National Library of Canada
Bibliothèque nationale du Canada

Canadian Theses Service
Services des thèses canadiennes

Ottawa, Canada
K1A 0N4

CANADIAN THESES

THÈSES CANADIENNES

NOTICE

The quality of this microfiche is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Previously copyrighted materials (journal articles, published tests, etc.) are not filmed.

Reproduction in full or in part of this film is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30. Please read the authorization forms which accompany this thesis.

AVIS

La qualité de cette microfiche dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

Les documents qui font déjà l'objet d'un droit d'auteur (articles de revue, examens publiés, etc.) ne sont pas microfilmés.

La reproduction, même partielle, de ce microfilm est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30. Veuillez prendre connaissance des formules d'autorisation qui accompagnent cette thèse.

THIS DISSERTATION HAS BEEN MICROFILMED EXACTLY AS RECEIVED

LA THÈSE A ÉTÉ MICROFILMÉE TELLE QUE NOUS L'AVONS REÇUE

NL 339 (r. 86/01)

Canada

0-315-28295-9

National Library of Canada | Bibliothèque nationale du Canada
CANADIAN THESES ON MICROFICHE | THÈSES CANADIENNES SUR MICROFICHE

NAME OF AUTHOR/*NOM DE L'AUTEUR* Pau Yen Yong

TITLE OF THESIS/*TITRE DE LA THÈSE* Minimization of Pages Fetches in Query Processing in Relational Databases

UNIVERSITY/*UNIVERSITÉ* Simon Fraser University

DEGREE FOR WHICH THESIS WAS PRESENTED/*GRADE POUR LEQUEL CETTE THÈSE FUT PRÉSENTÉE* Master of Science

YEAR THIS DEGREE CONFERRED/*ANNÉE D'OBTENTION DE CE GRADE* 1984

NAME OF SUPERVISOR/*NOM DU DIRECTEUR DE THÈSE* Tiko Kameda

Permission is hereby granted to the NATIONAL LIBRARY OF CANADA to microfilm this thesis and to lend or sell copies of the film.

The author reserves other publication rights, and neither the thesis nor extensive extracts from it may be printed or otherwise reproduced without the author's written permission.

L'autorisation est, par la présente, accordée à la BIBLIOTHÈQUE NATIONALE DU CANADA de microfilmer cette thèse et de prêter ou de vendre des exemplaires du film.

L'auteur se réserve les autres droits de publication; ni la thèse ni de longs extraits de celle-ci ne doivent être imprimés ou autrement reproduits sans l'autorisation écrite de l'auteur.

DATED/*DATÉ* Feb. 2, 84. SIGNED/*SIGNÉ*

PERMANENT ADDRESS/*RÉSIDENCE FIXE*

[redacted]

NL-91 (3-74)

MINIMIZATION OF PAGE FETCHES IN QUERY PROCESSING IN RELATIONAL DATABASES

by

Pau Yen Yong

B.Sc., Dalhousie University, 1981

THESIS SUBMITTED IN PARTIAL FULFILLMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

in the Department

of

Computing Science

© Pau Yen Yong 1984

SIMON FRASER UNIVERSITY

January 1984

All rights reserved. This work may not be
reproduced in whole or in part, by photocopy
or other means, without permission of the author.

PARTIAL COPYRIGHT LICENSE

I hereby grant to Simon Fraser University the right to lend my thesis, project or extended essay (the title of which is shown below) to users of the Simon Fraser University Library, and to make partial or single copies only for such users or in response to a request from the library of any other university, or other educational institution, on its own behalf or for one of its users. I further agree that permission for multiple copying of this work for scholarly purposes may be granted by me or the Dean of Graduate Studies. It is understood that copying or publication of this work for financial gain shall not be allowed without my written permission.

Title of Thesis/Project/Extended Essay

Minimization of Pages Fetches in Query Processing in Relational Databases

Author:

(signature)

Pau Yen Yong

(name)

Feb. 2, 84.

(date)

APPROVAL

Name: Pau Yen Yong

Degree: Master of Science

Title of Thesis:

Minimization of Page Fetches

in Query Processing in Relational Databases

Examining Committee:

Chairperson: Pavol Hell

Tiko Kameda
Senior Supervisor

wo-Shun Luk

Art Liestman

Toshihide Ibaraki
External Examiner
(in absentia)

Date Approved: Jan 12, 1984

ABSTRACT

This thesis investigates nested loops methods for computing joins in a relational database. These methods try to determine the optimal nesting order of relations in the evaluation program (that is, to minimize the number of page fetches). Since true minimization is in general NP-hard, several heuristic algorithms (C, D, and D2) have been developed and compared with the currently known algorithms (A and B). The heuristic solutions obtained from these algorithms are improved by interchanging adjacent relations. Experiments are conducted for testing the algorithms. The results indicate that among the four algorithms (which are applicable to general queries), algorithm D gives the best closeness-of-solution to the optimal one. For over 200 example queries involving 6 or fewer relations, the number of page fetches required by the solutions of algorithm D were never more than 38 percent greater than the optimal values, and on the average less than 5 percent greater than optimal. Two special cases have been identified which can be solved precisely : 1) the input query is a general query involving three relations; and 2) the input query is a loop query. Further experiments indicate that, for all the sample queries used above, the nested loops outperform the sort-merge method of query evaluation. Under the assumption that the 4-way merge sort is used and that every time two relations are merged they must be sorted, algorithm D did better than the sort-merge method for all of the sample queries used above.

To my parents in Johor Bahru.

ACKNOWLEDGEMENT

I wish to thank my senior supervisor Dr. Tiko Kameda for his guidance and patience throughout the course of this work. I am particularly grateful to Dr. Toshihide Ibaraki of Toyohashi University of Technology, Japan for his many thoughtful contributions to my work. Sincere thanks are due also to Dr. Wo Shun Luk, Dr. Art Liestman, Dr. Pavol Hell, and Mr. Charles G. Brown for making valuable comments and proofreading my thesis.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

The term "Relational Database System" has been widely used since E.F.Codd published his influential paper on this topic [CODD-70]. The relational model suggests that all data in a database management system may be represented by a set of simple two-dimentional tables named relations. Within each relation the columns are usually referred to as attributes (or domains), and the rows as tuples (or records). Each attribute is associated with a field name. An example of a relation describing students information is shown in Figure 1.1. This relation consists of three tuples, each of which is made up of three different fields : STUDENT_NUMBER, NAME, and ADDRESS.

STUDENT_NUMBER	NAME	ADDRESS
78233-7731	Ann Cooper	607 Keel St., Van.
79161-2435	Qing D. Li	1933 Esi Drive, Van.
79313-4971	Wen R. Lewis	565 High Park Ave., Van.

Figure 1.1 A relation R.

Tuples of relations are stored in physical (or secondary) storage space which is divided into fixed-size blocks called pages. The unit of

storage allocation and the unit of transfer between main storage and secondary storage is a page. The transfer of a page from secondary storage to main storage is called a page fetch. In order to access a tuple from a page, a page fetch is incurred if the page does not already reside in main storage.

Queries to a database for extracting desired data are posed through a specialized language called query language. The query language is usually designed from a high-level computer language and made easy to use for inexperienced users. Relational queries (questions about a relational database) can be expressed in high-level query languages simply and concisely. A wide variety of query languages have been proposed. Such as, QUEL, run under INGRES; QUERY-BY-EXAMPLE and SQL, run under SQL/Data System. There are a number of methods for accessing data in response to queries. Since a page fetch is costly, these methods attempt, through different approaches, to minimize the number of page fetches needed for answering a query.

Answering a query very often requires computing joins of relations. However, the join operation is the most time consuming and difficult operation, for a great number of page fetches are often required. Much has been done on the investigation of different approaches for computing joins of relations [AHO-79, ASTR-80, DEMO-80, IBAR-82, KIM-80, MERR-83, PERC-76, ROSE-80, SELI-79, WONG-70, YAO-79]. These optimization studies in query evaluation include simplistic exhaustive searches as well as complex heuristic methods. The purpose of this work is to investigate the nested loops methods for computing joins in a relational database. These methods attempt to determine the optimal nesting order of relations in the query

evaluation program. Optimal nesting order is defined as the nesting order which minimizes the number of page fetches.

In Chapter 2, important assumptions and different strategies for query evaluation programs will be presented and discussed.

Chapter 3 will discuss the nested loops method with special construction [IBAR-82]. The expected number of page fetches needed for answering a query is a function of the nesting structure of relations. This function is defined as the cost function we want to minimize. Since the true minimization of the expected number of page fetches is NP-hard, several heuristic algorithms are developed. It will be also shown that in two special cases the problem can be solved exactly when the assumptions of cost function hold.

Two sets of experiments will be conducted in Chapter 4. In the first set, the algorithms proposed in this thesis as well as the existing ones are run. The input query structures contain cycles. The heuristic solutions are compared with the optimal solutions obtained by exhaustive search. In the second set of experiments, "tree queries" will be tested. The comparison criteria for the experiments are 1) the closeness of the solution obtained by a heuristic algorithm to the optimal solution and 2) the running time of the algorithm.

In Chapter 5 a scheme is developed for improving the heuristic solutions obtained by the heuristic algorithms of Chapter 4. A set of experiments will be conducted for determining improvements achieved by this scheme.

In Chapter 6, we shall compare the number of page fetches required in the nested loops method with that of the sort-merge method.

The appendices consist of the notation used and the formal descriptions of the algorithms discussed in this thesis.

CHAPTER 2

QUERY EVALUATION

2.1. Notations and Assumptions

The relations referenced in a given query are denoted as $R_1, R_2, \ldots, R_n$. $R_i.A$ is the attribute A of relation R_i. A query Q is defined as a conjunction of simple predicates :

$$Q = P_1 \wedge P_2 \wedge \ldots \wedge P_n,$$

where

$$P_i = R_i.A \ \theta \ R_k.B$$

and θ is one of the following comparison operators,

$$\theta \in \{=, <, >, \leq, \geq\}.$$

It should be noted that the set of comparison operators does not include the not-equal ($\neq$) sign. Rosenkrantz [ROSE-80] proves that when $\neq$ comparisons between variables are allowed, minimization as well as most of the corresponding approximation problems are NP-hard.

The predicates are generally grouped into three categories:

1. $R_i.A\theta c$

   c represents a constant. This type of predicate is known as a <u>one-relation</u> predicate [WONG-76]. In order to test a predicate of this type, all the tuples of relation R_i have to be checked (that is, <u>scan</u> the relation -- all the data pages that store the tuples must be fetched) through the attribute A.

2. $R_i.A\theta R_i.B$

This form of predicate is also known as a one-relation predicate because there is only one relation involved. However, R_i has to be scanned only once.

3. $R_i.A\theta R_j.B$ $(i \neq j)$

This type of predicate, involving two relations, is called a join-predicate. Both of the relations R_i and R_j are examined in processing the predicate.

The join operation is the most time consuming and difficult operation. In the present thesis, only the JOIN operation is considered. The one-relation predicates are not of direct interest. Intuitively, the cost of processing a one-relation predicate is linear in the number of tuples of the relation involved. It is often desirable to process the one-relation predicates before the join-predicates are considered. It has been shown that this tactic is beneficial [STON-76]. For instance, after processing a one-relation predicate the size of the relation involved is smaller. The unqualified tuples — which do not satisfy the one-relation predicate — need not be examined in the following join operations.

In what follows, the sizes of relations are assumed to be those after the processing of one-relation predicates. The number of tuples of a relation R_i is denoted by N_i. M_i is the number of pages that is needed to store all the tuples of R_i. It is assumed that tuples from different relations do not co-reside in a page.

Given a predicate $P=R_i.A\theta R_j.B$, the selectivity factor f_p associated with P is defined as the probability that a randomly chosen tuple pair from

R_i and R_j satisfies P. The selectivity factors of predicates are assigned by the system program [SELI-79], which computes the selectivity factors from the known parameters such as N_i, M_i, number of distinct values in a join attribute. The distribution of the values in a join attribute is assumed to be uniform. This assumption is adopted from [IBAR-82].

Graphical Representation

A query may be represented graphically. Each relation is represented by a node in the graph. An edge between two nodes indicates the existence of a predicate P in which both relations are referenced. Such an edge is labelled by its associated predicate P.

$G(Q)=(V,E)$

where V=the set of relations referenced in query Q,

$E=\{(i,j) \mid i,j \in V$ and there exists some predicate of Q which references relation R_i and $R_j\}$.

Example

Given query : $Q=P_1 \wedge P_2 \wedge P_3 \wedge P_4$

where $P_1=(R_1.A=R_2.B)$, $P_2=(R_2.C=R_3.D)$,

$P_3=(R_3.E=R_4.F)$, $P_4=(R_1.G=R_3.H)$.

and $f_{P_1}=0.001$, $f_{P_2}=0.002$,

$f_{P_3}=0.002$, $f_{P_4}=0.003$.

The corresponding query graph G(Q) is shown in Figure 2.1.

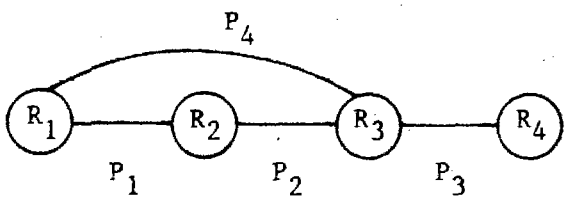


Figure 2.1 A query graph

The selectivity diagram of Q is the same as its G(Q) except that the label P of each edge is replaced by the corresponding selectivity factor f_P.

2.2. Different Strategies for Query Evaluation Program

The query evaluation programs attempt to process queries from different approaches. This section gives a brief introduction to the proposed strategies.

2.2.1. Sort-Merge

The major operations of this method are the sorting and merging of relations. Given a predicate, the first step is to sort the relations individually on their join attributes. If a z-way merge-sort is used for sorting, the cost (in number of page fetches) of sorting a relation R_i can be given as [BLAS-77] :

$$\text{sort cost} = 2\lceil M_i \log_z M_i \rceil$$

The next step is to merge the two sorted relations. Both of them need to be scanned only once since they are already sorted on the join attribute. The scanning cost (in terms of page fetches) is simply the total number of pages of the relations referenced.

Although the basic strategy of sort merge is rather straightforward, it can be costly to implement. For queries with many predicates, the sequence of evaluating the predicates obviously affects the number of page fetches required. The process of selecting the optimal order of predicates tends to be very complicated. If there are m predicates involved, there can be as many as m! different sequences. The overheads in computing the estimated costs are far too high if each sequence is attempted. Much research has been done on this problem of finding the optimal sequence [SELI-79, MERR-83].

2.2.2. Decomposition

The general procedure for this strategy [WONG-76] is to decompose the multi-variable query (a query which references more than one relation) into a sequence of single-variable (that is, single-relation) queries by applying reduction of query and tuple substitution. Reduction is a process of breaking off the components of the query which are joined to it by a single variable. When these components cannot be further detached to single-variable subqueries, tuple substitution is used to complete the decomposition. One of the variables is selected for substituting a tuple at a time.

2.2.3. Nested Loops Method

The loop is the major structure of this method. Figure 2.2 shows the basic structure of a nested loops program. Assume that relations R_1 and R_2 consist of M_1 and M_2 pages, respectively. In order to compute the join of the two relations, for each data page of R_1, all the data pages of R_2 must be examined if R_1 is nested outside (see Figure 2.2) R_2. Thus the total

total number of page fetches needed to compute the join is $M_1+M_1M_2$.

```
For i=1 to M1 do
  Begin
   For j = 1 to M2 do
      Begin
        Access the tuple pair(i,j)
      End
  End
```

Figure 2.2 A nested loops structure.

