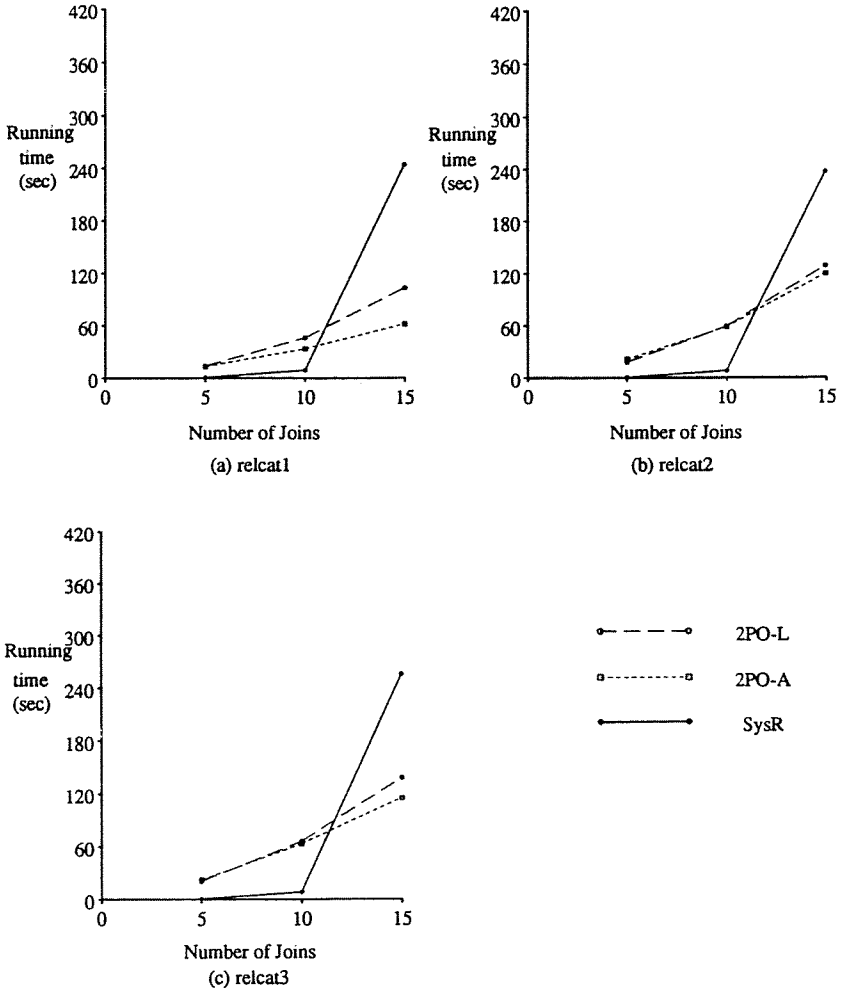


Figure 7.7: Average running time of 2PO-*L*, 2PO-*A*, and SysR under the cost model CM1

