*Coşar/ AKÜ Fen Bilimleri Dergisi 2009-01 31-40*

# Solving Large Multiple Query Optimization Problems

Ahmet Coşar
*Department of Computer Engineering, Middle East Technical University, 06531 Ankara*
*e-posta: cosar@metu.edu.tr*

**Abstract**

This work presents the multiple query optimization (MOQ) problem and state of the art optimal solutions for this problem. Then, we design 16 new methods to solve the MQO problem. These proposed methods are executed to solve randomly generated instances of MQO problem. Each method is run on 20 MQO problem instances and their averages are taken to calculate the performance of each method on the same instances. In order to compare the methods we calculate the rank of each method and the method with the best overall average rank is chosen as the best method.
*Key Words:* Relational databases, Multiple query optimization, A*

# Büyük Çoklu Sorgu Eniyileme Problemlerinin Çözülmesi

**Özet**

Bu çalışmada çoklu sorgu optimizasyonu (ÇSO) için varolan çağdaş ve eniyi optimal çözümler sunulmaktadır. Daha sonra ÇSO problemi için 16 yeni algoritma tasarlanmıştır. Bu önerilen algoritmalar kullanılarak rastgele üretilmiş ÇSO problemleri çözülmüştür. Her bir algoritma farklı 20 ÇSO problemi üzerinde çalıştırılmış ve bunların ortalaması o yöntemin performansı olarak hesaplanmış, ve aynı 20 problem üzerinde bütün algoritmalar çalıştırılarak ortalamaları bulunmuştur. Önerilen algoritmaları karşılaştırmak için ortalama maliyetlerine göre sıralanmış ve en iyi algoritma 1, en kötü algoritma 16 sıralaması verilmiştir. En küçük ortalama sıralamaya sahip olan algoritma en iyi algoritma olarak belirlenmiştir.
*Anahtar Kelimeler:* İlişkisel veritabanları, Çoklu sorgu optimizasyonu, A*

## 1. Introduction

The multiple query optimization (MQO) problem has been studied in the database literature since 1980s. MQO tries to lower the execution cost of a group of queries by evaluating common tasks only once, whereas traditional query optimization considers a single query at a time. MQO has been formulated in [1] as an optimization problem where several heuristic functions are used to direct an $A^*$ search. Later in [2], and [3] a more informed cost estimation function, which is more expensive to calculate, has been used to reduce the number of states explored by the $A^*$ search. We show that this improved heuristic can be improved significantly by modifying it to use a dynamic query ordering scheme, instead of the static query ordering heuristics used before. As the MQO techniques are now beginning to be used for large optimization problems such as materialized view selection, the performance gains reported here will be very useful in solving these large optimization problems in less time and using less memory.

The MQO problem can be divided into two phases. The first phase is to identify common tasks among a group of queries and prepare a set of alternative plans for each query, which will be the basis for the second phase of MQO. The second phase is the selection of exactly one plan for each query in the query set. As a result of second phase a global execution plan in the form of a directed acyclic graph of tasks is obtained, which will produce the required outputs for all queries when executed.

The first phase of MQO has been addressed in a recent paper [4] and it has been shown that for up to 10 queries near-optimal alternative plans can be generated in less than 2 seconds. What is even more exciting is that we can reduce the number of alternative plans generated for a query in a given group of queries, while we preserve the quality

of plans and obtain multi-plans always within 20%(on the average 12%) of optimal. Another interesting observation is that, as the number of queries input to MQO increases the multi-plans generated remain close to optimal.

The second phase of MQO is also very important because that is the most time consuming phase of the problem. For example, as reported in [1] the CPU time of the A* optimization algorithm could be as high as several hundred seconds for only 15 queries, while this time has been reduced to less than a second (for small MQO problems) using a better heuristic [2, 3], and in this work the optimization time is further reduced and it is experimentally shown that solving large problems with even one hundred queries and hundreds of alternative plans is now feasible.

**Example 1.1** (*How common tasks and alternative plans can be identified*). In Fig.1 an example task sharing is shown on how multiple-query optimization exploits common tasks and generate a global-plan to execute two queries in a group. In this example we have two queries:

Q1: Find the names of senior students in the computer science department.