This method is quite straightforward. However, for queries referencing many relations, the evaluation can be costly. Suppose, for example, that five relations are referenced in a given query. The corresponding sizes are M_1=10, M_2=20, M_3=30, M_4=40, M_5=50. The cost formula

$$M_1+M_1M_2+M_1M_2M_3+M_1M_2M_3M_4+M_1M_2M_3M_4M_5$$

gives a total number of page fetches, which is over twelve million.

One way to reduce the total number of page fetches is given by [KIM-80]. Instead of automatically scanning the relation R_2 of Figure 2.2 in a top-down manner(i.e., fetching the data pages from the first page to the last page), some conditions are added. After the first scan of relation R_2, the second page of R_1 is fetched. Since the last page of R_2 is still in the main memory, the join operation can be performed without fetching a new page from R_2. Thus the scanning of R_2 is carried out in bottom-up

manner and we save one page in fetching. The next scan of R_2 starts from the top again and one page is saved for the first page of R_2 is already in the memory. It can be seen that the savings are insignificant if the sizes of the relations are large.

Another way of reducing the number of page fetches is shown in [IBAR-82]. More structure is superimposed and a data page is fetched only if it is necessary to do so. We shall see this in Chapter 3.

CHAPTER 3

NESTED LOOPS METHOD FOR QUERY EVALUATION

3.1. Reasons for Using Nested Loops Method

The nested loops method in processing relational queries is the approach taken in the present thesis. It has various advantages over the other methods. The ease of programming is highly attractive to programmers. Furthermore, the number of page fetches needed in this method can be kept small in the situation where all the data pages of each relation are not automatically fetched [IBAR-82] (This differs from the automatic fetch tactic used, for example, in [PERC-76]). Some tests are performed before the above access is made, the details of which are presented in the following section.

3.2. Nested Loops Method with Special Pointers

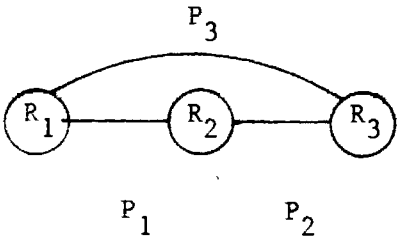
3.2.1. Basic Structure

The structure of the nested loops method with special pointers is based on the following idea :

> Some useful pointers associated with relations are created during the sorting of relations. These pointers are used in query evaluation programs to fetch only those pages of the corresponding relations which contain the tuples satisfying a predicate of an input query [IBAR-82].

Without loss of generality, $R_1, R_2, \ldots, R_n$ are assumed to be processed in that order. For a predicate $P_i = R_k.x \theta R_i.y, k<i$, R_i is assumed to be sorted on attribute y. For each tuple in R_k, two pointers associated with P_i are created. One of them points to the first page of R_i on which there is a tuple such that the tuple pair (i.e., a tuple from R_k and a tuple from R_i) satisfies the predicate. The other pointer points to the last such page. The following example will make this clear.

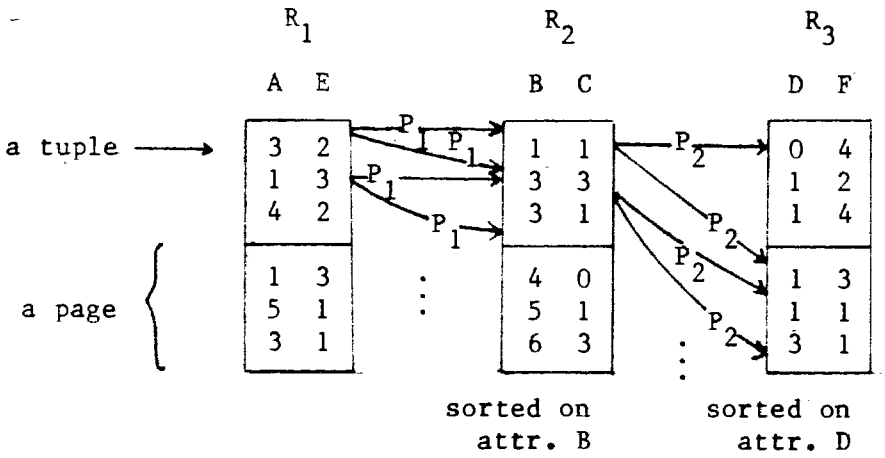
Given query graph :



$Q = P_1 \wedge P_2 \wedge P_3$

where $P_1 = (R_1.A = R_2.B)$, $P_2 = (R_2.C = R_3.D)$,

$P_3 = (R_1.E = R_3.F)$.



Relations R_1, R_2, and R_3 are processed in that order. R_2 and R_3 are sorted on the join attributes B and D respectively. For instance, the join

attribute A of the first tuple of R_1 has value 3. The pointers (associated with P_1) of the this tuple both point to the first page of R_2 because this page contains both the first and the last tuples which satisfy the predicate P_1. For the first tuple in R_2, the join attribute C has value 1. The tuples of R_3, which satisfy P_2 with $R_2.C=1$, span the first two pages of the relation. Thus the corresponding pointers of the first tuple of R_2 point to pages 1 and 2 in R_3 respectively. Note that there is no pointer created for P_3 as it can be processed after a page of R_3 is fetched by P_2.

3.2.2. Construction of the pointers

In [IBAR-82], the pointers from R_k to R_i with respect to P_i ($P_i=R_k.a\theta R_i.b$) are created in the following way, without any additional page fecthes:

(1) While sorting the relation R_i a list is constructed in main memory.

(2) For each page of R_i, the distinct values of $R_i.b$ are kept in the list.

(3) The list is maintained until the pointers for R_k with respect to P_i are constructed.

Step 2 can in fact be improved by storing, for each page of R_i, merely the minimum and maximum values of $R_i.b$.

3.2.3. Cost Function

Some definitions are first introduced.

$$pred(i) = \{P \mid P='R_k.a\theta R_h.b',\ k,h \leq i\}$$

$$F_i = \text{PRODUCT [FOR P IN pred(i)] OF } f_P.$$

Using the nested loops method without pointers, the total number of page fetches for R_i is $M_1M_2...M_i$. Let $P_i=R_k.a\theta R_i.b$, Ibaraki [IBAR-82] shows that using the pointers technique this number can actually be reduced to

$$H_i = F_{i-1}N_1...N_{i-1}f_{P_i}M_i, \qquad ...(3.1)$$

where $F_{i-1}N_1...N_{i-1}$ is the expected number of tuple combinations (from $R_1...R_{i-1}$) that satisfy the pred(i). $M_1...M_i$ is indeed reduced by the factor

$$(F_{i-1}N_1...N_{i-1}) * f_{P_i} / (M_1...M_{i-1}).$$

Note that formula (3.1) is only an approximation function of the general function derived in [IBAR-82]. The general function of (3.1) has been given as follows :

$$M_1...M_i[1-(1-f_{P_i})^{K_{i-1}}].$$

K_{i-1} is the average distinct $R_k.a$-value. Other important parameters used in the general cost function have been given as follows :

(1) $K_{i-1} = r_k[1-(1-1/r_k)^{J_{i-1}}]$

(2) $J_{i-1} = (s/t_k)[1-(1-t_k/s)^{I_{i-1}}]$

(3) $I_{i-1} = F_{i-1}N_1...N_{i-1}/M_1...M_{i-1}$

r_k is the number of distinct values that $R_k.a$ can take. J_{i-1} is the expected number of distinct tuples chosen from a page of R_k and I_{i-1} is the

tuple combinations per page combination of $R_1,\ldots,R_{i-1}$. Using the approximation formula

$$1-(1-a)^N \sim 1-\exp(-aN) \quad \text{for small } a$$
$$\sim aN \quad \text{for small } aN,$$

$[1-(1-f_{P_i})^{K_{i-1}}]$ is approximated as $(1-\exp(-f_{P_i} K_{i-1}))$ under the assumption that $f_{P_i} << 1$. Furthermore, K_{i-1} is approximated as I_{i-1} when f_{P_i}, J_{i-1}/r_k, and $t_k I_{i-1}/s$ are all small. Hence, if $f_{P_i} I_{i-1} << 1$, the function

$$M_1 \ldots M_i [1-\exp(-f_{P_i} I_{i-1})]$$

is approximated as (3.1). This approximation is valid only when the assumptions stated above are all valid. Thus, the assumptions made in this thesis are adopted from [IBAR-82].

There are n relations involved in a query. Only the relation nested in the outermost loop need not be sorted whereas the others have to be sorted at most once. The cost formula we want to minimize is :

$$COST = M_1 + \sum_{i=2}^{n} (H_i + S(P_i)) \qquad \ldots(3.2)$$

where $S(P_i)$ is the number of page fetches for sorting R_i on $R_i.b$, i.e.,

$$S(P_i) = \begin{cases} 0 & \text{if i=1 or } R_i \text{ is already sorted on } R_i.b \\ & \text{prior to the evaluation of query Q,} \\ 2\lceil M_i \log_z M_i \rceil & \text{otherwise,} \end{cases}$$

assuming that a z-way merge sort is used.

3.2.4. Role of Nesting Structure

The expected cost (i.e., number of page fetches) is a function of the nesting structure of relations. Given a nesting order, the corresponding estimated cost can be computed from (3.2). However, the greatest difficulty is the determination of the best order of the relations. To find the optimal order from among the n! possible choices by exhaustive search may not be practical. In fact true minimization is NP-hard. Therefore, we utilize some heuristic search methods.

3.3. Algorithms for Finding Good Nesting Order

3.3.1. Algorithm A. [IBAR-82]

This algorithm is applicable to general queries. The method is based on the assumption that H_n is the most dominant term in (3.2), followed by $H_{n-1},\ldots,H_1$. Thus, H_i's are minimized in this order. The algorithm is shown in Appendix 2.

Ties may occur in choosing R_i (step 2.1a of the Appendix) when there is more than one minimum H_i. In the worst case, all the possibilities are explored, and the running time of the algorithm may increase exponentially. This gives a time complexity of O(n!).

3.3.2. Algorithm B. [IBAR-82]

This algorithm applies to the special case where the input query is a "tree query". It gives the optimal solution of a tree query only if all the assumptions made for the approximation cost function (3.1) are valid. In the algorithm, rank values are computed for single relations or groups

of relations (see Appendix 3). Relations are then ranked in such a way that the corresponding rank values are in ascending order. The optimal solution is produced in polynomial time. The time complexity is $O(n^2 \log n)$. The details of the algorithm are shown in Appendix 3.

3.3.3. Algorithm C

This algorithm is applied to general queries. Its basic structure is supported by algorithm B. A minimum spanning tree of the input query graph is first determined. Using this tree, the nesting order of relations is then computed by B. Cost for the input query graph is finally computed from the formula (3.2).

The edges of the input query graph are weighted by selectivity factors. The smaller the selectivity factors used, the fewer the needed number of page fetches of relations. The graphical representation of the following example makes this clear. Given the query graph shown in Figure 3.1a, there are three possible spanning trees(3.1b,c,d).

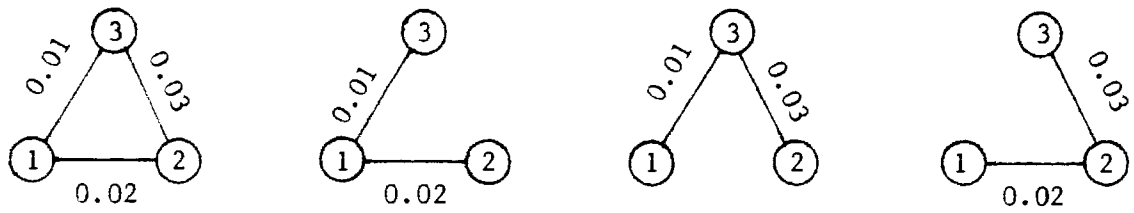


Figure 3.1 A 3-node query graph.

Suppose that relations 1, 2, and 3 are of the same size M, and have the same number of tuples N. For each spanning tree, the total number of page fetches needed is as follows :

spanning tree b : M+0.01NM+0.01*0.02NNM;

spanning tree c : M+0.01NM+0.01*0.03NNM;

spanning tree d : M+0.02NM+0.02*0.03NNM.

Figure 3.1b is clearly the best choice among the three. In choosing the predicate, the one with smaller selectivity factor has higher priority to be picked. The algorithm is shown in Appendix 4.

A drawback of this algorithm is that if a given n-node query graph is complete and all the selectivity factors are the same, there are n^{n-2} minimum spanning trees in which case the time complexity is $O(n^n \log n)$ for each tree requires $O(n^2 \log n)$ time. In order to avoid such excessive computation, in algorithm C, we test minimum spanning tree.

3.3.4. Algorithm D

We now take a closer look at the H_i formula :

$$H_1 = M_1$$

$$H_2 = F_1 N_1 f_{P_2} M_2$$

$$\vdots$$

$$H_i = F_{i-1} N_1 \cdots N_1 f_{P_i} M_i$$

$$H_{i+1} = F_i N_1 \cdots N_i f_{P_{i+1}} M_{i+1}$$

Suppose that R_1 to R_{i-1} have been chosen in the above order. The factor $F_{i-1} N_1 \cdots N_{i-1}$ (hereafter FN_{i-1} factor) in H_i cannot be changed at the later stages. This FN_{i-1} factor becomes the common factor in the

remaining H_i's $(H_{i+1}, H_{i+2}, \ldots, H_n)$. This implies that it is highly desirable to keep the smallest common factor at every step of choosing R_i. Thus, in order to choose R_i, we use a one-step look ahead method. The FN_i factor in H_{i+1} is first minimized. The relation associated with the smallest FN_i is chosen as R_i. The next step is to determine f_{P_i} from the set of selectivity factors which are associated with the predicates in pred(i)-pred(i-1). We simply pick the one with minimum value.

When there are more than one minimum FN_i, all the possible choices for R_i are tried. In the worst case, the time needed for running the algorithm grows exponentially. This is same as the time complexity of algorithm A (O(n!)). The complete algorithm D is presented in Appendix 5.

3.3.5. Algorithm D2

In algorithm D, the relation with minimum $F_i N_1 \ldots N_i$ is picked as R_i without considering the $f_{P_i} M_i$ factor in H_i. In this algorithm we try to take the neglected factor into account when R_i is being considered. Some useful symbols are defined as follows:

PRED(x,i)={P|P='R_k.aθR_x.b' and $k \leq i$}
(the set of predicates which reference only relations R_x and R_k with $k \leq i$),

F{1,2,...,i}=F_i (the product of all the selectivity factors associated with the predicates in pred(i)),

F{1,2,...,i|x}=the product of all the selectivity factors associated with the predicates in PRED(x,i).

Given $R_1, R_2, \ldots, R_{i-1}, R_x, R_y$, without loss of generality it is assumed $R_1, R_2, \ldots, R_{i-1}$ are chosen in that order. The next step is to choose R_i from R_x and R_y. There are two possible cases:

(1) $R_i = R_x$, $R_{i+1} = R_y$;

(2) $R_i = R_y$, $R_{i+1} = R_x$.

