A C++ IMPLEMENTATION AND EVALUATION OF ALTERNATIVE PLAN GENERATION METHODS FOR MULTIPLE QUERY OPTIMIZATION

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

DILIXIATI ABUDULA

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
COMPUTER ENGINEERING

NOVEMBER 2006

Approval of the Graduate School of Natural and Applied Sciences

Prof. Dr. Canan Özgen
Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

Prof. Dr. Ayşe Kiper
Head of Department

This is to certify that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.

Assoc. Prof. Dr. Ahmet Coşar
Supervisor

Examining Committee Members

Prof. Dr. Adnan Yazıcı	(METU, CENG)	
Assoc. Prof. Dr. Ahmet Coşar	(METU, CENG)	
Prof. Dr. Özgür Ulusoy	(BILKENT, CENG)	
Prof. Dr. Faruk Polat	(METU, CENG)	
Assoc. Prof. Dr.İsmail Hakkı Toroslu	(METU, CENG)	

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work

Name. Last Name : Dilixiati Abudula

Signature :

ABSTRACT

A C++ IMPLEMENTATION AND EVALUATION OF ALTERNATIVE PLAN GENERATION FOR MULTIPLE QUERY OPTIMIZATION

ABUDULA, Dilixiati
MS, Department of Computer Engineering
Supervisor: Assoc. Prof. Dr. Ahmet Coşar

November 2006, 53 pages

In this thesis, alternative plan generation methods for multiple query optimization(MQO) are introduced and an implementation in the C++ programming.language has been developed. Multiple query optimization, aims to minimize the total cost of executing a set of relational database queries. In traditional single query optimization only the cost of execution of a single relational database query is minimized. In single query optimization a search is performed to investigate possible alternative methods of accessing relational database tables and alternative methods of performing join operations in the case of multi-relation queries where records from two or more relational tables have to be brought together using one of the join algortihms (e.g. nested loops, sort merge, hash join,etc). The choice of join method depends on the availability of indexes, amount of available main memory, the existence of ORDER BY clause for sorted output, the sizes of involved relations, many other factors. A simple way of performing multiple query optimization is to take the query execution plans generated for each of the queries as input to a MQO algorithm, and then try to identify common tasks in those plans using the MQO algorithm. However, this approach will reduce the achievable benefits since a more expensive execution plan (thus discarded by a single query optimizer) could have more common operations with other query

execution plans, resulting in a lower total cost for MQO. .For this purpose we will introduce several methods for generating such potentially beneficial alternative query execution plans and experimentaly evaluate and compare their performances..

Keywords: Relational Database, Query Optimization, Multiple Query Optimization, Alternative Plan Generation.

ÖZ

ÇOKLU SORGU OPTİMİZASYONU İÇİN C++ DİLİNDE ALTERNATİF PLAN ÜRETME METOTLARININ GERÇEKLEŞTİRİLMESİ VE KARŞILAŞTIRILMASI

ABUDULA, Dilixiati

Yüksek Lisans, Bilgisayar Mühendisliği Bölümü

Tez Yöneticisi: Doç. Dr. Ahmet Coşar

Kasım 2006, 53 sayfa

Bu tezde çoklu sorgu optimizasyonu(ÇSO) için alternatif plan üretme yöntemleri araştırılmış ve C++ programlama dilinde kodlanarak gerçekleştirilmiştir. Çoklu Sorgu Optimizasyonu bir ilişkisel veritabanı üzerinde çalıştırılacak bir sorgu kümesinin toplam çalıştırılma süresini en aza indirmeyi amaçlar. Konvansiyonel tek sorgu eniyilemesinde sadece bir adet ilişkisel veritabanı sorgusunun işletme maliyetinin en aza indirilmesi amaçlanır. Tek bir sorgu eniyilemesi için ilişkisel veritabanında tablolara erişmek için kullabilecek alternatif yöntemler(sıradan, endeksli, vs.) ve aynı zamanda birden fazla tablo üzerinde sorgulama yapılıyorsa birleştirme (join) operasyonu için kullanılabilecek alternatif birleştirme yöntemleri (içiçe-döngü, sırala-birleştir, ve karıştır-birleştirme) incelenmeli ve en kısa zamanda sonucu verecek yöntemler seçilmelidir. Seçilecek birleştirme yöntemi elde bulunan endekslere, ana bellek miktarına, sorgu sonucunun SORTED BY ile sıralı olması istenmesi durumuna, üstünde işlem yapılacak tabloların büyüklüğüne, ve birçok başka faktöre bağlı olarak değişecektir. Çoklu sorgu eniyilemesi için basit bir yöntem, her sorgu için tek başına en iyi yöntemi belirlemek ve bir plan üretmek, sonra da bir ÇSO algoritması ile bu planlar arasındaki ortak görevlerin belirlenerek bütün sorguların cevaplarını üretecek bir ortak planın üretilmesidir. .Ancak bu yönemde elde edilebilecek yararlar sınırlı olacaktır, çünkü her bir sorgu için tek

başına en ucuz maliyetli olan plan aslında diğer planlarla olabilecek paylaşımları yeterince kullanmıyor olabilir, bu da olası en düşük maliyetli ortak planın bulunmasını engeleyecektir. Bu yöntemlerin arasında karşılaştırma yapmak ve hangisinin daha iyi sonuçlar ürettiğini görebilmek için deneyler yapılmıştır.

Keywords: İlişkisel Veritabanı, Sorgu Eniyilemesi, Çoklu Sorgu Eniyilemesi, Alternatif Plan Üretimi.

To my father and mother

ACKNOWLEDGEMENTS

I would like to express my deepest gratitude to my thesis supervisor Assoc. Prof. Dr. Ahmet Coşar for his guidance, advice, criticism, encouragements and insight throughout this research.

I would also like to thank to my thesis jury members for their valuable comments and suggestions.

TABLE OF CONTENTS

PLAGIARISM………………………………………………………………… iii
ABSTRACT…………………………………………………………………… iv
ÖZ…………………………………………………………………………... vi
DEDICATION……………………………………………………………… viii
ACKNOWLEDGEMENTS………………………………………………… ix
TABLE OF CONTENTS…………………………………………………… x
LIST OF TABLES………………………………………………………….. xii
LIST OF FIGURES………………………………………………………… xiii
CHAPTER
1. INTRODUCTION………………………………………………. 1
2. AN OVERVIEW OF QUERY OPTIMIZATION……………..… 6
2.1 Multiple query optimization formulation…………………... 6
2.2 A dynamic programming formulation………...……………. 8
2.2.1 General overview of dynamic programming……..…... 8
2.3 A dynamic programming formulation for finding optimal join orders…………………………………………………..... 11
2.3.1 A sample join order problem.………………………… 11
2.3.2 Plan generation for a query……………………...…… 12
2.4 Alternative plan generation……………………………… 14
2.4.1 Alternative plan generation for a query………………. 15
2.5 Join tree transformation operations………………………... 16
2.5.1 Commutativity transformation……………………….. 17
2.5.2 Associativity transformation…………………………. 17
2.5.3 Selection fragmentation transformation……………… 18
2.5.4 Projection fragmentation transformation…………….. 19

2.5.5 Selection and Projection propagation transformation 20
3. EXPERIMENTAL SETUP………………………………………… 22
3.1 Randomly generating queries……………………………… 23
3.2 The random query generation algorithm……………........... 24
3.3 Generating an optimal plan for a query……………………. 25
3.4 Generating alternative plans for a query…………………..... 25
3.5 Generating alternative plans for multiple queries 26
4. EXPERIMENTAL RESULTS…………………………………... 29
4.1 Introduction………………………………………………… 29
5. CONCLUSIONS AND FUTURE WORK………………...……. 35
5.1 Conclusions from experimental results…………………….. 35
5.2 Future works…………..……………………………………. 36
REFERENCES……………………………………………………... 38

LIST OF TABLES

Table 1 Task Description for Query 1………………………………… ….. 6
Table 2 Dynamic programming solution for the shortest path problem ….. 10
Table 3 Join cost estimation ……………..……………............................. 12
Table 4 Tasks in a multiple query optimization problem…………....……. 14
Table 5 Description of MQO plans ………………..……………...……. 14
Table 6 An example synthetically generated database ……………..……. 23