Q2: Find the id's of repeat students in the computer science department.

In Fig.1 only the select operation *dname="CS"* on the *Department* relation is common in both queries. It is possible to share the join operation between *Department* and *Student* relations, as well, but, for this, we must delay the selection operations *status=repeat* and *year=4*, as shown in Fig.2.

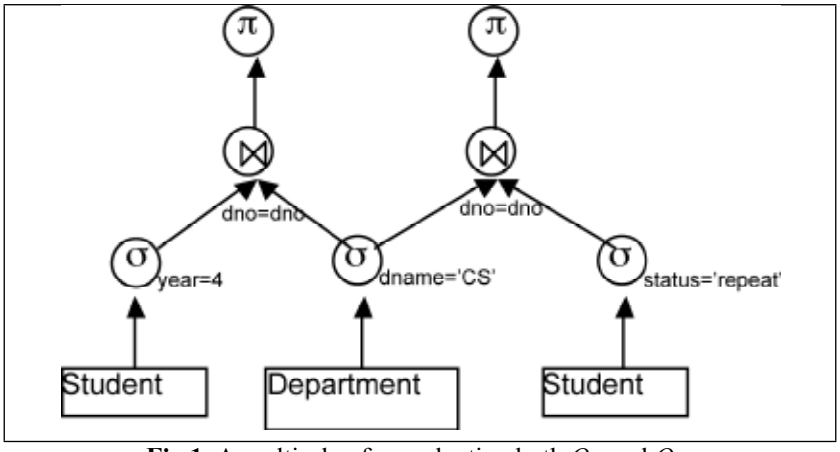


**Fig.1.** A multi-plan for evaluating both $Q_1$ and $Q_2$.

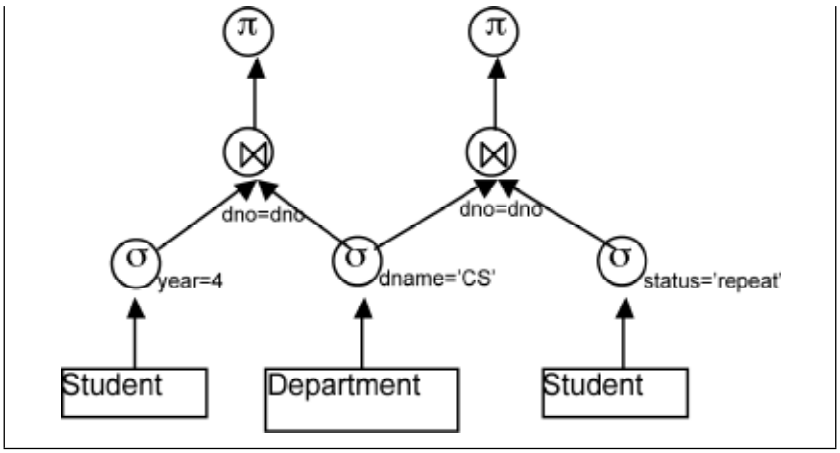


**Fig.2.** Another multi-plan for evaluating both $Q_1$ and $Q2$.

The important point to recognize in Fig.1 and Fig.2 is the delaying of selection operations. This is just opposite of the conventional policy of performing selections as early as possible in single query optimization. By delaying selections we increase the cost of evaluating each query

individually, however, due to the potential increase in common tasks this becomes a good technique to obtain alternative plans for individual queries and use these alternative plans as input for multiple-query optimization phase.

## 2. Materials And Methods

In order to find the best algorithm for finding optimal solution to a MQO problem, a C program was written to implement the A* heuristic and calculate both the number of search states expanded by the A* algorithm and also the time taken in milliseconds. A second program in C language was also designed and implemented. The second program generates a randomly produced MQO problem instance by using 3 parameters. (1) The range of costs of tasks (the minimum and maximum costs) and the number of such tasks. (2) The total number of plans and the range (minimum and maximum number of tasks) of number of tasks in any plan and the randomly selected tasks in each plan. (3) The total number of queries and the range of number of plans for each query and the actual selected plans for each query.