In the first case the total number of page fetches for H_1 to H_{i+1} is shown as

$$s(x,y) = H_1 + H_2 + \ldots + F\{1,2,\ldots,i-1\} N_1 \ldots N_{i-1} f_{P_x} M_x$$
$$+ F\{1,2,\ldots,i-1,x\} N_1 \ldots N_{i-1} N_x f_{P_y} M_y. \qquad \ldots(3.3)$$

$s(x,y)$ indicates that R_x is followed by R_y in the nesting order. The latter case is shown as

$$s(y,x) = H_1 + H_2 + \ldots + F\{1,2,\ldots,i-1\} N_1 \ldots N_{i-1} f_{P_y} M_y$$
$$+ F\{1,2,\ldots,i-1,y\} N_1 \ldots N_{i-1} N_y f_{P_x} M_x. \qquad \ldots(3.4)$$

We subtract (3.4) from (3.3) and obtain

$$\begin{aligned} s(x,y) - s(y,x) = {} & F\{1,2,\ldots,i-1\} N_1 \ldots N_{i-1} f_{P_x} M_x \\ & + F\{1,2,\ldots,i-1,x\} N_1 \ldots N_{i-1} N_x f_{P_y} M_y \\ & - F\{1,2,\ldots,i-1\} N_1 \ldots N_{i-1} f_{P_y} M_y \\ & - F\{1,2,\ldots,i-1,y\} N_1 \ldots N_{i-1} N_y f_{P_x} M_x \end{aligned}$$

If $s(x,y) - s(y,x) \geq 0$ then

$s(x,y) \geq s(y,x)$ <=>

$$f_{P_x} M_x (F\{1,\ldots,i-1\}-F\{1,\ldots,i-1,y\}N_y) \geq$$

$$f_{P_y} M_y (F\{1,\ldots,i-1\}-F\{1,\ldots,i-1,x\}N_x) \qquad \ldots(3.5)$$

<=>

$$(F\{1,\ldots,i-1,y\}N_y-F\{1,\ldots,i-1\})/f_{P_y} M_y \leq$$

$$(F\{1,\ldots,i-1,x\}N_x-F\{1,\ldots,i-1\})/f_{P_x} M_x$$

Both sides are divided by $F\{1,\ldots,i-1\}$,

$$(F\{1,\ldots,i-1|y\}N_y-1)/f_{P_y} M_y \leq (F\{1,\ldots,i-1|x\}N_x-1)/f_{P_x} M_x.$$

For a relation R_k, let

$$\text{Ratio}(k)=\frac{(F\{1,\ldots,i-1|k\}N_k-1)}{f_{P_k} M_k}$$

Therefore,

$$s(x,y) \geq s(y,x) \iff \text{Ratio}(x) \geq \text{Ratio}(y).$$

The derivation implies that the nesting order $R_1,R_2,\ldots,R_{i-1},R_y,R_x$ is better than $R_1,R_2,\ldots,R_{i-1},R_x,R_y$ if and only if Ratio(y) is less than Ratio(x). Therefore, to choose R_i from a set of relations we simply compute the Ratio for each of those relations. Among those relations the one with the smallest Ratio is selected as R_i. Similarly, ties may occur in choosing R_i, for there may be more than one minimum Ratio. If the Ratios corresponding to R_i are all same and each choice of R_i is attempted, the time complexity of this algorithm is O(n!).

Incorrect Ratio

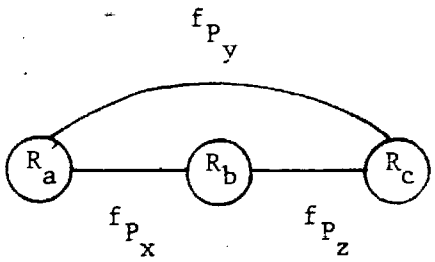
Let us take a closer look at the Ratio derived from the above section. We find that the f_{P_x} in (3.3) does not necessarily have the same value as the one in (3.4) because the former f_{P_x} is the minimum selectivity factor chosen from the set of selectivity factors associated with PRED(x,i-1), whereas the latter is chosen from the set associated with PRED(x,i). These two sets of predicates may contain different predicates. The choices of f_{P_x}'s in (3.3) and (3.4) can be different. Similarily in the case of f_{P_y}. In order to distinguish the difference, the f_{P_x} in (3.4) is replaced by f_{P_x}', and f_{P_y} in (3.3) by f_{P_y}'. The derivation in (3.5) is no longer valid because $f_{P_x} M_x$ and $f_{P_y} M_y$ cannot be factored. Instead, we have the following expression :

$$(F\{1,\ldots,i-1|x\}N_x f_{P_y}'-f_{P_y})/M_x \quad \geq \quad (F\{1,\ldots,i-1|y\}N_y f_{P_x}'-f_{P_x})/M_y.$$

Hence, the measure of the Ratio is actually not correct if $f_{P_x} \neq f_{P_x}'$ and $f_{P_y} \neq f_{P_y}'$. The drawback caused by the incorrect Ratios is shown in the experiments conducted in Chapter 4.

3.3.6. General Query Involving Three Relations

Given query Q with the following selectivity diagram SD(Q) :



$Q=P_x \wedge P_y \wedge P_z$,

where $P_x=(R_a.A=R_b.B)$, $P_y=(R_a.C=R_c.D)$, $P_z=(R_b.E=R_c.F)$,

and f_{P_x}, f_{P_y}, and f_{P_z} are the corresponding selectivity factors.

The query can be solved in constant time as follows : The three given relations are named R_a, R_b and R_c. The choice of R_a is important whereas the choice of R_b and R_c is not important. Each given relation referenced in the query is chosen as R_a in turn. For each choice of R_a, the nesting order $R_aR_bR_c$ is better than $R_aR_cR_b$ if and only if the following is true :

$$\frac{(f_{P_x} N_b \mathrm{MIN}(f_{P_y}, f_{P_z}) - f_{P_y})}{M_b} \leq \frac{(f_{P_y} N_c \mathrm{MIN}(f_{P_x}, f_{P_z}) - f_{P_x})}{M_c}$$

Proof.

Let s(a,b,c) denote the total cost of the nesting order $R_aR_bR_c$, and let $\mathrm{MIN}(f_{P_x}, f_{P_y})$ be the minimum of f_{P_x} and f_{P_y}. We have

$$s(a,b,c) = M_a + N_a f_{P_x} M_b + f_{P_x} N_a N_b \mathrm{MIN}(f_{P_y}, f_{P_z}) M_c$$

$$s(a,c,b) = M_a + N_a f_{P_y} M_c + f_{P_y} N_a N_c \mathrm{MIN}(f_{P_x}, f_{P_z}) M_b$$

$$\begin{aligned} s(a,b,c) - s(a,c,b) &= N_a f_{P_x} M_b + f_{P_x} N_a N_b \mathrm{MIN}(f_{P_y}, f_{P_z}) M_c \\ &\quad - N_a f_{P_y} M_c - f_{P_y} N_a N_c \mathrm{MIN}(f_{P_x}, f_{P_z}) M_b \\ &= N_a M_b [f_{P_x} - f_{P_y} N_c \mathrm{MIN}(f_{P_x}, f_{P_z})] \\ &\quad - N_a M_c [f_{P_y} - f_{P_x} N_b \mathrm{MIN}(f_{P_y}, f_{P_z})] \end{aligned}$$

If $s(a,b,c) - s(a,c,b) \geq 0$ then

$$s(a,b,c) \geq s(a,c,b) \iff$$

$$M_b [f_{P_x} - f_{P_y} N_c \mathrm{MIN}(f_{P_x}, f_{P_z})] \geq M_c [f_{P_y} - f_{P_x} N_b \mathrm{MIN}(f_{P_y}, f_{P_z})]$$

$$\iff [f_{P_x} - f_{P_y} N_c \mathrm{MIN}(f_{P_x}, f_{P_z})]/M_c \geq [f_{P_y} - f_{P_x} N_b \mathrm{MIN}(f_{P_y}, f_{P_z})]/M_b$$

$$\iff [f_{P_x} N_b \mathrm{MIN}(f_{P_y}, f_{P_z}) - f_{P_y}]/M_b \geq [f_{P_y} N_c \mathrm{MIN}(f_{P_x}, f_{P_z}) - f_{P_x}]/M_c .$$

Q.E.D.

Since each given relation is chosen as R_a in turn, the optimal solution is, therefore, determined from the three choices. The algorithm is formally described in Appendix 7.

3.3.7. Loop Queries

We can now describe a polynomial algorithm, which gives the optimal solution, when the input query is a loop query. The query graph of a loop query involving n relations is a cycle with exactly n nodes and n edges. By removing one edge at a time, there are at most n spanning trees. Each of them can be solved individually by applying algorithm B. Since the removed edge of each tree would not be used in the cost formula (3.1), algorithm B gives the exact ranks and provides the optimal nesting order for that spanning tree. Among the n nesting orders produced from different spanning trees, the one with smallest cost is chosen as the optimal nesting order. The details of the algorithm are shown in Appendix 8. The time complexity is $O(n^3 \log n)$.

CHAPTER 4

PERFORMANCE EVALUATION

4.1. Assumptions

Performance is a significant factor in the choice of algorithms. The predictability of algorithm performance is important to the user. In this section we investigate the performance of the various algorithms introduced in Chapter 3. Simulation experiments are conducted to evaluate the algorithms. The major criteria of evaluation are : closeness of the solution obtained by an algorithm to the optimal solution, and the running time of the algorithm. The following data for the experiments are generated by random number generators :

(1) M_i : number of pages of relation R_i.

(2) s/t_i : number of tuples per page of relation R_i.
N_i : number of tuples of R_i is computed by $M_i * s/t_i$.

(3) Selectivity factors associated with predicates.

The following ranges are assumed :

(1) $10 \leq M_i \leq 500$, for all i;

(2) $10 \leq s/t_i \leq 30$, for all i;

(3) $0 < f_p \leq 0.02$, for all P;

(4) z = 4, i.e., 4-way merge sort is used;

(5) s = 512 bytes, page size.

We believe that the ranges (1 and 2) chosen above are sufficiently large because the sizes of relations are assumed to be the sizes after processing the one-relation predicates. The irrelevant tuples and columns are eliminated. The range of s/t_i is adopted from the example used in [BLAS-77]. In his paper an average number of tuples per page from a relation is 20. The s/t_i range chosen above includes this value.

The choice of f_p is based on the assumptions made for the H_i cost formula (3.1). The range of f_p must be sufficiently small (see Chapter 3) such that the approximation is valid. For convenience, we adopt the range used in [IBAR-82].

The choice of 4 as the merge factor in sorting is not restrictive because the relationships between two merge factors, say z1 and z2, can always be found as follows :

$$\log_{z1}X = \frac{\log_2 X}{\log_2 z1} \quad \text{and} \quad \log_{z2}X = \frac{\log_2 X}{\log_2 z2}$$

Hence,

$$\log_2 z1 * \log_{z1}X = \log_2 z2 * \log_{z2}X$$

and,

$$\log_{z2}X = \frac{\log_2 z1 * \log_{z1}X}{\log_2 z2}$$

Therefore, if we want to change the merge factor z1 to z2 we simply multiply the sort cost $M_i \log_{z1} M_i$ (before it is doubled) by the factor $\log_2 z1 / \log_2 z2$.

Typically a page size ranges from 256 bytes to 4K bytes. In these experiments, we have adopted 512 bytes as our page size.

4.2. Normalization

In order to normalize the performance of the algorithms, heuristic solutions are compared with the optimal solutions. These optimal solutions can be obtained by exhaustive search. The normalization of a heuristic solution is denoted by performance ratio (PR), which is defined as follows :

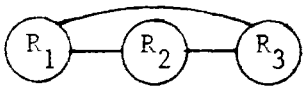
$$PR = \frac{\text{cost obtained by heuristic solution}}{\text{optimal cost}} \qquad \ldots(4.1)$$

The PR indicates the closeness of the heuristic solution to the optimal solution. It is either greater than or equal to 1. It cannot be less than 1 as the cost obtained by a heuristic cannot be smaller than the optimal cost.

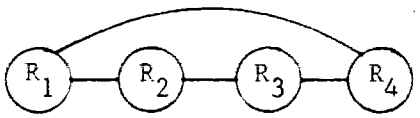
4.3. General Queries

In the following experiments, algorithms A, C, D, and D2 are evaluated. A collection of 11 query structures were chosen :

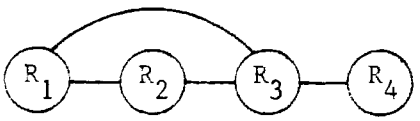
(1) Q1 :



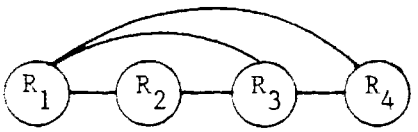
(2) Q2 :



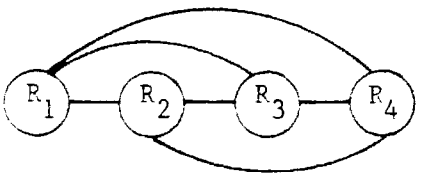
(3) Q3 :



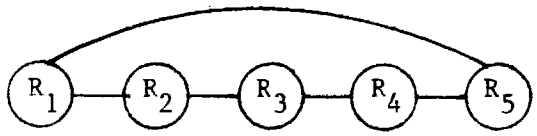
(4) Q4 :



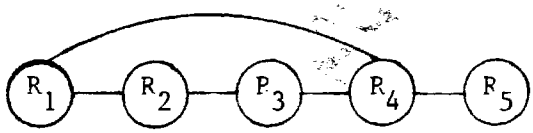
(5) Q5 :



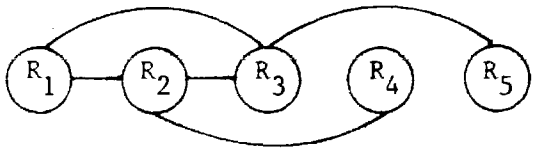
(6) Q6 :



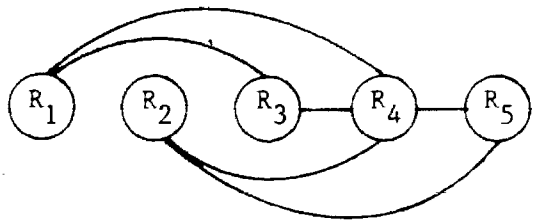
(7) Q7 :



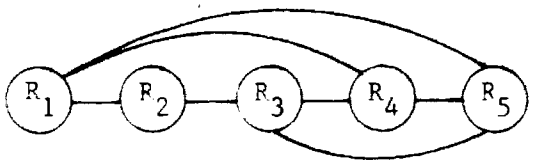
(8) Q8 :



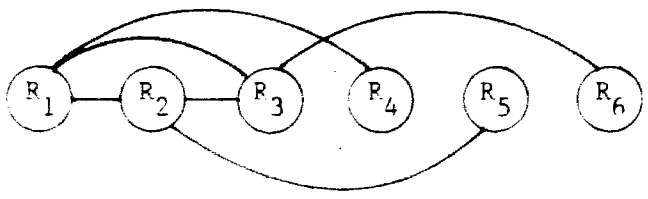
(9) Q9 :



(10) Q10 :



(11) Q11 :



4.3.1. Page Fetches

For each query structure, 24 sets of random numbers were generated. The heuristic solutions to these inputs are normalized as explained above. The following values are computed for the purpose of comparing the performance of the different algorithms :

(1) PR range.

(2) Average PR for each query structure.

(3) Range of average PR's.

(4) Standard deviation of the PR's.

The PR range is defined to be the difference between the largest and the smallest values in a set of PR's. The PR range of each algorithm is shown in the following table :

A : 1 -- 1.6087
C : 1 -- 2.8660
D : 1 -- 1.3750
D2 : 1 -- 109.5281

Table 4.1 The ranges of PR's of various algorithms.

Among the four algorithms, D has the smallest range of PR's. The expected number of page fetches needed in answering a query can be 37.50 percent more than the optimal solution. The range of PR's of algorithm A is slightly larger than this. The worst heuristic solution obtained for algorithm A in the above experiments is 60.87 percent larger than the

optimal solution. The PR range of C is fairly large. It varies from 1 to about 2.9. In other words, a heuristic solution produced by C can be almost triple the exact solution. The most unsatisfactory algorithm, as far as the PR range is concerned, is D2. One sample query produced a very large PR of 109.5281. This shows that the heuristic solution by D2 can be very costly.