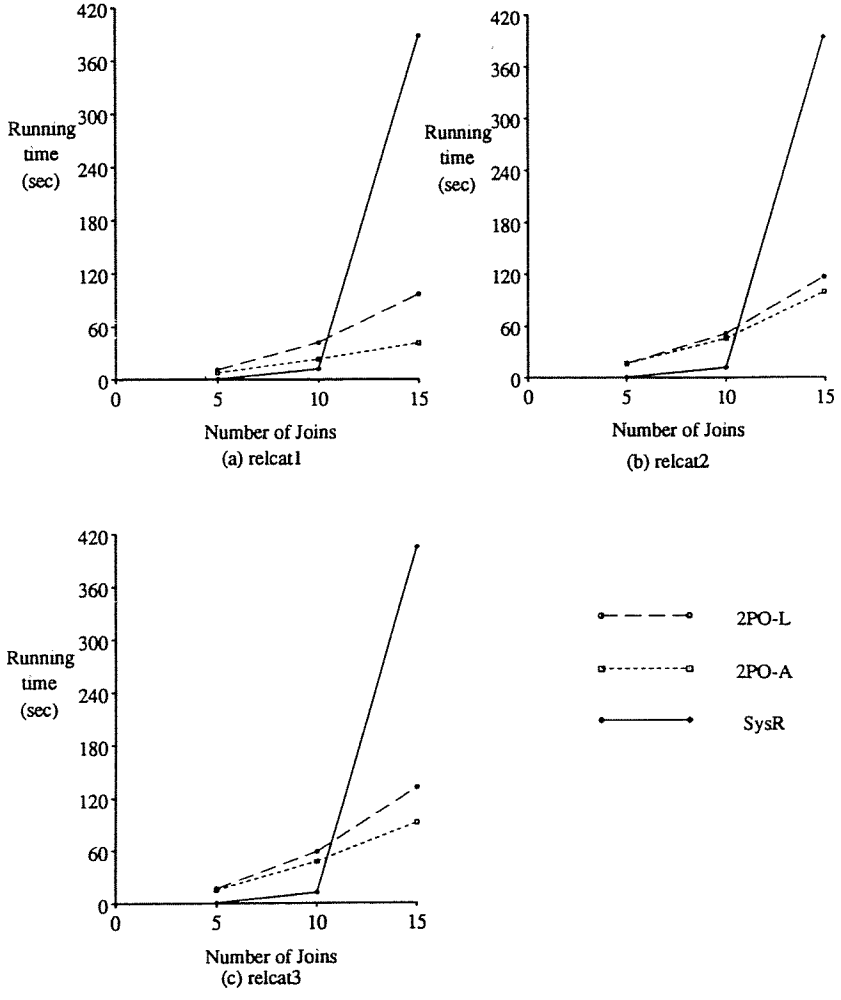


Figure 7.8: Average running time of 2PO-*L*, 2PO-*A*, and SysR under the cost model CM2

'relcat2', which was the case in which in general 2PO-*A* could not improve on the output cost of 2PO-*L*. This shows that the performance of 2PO-*A* becomes similar to that of 2PO-*L* with respect to both output quality and running time when there are not many bushy strategies in the low cost area. On the other hand, when there are many such strategies in the low cost area, 2PO-*A* produces a lower output strategy in less time than 2PO-*L*.

7.4. Summary

We tested the performance of randomized algorithms for optimizing small join queries and compared their output quality and running time with those of the System R algorithm. The results can lead to several conclusions with respect to several issues. First, they verify the developed theory on the form of the space shape for both *A* and *L* spaces and they establish the superiority of 2PO over II once more. Second, they come in support of current optimization technology for small queries, as it is represented by the System R algorithm. Although it performs an almost exhaustive search, SysR is much faster than randomized algorithms on very small queries. On the other hand, beyond a certain point the running time of SysR increases dramatically. In that case, randomized algorithms, as represented by 2PO, are superior, since they are very close to the global minimum in terms of output quality while needing much less time than SysR. Third, they establish once again the superiority of the *A* space over the *L* space for query optimization by randomized algorithms. In most cases, for the same query, 2PO running in the *A* space produces strategies of lower cost and takes less time than

2PO running in the *L* space.

The above analysis suggests that future database systems may need to support multiple query optimization algorithms, so that based on the characteristics of each query, a simple "meta-optimization" algorithm can choose the appropriate one among them. Given the above results, a very simplified form of this meta-optimization algorithm would choose SysR in the *L* space for small queries (up to approximately 10 joins) and 2PO in the *A* space for larger queries.

Chapter 8

RELATED WORK

Query optimization has been an active area of research ever since the beginning of the development of relational DBMSs. Various query optimization algorithms have been developed as part of the research projects of INGRES [Ston76, Wong76, Kooi80] and System R [Astr76, Blas76, Seli79, Mack86a]. Theoretical approaches to query optimization have also been employed, attempting to apply general theorems to help the query optimizer in its task [Aho79, Rose80]. Query optimization algorithms in a distributed DBMS environment have received considerable attention as well [Epst78, Bern81, Ceri84, Mack86b]. Multiple query optimization has also been studied in the past [Gran81, Sell88]. Most of the existing work on query optimization has focused on optimizing conjunctive queries. The work of Muralikrishna [Mura88] on optimizing disjunctive queries is an exception that verifies the rule. Good surveys on query optimization and other related issues can be found in the article by Jarke and Koch [Jark84] and the book by Kim, Reiner, and Batory [Kim86].

Most of the techniques proposed in the literature have a limited scope and cannot be applied to large queries. For example, System R and commercial Ingres employ algorithms that essentially perform an exhaustive search (Section 7.1). (This is adequate in their case because they do not allow large queries, e.g., > 15 joins, in the system.) In this section, we briefly describe the query optimization techniques that have some commonalities with our approach. These techniques can be broadly classified into three categories: rule-based query optimization, non-randomized query optimization for large queries, and randomized query optimization.

8.1. Rule-based Query Optimization

Extensible DBMSs require a modular design for the query optimizer, which emphasizes separating the strategy representation, the cost function, and the search algorithm. In these extensible query optimizers, alternative strategies are examined using a set of rules for flexibility and extensibility. One can distinguish two kinds of such rules. As in randomized algorithms, Freytag [Frey87] and Graefe/DeWitt [Grae87a, Grae87b] proposed using strategy *transformation* rules to explore alternative strategies. Search algorithms based on this approach behave similarly to the randomized algorithms discussed in this thesis. Abstractly, strategies are nodes in a graph, edges are defined based on the transformation rules, and the algorithms traverse the graph by transforming strategies. On the other, hand Lohman [Lohm88] suggested using grammar-like *generation* rules to construct alternative strategies. Search algorithms based on this approach behave similarly to the System R algorithm. They

successively construct increasingly more complex strategies, in each step applying the generation rules to the partial strategies constructed in the previous step. In the last step, strategies are constructed that can answer the given query, which are essentially all the nodes of the graph searched by the transformation rule algorithms. The lowest cost strategy among them is chosen by direct comparison. We compare these rule-based extensible optimization approaches with our approach in the context of large relational query optimization.