Then, 16 possible algorithms are generated. For this purpose 4 different ordering criteria, DQO1 (current shared cost), DQO2 (ratio of current shared plan cost to unshared plan cost), DQO3 (difference of current shared cost from unshared plan cost) and DQO4 (estimated current plan cost) are defined. For each of these ordering criteria we select the MIN (the query with the minimum cost plan among all plans), MAX (the query with the maximum cost plan among all plans), MIN+AVG (the query with the minimum average plan cost, obtained by calculating the average of all plans for a query), and MAX+AVG (the query with the maximum average plan cost, obtained by calculating the average of all plans for a query). Each of these methods are implemented as a C function and depending on the selected algorithm the corresponding C function is executed to choose the next query to be expanded in the A* optimization algorithm.

Before comparing the DQO1-DQO4 algorithms, we compare static query ordering and dynamic query ordering by using the same randomly generated MQO problem instances, and experimentally show that dynamic query ordering is vastly superior to static query ordering.

## 3. Related Work

Query optimizers for relational databases use relational algebra (RA) [5] for internal query representation. Most of the theoretical work on databases has focused on select-project-join queries and this article's experiments have also been limited to those operations. It is assumed that the set of alternative plans will be generated by a multiple query optimizer as described in[6], or by post-processing of single query optimizer generated plans to obtain *more sharable* alternative plans as described in [7] and [8]. Once common tasks and their execution time estimates are provided, our optimization algorithm is applicable for the complete set of RA operations and vendor specific extensions.

There has been a lot of recent interest in using multi-query optimization techniques in the context of materialized views and data warehouses. In [9] it is shown that multi-query optimization can be easily added to existing optimizers and is feasible for use in complex decision support systems. In [10] multi-query optimization is used to identify and exploit common sub expressions for materialized view maintenance. Another possibility is to materialize views such that it would improve query response times by considering frequently used common sub expressions as candidates for view materialization, a very detailed survey of previous work on answering queries using views is given in [11]. An extension of multiple query optimizations to queries with aggregate operators has also been proposed in [12] and shows the great potential MQO has for expensive queries. The maintenance and verification tasks on relational databases can be improved by employing multi-query optimization techniques as discussed in [13], as well. The query model used in this study doesn't include the effect of pipelining which would make it very difficult to calculate a given plan's cost. It would cause the cost of a sub-query to depend on ordering of relations while in our

model a sub-query is a set of relations where ordering of relations is immaterial. Recently, there has been renewed interest in multiple query optimization in the domain of sensor networks. The need for executing the same query on multiple sensor outputs and also executing multiple queries on a given sensor output naturally gives rise to MQO problems[14].

In addition to A*, in recent work other optimization techniques have been proposed for solving MQO problem. One of these is dynamic programming (DP) and it has been studied in [15] where it was shown that DP has comparable performance to depth-first branch-and-bound (DFBB) but its memory overhead is much larger than DFBB. Genetic Algorithm (GA) is also a recent technology which can be used for optimizing a variety of problems and its performance on MQO problem has been reported in [16] which show that optimal solutions can be obtained in a reasonable time. The advantageous side of GA is its ability to provide a solution at any time during optimization which makes it very attractive for real-time constrained applications. The ability of GA to limit the solution pool to minimize the used memory also makes it perfect for real-time limited memory environments.

Single query optimization aims to minimize the time required to calculate the output for a given query. This problem has been addressed in great detail by database researchers since 1970s [17, 18]. Because of the large number of alternative join methods and join orders, state of the art optimizers search only a subset of the possible query plans, to be specific, left-deep (non-bushy) join trees and generate a single execution plan in the form a tree with RA operations as its internal nodes and base relations as leaf nodes.

Even though above approach is acceptable for single queries, we are faced with the problem that in MQO we may prefer a sub-optimal plan for a query if it results in more common tasks with the other queries in the group and thus yield a better global plan for the query group. This can be seen easily from Example 3.1. We are assuming that common tasks have been already determined by the first phase of MQO, alternative plan generation.