LIST OF FIGURES

Figure 1 An example SQL language statement to define a relational table.... 1
Figure 2 An example SQL language statement using INSERT command…. 2
Figure 3 An example SQL language statement using SELECT command… 2
Figure 4 A similar example with Q3……………………………..……….. 4
Figure 5 A plan for executing Query1……………………………..……….. 7
Figure 6 An example shortest path problem. …………………….………... 9
Figure 7 A sample query tree and its task generation...……......………….... 13
Figure 8 Pairwise alternative plan generation……………………………..... 16
Figure 9 A commutativity transformation example ……………………...... 17
Figure 10 An associativity transformation example …….…………….…..... 18
Figure 11 A selection fragmentation transformation example 19
Figure 12 A projection fragmentation example 20
Figure 13 A selection and projection propagation transformation example.. 21
Figure 14 A sample left-deep join tree………………..………………....… 25
Figure 15 The pairwise algorithm……………..………………………....… 26
Figure 16 The complete algorithm….……………………………………....… 27
Figure 17 The sharing factor heuristic...………………………………....… 28
Figure 18 APG times as number of queries increases (2 joins per query) ... 31
Figure 19 Multiplan quality as number of queries increases (2 joins per query) ……………………..……………………………………...…… 31
Figure 20 APG times as number of queries increases (3 joins per query)…. 32
Figure 21 Multiplan quality as number of queries increases (3 joins per query)……………………..…………………………………………....... 32
Figure 22 APG times as number of queries increases (4 joins per query) …. 33
Figure 23 Multiplan quality as number of queries increases (4 joins per query) ……….…………….………………………………………….…. 33

Figure 24 APG times as number of queries increases (5 joins per query)….. 34
Figure 25 Multiplan quality as number of queries increases (5 joins per query) ……………………………………………………………………….. 34

CHAPTER I

INTRODUCTION

Database management systems (DBMS) form the backbone of almost all commercial application software systems. A DBMS can be used for storing alphanumeric data as well as multimedia data such as music, human voice, pictures, and videos. The state of the art standard for DBMS are relational DBMS (e.g. Oracle, Sybase, MS SQL Server, IBM DB2, etc.) systems which store information in tables consisting of a number of columns and rows in the form of N-tuples where each column is used for storing information about an attribute[Ramakrishna 2003].

The standard way of defining and accessing relational databases is to use a standard query language, namely the SQL query processing language, which is implemented by all of the available commercial relational DBMSs, and has contributed largely to the success of relational DBMS software. An example SQL language statement to define a relational table is given below..

```
Q1:
CREATE TABLE STUDENTS(
   FIRSTNAME VARCHAR(10),
   LASTNAME VARCHAR(10),
   DEPARTMENT_ID INTEGER)
```

Figure 1: An example SQL language statement to define a relational table

Each table in a relational database must be given a unique name which is used to refer to that table. Each column in a table must also be given a name that is unique to that table. Once a table is defined and populated with information using INSERT commands as given below in Q2, it becomes possible to issue SELECT queries (see Q3) on that table.

```
Q2:

INSERT INTO STUDENTS
  VALUES("AHMET","COSAR", 1 )
```

Figure 2: An example SQL language statement using INSERT command

A query to find those students who are in department number 1, is given below.

```
Q3:

SELECT FIRSTNAME,LASTNAME
FROM STUDENTS
WHERE DEPARTMENT_ID=1
```

Figure 3: An example SQL language statement using SELECT command

The number of SELECT statements that can be answered by a DBMS critically determines the performance of that DBMS and the number of users/clients that can be served by the DBMS.

In order to increase the number of SELECT statements answered by a DBMS(in a fixed amount of time) the contents of a table can be buffered by the DBMS in main memory and queries can be answered by reading a table's content from main memory (which is much faster) rather than magnetic disk. Thus, it becomes possible to speed up execution time of a query by even hundreds of times.

Unfortunately, when the size of a table is large it becomes impossible to store a reasonable portion of a relational table. Thus, queries that use such a large table cannot be executed any faster by buffering in main memory some of the tables in a relational database.

One way to execute such queries faster is to read common tables only once and evaluate the answer for two or more queries at once. Using this technique two or more queries will be answered by reading a table only once. Since, it may take in the order of several seconds to read a large table the gains in the total execution time will be considerable.

As an example, the query given below, Q4, is very similar to the query, Q3, and both queries can be answered by reading once all of the student records in the table.

```
Q4:

SELECT * FROM STUDENTS
WHERE DEPARTMENT_ID=2
```

Figure 4: A similar example with Q3

The problem of identifying common parts in queries, such as scanning common relations for locating records that satisfy similar or other conditions, is called Multiple Query Optimization(MQO) and has been studied in great detail since 1980s.

Some special database applications such as deductive query processing, batch query processing and recursive query processing require a group of very similar queries to be executed by the DBMS. It would be very profitable to come up with a single global multiplan that would generate the results for all of these queries at once.

Sellis[Sellis 80] has developed an optimization model for MQO and has given an A* formulation with associated heuristics. Randomized optimization techniques such as Simulated Annealing [Cosar 1993] and Genetic Algorithm have been used successfully for solving this problem[Bayir 2006].

Another recent attempt [Toroslu 2005] for solving MQO problem has used a Dynamic Programming(DP) formulation and the performance of DP has been shown to be comparable to A*.

In Chapter 2 an overview of A* and Dynamic Programming formulations is given. In Chapter 3 we present the system used for generating a synthetic database schema and a set of synthetic queries that will be used for comparing the performance of all of the algorithms given in Chapter 2. Finally, Chapter 4 and Chapter 5 present the results of our experiments and our conclusions.

CHAPTER II

AN OVERVIEW OF QUERY OPTIMIZATION

2.1 Multiple Query Optimization Formulation

The following definitions are needed for defining the MQO problem.

Definition(*Task*). A task is a relational database operation. For the purposes of this thesis we consider only *scan*(reading all the records in a table sequentially), *select* (reading and outputting only the records of a table that satisfy a given predicate) *project* (reading and outputting only the required columns of a table), *join* (performs that relational database join operation that matches records of two tables using a "join predicate" giving the conditions that must be matched by any two corresponding tuples). The input(s) to a task is a relational table, and the output of any task is again a relational table that could form the input to another task.

Example(*Task*). The tasks in Query1 are:

Table 1: Task Description for Query 1

Task1	read(STUDENTS)
Task2	select(DEPARTMENTID=1)
Task3	project(FIRSTNAME,LASTNAME)

Definition (*Plan*).A plan is a directed graph of “task”s that perform relational database operations on individual tables and outputs of “task”s. The graph corresponding to Query1 is given in the below figure.

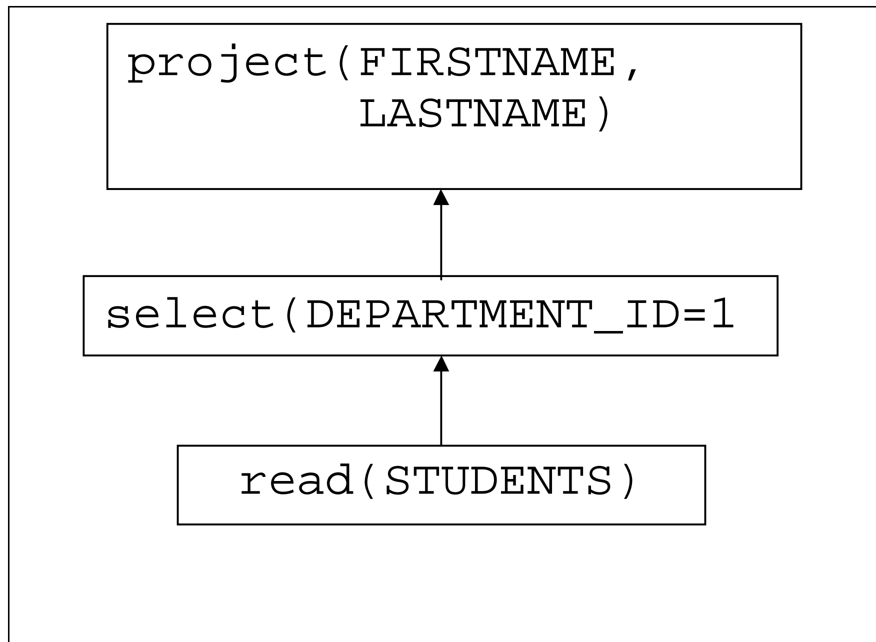

Figure 5 : A plan for executing Query1.