Lee et al. [Lee88] describe a design for strategy generation in an optimizer by interpreting Lohman's strategy alternative rules. This has been employed in the Starburst extensible database system [Haas89, Hasa88], where all strategies are constructed in a bottom-up fashion. Ono and Lohman [Ono90] describe the specific algorithm used to enumerate all the join orders that are equivalent to a query, parameterizing the types of strategies to be considered (e.g., whether or not bushy trees should be considered). This process is functionally equivalent to an exhaustive search and cannot be used for query optimization when the search space is very large. It is suggested in [Lee88] that the search can be guided using some priorities on the rules when the alternative strategies are generated. However, no specific technique is given that can be used effectively for a large search space.

Graefe [Grae87b] designed the query optimizer generator for the EXODUS extensible database system [Care86, Grae87c]. He proposed a generic search strategy that can be used in any generated optimizer. As in randomized algorithms, operator

trees are used to represent strategies. The input to the optimizer generator includes algebraic transformation rules of the operator trees. Each algebraic transformation rule is associated with an expected cost improvement factor that is used to estimate the cost after applying the transformation. The search starts at some initial operator tree and is directed by the cost improvement factors so that, at any point, the transformation rule that promises the most cost improvement is applied.

The values of cost improvement factors are determined by the EXODUS optimizer based on experience from past query optimizations. In particular, the cost improvement factor of a transformation rules is equal to the geometric average of the cost improvement ratios obtained by using the rule in the past. However, cost improvements obtained by applying a transformation rule may vary significantly among different queries, depending on the characteristics of each query and its optimal operator tree shape. Hence, the effectiveness of the above learning method is rather questionable. A key advantage of the discussed randomized algorithms over the EXODUS search strategy is that the former decide on the usefulness of a transformation rule at run time, by first applying and then accepting or rejecting the move depending on details of the move itself. Thus, the characteristics of the specific query, the part of the space explored at that time, and the current phase of the search are taken into account, resulting in a much more "educated" evaluation of the usefulness of a rule than in the EXODUS optimizer.

During the search, the EXODUS algorithm maintains all relevant information about the explored strategies, whereas randomized algorithms maintain only the current strategy. By comparing every newly generated strategy against formerly visited strategies, the EXODUS optimizer enforces that each strategy is visited at most once, whereas the randomized algorithms allow a strategy to be visited multiple times. Although the former approach may be reasonable for small queries, as query size grows, the probability of visiting a strategy more than once becomes very small anyway. Randomized algorithms can thus visit more strategies in a given time because they do not have the overhead associated with maintaining and searching all the visited strategies. Moreover, although in the EXODUS approach the previously visited strategies are stored in a tree structure so that they share common subtrees (common partial strategies), the memory requirements for keeping all relevant information become very large quickly, especially for large queries. These memory requirements limit the applicability of the EXODUS search algorithm for large query optimization and are the second main disadvantage of this approach compared to randomized algorithms.

8.2. Non-randomized Large Query Optimization Algorithms

Ibaraki and Kameda proposed a heuristic algorithm for finding an optimal nesting order of joins when a *nested loop N-ary join method* is used [Ibar84]. In this method, the N base relations are joined using pipelining without materializing any temporary

relation. Strategies using this method correspond to left-deep trees. Ibaraki and Kameda derived a cost formula for the expected number of page accesses for a particular nested loop N-ary join method. They showed that minimization of the expected number of page accesses for that problem is NP-complete. They proposed using an approximation of the cost formula and showed that for tree queries, the approximate formula satisfies the ASI property (Section 3.3) for the strategies with a given relation being the first to join. Based on this property they were able to devise an $O(N^2 \log N)$ algorithm to find the optimal left-deep strategy for a tree query for the specific join method and associated cost function.

Krishnamurthy, Boral, and Zaniolo [Kris86] generalized the above optimization algorithm to include several additional join methods. Based on the specific form of the cost formula for all included join methods, they devised an algorithm that avoids much duplicate work that was done by the Ibaraki and Kameda algorithm and thus runs in $O(N^2)$ time.

Both algorithms above [Ibar84, Kris86] rely on a specific form of the cost formulas for the join methods to derive a good query optimization algorithm for the L space. However, as we discussed in Section 4.2, there are join methods whose cost formulas are not of the required form, e.g., merge-scan. Hence the applicability of both algorithms is limited.