Since the PR range of each algorithm appeared to be greatly affected by the query structure, the average of PR's was computed for each algorithm and query structure. For each query structure, 24 sets of random numbers were generated and the mean of 24 PR's was computed. In Figure 4.1, the curves show the relationship between the average PR's and the query structures used, Q1 to Q11. Each curve corresponds to one of the four algorithms. Among the four curves in the figure, curve A and D can hardly be distinguished, whereas curve C and D2 have some obvious high peaks which have very large values. In order to examine the curves A, C, and D more closely, curve D2 is not shown in Figure 4.2. It should be noted that the Y axis has been expanded in the figure. Table 4.2 below shows the ranges in terms of numerical values :

A	1.0009 — 1.0967
C	1.0061 — 1.2810
D	1.0044 — 1.0410
D2	1.0004 — 5.5885

Table 4.2 Average PR ranges of various algorithms.

It can be seen that the upper end of the range of average PR's for D2 is much smaller than the corresponding value in Table 4.1. This indicates that the PR's obtained by D2 are not uniformly distributed in the range shown in Table 4.1. The heuristic solutions obtained by D2 can be very bad. Due to this large variation in PR's, D2 is dropped from the experiments. Our interest will focus on algorithms A, C, and D. In Table 4.2, D, again, shows the smallest variation. Out of eleven average PR's, there is only one case (Q2) in which D has a value larger than the runner-up (A). Out of the eleven average PR's, there are seven values of A which are smaller than those of C. In addition, the range of average PR's for A is much smaller than that for C.

By comparing Tables 4.1 and 4.2 one can easily see that the PR range and the range of average PR's of each algorithm are very different. Variance (Sx^2), defined as follows, is used to measure the variation in PR's :

$$Sx^2 = \left(\sum_{i=1}^{n'} (xi^2 - n'X^2) \right) / (n'-1)$$

where xi = data value,
X = the mean of the data values,
n' = number of data values.

In this experiment, data value, xi, is the PR of each example; X is the average PR for each query structure; and n' is the number of samples tested (which is 24). We want to compute the variance for each query structure. Data fluctuating over a wide range of values will have a large variance. Conversely, a large variance indicates that the data have a wide spread about the mean. The standard deviation is defined as the positive

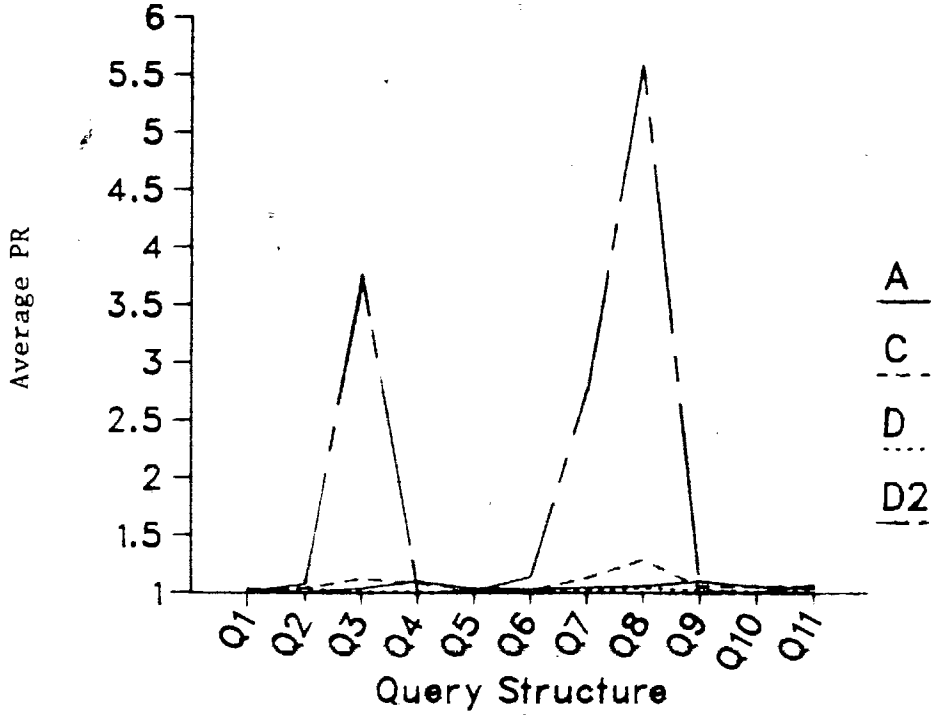


Figure 4.1 Average PR's for general query sturctures : A, C, D, D2

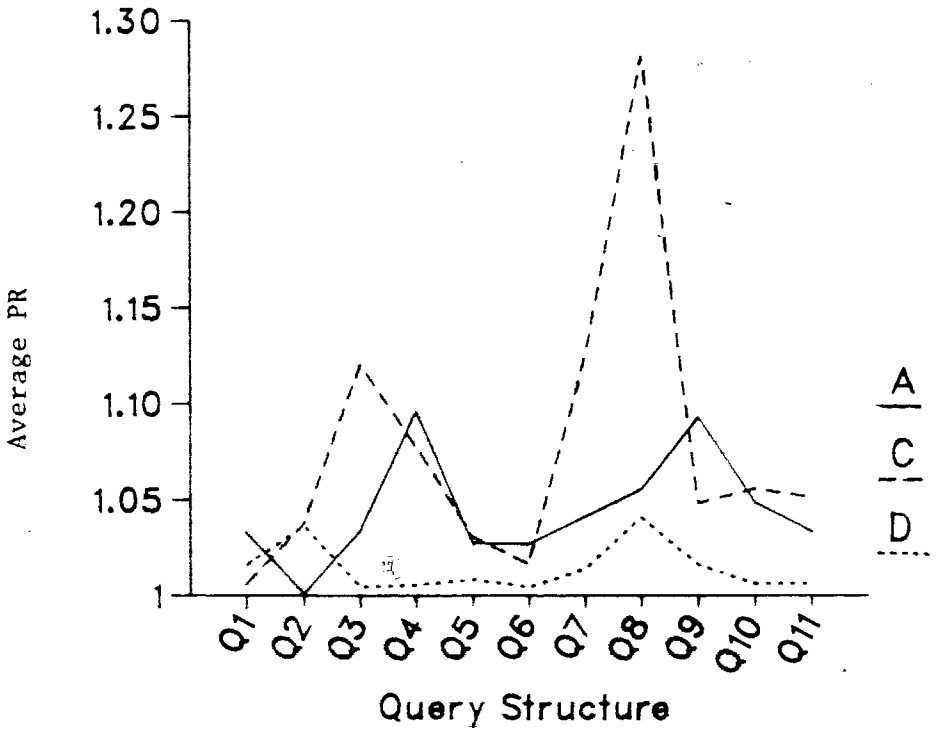


Figure 4.2 Average PR's for general query structures : A, C, D

square root of the variance.

$$Sx = \sqrt{Sx^2}$$

Figure 4.3 shows the standard deviation of each query structure. It can be seen that the standard deviation curves are similar to the corresponding average PR curves in Figure 4.2. That is to say, the algorithms with large average PR's also have large standard deviations. For example, for algorithm C, Q3 and Q8 have large average PR's as well as large standard deviations. This indicates that an algorithm that performs poorly does not perform poorly all the time. Namely, the PR is not consistently large for such an algorithm.

At this point a general conclusion can be stated. Among the four algorithms, D tends to produce the best heuristic solutions, whereas A gives reliable heuristic solutions which do not fluctuate widely. C is not as stable as the two algorithms. It tends to produce "unacceptable" as well as acceptable solutions.

4.3.2. Computation Time

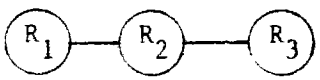
In this section the average running time of each algorithm for 24 examples of each query structure is presented. Figure 4.4 shows the running time for each algorithm. There is no significant difference between the exhaustive search and the heuristic methods when the number of relations referenced in a query is less than 4. As for the exhaustive search method, when the number of relations is greater than 4, the time required grows exponentially, while the time required for the heuristic methods appears to grow much slower than the exhaustive search. Among the

heuristic methods, A requires more time than the others for most of the query structures, wheareas C requires less time than A, and D needs the least time for computing a heuristic solution.

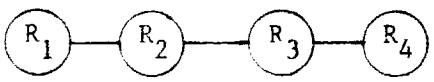
4.4. Tree Queries

When the assumptions being made in Chapter 3 for the approximation cost function are all valid, an optimal nesting order for a tree query can be solved efficiently. Algorithm B gives an optimal solution in polynomial time. In order to test the performance of algorithms A, and D when applied to tree queries, the following eight tree query structures were chosen as the input to the second set of experiments. For each query structure, 12 examples are tested.

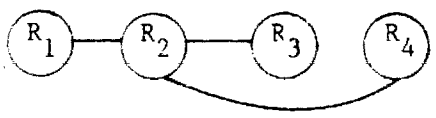
(1) Q1 :



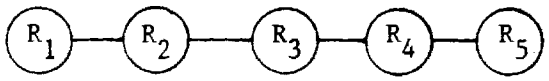
(2) Q2 :



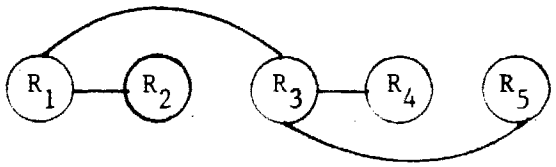
(3) Q3 :



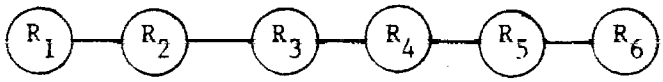
(4) Q4 :



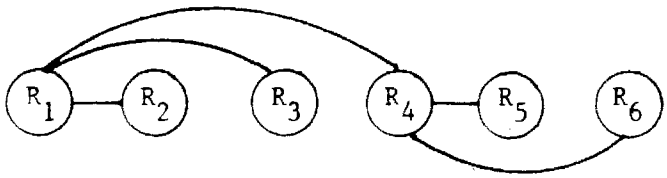
(5) Q5 :



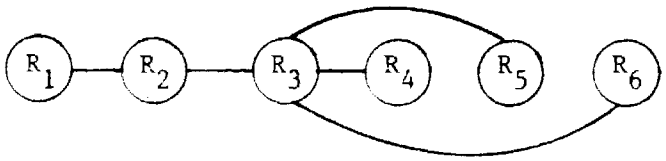
(6) Q6 :



(7) Q7 :



(8) Q8 :



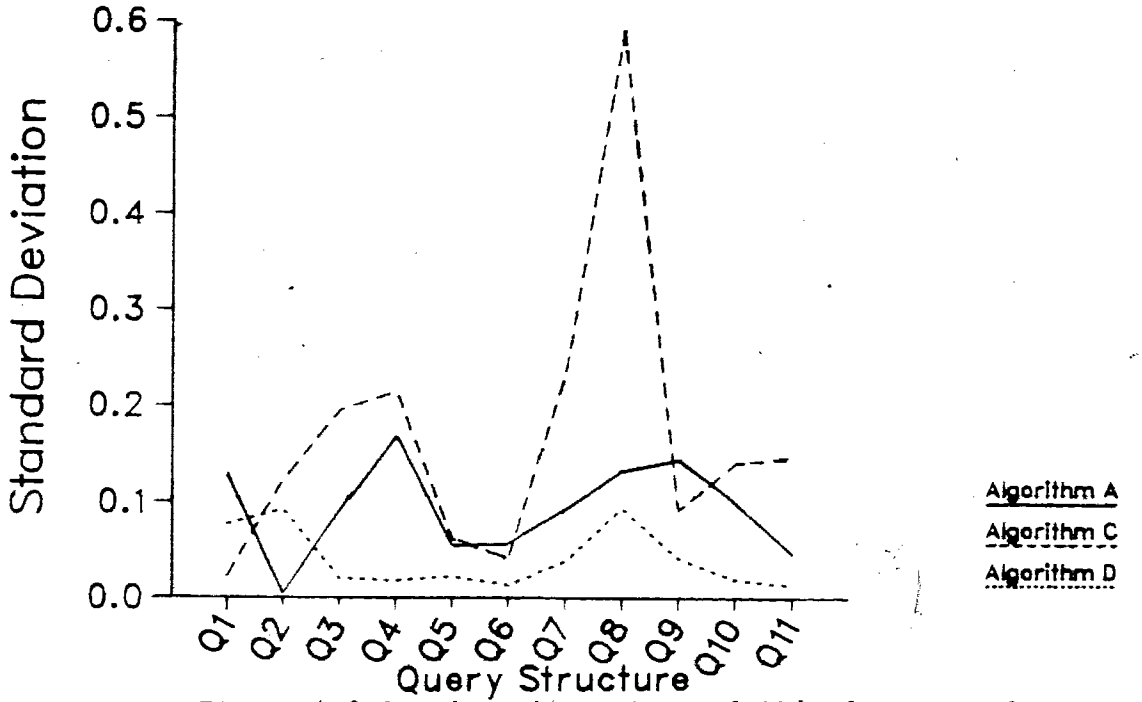


Figure 4.3 Standard deviations of PR's for general query structures : A, C, D

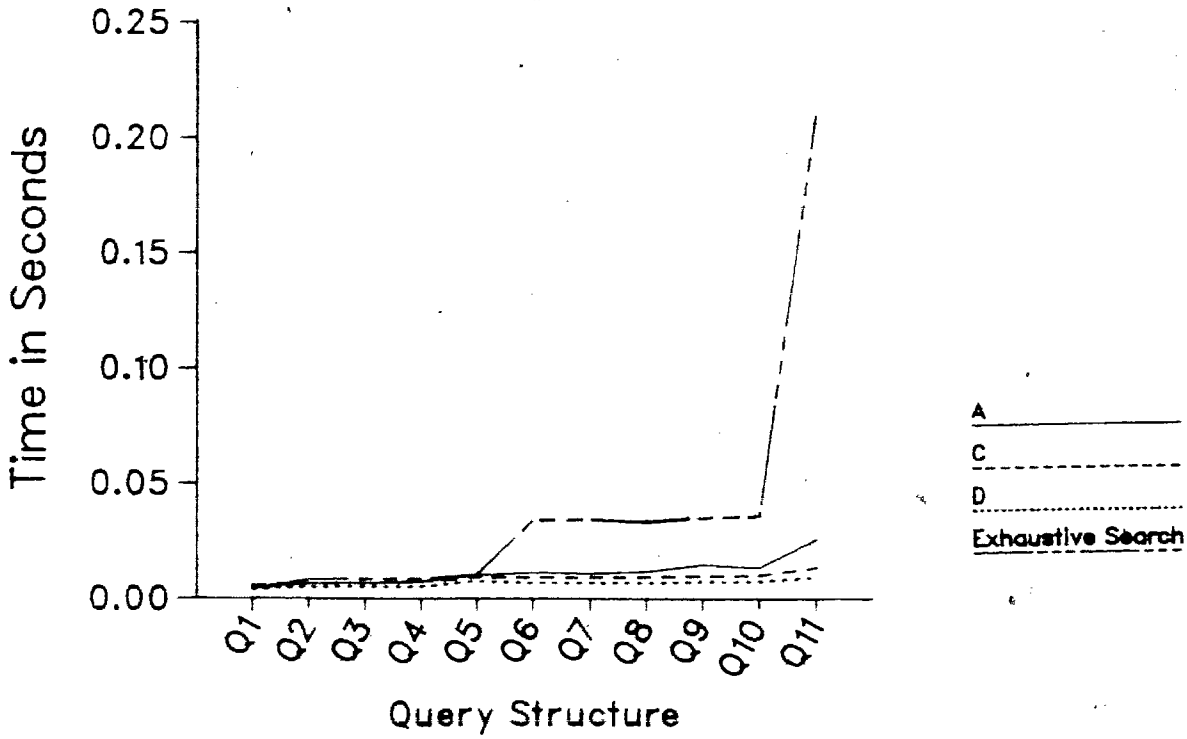


Figure 4.4 Average running time for samples used of each general query structure

4.4.1. Page Fetches

In this set of experiments, the heuristic solutions obtained from algorithms A and D are also normalized as PR's. The optimal cost in formula (4.1) is substituted by the optimal cost obtained from algorithm B. Thus, the same set of statistics (that is, PR range, average PR for each query structure, range of average PR's, standard deviation of PR's) was collected for analysis. The average PR's for different query structures are shown in Table 4.3 and 4.4.