Definition (*MQO*).Given $\mathbf{P_i}$ alternative plans to solve each a query Q_i, where there are **N** queries, select exactly one plan for each query such that the total cost of all the tasks in those chosen plans is minimal.

2.2 A Dynamic Programming Formulation

2.2.1 General Overview of Dynamic Programming

In computer science many problems can be solved by starting from small subsets and building on top of those small subsets to solve progressively larger subsets, eventually obtaining all possible solutions to the original problem instance. In doing this process care is taken so as not to repeat calculations for solving a previously optimized subset, this will guarantee an algorithm. that runs in optimal time. The method was invented by Richard Bellman[xxx] in 1953.

A very famous and popular problem that has the "optimal substructure" property is the "shortest path" problem where the shortest path (an ordered list of nodes to visit with the smallest possible total distance).must be found from an initial node, A, to a destination node, B. The solution to this problem will be an ordered list of nodes starting with 'A' and ending with 'B'. Due to the nature of the problem each node can appear at most once in this list and the path given by the list of nodes between any two pairs, A_m, and A_n (where $m<n$), in this optimal path (A, A_1, …, A_m, A_{m+1}, …A_{n-1},A_n, A_{n+1}, …, B) should also be optimal.

An example instance of the shortest path problem is given in the below figure.

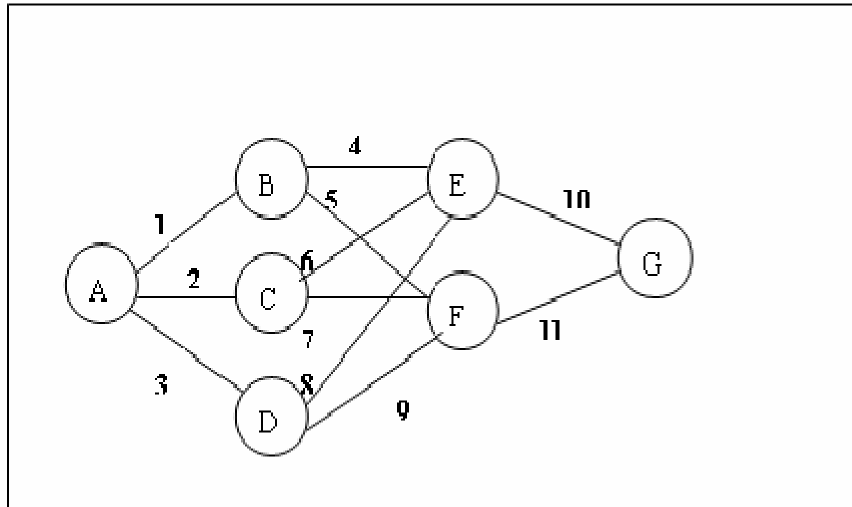

Figure 6: An example shortest path problem

The below table shows the way DP proceeds to solve this shortest path problem of going from node=(A) to node(G). The nodes which in real life could represent cities are connected by lines that could represent roads connecting cities. The distance between two cities is represented as the label of the line between two nodes. First, all adjacent pairs of nodes starting with "A" are recorded in the solutions space used resulting in below list.

SOLUTIONS-1: <A,B,1>,<A,C,2>,<A,D,3>

Then, these solutions are extended with solutions that include nodes that could be reached from the second node in SOLUTIONS-1 list. For example, node(E) can be reached from node(B) therefore <A,B,1> is used to insert a new solution for <A,B, E,5> since the distance between node(B) and node(E) is 4, and the distance between node(A) and node(B) is recorded as "1" in SOLUTIONS-1. Thus, we can obtain the below table for the second step of DP(solutions stored by DP are shown in bold face).

Table 2: Dynamic programming solution for the shortest path problem.

Solutions-1	Solutions-2	Solutions-3
<A,B,1>	**<A,B,1>**	**<A,B,1>**
<A,C,2>	**<A,C,2>**	**<A,C,2>**
<A,D,3>	**<A,D,3>**	**<A,D,3>**
	~~**<A,B,E,5 >**~~	**<A,B,F,6>**
	<A,B,F,6>	**<A,D,E,4 >**
	<A,C,E,8 >	~~**<A,B,F,G,17 >**~~
	<A,C,F,9 >	< **A,D,E,G,14** >
	<A,D,E,4 >	
	<A,D,F,12 >	

You can see that we have recorded both solutions for going from node(A) to node(E) using node(B) as an intermediary, and going from node(A) to node(F) again using node(B) as an intermediary. Now, we need to continue generating solutions that can be generated using <A,C,2> and <A,D,3>. From node(C) we can go to node(E) and node(F) with distances of 6 and 7, respectively. When we combine <A,C,2> with node(E) and add the distance 6 to the value in <A,C,2> we get a new solution, <A,C,E,8 >. You can see that in above table we already have a solution for <A,B,E,5> with a distance of 5 which is 3 less than 8. Therefore, this new solution of <A,C,E,8 > will not be inserted into the SOLUTIONS-2 list. Using the same procedure we obtain <A,C,F,9 > which is a solution for going from node(A) to node(F) with a total distance of 9. Similarly, we already have a solution of

<A,B,F,6> which has a smaller total distance, therefore this solution will not be entered into SOLUTIONS-2 either. Finally, for SOLUTIONS-2, we will find out the nodes reachable using <A,D,3>. Node(D) has two links to node(E) and node(F) with distances 1 and 9, respectively. Using these two links we obtain <A,D,E,4> and <A,D,F,12>. The first solution <A,D,E,4> is better than the existing solution of <A,B,E,5> so <A,D,E,4> will be recorded over <A,B,E,5> (represented by striking through this solution in the table) and <A,D,F,12>, will be discarded since existing solution <A,B,F,6> has a distance of 6, much lower.than 12. In the final step of DP, we generate <A,B,F,G,17> and <A,D,E,G,14> where we note that <A,D,E,G,14> is shorter than <A,B,F,G,17>. As we cannot find any more new paths through the graph we conclude that the shortest path from node(A) to node(G) is given by the ordered list <A,D,F,G> and its cost is

Next, a DP formulation is presented for generating optimal join orders for a relational join query.

2.3 A Dynamic Programming Formulation for Finding Optimal Join Orders

Given an relational query that requires combining information from N relations, namely R_1, R_2,…, R_N, the single query optimization problem aims to determine in which order these relations need to be joined will result in a minimal total cost of (N-1) join operations that will be executed. An example is given next for three relations.

2.3.1 A sample join order problem

The join requested in relational algebra notation: ((R1 $\bowtie$ R2) $\bowtie$ R3) $\bowtie$ R4)

The cardinalities of relations: R1(1000), R2(10000), R3(100000), R4(100)

The cardinalities of join results: R1 ⋈ R2 (1000, i.e. selectivity=10^{-4}), R2 ⋈ R3 (10, i.e. selectivity=10^{-8}), and R3 ⋈ R4 (10000, i.e. selectivity=10^{-3})

Since (R2 ⋈ R3) will result in a much smaller number of records in the intermediate result, that must be produced and saved in memory or on disk, before the second join with R1 is performed, the execution of (R2 ⋈ R3) ⋈ R1 will take much less time than (R1 ⋈ R2) ⋈ R3. Assuming that the processing time of a join operation is proportional to the size of its inputs and the size of result it generates, the dynamic programming will proceed as follows to find the optimal order of join operations.

Table 3: Join cost estimation

R1 ⋈ R2, size=10^3, cost=12000	(R1 ⋈ R2) ⋈ R3, size=1,cost=13001
R1 ⋈ R3, size=10^8, cost=10^8+101K	(R1 ⋈ R3) ⋈ R2, size=1, cost=10^8+201K
R1 ⋈ R4, size= 10^5,cost=101100	(R1 ⋈ R2) ⋈ R4,size=
R2 ⋈ R3, size= 10,cost=110010	
R2 ⋈ R4, size= 10^6,cost=1010100	
R3 ⋈ R4, size= 10^4,cost=100100	

2.3.2 Plan generation for a query

For the evaluation purposes of this thesis we consider only select, project, and join operations. The types of tasks generated by the developed software are as follows:
Read task: these tasks have the name the name the of the relation that will be input. We consider that all of the records in a relation will be read by a read task.

Select task: A select task will apply a filter condition on the input records and output only those records for which the condition evaluates to true. The conditions are generated randomly and has selectivities between 10% and 100%.