**Example 3.1** (*A sample MQO problem*). Given two queries, $Q_1$ and $Q_2$, and below alternative plans for them, where $P_{i,j}$ is the j-th alternative plan for $Q_i$:

$P_{1,1}=\{t_1, t_2, t_3\}=75$ $P_{1,2}=\{t_4, t_5\}=55$

$P_{2,1}=\{t_1, t_6, t_7\}=55$ $P_{2,2}=\{t_2, t_8, t_9\}=45$ $P_{2,3}=\{t_5,t_{10}\}=50$

Given task costs: $t_1=40$, $t_2=30$, $t_3=5$, $t_4=35$, $t_5=20$, $t_6=10$, $t_7=5$, $t_8=5$, $t_9=10$, $t_{10}=30$.

The lowest cost plans for $Q_1$ and $Q_2$ are $P_{1,2}$ and $P_{2,2}$, respectively. For multiple query optimization, however, the preferred plans for $Q_1$ and $Q_2$ will be $P_{1,2}$ and $P_{2,3}$, respectively, since these two plans share $t_5$ and give a global plan cost of *85*, while $P_{1,2}$ and $P_{2,2}$ have no common tasks and a global plan cost of *100*.

## 4. MQO Problem Formulation

This formulation has been given in [1]. It defines the search space for A* and also a heuristic function, ht, to direct the search.

*Initial State*: This state has all null values denoting that no plan has been selected for any queries yet.

*Intermediate State*: These are states where at least one plan has been assigned for one or more queries, but there is at least one query left for which no plan has been assigned. In our notation the i-th position holds the assigned plan for Qi.

*Solution State*: A state where all queries are assigned a plan.

### *4.1 Heuristic Function ($h_t$)*

Assume that the state after selecting plans for queries $Q_1$ through $Q_k$ is

$S_k=<P_{1,j1},...,P_{k,jk}, \text{null}, ...,\text{null}>$.

$$\text{est\_cost}(t_i)=\frac{\cos t(t_i)}{n_i}$$

$$\text{est\_cost}(P_{i,j})=\sum_{t_i \in P_{i,ji}} est\_\cos t(t_i)$$

Here, $n_i$ is the number of queries, among the original set of $n$ queries, with a plan containing $t_i$. Then,

$$h_t(S_k)= \sum_{1\le i\le k} est\_cost(P_{i,j_i})+ \sum_{1\le i\le k} \min(est\_cost(P_{i1}),.., \text{est\_cost}(P_{i,ni}))$$

A more informed heuristic for plan cost estimation has been defined in [2] and [3], which improves the performance of MQO drastically. The only difference is the use of $m_i$, instead of $n_i$, which is the number of queries, among the remaining (instead of all queries as in $n_i$) set of queries without an assigned plan, with an alternative plan containing $t_i$. By using $m_i$ now it becomes possible to use the actual cost of the partial global plan for cost estimation as the *admissibility* [19] is guaranteed as proven in

*4.2.Heuristic Function $h_s$*

Let $t_{sel}= \bigcup_{1\le i\le k} P_{i,j_i}$ be the set of tasks in the selected plans for queries $Q_1$ through $Q_k$.

$\text{est\_cost}(t_i)= \dfrac{real\_cost(t_i)}{m_i}$, if $t_i$ Ït$_{sel}$, and zero otherwise.

$$h_s(S_k)= \sum_{t_x \in t_{sel}} real\_cost(t_x)+ \sum_{k< i\le n} \min(\text{est\_cost}(P_{i,1}),.,\text{est\_cost}(P_{i,ni}))$$