A : 1 -- 169.5106
D : 1 -- 1.1707

Table 4.3 The ranges of PR's of algorithms A and D.

A : 1 -- 15.2193
D : 1 -- 1.0207

Table 4.4 Average PR ranges of algorithms A and D.

It is surprising to note that algorithm A does not behave as well as it does in processing general queries (see Figure 4.2). Both ranges are much larger than the ranges shown in Tables 4.1 and 4.2. This makes algorithm A relatively unattractive compared to algorithm D.

Since algorithm B gives optimal solutions for tree queries, its average PR's have value 1. Figure 4.5 is included to show how the average PR's of D behave relative to those values 1 of B (A is not shown because

the corresponding range is too large). In general, the average PR's of D increase as the number of nodes in the query graph increase. That is, when a query references fewer relations, the heuristic solutions obtained from D are more accurate. The worst case appearing for D in the experiments is only 18 percent more than the optimal solution.

The standard deviations of heuristic solutions obtained from each algorithm are computed. Since the optimal curve B does not fluctuate at all, the standard deviation for algorithm B is zero. Among algorithms A and D, A has very large range whereas D does not deviate very much. In the worst case, Q8, the average PR given by D deviates only about 5 percent. Figure 4.6 shows the standard deviations of D in comparison with those of B.

4.4.2. Computation Time

Figure 4.7 shows the time required for running each algorithm. Algorithm A obviously requires more time than the others to produce a heuristic solution, whereas B and D need about the same amount of time. Generally the time required increases with the number of relations. However, as shown in Figure 4.7, the amount of time needed by algorithm A grows more rapidly than that required by the other algorithms.

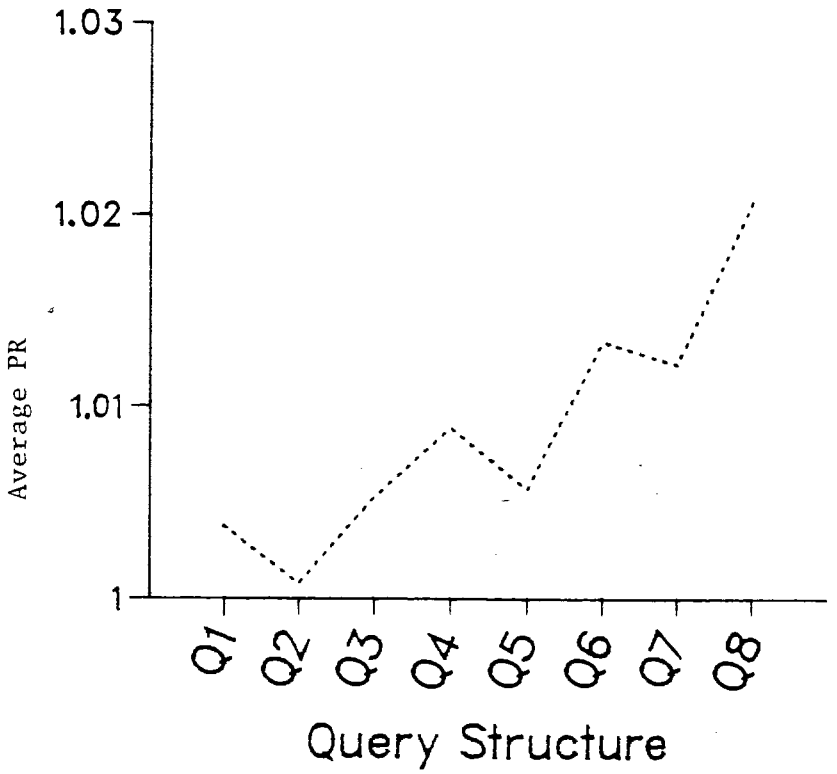


Figure 4.5 Average PR's for tree query structures : D

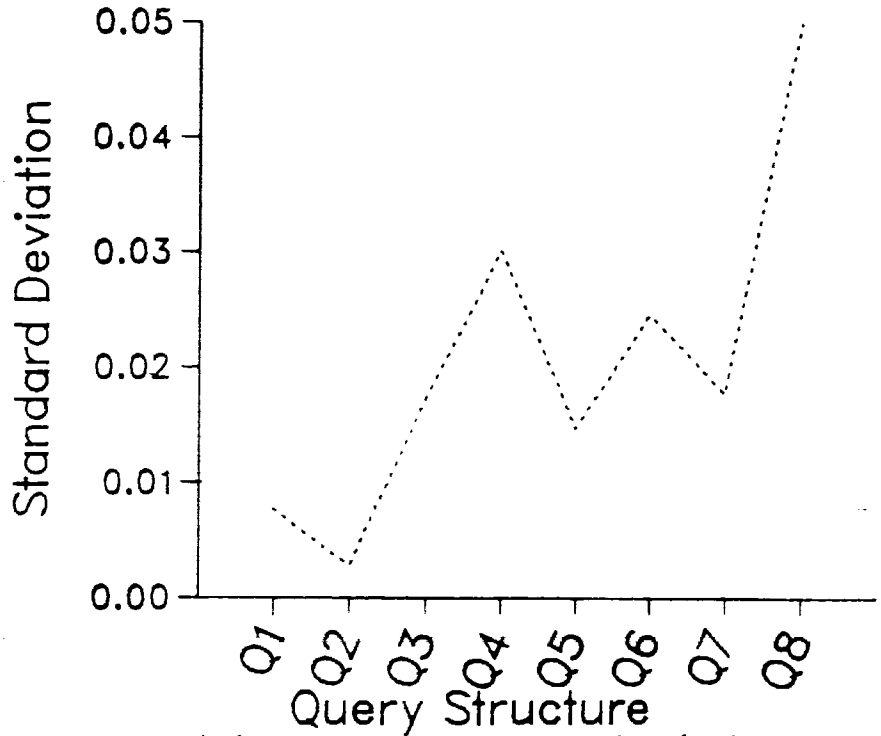


Figure 4.6 Standard deviations of PR's for tree queries : D

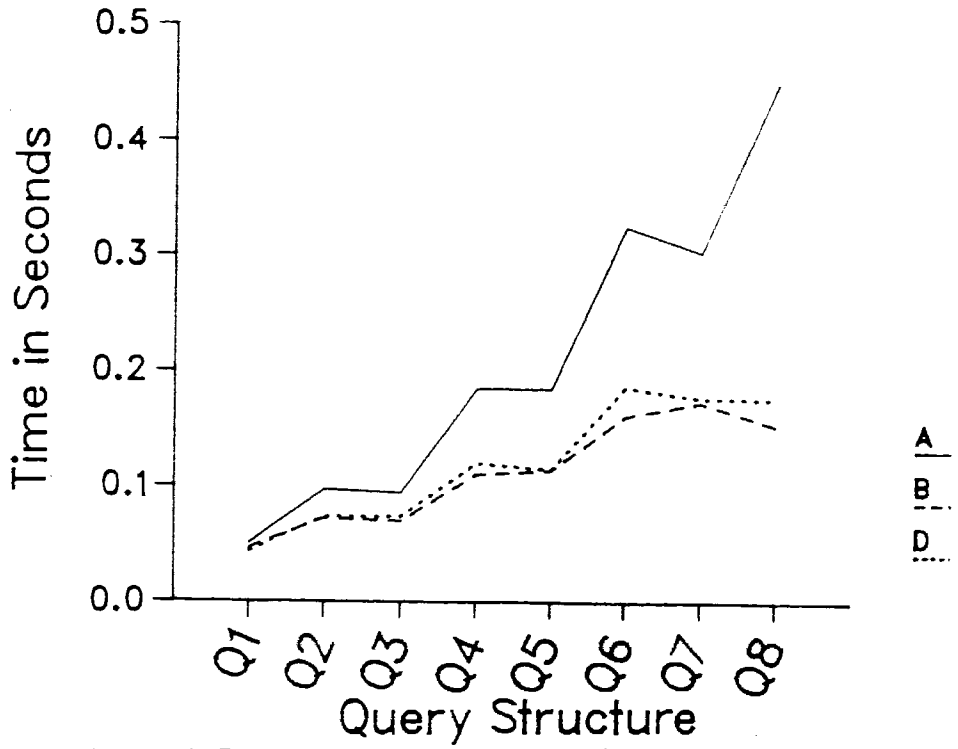


Figure 4.7 Average running time for samples used of each tree query structure : A, B, D

CHAPTER 5

IMPROVING A HEURISTIC SOLUTION

5.1. Interchanging Adjacent Relations

As the previous chapter has shown, the nesting orders of relations given by the four heuristic algorithms are not always optimal. Comparing the heuristic sequences with the optimal ones, we find that most of the heuristic sequences are near-optimal, in the sense that they are very similar to the optimal sequences except that a few relations are out of order. This suggests that if those adjacent relations in reversed order are interchanged, the near-optimal solutions could be improved or even upgraded to the desired optimal solutions. Thus, an optimal solution can be obtained with sufficient interchanges. For the purpose of achieving efficiency, the interchanging tactic must be done within a reasonable and acceptable time.

In this thesis, the interchanging tactic is applied as follows : given a sequence of relations corresponding to a heuristic solution, an adjacent pair of relations is interchanged. If the resulting sequence gives a smaller number of page fetches, the former sequence is discarded and the same process is repeated with the latter. Otherwise the latter sequence is ignored and a new adjacent pair of nodes in the former is chosen. Interchanging is repeated until all adjacent pairs have been tried without improvement in the number of page fetches. At this point the current sequence is considered as the solution.

There are many ways to find an adjacent pair of nodes in a sequence. Relations can be scanned, one by one, from left to right or from right to left. Assuming that the left to right method is used, the leftmost two adjacent relations are examined first. If they are not interchanged, we then try the second and the third relations, and so forth. If no pair is interchanged this process stops when the end of the sequence is reached. Suppose that a pair of adjacent relations is interchanged at some point (see Figure 5.1). The question is how to proceed from here. The left part of the sequence has been scanned without succeeding in reducing of cost. Therefore, it is not necessary to rescan the entire left subsequence as the interchanges involved do not alter its order of relations. We simply backtrack one position to the left and repeat the same process.

The left to right scanning method is adopted in the algorithm used in this thesis (shown in Appendix 9). The results of these experiments are shown in the next section.

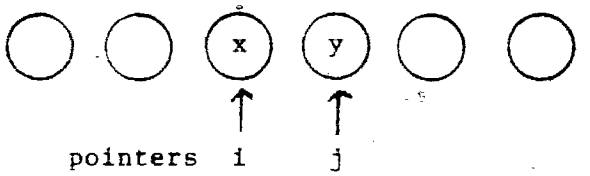
5.2. Performance of Heuristics with Interchanging

In order to compare the performance of the algorithms with and without interchanging, the same set of general query structures and data as used in Chapter 4 will be used again. Algorithm A with interchanging is referred to as A*, C with interchanging as C*, and so forth. Furthermore, the same parameters, such as the PR range, are computed for comparison.

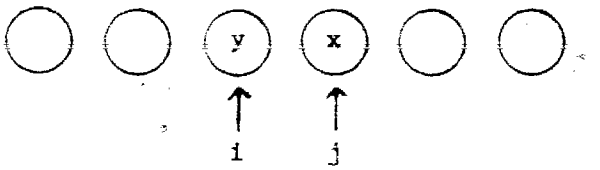
5.2.1. Page Fetches

Tables 5.1 and 5.2 show the PR ranges and the average PR ranges for the three algorithms with interchange. The ranges are all strictly smaller

Current sequence of relations:



After interchanging of relations x and y:



Backtrack one position to the left:

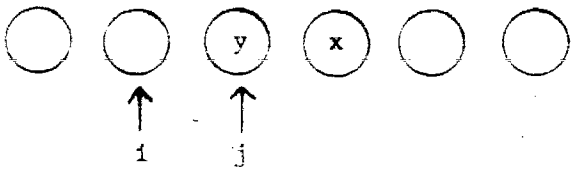


Figure 5.1 Starting position after interchanging

than the corresponding ranges in Tables 5.1 and 5.2.

A* : 1 — 1.5457
C* : 1 — 2.6531
D* : 1 — 1.3540

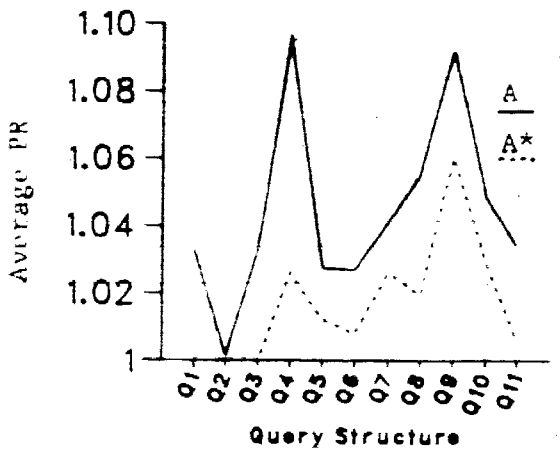
Table 5.1 The ranges of PR's of various algorithms after interchanging

A* : 1.0000 — 1.0607
C* : 1.0017 — 1.0999
D* : 1.0000 — 1.0308

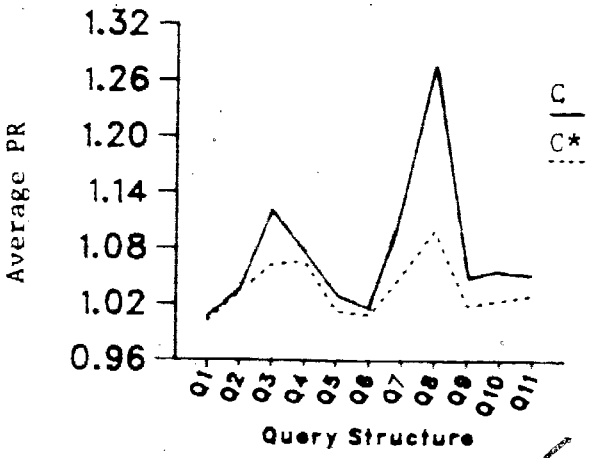
Table 5.2 Average PR ranges of various algorithms after interchanging

Figures 5.2(a,b,c), with different scales on the Y axes, graphically show the changes in the average PR's for different query structures. The solid curves correspond to the algorithms without interchange. The dotted curves correspond to the algorithms with interchanging. It is seen that all the dotted curves are below the solid curves. The amount of reduction in the average PR's is also significant. For example, the average PR of Q8 for C is reduced from 1.2810 to 1.0999.