Join task: A join task will combine records from a two input relations and output the resulting records. The number of records generated by joining relations R1 (with N_{r1} records) and R2 (with N_{r2} records) will be determined by the join selectivity factor which is between 10% to 100%. Thus, the number of records in the join result can be calculated by using the formula, $N_{r1} * N_{r1} * SF(R1,R2)$.

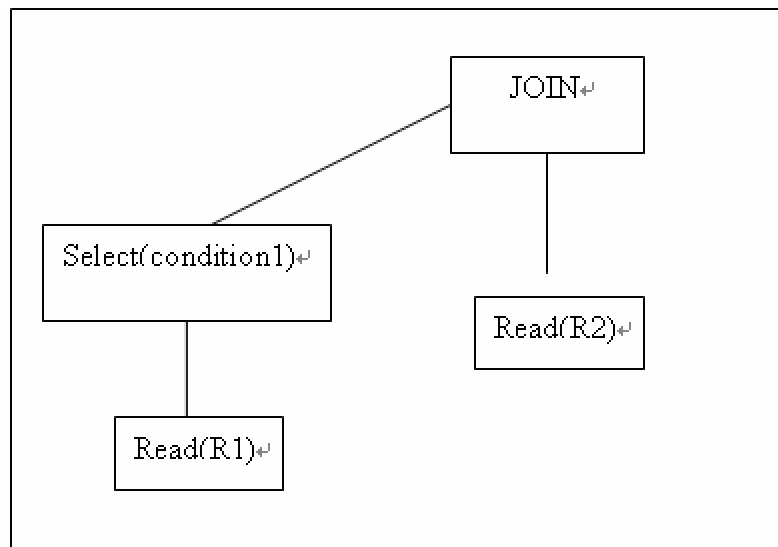

Figure 7: A sample query tree and its task generation.

The plan generated for the above query tree will consist of the tasks:

T1: read (R1)

T2: apply select condition1 on output of T1

T3: Read(R2)

T4: Join the output of T2 with the output of T3

2.4 Alternative plan generation

Given a multiple query optimization problem instance with tasks

Table 4: Tasks in a multiple query optimization problem

T1	T2	T3	T4	T5	T6	T7	T8	T9
10	20	20	5	10	30	25	15	5

and, plans for three queries, $P_{i,j}$ which is the j-th plan for query Q_i.

Table 5: Description of MQO plans

$P_{1,1}$	$P_{1,2}$	$P_{2,1}$	$P_{2,2}$	$P_{2,3}$	$P_{3,1}$	$P_{3,2}$
T1,T2,T3 (50)	T1,T4,T5 (25)	T4,T6,T7 (60)	T1,T3,T8 (45)	T2,T4,T7 (50)	T1,T2 (30)	T3,T9 (25)

The smallest cost plans for individual queries are as follows:

$P_{1,2}$, $P_{2,2}$ and $P_{3,2}$ are lowest cost plans for Q_1, Q_2, and Q_3, respectively. However, the plan combination with the lowest total cost for all three queries is $\{P_{1,1}, P_{2,2}, P_{3,1}\}$ with a total cost of 65. You can see that for Q1 and Q3 the lowest cost plans are not selected when multiple query optimization is used. Also, the total cost of evaluating all three queries is reduced from 95 to 65 since each shared task is executed only once, resulting in these savings.

2.4.1 Alternative plan generation for a query

In order to be able to achieve the maximum benefit out of multiple query optimization and reduce the total cost of executing a set of queries, we need to be able to generate alternative plans for executing a query. Also, these alternative plans must be generated in such a way that there must be as much sharing as possible between a generated alternative plan and the plans already available for other queries.

For achieving this purpose three methods have been proposed in the literature, to the best of our knowledge. First method is called "pairwise alternative plan generation" and it proceeds as follows:

Step1: FOR i:=1 to NumberOfQueries

Step1.1: FOR j:=1 to NumberOfQueries

Step1.1.1: IF(i= =j) CONTINUE;

Step1.1.2: CR:= the common relations between Qi and Qj

Step1.1.3: CC:= the common conditions between Qi and Qj (on CR relations)

Step1.1.4: IF (| CR | < 1) continue; // there is no common relation

Step1.1.5: IF (| CR | > 1)

Step1.1.5.1: …JOINTREE := BEST-LDEEP-JOIN-TREE(CR, CC)

Step1.1.5.2: Qi' := JOINTREE join Qix

Step1.1.5.3: Qj' := JOINTREE join Qjx

Figure 8: Pairwise alternative plan generation

We modify Qi and Qj (obtaining Qi' and Qj') so that all the select conditions on CR but not in CC will be performed on the output of JOINTREE. Thus we will have two alternative plans, one for each of Qi and Qj, so that all of the join task(s) to calculate JOINTREE can be shared.

2.5 Join tree transformation operations

In order to modify the query join trees in such a way that shared relations and conditions are collected together so that they can be executed separately from the rest of the query tree, and its results can be shared with other join trees. Since the modified join tree must generate the same query result set, we show the equivalency of used query tree transformation operations and explain how they work.(xxx ⋈)

2.5.1 Commutativity transformation

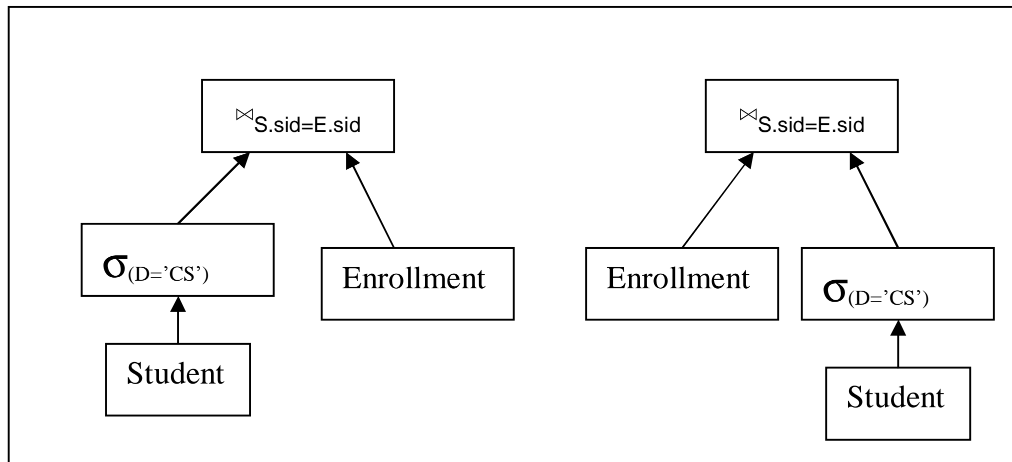

Figure 9: A commutativity transformation example.

The commutativity transformation is used for changing the order of inputs to a join operation. For example, for the nested-loops-join method the order of inputs is used to decide which input relation is used in the outer loop, and the other one is used in the inner loop. For hash-join method it could represent the relation that would be used for building the hash table used in the hash join.

2.5.2 Associativity transformation

The join operation is associative. This allows joins in a query to be ordered in any way chosen and makes this an ideal transformation method for increasing the number of sharable join operations between two query plans. Since join operations

are usually very expensive, there is potential for huge reductions in total execution costs in a multiple query optimization problem instance.

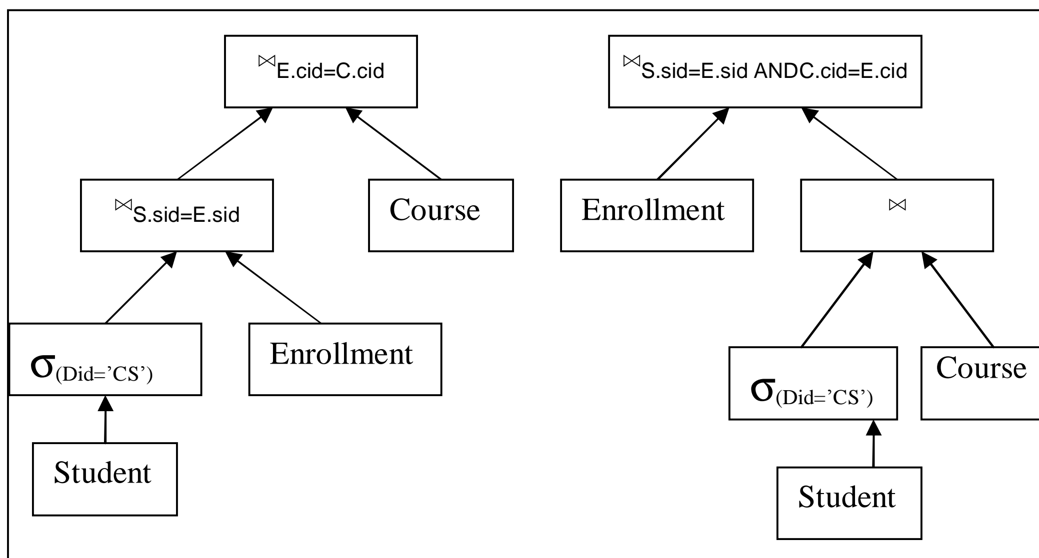

Figure 10: An associativity transformation example.