Using a completely different approach, Yoo and Lafortune [Yoo89] succeeded in efficiently applying the A^* algorithm [Wins84] to optimize semijoin sequences for

distributed queries. A^* is a branch and bound algorithm and it requires a tight lower bound on strategy costs to prune partial plans effectively. Applying the A^* algorithm on other query types for a centralized database appears to be very hard because of the tight lower bound requirement for the algorithm to work well [Lohm89].

8.3. Randomized Query Optimization Algorithms

Randomized optimization algorithms such as Simulated Annealing and Iterative Improvement have been shown to be effective in many hard combinatorial optimization problems, including query optimization. Many theoretical investigations have been performed on the behavior of the Simulated Annealing algorithm [Rome84, Rome85, Sasa88]. Experimental studies have been performed as well, applying Simulated Annealing and Iterative Improvement to more traditional optimization problems such as graph partitioning [John87].

Ioannidis and Wong first applied the Simulated Annealing algorithm in query optimization [Ioan87]. They defined an algebraic model that can be used to represent strategies of recursive queries. Based on the algebraic properties of the model, they defined transformation rules and applied Simulated Annealing to the optimization of such recursive queries. They performed a small set of experiments and found the performance of Simulated Annealing to be encouraging. However, the experiments that they performed were preliminary, and they were not directed towards relational query optimization.

Swami and Gupta applied Simulated Annealing and Iterative Improvement to large join query optimization [Swam88]. Their study is the closest one to the work presented in this thesis and was discussed in more detail in Section 6.4. We would only like to add that a follow-up on their work by Swami [Swam89] investigated the use of heuristics in conjunction with II and SA and discussed the performance of various algorithm-heuristic combinations.

The work described in this thesis has established that among all proposed randomized algorithms, 2PO is the one of choice for join query optimization. It is part of our future plans, discussed in the next chapter, to investigate its effectiveness in other situations.

Chapter 9

SUMMARY AND FUTURE RESEARCH DIRECTIONS

9.1. Summary

When the strategy space for query optimization is too large to perform an exhaustive search, we need to use other efficient algorithms. Our work has shown that this need can be fulfilled by several known randomized combinatorial optimization algorithms, SA and II, at least for optimizing select-project-join queries in the space that includes deep and bushy trees and also in the space that includes left-deep trees only. Several analytical and experimental results on these two strategy spaces have shed some light on the shape of their cost function. In particular, it has become evident that both spaces essentially form a 'well', but of a distinctly different quality.

We have introduced the 2PO algorithm that takes advantage of the cost function shape and almost always dominates the other randomized algorithms in terms of both

output quality and running time. To achieve a more stable behavior and obtain output of higher quality for large join queries, it is better to run 2PO several times and choose the lowest cost output among them (as opposed to giving more time to a single run by changing its parameters). This is because a single run cannot cover the entire 'well' bottom area. A higher initial temperature (implying more time) does not solve the problem because the search tends to escape to the high cost area. Hence, multiple runs appears to be the best approach. The number of runs are expected to increase with the query size, but five runs seems to be enough for up to quite large queries.

In addition, for small join queries, when 2PO is run on the space of deep and bushy trees, most often it produces output of better quality than the System R algorithm. Although, the running time of SysR is lower than that of 2PO for very small queries, beyond the point of about 10 joins the SysR running time becomes higher and continues to grow exponentially. Hence, 2PO is quite effective for small join query optimization as well.

A final comment on the merits of 2PO is that due to its generic randomized algorithm nature, 2PO is modular, flexible, and simple to implement; it also requires minimal memory. Given all of its advantages, we believe that 2PO is the most promising candidate that has been proposed until now for optimizing large join queries.

The obtained results have also allowed a comparison between the A and L spaces, which has led to the conclusion that optimization with the space of both deep and

bushy trees is easier than with the space of left-deep trees only. Since the former contains a superset of the strategies of the latter, one expects that it would produce output of better quality as well. Hence, combining these two facts indicates that, at least with the transformation rules used in this work, the larger space should be the one used for randomized query optimization.

Finally, our investigation of the shape of the cost function of the spaces of interest to query optimization has led into several results that may have much wider applicability. Specifically, we have demonstrated that the cost distribution of local minima in an arbitrary space is affected by the cost distribution of all states in the space, by the degree of the states, and the cost difference of neighboring states. Also, the connection cost between local minima is affected again by the cost distribution of all states in the space and, indirectly, by the degree of the states, but also by the specific form of the cost function. These results have served their purpose well in analyzing the particular spaces of interest to us, but need further study before the extent of their applicability can be assessed.

9.2. Future Research Directions

Our plans for future work include extending the results presented in this thesis in two directions. First, we want to look into the optimization of other types of queries, both of the relational and other data models. Second, we want to look into other domains where optimization problems arise and test the validity of our generic results on the shapes of cost functions. The goal of both efforts is to examine whether a 'well'

is formed or not in the corresponding state spaces and thus to test the applicability of algorithms like 2PO in those cases. Regarding this plan, we have only done some preliminary work on optimizing relational queries of types other than those considered here (star and tree). While we have shown that the randomized optimization algorithms (especially 2PO) perform well for tree and star queries, which are the most common type of relational queries, they also can be applied to other types of query optimization.