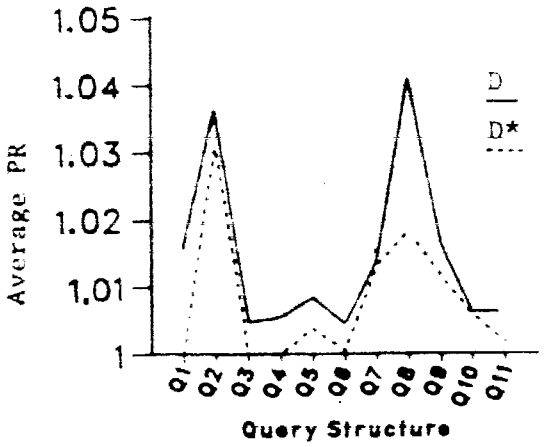
In Figure 5.2(a) it is seen that the average PR for A* for each query structure is smaller than the corresponding average PR for A. The amount of reduction is as large as 0.07. The curves D and D* in Figure 5.2(c) show the amount of reduction in the average PR's is not as significant as for A. However, putting curves A* and D* in the same figure (Figure 5.3),



(a)



(b)



(c)

Figure 5.2 Average PR's of solution after interchanging

curve D* still has the smaller average PR's, although there is one exception (for Q2). (A similar situation occurs in Figure 4.2). For query structure Q2, the difference in the average PR's for D* and A* is only about 0.03, which is not significant in the overall performance of algorithm D*.

The standard deviations of the PR's for the algorithms with interchanging are shown in Figures 5.4(a,b,c). The dotted curves show that the standard deviations of the average PR's decrease as the average PR's in Figures 5.2 decrease. There are many query structures--such as Q1, Q2, and Q3 for A*, Q1 for C*, Q1, Q3, Q4, and Q6 D*--having the standard deviations reduced to zero or near-zero. This means that the variations of average PR's obtained by the algorithms with interchange are very small. In most cases the average PR's decrease by a great amount. Figure 5.5 shows the relationships of the three standard deviation curves for A*, C*, and D*. Algorithm D* still has the best performance. A* is the runner-up and C* is the third.

5.2.2. Computation Time

The amounts of time required for running A*, C*, and D* are shown in Figures 5.6(a,b,c) respectively. The gap between the two curves in a figure is the amount of time required to implement interchanging. The gaps show that, on the average, the application of interchanging to a sequence requires only 0.005 seconds. However, this extra time has different relative meaning to the time shown by the solid curves. The Table 5.3 shows the relative relationships of the dotted and solid curves of each algorithm.

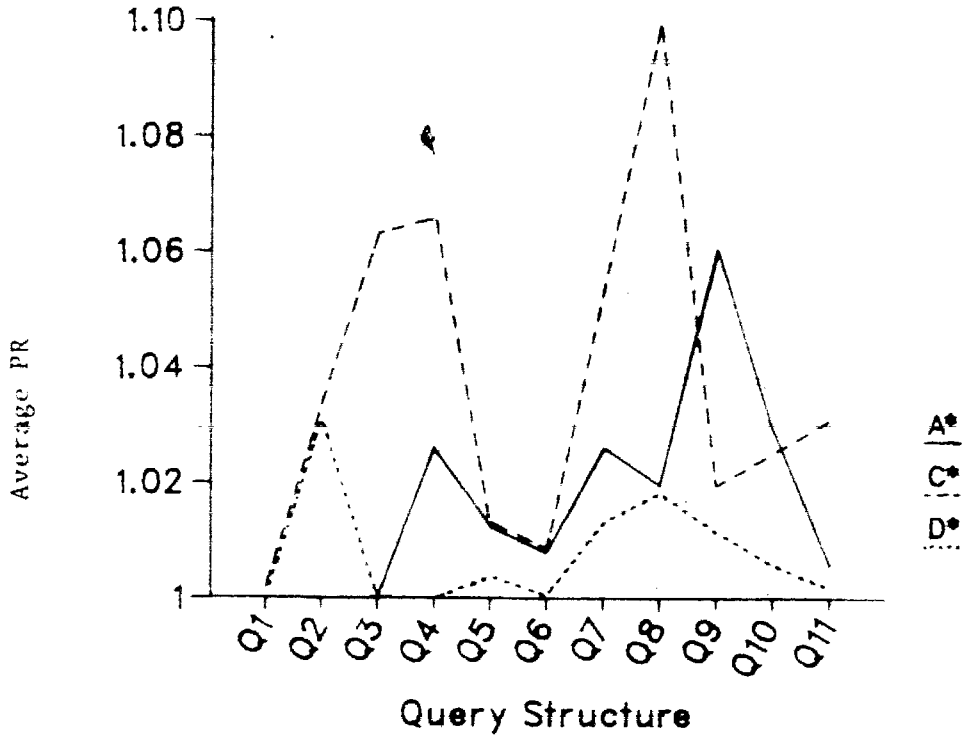
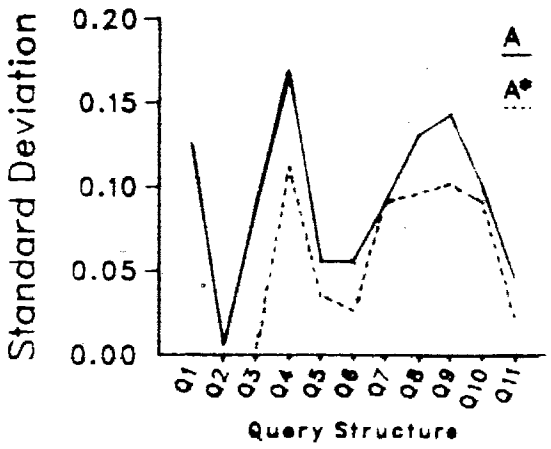
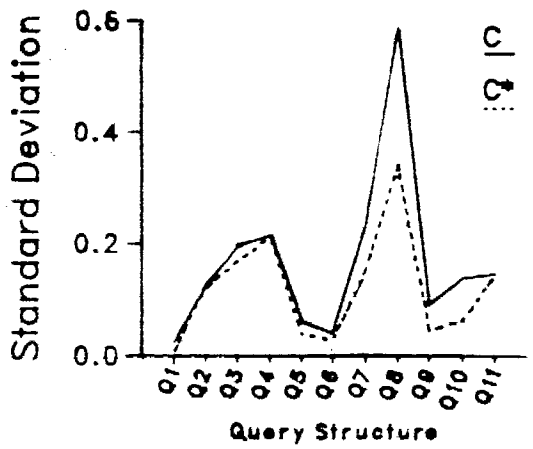


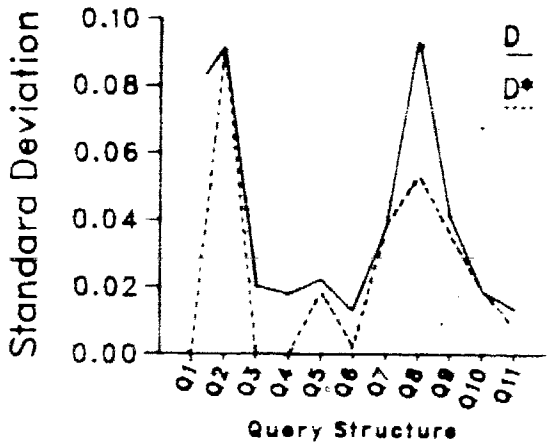
Figure 5.3 Average PR's of various algorithms with interchanging



(a)



(b)



(c)

Figure 5.4 Standard deviations of PR's of solutions after interchanging

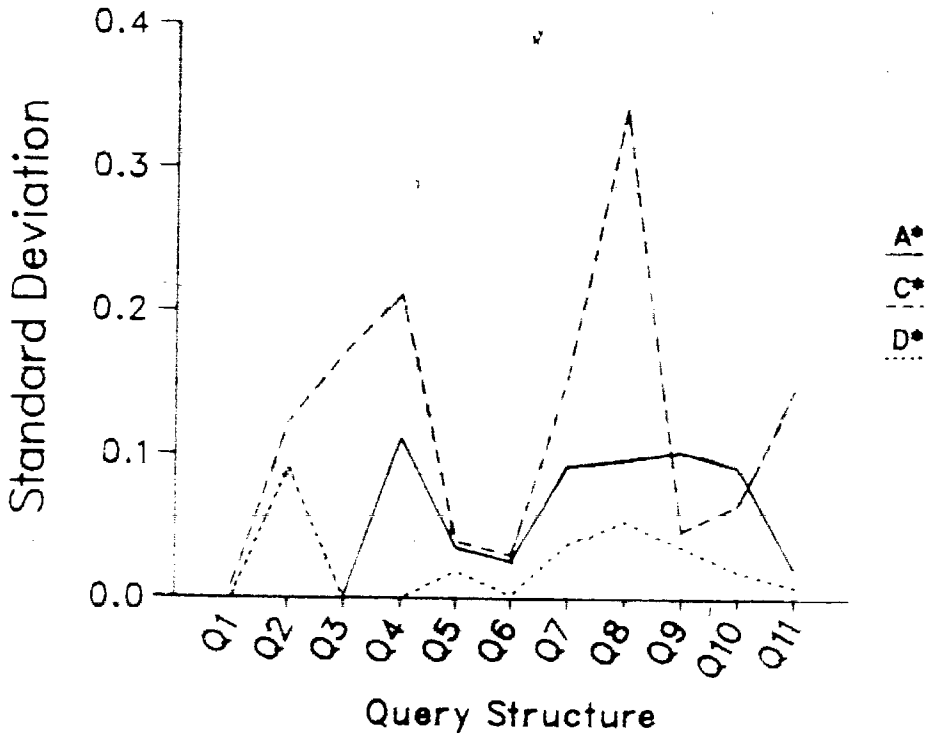
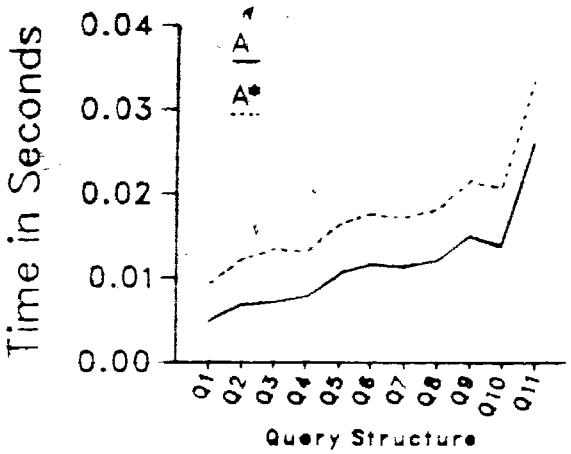


Figure 5.5 Standard deviations of various algorithms with interchange

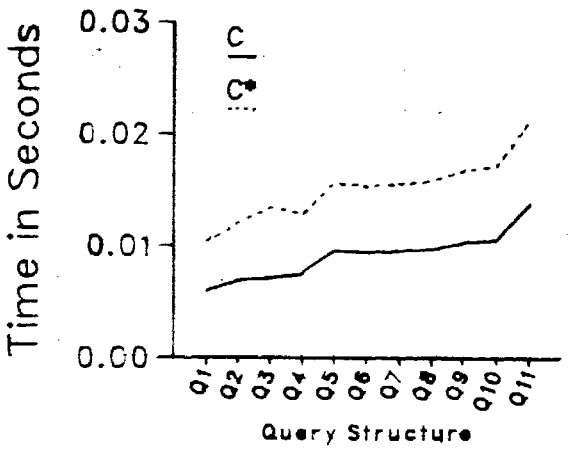
	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Q11
A*	92	75	89	68	56	51	52	51	43	48	29
C*	74	75	89	70	64	64	64	63	63	65	54
D*	106	99	110	95	75	83	84	87	84	85	74

Table 5.3 Time (in percent) needed for implementing interchange.

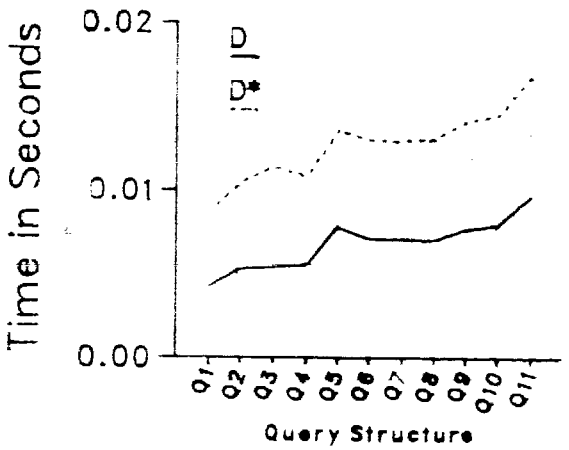
The amount of time (in percentage) needed for implementing the interchange scheme varies for each query structure. The time needed for interchanging can be as high as 110 percent and as low as 29 percent of the running time of the heuristics. However, this does not affect the overall performance of the interchange algorithm because as the number of relations referenced in a query structure increases, the percentage of time needed for interchanging decreases. On the other hand, when compared to exhaustive search, the extra time for improving the heuristic solution is worthwhile. Figure 5.7 clearly shows that, if the number of relations involved in a query structure is greater than 4, the running time of the heuristic algorithms with interchanging is much less than the running time of the exhaustive search. Therefore, the price of applying the interchanging tactic is not significant in comparison with the running time of exhaustive search.



(a)



(b)



(c)

Figure 5.6 Running times of algorithms with and without interchange

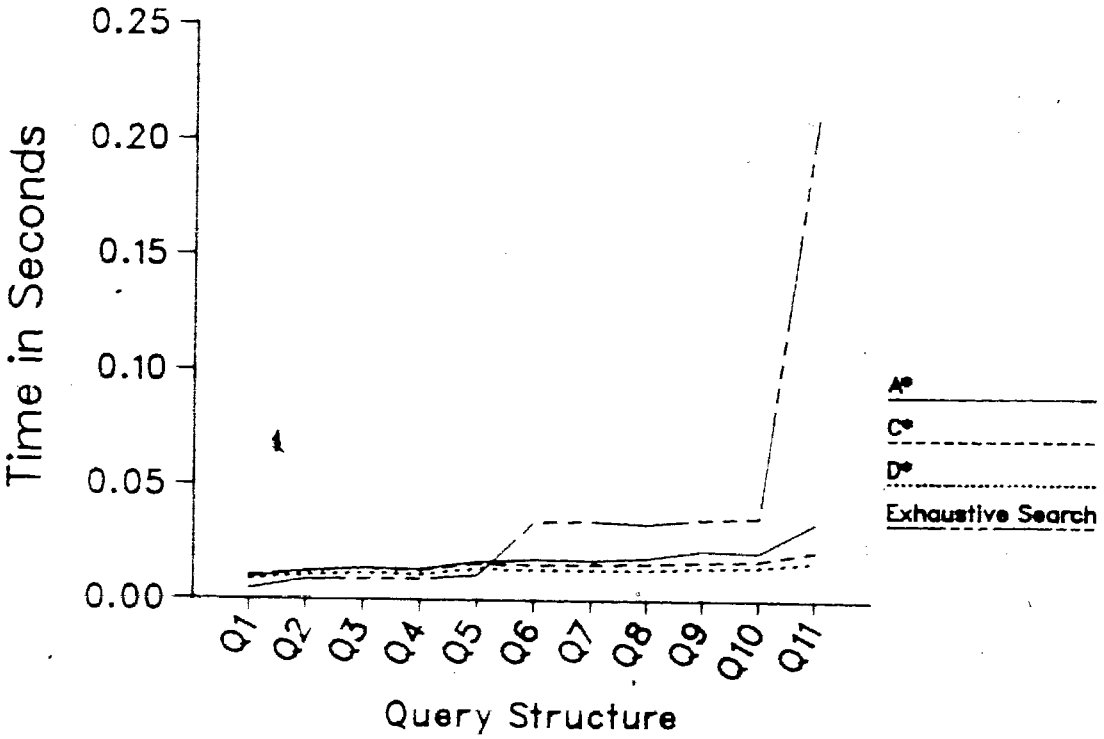


Figure 5.7 Running times of algorithms with interchange of relations

CHAPTER 6

COMPARISON OF NESTED LOOPS METHOD WITH SORT-MERGE METHOD

6.1. Assumptions

In this chapter a set of experiments was conducted for comparing the nested loops method (NLM) with the sort-merge method (SMM) [BLAS-77] which is described in Chapter 2. We assume that there is no special access path (such as index) available. In the sort-merge method, the relations (say R_i and R_j) referenced in a predicate are first sorted, using the 4-way merge sort, and then merged into one relation (say R_k), which is called an intermediate relation. The size of the intermediate relation, R_k, can be computed as follows. Let $t_i(t_j)$ be the length of each tuple of relation $R_i(R_j)$. After joining a tuple pair from R_i and R_j the longest possible length of the tuple of R_k is t_i+t_j. It is assumed that a page can hold s bytes. Thus we need $N_k(t_i+t_j)/s$ pages to store N_k tuples. The parameters of R_k are obtained as follows :

$$N_k = \lceil f_{P_i} N_i N_j \rceil$$

$$M_k = \lceil N_k((t_i+t_j)/s) \rceil$$

$$= \lceil N_k(((sM_i/N_i)+(sM_j/N_j))/s) \rceil$$

$$= \lceil N_k(M_i/N_i + M_j/N_j) \rceil$$

The cost (in terms of the number of page fetches) for scanning the entire relation R_i is M_i, since all the data pages of R_i must be fetched once and only once. To perform a merging operation on the two relations, R_i and R_j, both of them need be scanned once since they are sorted on the join attribute. Hence, the merging cost in the number of page fetches is :