2.5.3 Selection fragmentation transformation

Since the number of relations in database is relatively small, the variety of queries mostly result from the use of different selection criteria used for specifying which records in a relation must be returned by a query. We could still obtain some shared tasks from such selection operations by considering "subsumption" of conditions. An example for such subsumption could be two conditions where one includes an extra predicate (e.g. dept='CS' AND year='4'). By fragmenting this condition into two parts it could be possible to share either "dept=CS" or "year=4" with another query.

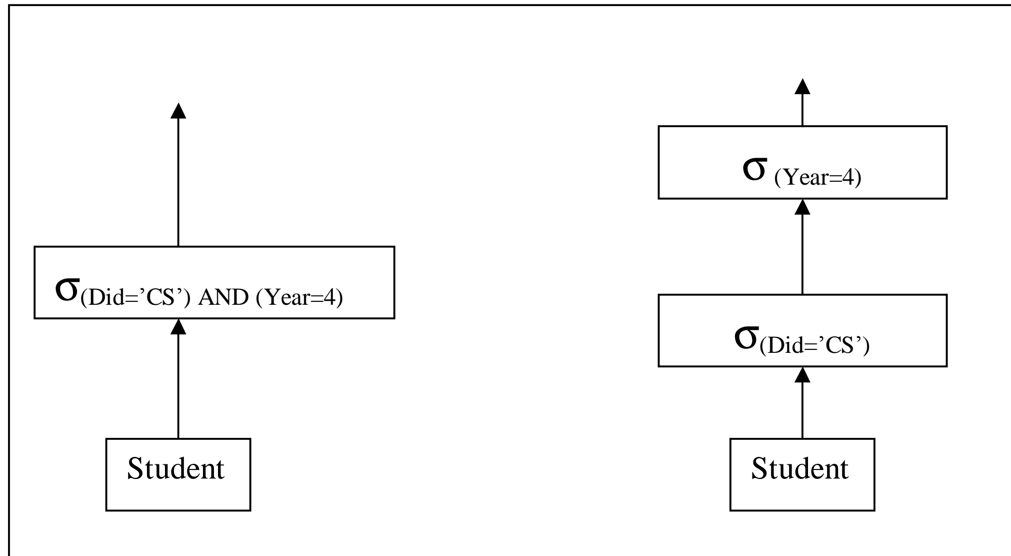

Figure 11: A selection fragmentation transformation example.

2.5.4 Projection fragmentation transformation

This operation is similar to selection fragmentation and by keeping attributes of a relation in an intermediate result, it becomes possible to share an intermediate result with another query plan.

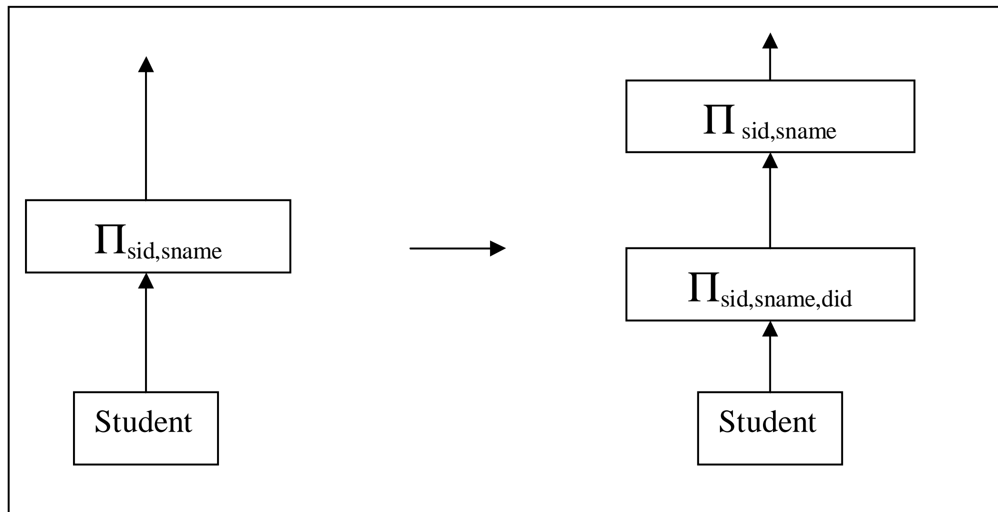

Figure 12: A projection fragmentation transformation example.

2.5.5 Selection and Projection propagation transformation

This operation is used to delay selection and projection operations so that any following join operation could use the same intermediate results with another query plan, thus allowing shared join operations with the same inputs. Although the cost of a join operation would be increased by delaying a selection or projection operation (because of size of input relation becomes larger taking more space in memory), the total multiple query execution cost could be reduced by allowing a join operation to be shared.

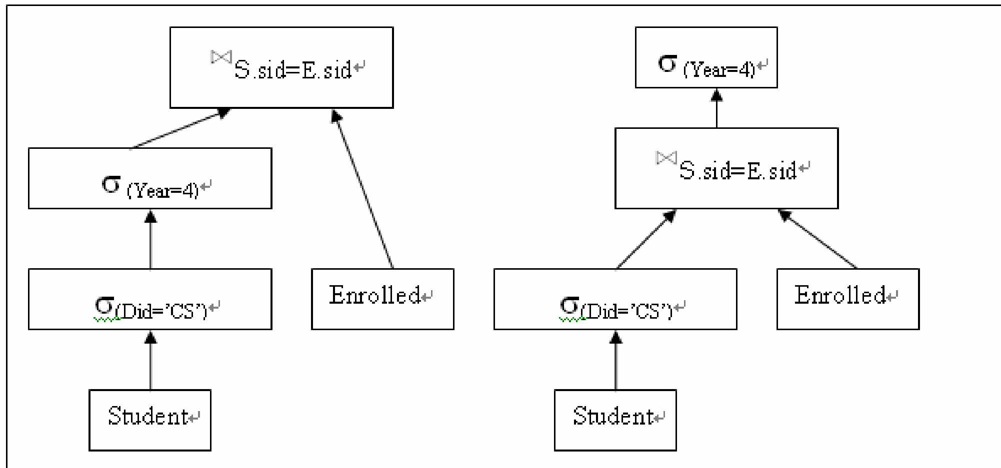

Figure 13: A selection and projection propagation transformation example.

CHAPTER III

EXPERIMENTAL SETUP

In order to compare alternative plan generation methods we synthetically generate a database with a number of tables. Each table is randomly assigned a number of records, a number of attributes(one key and several non-key attributes) that will be used to determine the length of that table's records. Once the requested number of base relations are generated, relationship tables which can be joined by other base relations are randomly generated between pairs of randomly chosen base relations.

Table 6: An example synthetically generated database

Base Relations			Relationship Relations		
Rel#	# Key attr	Num. Dep.Attr.	Rel#	# Key Attr.	Num.of Dep. Attr.
R0	1	4	R9	2	3
R1	1	5	R10	2	8
R2	1	7	R11	2	3
R3	1	2	R12	2	9
R4	1	1	R13	2	6
R5	1	8	R14	2	3
R6	1	4	R15	2	9
R7	1	6	R16	2	1
R8	1	10	R17	2	1

3.1 Randomly Generating Queries

Once the underlying database tables have been generated, the next step is the generation of relational queries on these tables. In order to make sure the generated query is a realistic one it is taken great care so that only relations that have a "foreign key" connecting them will appear in the "FROM" clause of the query. In order to achieve this first a so-called "root relation" is selected from among the so-called "relationship relations".

Once the "root relation" is selected, then we continue to select "base relations" and "relationship relations" that are "connected" to this "root relation" by one or more

"foreign key" attributes. This way, the generated query is guaranteed to be free of any cartesian product operations and contain meaningful equijoin operations only.