One such query type is cyclic queries. We have performed a preliminary set of experiments on cyclic queries in the *A* space with the CM1 cost model and the 'relcat1' catalog. The results are very similar to those of tree queries. 2PO performs better than the other randomized algorithms. This is to be expected, since cyclic queries generate spaces with higher degree of strategies and therefore the space is expected to have an even more definite shape of a 'well'. However, cyclic queries take more time to optimize than tree queries. This is partly because the spaces tend to be larger, and is also because it appears that low cost strategies tend to be left-deep having nested-loops as the method for most joins. Therefore, much time is spent on sort order propagation because sort order changes tend to be propagated farther. We plan to complete the experiments for cyclic queries and investigate how the space shape of these queries changes. We will also investigate how sort order propagation affects the output quality and whether or not heuristics can be used to minimize that effort.

We have also made some progress in extending our approach to optimizing queries including unions. The union operator makes random state generation very difficult [Jerr86, Sinc89]. For II, an efficient way to generate a random state is crucial, whereas for SA and 2PO it is much less so. We plan to look into this problem further, but given its difficulty, II will most likely not be used for optimization of such queries.

When queries with union are considered, we need to represent a state in a more general form than for select-project-join queries to be able to express common subexpressions. In particular, an acyclic directed operator graph will be used to represent a state instead of an operator tree. In addition to the transformation rules discussed in Section 2.2 for the *A* space, transformation rules representing union commutativity, union associativity, and distributivity of join over unions will be used to define state neighbors. Hence, the complete set of transformation rules will be as follows.

(1) $a \bowtie_{method\,i} b \rightarrow a \bowtie_{method\,j} b$

(2) $a \bowtie b \rightarrow b \bowtie a$

(3) $(a \bowtie b) \bowtie c \leftrightarrow a \bowtie (b \bowtie c)$

(4) $(a \bowtie b) \bowtie c \rightarrow (a \bowtie c) \bowtie b$

(5) $a \bowtie (b \bowtie c) \rightarrow b \bowtie (a \bowtie c)$

(6) $a \cup b \rightarrow b \cup a$

(7) $(a \cup b) \cup c \leftrightarrow a \cup (b \cup c)$

(8) $(a \cup b) \bowtie c \leftrightarrow (a \bowtie c) \cup (b \bowtie c)$

(9) $a \bowtie (b \cup c) \leftrightarrow (a \bowtie b) \cup (a \bowtie c)$

Defining the space as above, we plan to study the applicability and performance of randomized optimization algorithms for queries involving unions in the same way that we did for join queries.

APPENDIX

We summarize here the cost formulas that we used for the three cost models CM1, CM2, and CM3. The necessary parameters are listed in Table A.1. The CPU time for various operations have been adopted from elsewhere [Shap86]. We assume a join of two relations R and S on the attribute A of both relations.

Parameter	Meaning
$\|R\|$	size of R in pages
$\{R\}$	size of R in tuples
v(A,R)	number of unique values in an attribute A of R
$P(I_R)$	size of index I of R in pages
$lp(I_R)$	number of leaf pages in a B-tree index I of R
$d(I_R)$	fanout of a B-tree index I of R
W	number of buffers
comp	time to compare keys (3 micro seconds)
hash	time to hash a key (9 micro seconds)
move	time to move a tuple (20 micro seconds)
F	fudge factor for hash overflows (1.2)

Table A.1: Parameters used in the cost formulas

The I/O cost in pages for nested-loops with a join predicate R.A=S.A when R is the outer relation for CM1 is summarized in Table A.2.

S.A index	I/O cost (pages)
Heap	$\|R\| + \{R\}\|S\| + \|R \bowtie S\|$
Hash(prim)	$\|R\| + \{R\}\|S\|/v(A,S) + \|R \bowtie S\|$
Hash(sec)	$\|R\| + \{R\}(P(I_S)/v(A,S)+\{S\}/v(A,S)) + \|R \bowtie S\|$
B-tree(prim)	$\|R\| + \{R\}(d(I_S) + \|S\|/v(A,S)) + \|R \bowtie S\|$
B-tree(sec)	$\|R\| + \{R\}(d(I_S) + lp(I_S)/v(A,S)+\{S\}/v(A,S)) + \|R \bowtie S\|$

Table A.2: I/O cost for nested-loops

The I/O cost for merge-scan with a join predicate R.A = S.A for CM1 is summarized in TableA.3.