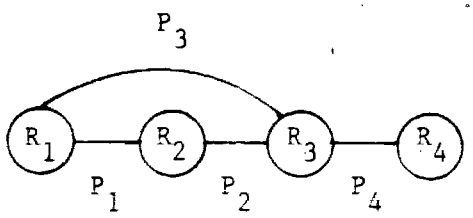
$$\text{merge cost} = M_i + M_j.$$

We assume that a given query has already been preprocessed, i.e., there is no one-relation predicate involved in the given query. However, in the sort-merge method, after some merging operations are performed, some one-relation predicates may be produced (see Figure 6.1), and the sizes of the intermediate relations resulting from this type of predicates must be treated differently from the N_k and M_k described above, for the resulting tuples do not lengthen and the number of tuples per page is not changed. Suppose, for example, that the one-relation predicate is $P_1=(R_1.A=R_1.B)$ with the selectivity factor f_{P_1}. Relation R_1 has N_1 tuples and M_1 pages. The sizes of the resultant relation R'_k after performing P_1 are given as follows :

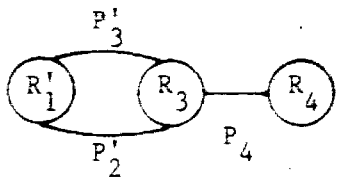
$$N'_k = \lceil f_{P_1} N_1 \rceil$$

$$M'_k = \lceil f_{P_1} M_1 \rceil$$

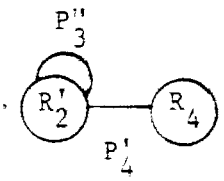
Given the following query graph :



After P_1 is computed :



After P'_2 is computed :



P''_3 is now an one-relation predicate.

Figure 6.1 One-relation predicate yielded from merging of relations

Therefore, there are two types of predicates to be considered :

(1) Predicate referencing two relations, R_i and R_j.

Cost for evaluating this type of predicate

= sort cost of R_i + sort cost of R_j + merge cost

(2) Predicate referencing only one relation, R_i.

Cost for evaluating this type of predicate

= scan cost

In the experiments the same samples as we used in the previous sections were used. The costs used in the comparisons are those of the optimal solutions for the nested loops method and the sort-merge method. Therefore, exhaustive search was applied to both the nested loops and the sort-merge methods. In the sort-merge method, all the possible orders of evaluating the predicates were attempted in order to find the best order. The costs for the two methods are compared by the relative cost (rc).

$$rc = \frac{\text{cost for sort-merge method solution}}{\text{cost for nested loops method solution}}$$

The relative cost can be any positive value. If it is greater than 1, the cost for the sort-merge method solution is greater than that for the nested loops method solution. Similarly, if the rc is smaller than 1 the former cost is smaller than the latter. The results of the experiments are shown in the next section.

6.2. Results

6.2.1. Page Fetches

Figure 6.2 shows the average relative cost for each query structure. The solid curve corresponds to the sort-merge method. The dotted curve corresponds to the nested loops method. These curves do not overlap each other. The dotted curve, which is a straight horizontal line at rc=1, is well below the solid curve. This shows that the costs for the optimal solutions obtained by the sort-merge method are larger than the corresponding costs by the nested loops method for all the over 200 samples used. The smallest average rc (for Q9) of the solid curve is 2. In other words, the smallest average cost obtained by the sort-merge method is about double the corresponding cost by the nested loops method. The most unsatisfactory case is Q6. Its average cost by the sort-merge method is about 6 times that by the nested loops method.

Based on the assumptions of this set of experiments, the cost for sorting a relation is larger than that for merging two relations. Suppose that there are m predicates and n relations in a given query. In the sort-merge method, at most m mergings but at least n sortings are necessary. However, in the nested loops method, only n-1 sortings are required and the number of page fetches needed for each relation (i.e., H_i) is kept small. Therefore, due to the fact that the cost needed for evaluating a query by the sort-merge method is dominated by the sort cost, it is not surprising to see the results shown in Figure 6.2. If the sort cost is reduced, the total cost is also reduced significantly. To improve the cost of the sort-merge method, we can use a larger z merge factor, say

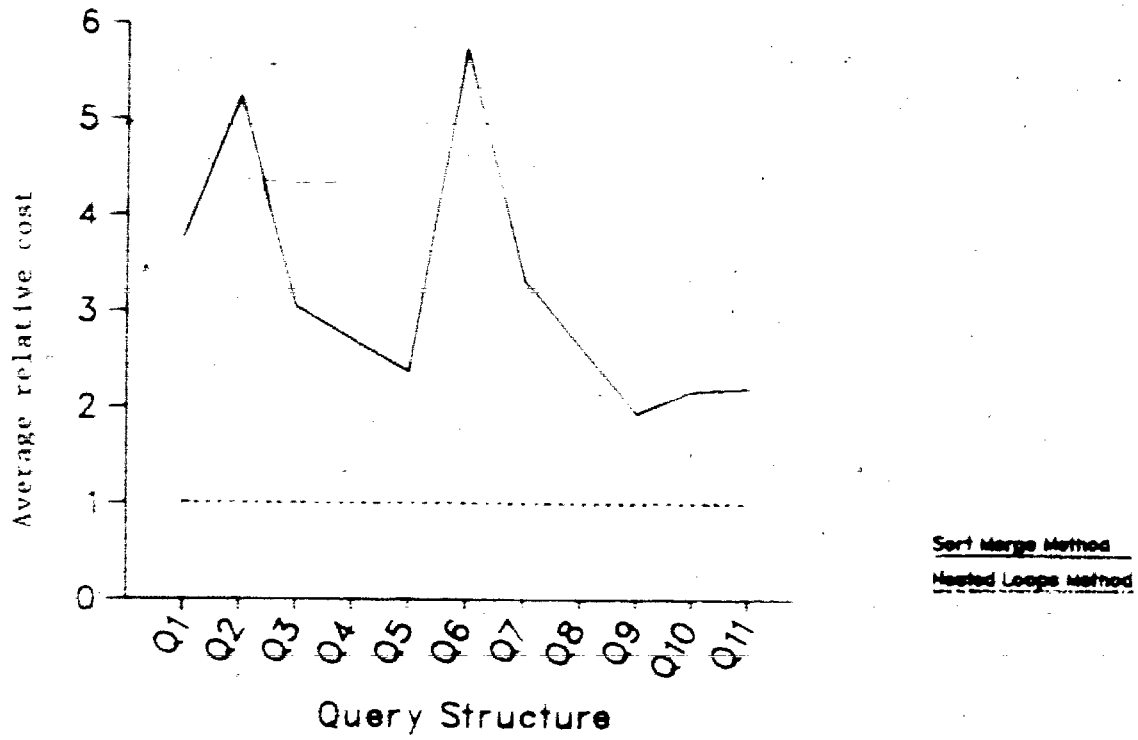


Figure 6.2 Average relative costs for SMM and NLM solutions

z=8, in the sorting. Figure 6.3 shows sorting cost for the z-way merge sort for different values of z. It is seen that as z increases the sort cost decreases. However, as we double z, say from 4 to 8, the sort cost is not reduced by half. Therefore, the solid curve in Figure 6.2 would not come down to half its height through this change. Thus the solid curve would still stay above the dotted curve. Furthermore, if an 8-way merge sort is also used in the nested loops method, the cost required for evaluating a query will be reduced as well (the relative costs remain as 1's). Therefore, the total cost of the sort-merge method will be still higher than that of the nested loops method for all the samples tested.

6.2.2. Computation Time

Figure 6.4 shows the amounts of time required for running the sort-merge and nested loops algorithms. The dotted curve corresponds to the nested loops algorithm with exhaustive search for the optimal nesting order. This curve is the same as exhaustive search curves shown in Figures 4.4 and 5.7 (except that the scale is different). It is seen that the amount of time required by the sort-merge algorithm grows more rapidly than that needed by the nested loops algorithm.

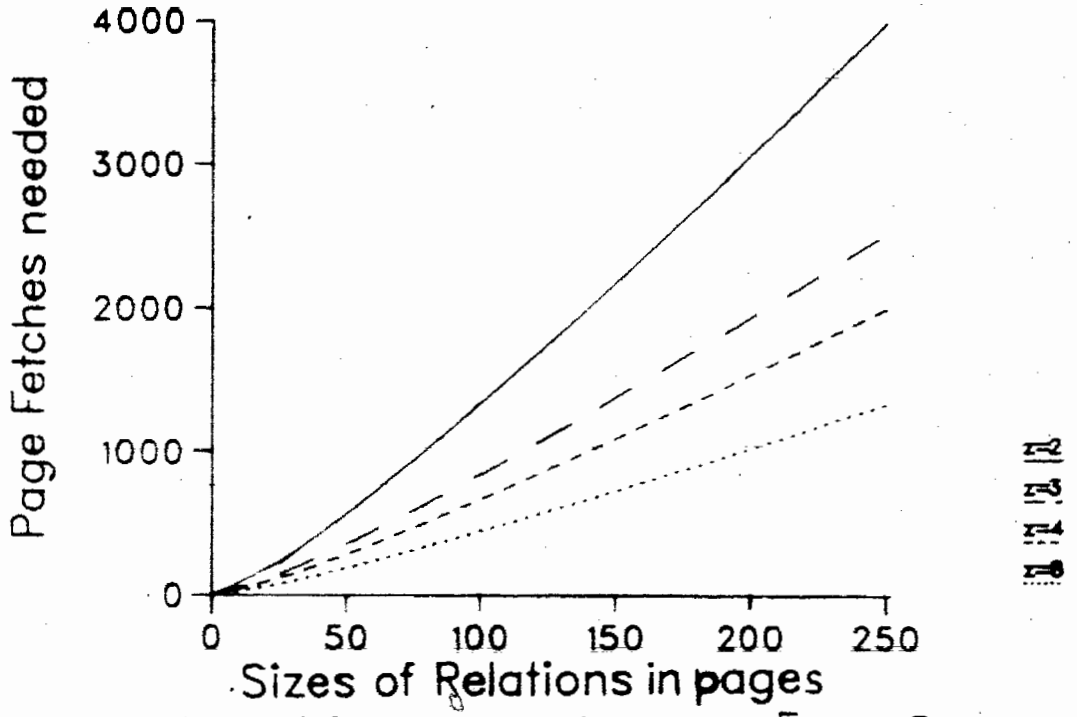


Figure 6.3 Sorting cost function $f=2\lceil M_i \log_z M_i \rceil$

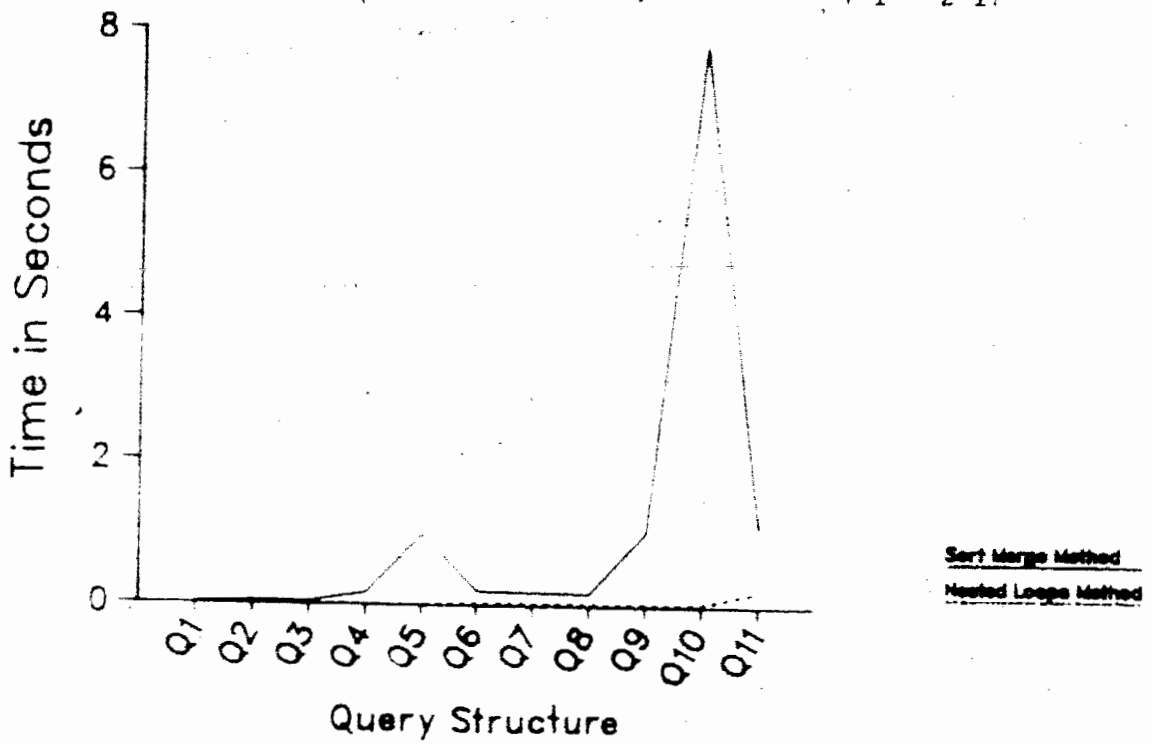


Figure 6.4 Running times for sort-merge and nested loops methods

CHAPTER 7

CONCLUSIONS AND OPEN PROBLEMS

7.1. Conclusions

From the results obtained in previous Chapters, we come to the following conclusions :

(1) Among the heuristic algorithms A, C, D, and D2, D gives the best heuristic solutions for both general and tree queries. The running time of D does not grow rapidly as the number of relations involved in a query increases. Ties in choosing R_i do not happen frequently.

(2) Algorithm A is the runner-up. It gives acceptable heuristic solutions to all of the general query samples used. But for tree queries, some unacceptable heuristic solutions are produced. This makes the algorithm unattractive compared to D. The amount of running time required for A is larger than that for D. Ties occur more frequently than for D.

(3) Algorithm C is the third best heuristic. It produces both acceptable and unacceptable heuristic solutions. However, unacceptable heuristic solutions occur more frequently than in A. Ties do not occur as frequently as they do in A. The running time is shorter than A but longer than D.

(4) Algorithm D2 is considered the least desirable algorithm. The heuristic solutions given are not stable in the sense that they are sometimes as good as those given by other algorithms, and sometimes much larger than the acceptable solutions.