When an individual base relation is chosen for inclusion in a query, a selection operation is also generated on it, randomly. For this purpose, one of the attributes in that relation is randomly selected as the subject of the select operation on that relation. It is also possible to assign more than one attributes (in the same table) for the select operation, in which case we restrict the two select conditions to be combined with an "logical and " operator only. The case of "logical or" operations can be handled by generating two queries each with one of the selected attributes.

We assume that each attribute can have at most five distinct conditions defined on them. Therefore, $C_{1,1}$ and $C_{1,2}$ refer to two distinct conditions on attribute A_1.

3.2 The random query generation algorithm

The algorithm for above described procedure is given below:

INPUT: The base relations and relationship relations
OUTPUT: The selected relations, the select attributes, the select conditions
// RGN ( ) – randomly generated number

```
Step 1.1 RR= RGN(1, NumberOfRelationshipRelations)
Step 1.2 NR= RGN(1, MaxNumberOfRelationsPerQuery)
Step 1.3 SSR= { } // set of selected relations
Step 1.4 SSC= { } // set of conditions
Step 1.5 for(i=1; i<NR; i++)
Step 1.5.1 SR= RGN(1, NumberOfRelations)
Step 1.5.2 while( (SR already in SSR) OR ( SR not connected to SSR) )
```

Step 1.5.2.1 SR= RGN(1, NumberOfRelations)

Step 1.5.3 SSR = SSR union SR

Step 1.5.4 SA= RGN(1,NumberOfAttributesInSR)

Step 1.5.5 SSC = SSC union SA * RGN(1,MaxNumOfCondsPerAttr)

3.3 Generating an Optimal Plan for a Query

For a single query finding an optimal query plan has been reduced to finding the order in which relations will be joined to a partial query result to form the final answer. This is called finding a so-called "left-deep" join tree for evaluating a given query. For a sample "left-deep" join tree see Figure 3.2.

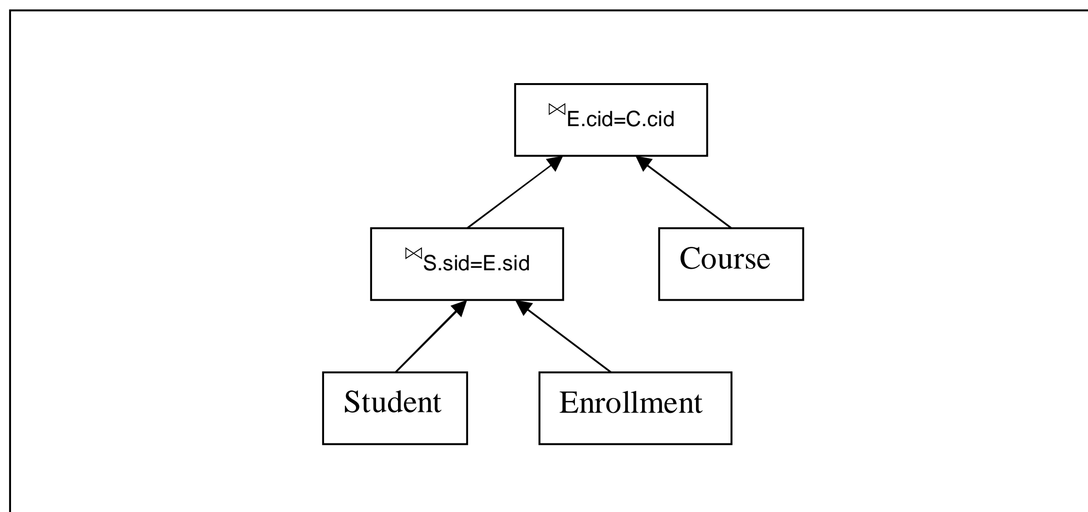

Figure 14: A sample left-deep join tree.

3.4 Generating Alternative Plans for a Query

An optimal plan for a single will always have its selection and projection operations performed as early as (at the lower levels of the join tree) possible, that is perform

projection when the extra attributes are not needed by the rest of the query anymore, and perform selection operations as soon as it becomes possible to evaluate the conditions, which is for conditions based on a single table do the selection operation just after reading the input relation, and for join conditions perform them when the join operation is being performed. For the purposes of this thesis, aggregate operations (those involving GROUP BY clause) and aggregate conditions (such as AVG(grade) > 3.0) are not considered during these experiments.

3.5 Generating Alternative Plans for Multiple Queries

In order to generate alternative plans for more than query, we consider two different algorithms, *pairwise* and *complete*.

```
NUMBER_OF_PLANS = number_of_plans;
for (i =1; i <= number_of_plans; i++)
   for( j = i + 1; j<= number_of_plans ; j++ )
      APPLY TRANSFORMATIONS ON Plan[ i ] so that an
      alternative plan
      Plan [ ++NUMBER_OF_PLANS ] Is obtained for
      it with maximal possible sharings with
      Plan[ j ]
```

Figure 15: The pairwise algorithm

```
for (i =1; i <= NUMBER_OF_PLANS; i++)
   for( j = i + 1; j<= NUMBER_OF_PLANS ; j++ )
      APPLY TRANSFORMATIONS ON Plan[ i ] so that
      an alternative plan
      Plan [ ++NUMBER_OF_PLANS ] is obtained for
      it with maximal possible sharings with
      Plan[ j ]
```

Figure 16: The complete algorithm

As you can see from the code for the "complete" algorithm, newly generated plans are included when generating other alternative plans while in "pairwise" only the initial set of plans is used when generating new alternative plans.

The "sharing factor" heuristic first proposed by [Cosar 2006] and developed by Dr. Ahmet Cosar and Dr. Faruk Polat in 1998, and tested in [Gunay 1998] is as follows;

```
sumSF=0.0; countSF=0; sumSFkare=0.0; mysigma=0.0;
 int nq= Number_of_Queries;
 for(i=1;i<nq;i++) {
      for(j=i+1;j<=nq;j++) {
         lcr.intersectList(Qs[i]->rels,Qs[j]->rels);
/*LCR is array of lcr (list of common relations) */
         LCR[i][j].copyList(lcr);
         if(lcr.cntf()<2) continue;
         lcc.intersectList(Qs[i]->conds,Qs[j]->conds);
/*LCC is array of lcc (list of common conditions) */
         LCC[i][j].copyList(lcc);
         lca.intersectList(Qs[i]->attrs,Qs[j]->attrs);
         nckattr=common_kattrs(lca);
/lca is the list of common key attributes */
        factor1=sum_relation_sizes(lcr,&avgcard);
        factor2=sum_join_relation_sizes(lcr);
        factor3=lcc.cntf()*lca.cntf()*avgcard;
        SF[i][j] = factor1 + factor2 + factor3 ;
        sumSF += SF[i][j];
        sumSFkare += SF[i][j]*SF[i][j];
        countSF++;

     }
   }
 *avgSF=sumSF/(double)countSF;
 *sigma=sqrt((double)((sumSFkare*countSF-
    sumSF*sumSF)/(double)(countSF*(countSF)))) ;
 *sigma = (*sigma)/2.0 ;
```

Figure 17: The sharing factor heuristic

This heuristic uses the number of common key attributes, number of common conditions, and number of common relations to calculate a heuristic value that is used to filter those plan combinations that are not very promising.

CHAPTER IV

EXPERIMENTAL RESULTS

4.1 Introduction

The performances of Pairwise and Complete alternative plan generation methods have been measured and compared experimentally. The results are reported in this chapter. The goal in performing these tests was to verify the results obtained in a previous MS thesis [Gunay 1998-2] and correct any mistakes that were made in analysis. Our results clearly show that the execution time measurement method (total (user,system,idle) CPU running times were used instead of using the Unix system's "time" command which calculates and reports the "user", "system" and "idle" cpu execution times separately) used in the above mentioned thesis resulted in gross exaggeration of obtained benefits by scaling the execution time of complete APG heuristic and thus exaggerated the benefits of sharing factor heuristic. We plan to perform another study to verify whether a similar improvement in APG execution time can be obtained by using a random dropping of same percentage of plans as that done by the sharing factor calculation heuristic.

After execution times of earlier exhaustive versions of "complete" and "pairwise" APG heuristics are measured, the performance of the same methods are measured when the "sharing factor" filtering heuristic is used to reduce the execution time of alternative plan generation phase. This way we can compare the running times of

each heuristic and measure how much CPU time is gained by employing the sharing factor heuristic.