R.A \ S.A	Heap	B-tree(prim)	B-tree(sec)
Heap	$2\|R\|(1+\log_2\|R\|)$ $+2\|S\|(1+\log_2\|S\|)$ $+\|R\|+\|S\|+\|R \bowtie S\|$	$2\|R\|(1+\log_2\|R\|)$ $+\|R\|+\|S\|+\|R \bowtie S\|$	$2\|R\|(1+\log_2\|R\|)$ $d(I_S)+lp(I_S)+\{S\}$ $+\|R\|+\|R \bowtie S\|$
B-tree(prim)	$2\|S\|(1+\log_2\|S\|)$ $+\|R\|+\|S\|+\|R \bowtie S\|$	$\|R\|+\|S\|+\|R \bowtie S\|$	$d(I_S)+lp(I_S)+\{S\}$ $+\|R\|+\|R \bowtie S\|$
B-tree(sec)	$d(I_R)+lp(I_R)+\{R\}$ $+2\|S\|(1+\log_2\|S\|)$ $+\|S\|+\|R \bowtie S\|$	$d(I_R)+lp(I_R)+\{R\}$ $+\|S\|+\|R \bowtie S\|$	$d(I_R)+lp(I_R)+\{R\}$ $d(I_S)+lp(I_S)+\{S\}$ $+\|R \bowtie S\|$

Table A.3: I/O cost for merge-scan

The cost formulas for hash-join for CM2 is summarized in Table A.4 [Shap86]. They assume that $|R|<|S|$ and use the following new parameters:

$$A=\left\lceil \frac{|R|*F}{W} \right\rceil$$

$$B=\left\lceil \frac{|R|*F-W}{W-1} \right\rceil$$

$$q=\frac{W-B}{|R|}$$

Memory size	Algorithm	I/O cost (pages)
$W \geq \sqrt{F\,|R|}$	Hybrid hash-join	$2(1-q)(|R|+|S|)$
$W < \sqrt{F\,|R|}$	Simple hash-join	$2(A-1)(|R|+|S|)-A(A-1)\frac{W}{F}(1+\frac{|S|}{|R|}$

Table A.4: I/O cost for hash join

In CM2, for nested-loops the cost formula in Table A.2 was slightly modified to incorporate pipelining. The term |R| was not added since the outer relation of the join was not read if R was not a base relation. Also for all join methods, |R ⋈ S| was not added either if the next join used nested-loops and had R ⋈ S as the outer relation since the result relation was not written. In addition, for merge-scan in CM2, the formula in Table A.3 was slightly modified, using W as the base of log, since with additional buffers a W-way merge sort was assumed.

For CM3, the formula for the cpu cost of hash-join that was used assumed that {R}<{S} and is given below [Shap86]:

({R}+{S}) *hash* + {R} *move* + {S} *comp* *F* .

REFERENCES

[Aho79]
A. Aho, Y. Sagiv, and J. Ullman, "Equivalences Among Relational Expressions," *SIAM Journal on Computing* **8**(2) pp. 218-246 (May 1979).

[Astr76]
M. Astrahan et al., "System R: Relational Approach to Database Management," *ACM Transactions on Database Systems*, pp. 97-137 (June 1976).

[Bane87]
J. Banerjee et al., "Data Model Issues for Object Oriented Applications," *ACM Trans. on Office Information Systems* **5**(1) pp. 3-26 (January 1987).

[Bato88]
D. Batory et al., "GENESIS: An Extensible Database Management System," *IEEE Transactions on Software Engineering* **14**(11) pp. 1711-1730 (November 1988).

[Bern81]
P.A. Bernstein et al., "Query Processing in a System for Distributed Databases(SDD-1)," *ACM Transactions on Database Systems*, pp. 602-625 (December 1981).

[Blas76]
M.W. Blasgen and K.P. Eswaran, "On the Evaluation of Queries in a Relational Data Base System," Research Report RJ-1745, IBM San Jose (April 1976).

[Care86]
M. Carey et al., "The Architecture of the EXODUS Extensible DBMS," *Proc. of the 1986 International Workshop on Object-Oriented Database Systems*, pp. 52-65 (September 1986).

[Care88]
M.J. Carey, D.J. DeWitt, and S.L. Vandenberg, "A Data Model and Query Language for EXODUS," *Proceedings of the 1988 ACM SIGMOD Conference*, pp. 413-423 (June 1988).

[Ceri84]
S. Ceri and G. Pelagatti, *Distributed Databases: Principles and Systems*, McGraw-Hill, New York, NY (1984).

[Codd70]
E.F. Codd, "A Relational Model of Data for Large Shared Data Banks," *Communications of the ACM*, pp. 377-387 (1970).

[Dada86]
P. Dadam et al., "A DBMS Prototype to Support Extended NF ^{2} Relations: An Integrated View on Flat Tables and Hierarchies," *Proceedings of the 1986 ACM SIGMOD Conference*, pp. 356-367 (May 1986).