(5) The heuristic solutions are generally improved with the application of interchanging of adjacent relations. The overall ranking of the performance of the heuristic algorithms do not change, i.e., D is still the best whereas A is the second best and so forth. The amount of time needed for implementing the interchange scheme varies in each query structure. However, as the number of relations referenced in a query structure increases, the percentage of time for interchanging decreases.

(6) The evaluation of nested loops and sort-merge methods is based on the assumptions that the selectivity factors are very small and the unit of transfer between main storage and secondary storage is a page. The results show that the optimal solutions for the nested loops method tend to give the smallest number of page fetches in all the samples used. The optimal solutions for the sort-merge method are at least double those for the former. The running time of the former method is much less than that of the latter.

(7) When the assumptions for the approximation function H_1 (3.1) are all valid, two special cases can be solved precisely. Using algorithm LQ, solutions for Algorithm 3R is applied to a general query referencing three relations. It gives the optimal nesting order of relations.

7.2. Open Problems

There are several open problems that can be looked into in future work.

(1) We use algorithm D to solve a general query. After proceeding through some stages, the remaining unordered relations form a tree query. Can the performance be improved if we use algorithm B to obtain the optimal solution of the tree query ?

(2) The cost function used in this work is derived under the assumption that the parameters, such as the values of attributes, are uniformly distributed. If a normal distribution is considered instead, what cost function do we obtain ?

(3) Much can be done on the comparison of the nested loops and sort-merge methods. This might occur when f_p is not very small, and more assumptions can be made for the sizes of the intermediate relations obtained after the merging (for example, one of the join fields can be eliminated from the composite tuple which has length t_i+t_j). Supposing that a sort-merge algorithm uses a block of k pages, instead of one page, as the unit of transfer between main storage and secondary storage, the sort cost function becomes $2\lceil(M/k)\log_z(M/k)\rceil$. How well do the nested loops and sort-merge algorithms perform ?

APPENDIX. I : NOTATION

1. $R_1, R_2, \ldots, R_n$: relations referenced in a given query.

2. N_i : number of tuples of R_i.

3. M_i : number of pages needed to store all tuples of R_i.

4. t_i : length (in bytes) of each tuple of R_i.

5. s : page size (in bytes).

6. $R_i.A$: attribute A of relation R_i.

7. $R_i.A\theta R_i.B$: one-relation predicate.

8. $P = R_i.A\theta R_j.B$, $i \neq j$: join predicate.

9. f_p : selectivity factor (the expected fraction of tuple pairs from R_i and R_j satisfying P).

10. pred(i) : $\{P \mid P='R_k.A\theta R_h.B', k, h \leq i\}$.

11. PRED(x,i) : $\{P \mid P='R_k.a\theta R_x.b', k \leq i\}$.

12. F_i : PRODUCT [FOR P IN pred(i)] OF f_p.

    (the product of all the selectivity factors associated with the predicates in pred(i)).

13. $F\{1,2,\ldots,i\}$: F_i.

14. $F\{1,2,\ldots,i|x\}$: PRODUCT [FOR P IN PRED(x,i)] OF f_p.

APPENDIX. 2 : ALGORITHM A

Algorithm A

Input:

Query Graph G(Q), the set R* of relations referenced in Q, and the predicates of Q together with their selectivity factors.

Output:

Nesting order and the directed spanning tree used for the nested loops method, together with the attribute of each relation on which it is to be sorted.

1. Construct the selectivity diagram SD(Q)=(V,E).

2. For each R∈R* let R_1=R and $\underline{R}$=R*-{R}, and repeat Steps 2.1 and 2.2.

   2.1. For i=n,n-1,...,3, repeat Steps a) and b).

   a) Choose R_i (from $\underline{R}$) and P_i which minimize H_i.

   b) Let $\underline{R}$=$\underline{R}$-{R_i} and delete from SD(Q) the node R_i and the edges incident on it.

   2.2. Let R_2 be the single remaining node in $\underline{R}$ and compute the cost C of (3.2).

3. Output the choices {$R_1,R_2,\ldots,R_n$} and {$P_2,\ldots,P_n$} that led to the minimum C in Step 2.2.

APPENDIX. 3 : ALGORITHM B .

procedure NORMALIZE(S);

commemt : S is the input sequence of strings;

while (starting from the beginning of S) there is a pair of adjacent strings, S_1 followed by S_2, in S such that $r(S_1) \geq r(S_2)$ do

begin replace S_1 and S_2 by a new string $(S_1 S_2)$ end;

Algorithm B

Input:

Tree query with a specified root R_1, the relations referenced in Q, their sizes, and the predicates of Q together with their selectivity factors.

Output:

Nesting order.

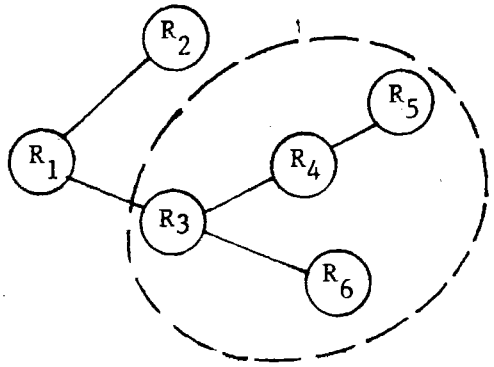
1. Let a string be a sequence of relation(s). Consider each node as a string.

2. If the tree is a single chain then stop (the desired nesting order is given by the chain).

3. Find a wedge (see example in next page) in the tree.

4. Apply NORMALIZE to the sequence corresponding to each chain of the wedge. The sequence of nodes corresponding to a component string in the output of NORMALIZE is considered as a unit (a supernode) from now on.

5. Merge the two chains into one by ordering supernodes by the ranks of the assiciated strings (from the smallest to the largest), and go to Step 2.



Example: The structure of a wedge is two chains connected by a node. A wedge is circled in the given tree.

APPENDIX. 4 : ALGORITHM C

Algorithm C

Input:

Query Graph G(Q), the set R* of relations referenced in Q, the predicates of Q together with their selectivity factors, and the associated minimum spannings tree of Q.

Output:

Nesting order and the directed spanning tree used for the nested loops method, together with the attribute of each relation on which it is to be sorted.

1. Construct the selectivity diagram SD(Q)=(V,E).

2. For each minimum spanning tree for SD(Q), do step 3 :

3. For each R∈R* repeat steps 3.1 and 3.2.

   3.1. Let root=R, apply algorithm B.

   3.2. Using the nesting order obtained in step 3.1 compute cost from formula (3.2).

4. Output the choices $\{R_1,R_2,\ldots,R_n\}$ and $\{P_2,\ldots,P_n\}$ that led to the minimum cost in step 3.2.

APPENDIX. 5 : ALGORITHM D

Algorithm D

Input:

Query Graph Q, the set R* of relations referenced in Q, the predicates of Q together with their selectivity factors.

Output:

Nesting order and the directed spanning tree used for the nested loops method, together with the attribute of each relation on which it is to be sorted.

1. Construct the selectivity diagram SD(Q)=(V,E).

2. For each $R \in R^*$ let $R_1=R$, $\underline{R}=R^*-[R]$, and $R'=[R]$, repeat steps 3 and 4.

3. For i=2 to n-1 repeat the following steps(3.1, 3.2, 3.3):

   3.1. For each $R_j \in \underline{R}$ do the following:

   3.1.1. Draw the graph Gp(i)=(Vp,Ep) from SD(Q)

   where $Vp=\{v \mid v \in R'+[R_j]\}$

   $Ep=\{(a,b) \mid (a,b) \in E \text{ and } a,b \in Vp\}$.

   3.1.2. Compute $FN = F\{1,2,\ldots,i-1 \mid j\} * N_j$

   where $F\{1,2,\ldots,i-1 \mid j\}$ = the product of the selectivity factors associated with the predicates in PRED(j,i-1),

   N_j = number of tuples in R_j

3.2. Choose f_{P_i} from pred(i)-pred(i-1) and compute H_i.

3.3. Let $\underline{R}=\underline{R}-[R_i]$ and $R'=R'+[R_i]$.

4. Let R_n be the single remaining node in $\underline{R}$, and f_{P_n} be the minimum selectivity factor chosen from pred(n)-pred(n-1).

Compute cost from formula (3.2).

5. Output the choices $\{R_1,\ldots,R_n\}$ and $\{P_2,\ldots,P_n\}$ that led to the minimum cost in step 4.

APPENDIX. 6 : ALGORITHM D2

Algorithm D2

Input:

Query Graph Q, the set R* of relations referenced in Q, the predicates of Q together with their selectivity factors.

Output:

Nesting order and the spanning tree used for the nested loops method, together with the attribute of each relation on which it is to be sorted.

1. Construct the selectivity diagram SD(Q)=(V,E).

2. For each R$\in$R* let R_1=R, $\underline{R}$=R*-[R], and R'=[R], repeat steps 3 and 4.

3. For i=2 to n-1 repeat the following steps(3.1,3.2):

   3.1. For each $R_j \in \underline{R}$ do the following:

   Compute Ratio(j) = $(F\{1,2,\ldots,i-1|j\}*N_j-1)/f_{P_j}M_j$

   where $F\{1,2,\ldots,i-1|j\}$ = the selectivity factors associated with the predicates in PRED(j,i-1),

   N_j = number of tuples in R_j,

   f_{P_j} = the minimum selectivity factor associated with the predicates in PRED(j,i-1).

   3.2. Choose the relation with minimum Ratio as R_i.

   Choose f_{P_i} from pred(i)-pred(i-1) and compute H_i.

   3.3. Let $\underline{R}=\underline{R}-[R_i]$ and $R'=R'+[R_i]$.

4. Let R_n be the single remaining node in $\underline{R}_n$ and f_{P_n} be the minimum selectivity factor chosen from pred(n)-pred(n-1).

   Compute cost from formula (3.2).

5. Output the choices $\{R_1,\ldots,R_n\}$ and $\{P_2,\ldots,P_n\}$ that led to the minimum cost in step 4.

APPENDIX. 7 : ALGORITHM 3R

Algorithm 3R

Input:

Query Graph G(Q), the set R* of relations referenced in Q, the predicates of Q together with their selectivity factors.

Output:

Nesting order and the spanning tree used for the nested loops method, together with the attribute of each relation on which it is to be sorted.

1. Construct the selectivity diagram SD(Q)=(V,E).

2. For each R∈R* do the following :

   2.1. Let R_a=R, and the remaining relations be R_b and R_c respectively.
   Let the predicate referencing R_a and R_b be P_x;
   the predicate referencing R_a and R_c be P_y;
   the predicate referencing R_b and R_c be P_z.

   2.2. Compute : $u=(f_{P_x} N_b \mathrm{MIN}(f_{P_y}, f_{P_z}) - f_{P_y})/M_b$;

   $v=(f_{P_y} N_c \mathrm{MIN}(f_{P_x}, f_{P_z}) - f_{P_x})/M_c$.

   2.3. If u < v pick $\{R_a R_b R_c\}$ as the desired order, otherwise pick the sequence $\{R_a R_c R_b\}$. Compute the cost using formula (3.2).

3. Output the choices $\{R_1, R_2, \ldots, R_n\}$ and $\{P_2, \ldots, P_n\}$ that led to the minimum cost in step 2.3.

APPENDIX. 8 : ALGORITHM LQ

Algorithm LQ

Input:

Query Graph G(Q), the set R* of relations referenced in Q, the predicates of Q together with their selectivity factors.

Output:

Nesting order and the directed spanning tree used for the nested loops method, together with the attribute of each relation on which it is to be sorted.

1. Let the query graph be G(Q)=(V,E). For each e∈E, do the following :

   1.1. Delete e from G(Q).

   1.2. For each R∈R* do Steps a) and b).

   a) Let root=R, apply algorithm B.

   b) Compute cost from (3.2) by using the order output from step a).

2. Output the choices $\{R_1,R_2,\ldots,R_n\}$ and $\{P_2,\ldots,P_n\}$ that led to the minimum cost in Step 1.2b).

APPENDIX. 9 : INTERCHANGE

Algorithm Interchange

Input:

A sequence of relations S_0 (a heuristic solution) together with its cost C_0 (in number of page fetches).

Output:

Nesting order and the corresponding cost.

1. Let sequence $S_1=S_0$ and cost $C_1=C_0$.

2. Starting from the leftmost relation of S_1, let the pointers i and j point to the first and second relations respectively.

3. Interchange the relations of S_1 pointed to by i and j and obtain sequence S_2.

4. Compute cost C_2 from formula (3.2).

5. If cost C_1 is smaller than or equal to cost C_2, discard S_2 and C_2, advance the pointers of S_1 one position to the right. Go to step 6 if the end of the sequence S_1 is reached. Otherwise, repeat the process from step 3. If cost C_1 is larger than cost C_2, let $S_1=S_2$, $C_1=C_2$, j=i, and i=i-1. Repeat the process from step 3.

6. Output sequence S_1 and cost C_1.

REFERENCES

[AHO-79]
Aho, A.V., C. Beeri, & J.D. Ullman, The Theory of Joins in Relational Databases, ACM Trans. on Database Systems, Sept. 1979, pp.297-314.

[ASTR-80]
Astrahan, M.M., et. al., Performance of the System R Access Path Selection Mechanism, Information Processing 80, 1980, pp.487-491.

[BLAS-77]
Blasgen, M.W. & K.P. Eswaran, Storage Access in Relational Data Bases, IBM System Journal, Vol. 16, 1977, pp.363-378.

[CODD-70]
Codd, E.F., A Relational Model of Data for Large Shared Data Banks, Comm. ACM, Vol.13, June 1970, pp.377-387.

[DEMO-80]
Demolombe, R., Estimation of the Number of Tuples satisfying a Query Expression in Predicate Calculus Language, Proc. 6th Inter. Conf. on Very Large Data Bases, Oct. 1980, pp.55-62.

[IBAR-82]
Ibaraki, T. & T. Kameda, On The Optimal Nesting Order for Computing N-Relational Joins, Technical Report TR 82-15, Department of Computing Science, Simon Fraser University, 1982. To appear in ACM Trans. on Database Systems, 1984.

[KIM-80]
Kim, W., A New Way To Compute The Product and Join of Relations, Proc. ACM-SIGMOD Inter. Conf. on Manag. of Data, 1980, pp.179-187.

[MERR-83]
Merrett, T.A., Why Sort-Merge Gives the Best Implementation of the Natural Join, ACM-SIGMOD Record, Jan. 1983, pp.39-51.

[PERC-76]
Percherer, R.M., Efficient Exploration of Product Spaces, Proc. ACM-SIGMOD Inter. Conf. on Manag. of Data, 1976, pp.169-177.

[ROSE-80]
Rosenkrantz, D.J. & H.B. Hunt, Processing Conjunction Predicates and Queries, Proc. 6th Inter. Conf. on Very Large Data Bases, Oct. 1980, pp.64-72.

[SELI-79]
Selinger, P.G., et. al., Access Path Selection in a Relational Database Management Systems, Proc. ACM-SIGMOD Inter. Conf. on Manag. of Data, May 1979, pp.23-34.

[STON-76]
Stonebraker, M., et. al., The Design and Implementation of INGRES, ACM Trans. on Database Systems, Vol.1, Sept. 1976, pp.189-222.

[WONG-76]
Wong, E. & K. Youssefi, Decomposition - A Strategy for Query Processing, ACM Trans. on Database Systems, Vol.1, Sept. 1976, pp.223-241.

[YOUS-78]
Youssefi, K. & E. Wong, Query Processing in a Relational Database Management System, University of California, Berkeley. College of Engineering, March 1978.

[YAO-79]
Yao, S.B., Optimization of Query Evaluation Algorithms, ACM Trans. on Database Systems, Vol.4, June 1979, pp.133-155.