Finally, the quality of alternative plans produced when "sharing factor" filtering technique is used, to decide whether the execution time savings in alternative plan generation, results in too much degradation in alternative plan quality.

For this purpose, we take the multiplan obtained by not generating any alternative plans as the basis of comparison since it always provides the global multiplan with the highest total execution cost. The complete and pairwise APG heuristic and sharing factor modified versions of complete & pairwise APG heuristics will always produce multiplans with lower costs. Thus, we divide the total cost of the best multiplan found by each heuristic by the best cost of multiplan found by running multiple query optimization on the initial input plans for each individual query. Therefore, the results reported in the comparison graphs have values between 0 to 1 where "1" corresponds to the worst multiplan found by not using any alternative plan generation at all.

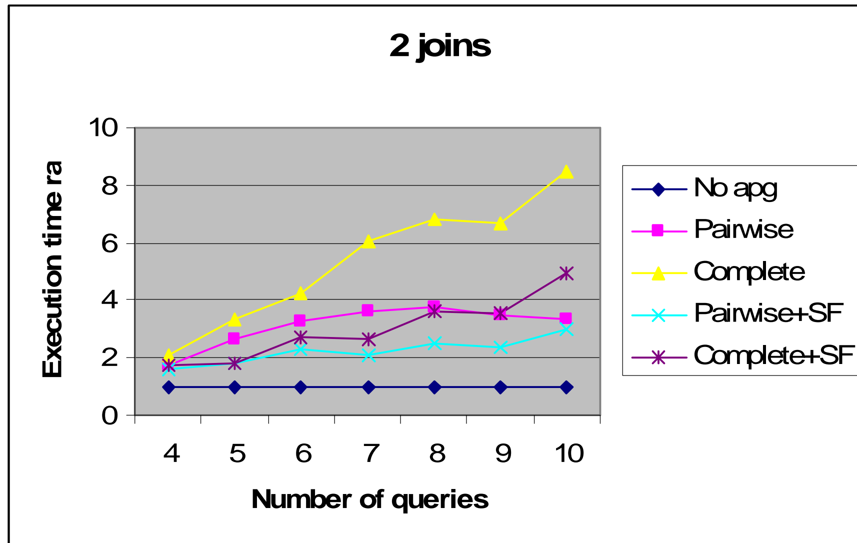

Figure 18: APG times as number of queries increases (2 joins per query).

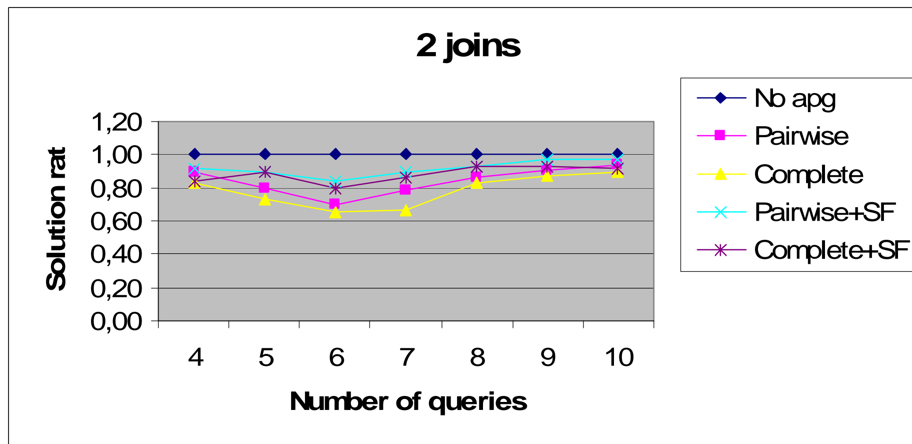

Figure 19: Multiplan quality as number of queries increases (for 2 joins per query).

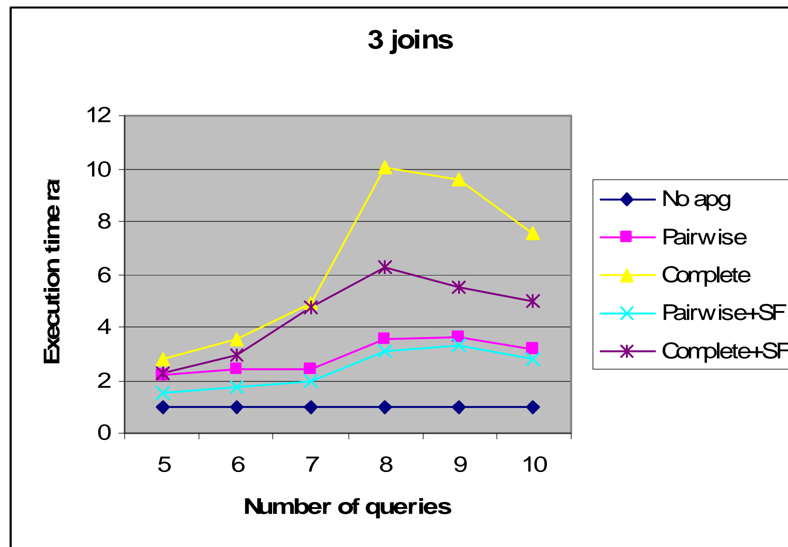

Figure 20: APG times as number of queries increases (for 3 joins per query).

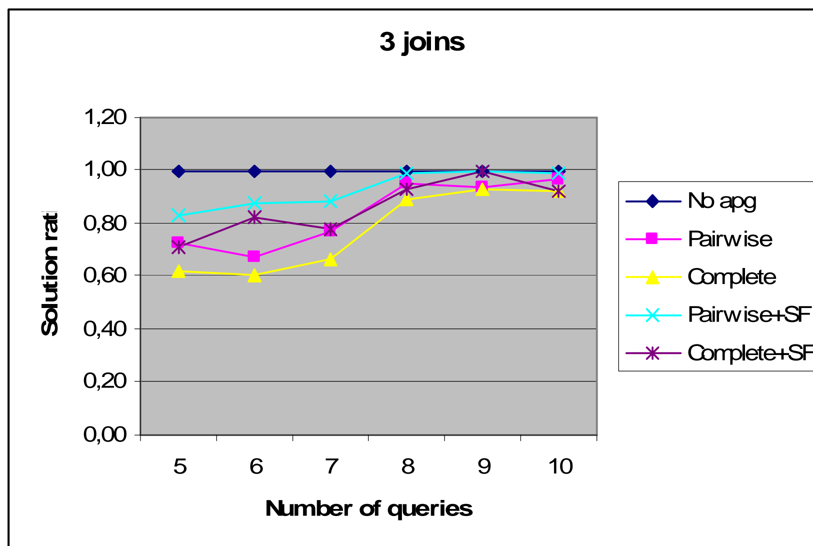

Figure 21: Multiplan quality as number of queries increases (for 3 joins per query).

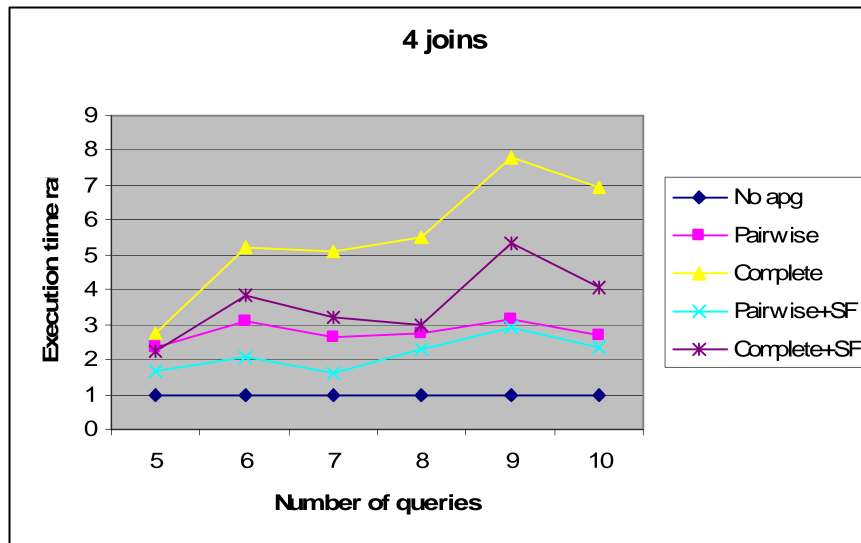

Figure 22: APG times as number of queries increases (4 joins per query).