[Epst78]
R. Epstein, M. Stonebraker, and E. Wong, "Distributed Query Processing in a Relational Data Base System," *Proceedings of the 1978 ACM SIGMOD Conference*, pp. 169-180 (May 1978).

[Fish87]
D. Fishman et al., "Iris: An Object-Oriented Database Management System," *ACM Trans. on Office Information Systems* **5**(1) pp. 48-69 (January 1987).

[Frey87]
J.C. Freytag, "A Rule-Based View of Query Optimization," *Proceedings of the 1987 ACM SIGMOD Conference*, pp. 173-180 (May 1987).

[Grae87a]
G. Graefe and D.J. DeWitt, "The EXODUS Optimizer Generator," *Proceedings of the 1987 ACM SIGMOD Conference*, pp. 160-172 (May 1987).

[Grae87b]
G. Graefe, "Rule-Based Query Optimization in Extensible Database Systems," Ph.D. Thesis, Computer Science Dept, University of Wisconsin, Madison, WI (August 1987).

[Grae87c]
G. Graefe, "Software Modularization with the EXODUS Optimizer Generator," *IEEE Database Engineering*, (December 1987).

[Gran81]
J. Grant and J. Minker, "Optimization in Deductive and Conventional Relational Database Systems," *Advances in Data Base Theory, Vol. I*, pp. 195-234 Plenum Press, (1981).

[Haas89]
L.M. Haas, J.C. Freytag, G.M. Lohman, and H. Prirahesh, "Extensible Query Processing in Starburst," *Proceedings of the 1989 ACM SIGMOD Conference*, pp. 377-388 (June 1989).

[Hasa88]
W. Hasan and H. Pirahesh, "Query Rewrite Optimization in Starburst," Research Report RJ6367, IBM Almaden Research Center (August 1988).

[Ibar84]
T. Ibaraki and T. Kameda, "On the Optimal Nesting Order for Computing N-Relational Joins," *ACM Transactions on Database Systems* **9**(3) pp. 482-502 (September 1984).

[Ioan87]
Y.E. Ioannidis and E. Wong, "Query Optimization by Simulated Annealing," *Proceedings of the 1987 ACM SIGMOD Conference*, pp. 9-22 (June 1987).

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

[Jerr86]
M.R. Jerrum, L.G. Valiant, and V.V. Vazirani, "Random Generation of Combinatorial Structures from a Uniform Distribution," *Theoretical Computer Science 43*, pp. 169-188 (1986).

[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) pp. 671-680 (May 1983).

[Kooi80]
R.P. Kooi, "The Optimization of Queries in Relational Databases," Ph.D. Thesis, Computer Science Dept., Case Western Reserve University (September 1980).

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

[Lecl88]
C. Lecluse, P. Richard, and F. Velez, "O_2, an Object-Oriented Data Model," *Proceedings of the 1986 ACM SIGMOD Conference*, (June 1988).

[Lee88]
M.K. Lee, J.C. Freytag, and G.M. Lohman, "Implementing an Interpreter for Functional Rules in a Query Optimizer," *Proceedings of the 14th VLDB Conference*, pp. 218-229 (August 1988).

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

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

[Lohm89]
G.M. Lohman, "Is Query Optimization a 'Solved' problem?," *Workshop on Database Query Optimization*, Portland, Oregon, (May 1989).

[Mack86a]
L.F. Mackert and G.M. Lohman, "R^* Validation and Performance Evaluation for Local Queries," *Proceedings of the 1986 ACM SIGMOD Conference*, pp. 84-95 (May 1986).

[Mack86b]
L.F. Mackert and G.M. Lohman, "R^* Validation and Performance Evaluation for Distributed Queries," *Proceedings of the 12th International VLDB Conference*, pp. 149-159 (August 1986).

[Monm79]
C.L. Monma and J.B. Sidney, "Sequencing with Series-Parellel Precedence Constraints," *Mathematics of Operations Research 4, 3*, pp. 215-224 (August 1979).

[Morr86]
K. Morris, J.D. Ullman, and A. VanGelder, "NAIL! System Design Overview," *Proceedings of the 3rd International Conference on Logic Programming*, pp. 554-568 (July 1986).

[Mura88]
M. Muralikrishna and D. DeWitt, "Optimization of Multiple-Relation Multiple-Disjunct Queries," *Proceedings of the 7th SIGACT-SIGMOD Symposium on Principles of Database Systems*, pp. 263-275 (March 1988).

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

[Ono90]
K. Ono and G.M. Lohman, "Extensible Enumeration of Feasible Joins for Relational Query Optimization," *Proceedings of the 16th International VLDB Conference*, pp. 314-325 (August 1990).

[Rome84]
F. Romeo, A. Sangiovanni-Vincentelli, and C. Sechen, "Research on Simulated Annealing at Berkeley," *Proceedings of IEEE International Conference on Computer Design*, pp. 652-657 (October 1984).