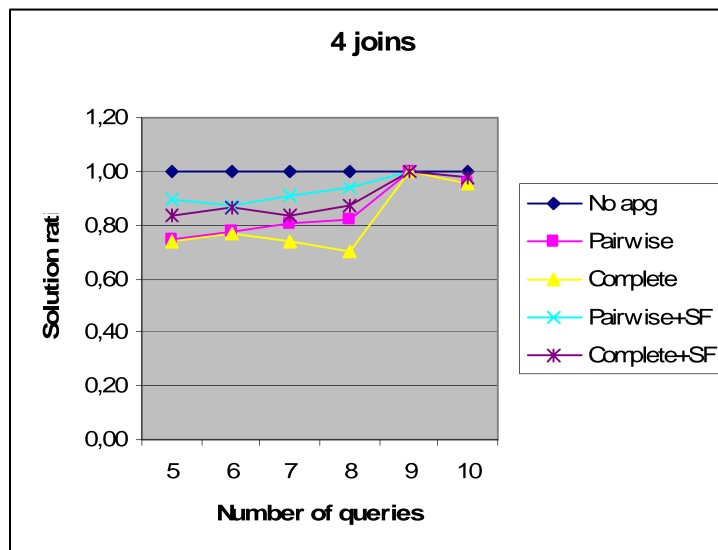

Figure 23: Multiplan quality as number of queries increases (4 joins per query).

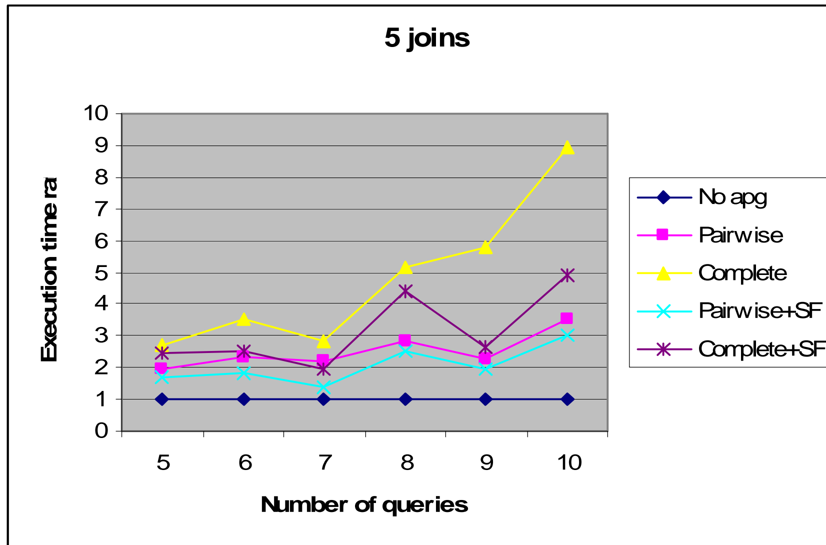

Figure 24: APG times as number of queries increases (5 joins per query).

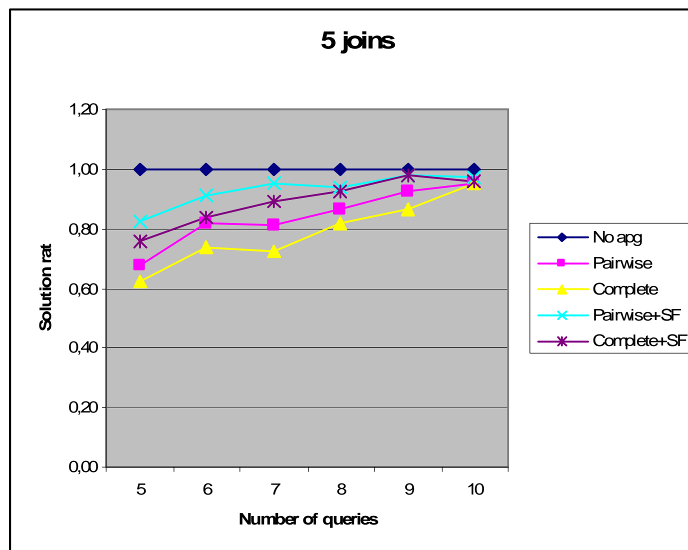

Figure 25: Multiplan quality as number of queries increases (for 5 joins per query).

CHAPTER V

CONCLUSIONS & FUTURE WORK

5.1 Conclusions from experimental results

After obtaining the results from running the experiments we analyzed the results and formed them into line graphics so that the effects of employing heuristics for alternative plan generation can be observed. The generated graphs for 2, 3, 4, and 5 join operations per query with up to 10 such queries in a multiple query optimization problem instance show that;

- The execution times of Pairwise and Complete APG methods are less than 100ms in all of the experiments
- For small (5-7) number of queries "Complete" heuristic takes little extra time and has high benefit
- As the number of queries increases the benefit from APG decreases

The last observation may seem to have a negative meaning on the potential benefit from multiple query optimization, but this result must be expected since as the number of queries all of the tables in a database will be included in at least one of the queries input to the multiple query optimization problem, and thus the total cost of a multiplan will be dominated (and limited) by reading all of the tables in the database once. Therefore, in all of the graphs as the number of queries grows

towards 10 (which is the total number of base tables in the experimental database) all of the heuristics start producing multiplans with very similar costs.

The most important conclusion from this work, however, is the finding that the reported benefits from [Gunay 1998-2] have been shown to be grossly inaccurate and no such large benefits can be expected from sharing factor calculation heuristic during alternative plan generation.

Another important finding the observation that by employing dynamic programming[Toroslu 2004] class (developed for the purposes of this thesis by Dr. Ahmet Cosar) alternative plan generation will take a very small amount of time, usually less than 100ms.

5.2 Future Works

After re-writing in C++ of the multiple query optimization system code that had been developed by Dr. Ahmet Cosar (for his PhD thesis work), now it has become possible to run more advanced and detailed experiments on multiple query optimization and alternative plan generation. Combined with the ability to generate synthetic randomly generated databases and MQO problem instances on that synthetic database, from within the program, it is possible to explore below future works;

- Inclusion of OR in WHERE clause.
- Inclusion of “<“ and “>” when deciding shared subsets of query results. (e.g. “year<4” and “year<2”

Upto now, it was assumed that a query with an “OR” logical operation in the “where” clause of the SQL statement would be input as two separate queries replicated for each side of the OR operation. Although, this didn’t change that cost

of the global Multiplan, it certainly affected the alternative plan generation cost since the number of queries input to the multiple query optimization is increased this way.

Also, the limitation of the detection of logical predicates to the “=” operator limits the benefits that can be obtained from multiple query optimization by allowing logical predicates like “age<30” and “age<20” to be shared and obtain the result of “age<20” from output of “age<30”.

REFERENCES

[Cosar 1993] Cosar A., Lim E-P., Srivastava J., “Multiple Query Optimization Using Depth-First Branch-and-Bound and Dynamic Query Ordering,” CIKM-93, USA, 1993.

[Cosar 1999] Cosar A., “Alternative Plan Generation for Multiple Query Optimization.” Book chapter in **Current Trends in Data Management Technology**, pp.113-129, 1999.

[Gunay 1998-1] Gunay M., Polat F., Cosar A., “Alternative Plan Generation Methods for Multiple Query Optimization,” ISCIS-98, Antalya, Turkey, 1998.

[Gunay 1998-2] Gunay M..“Alternative Plan Generation Methods for Multiple Query Optimization,” MS thesis, Middle East Technical University, 1998.

[Polat 2001] Polat F., Cosar A., Alhajj R., “The Semantic Information Based Alternative Plan Generation for Multiple Query Optimization,” Information Sciences, vol. 137/1-4, pp. 103-133, 2001.

[Ramakrishna 2003] Ramakrishna R., Gehrke J., **Database Management Systems**, 3rd Ed., USA, 2003.

[Rosenthal 1988] Rosenthal A., Chakravarthy U.S., “Anatomy of a Modular Multiple Query Optimizer,” Proc.of 14th Int.Conf. on VLDB, pp.230-239, 1988.

[Toroslu 2004] Toroslu I.H., Cosar A., “Dynamic Programming Solution for Multiple Query Optimization,” Information Processing Letters, Vol.92/3, pp.149-155, 2004.