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

[Rose80]
D.J. Rosenkrantz and H.B. Hunt III, "Processing Conjunctive Predicates and Queries," *Proceedings of the 6th International VLDB Conference*, pp. 64-72 (October 1980).

[Roth86]
V. Rothschild and N. Logothetis, *Probability Distributions*, John Wiley & Sons, New York, NY (1986).

[Sasa88]
G.H. Sasaki and B. Hajek, "The Time Complexity of Maximum Matching by Simulated Annealing," *JACM* 35(2) pp. 387-403 (April 1988).

[Schw86]
P. Schwarz et al., "Extensibility in the Starburst Database System," *Proc. of the 1986 International Workshop on Object-Oriented Database Systems*, (September 1986).

[Sedg83]
R. Sedgewick, *Algorithms*, Addison Wesley, Reading, MA (1983).

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

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

[Shap86]
L. Shapiro, "Join Processing in Database Systems with Large Main Memories," *ACM Transactions on Database Systems* **1**(3) pp. 239-264 (September 1986).

[Shek90]
E.J. Shekita and M.J. Carey, "A Performance Evaluation of Pointer-based Joins," *Proceedings of the 1990 ACM SIGMOD Conference*, pp. 300-311 (May 1990).

[Sinc89]
A. Sinclair and M. Jerrum, "Approximate Counting, Uniform Generation and Rapidly Mixing Markov Chains," *Information and Computation 82*, pp. 93-133 (1989).

[Smit75]
J.M. Smith and P.Y. Chang, "Optimizing the Performance of a Relational Algebra Database Interface," *Communications of the ACM* **18** pp. 568-579 (October 1975).

[Ston76]
M. Stonebraker, E. Wong, P. Kreps, and G. Held, "The Design and Implementation of INGRES," *ACM Transactions on Database Systems*, pp. 189-222 (September 1976).

[Ston86]
M. Stonebraker and L.A. Rowe, "The Design of POSTGRES," *Proceedings of the 1986 ACM SIGMOD Conference*, pp. 340-355 (May 1986).

[Swam88]
A. Swami and A. Gupta, "Optimization of Large Join Queries," *Proceedings of the 1988 ACM SIGMOD Conference*, pp. 8-17 (June 1988).

[Swam89]
A. Swami, "Optimization of Large Join Queries: Combining Heuristics and Combinatorial Techniques," *Proceedings of the 1989 ACM SIGMOD Conference*, pp. 367-376 (June 1989).

[Tsur86]
S. Tsur and C. Zaniolo, "LDL: A Logic-Based Data-Language," *Proceedings of the 12th International VLDB Conference*, pp. 33-41 (August 1986).

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

[Whan85]
K.Y. Whang, "Query Optimization in Office-by-Examples," IBM Research report, RC 11571 (1985).

[Wins84]
P. Winston, *Artificial Intelligence*, Addison-Wesley, Reading, MA (1984).

[Wong76]
E. Wong and K. Youssefi, "Decomposition - A Strategy for Query Processing," *ACM Transactions on Database Systems* **1**(3) pp. 223-241 (September 1976).

[Yoo89]
H. Yoo and S. Lafortune, "An Intelligent Search Method for Query Optimization by Semijoins," *IEEE trans. on Knowledge and Data Engeneering* **1**(2)(June 1989).

Listed below is the schedule for **Benjamin Liblit's** visit to the Department of Computer Science.

AREA: Software Engineering
PHD: Univ of California-Berkeley
Host: Marvin Solomon

DATE: April 12 (Monday):
Breakfast at Collins House: Professor Marvin Solomon

9:00-9:30	Dean Herbert Wang, 301 South Hall
9:45-10:30	Professor Suman Banerjee, Room 7391
10:40-11:15	Professor Mark Craven, 6730 MSC
11:15-11:50	Professor David Page, 6743 MSC
12:00-1:30	Lunch – Professor Somesh Jha
1:30-2:15	Professor Susan Horwitz, Room 5391
2:15-3:00	Professor Remzi Arpaci-Dusseau, Room 7357
3:00-3:30	Talk preparation
3:30-4:00	Cookies/Coffee, Room 2310
4:00-5:00	Talk, Room 1221

DATE: April 13 (Tuesday):
Breakfast at Collins House:

9:00-9:45	Professor Jeff Naughton, Room 7369
9:45-10:30	Professor Guri Sohi, Room 6375
10:30-11:15	Professor Bart Miller, Room 6381
11:15-12:00	
12:00-1:30	Lunch – Professor Mark Hill
1:30-2:15	Professor Charles Fischer, Room 5397
2:15-3:00	Professor Tom Reps, Room 5387
3:00-3:45	Professor David DeWitt, Chair, Room 7363

Computer Science Department
University of Wisconsin-Madison 1210 W. Dayton Street Madison, Wisconsin 53706
608/262-6616 Fax: 608/262-9777 E-mail: cathy@cs.wisc.edu