*The DFBB optimization algorithm:*

```
S:= <null,null,null,…,null>;
Q:= makeDEQueue( S); // a double ended queue
OptimalSolution:=
  chooseLowestCostPlanForEachQuery();
While isNotEmpty( Q ) Do
  S := delFront (Q);
Qnum:= chooseQuery(S); // select a query not
                       // assigned a plan yet
  For Pnum:=1 to NumberOfPlansForQuery[Qnum]
    // assign plan to Query[Qnum]
    NewState := assignPlan (S, P[Qnum][Pnum] );
If( estimatedCost(NewsState) >
      real_cost(OptimalSolution)) Continue;
    If( isASolution(NewState) ) then
     Q:= removeStatesWithHigherEstimatedCost(Q,
                  real_cost(NewState) );
     OptimalSolution:= NewState;
    Else  // add new state at head of queue
     Q:= addFront(Q, NewState);
    End If
  End For
End While
```

*4.3 Proof of Admissibility of $h_s$*

In $h_t$, the first term corresponds to the plans already selected and a lower bound is calculated for cost. In $h_s$, the real shared cost for plans is used. Thus, the first term of $h_s$ is at least as large as the first term of $h_t$. The second term of $h_s$ uses mi istead of ni and $m_i \le n_i$, so the second term of ht is at least as large as the second term of ht. Hence, $h_s \le h_t$, and hs is more informed than ht.

In order to show admissibility we note that the first part of summation for hs gives real shared cost of selected plans for Q1 through Qk. The second part calculates a lower bound for remaining queries by assuming maximum possible sharing of all tasks in the queries Qk+1 through Qn. Therefore, it is guaranteed that sum of these two values will not exceed the actual cost of the best possible solution reachable from that state in search, and thus $h_s$ is admissible like $h_t$ while it is more informed than $h_t$.

*4.4 Query Ordering Heuristics Defined in [1].*

In order to reduce the estimation error in heuristic function $h_t$ [1] defines and experimentally evaluates six alternative query ordering heuristics (1: original query order, 2: increasing number of plans, 3: decreasing average query cost, 4:

decreasing average estimated query cost, 5: decreasing average query cost per number of plans, 6: decreasing average estimated query cost per number of plans) for determining the order of alternative plan assignment for each query in the MQO query set. This order is calculated before MQO search begins and remains static throughout the A* search. As a result of these experiments it is reported that the third ordering heuristic, which orders queries in decreasing average query costs, gives the best performance. This result is verified in [2]; therefore, in our experiments we use this query ordering heuristic for comparison of performance results.

### *4.5 Comparison of $h_t$ and $h_s$*

We explain $h_t$ and $h_s$ using their trace Figure 3 and Figure 4 for the problem instance defined in Example 2.1. Initial upper bound is 85.

| state | estimated cost | real cost | action |
|---|---|---|---|
| <-,-> | 70 | - | expand |
| <$P_{1,1}$,-> | 70 | - | expand |
| <$P_{1,1}$,$P_{2,2}$> | 70 | 100 | prune |
| <$P_{1,1}$,$P_{2,1}$> | 75 | 90 | prune |
| <$P_{1,2}$,-> | 75 | - | expand |
| <$P_{1,2}$,$P_{2,2}$> | 75 | 100 | prune |
| <$P_{1,2}$,$P_{2,1}$> | 80 | 110 | prune |
| <$P_{1,1}$,$P_{2,3}$> | 80 | 125 | prune |
| <$P_{1,2}$,$P_{2,3}$> | 85 | 85 | optimal |

**Fig.3.** Execution of A* with $h_t$.

| state | estimated cost | real cost | action |
|---|---|---|---|
| <-,-> | 70 | - | expand |
| <$P_{1,1}$,-> | 90 | - | prune |
| <$P_{1,2}$,-> | 85 | - | expand |
| <$P_{1,2}$,$P_{2,1}$> | 90 | - | prune |
| <$P_{1,2}$,$P_{2,2}$> | 90 | - | prune |
| <$P_{1,2}$,$P_{2,3}$> | 85 | 85 | optimal |

**Fig.4.** Execution of A* with $h_s$.

## 5. Comparison of dynamic query ordering with static query ordering

As reported in [1] the number of states expanded by A* for MQO problem is greatly affected by the order in which queries are considered for alternative plan selection. This is because the accuracy of state cost estimation greatly depends on the query order. By using a static order determined at the beginning of search we ignore the fact that, due to task sharing between alternative plans, as the search progresses the query order may need to be changed. For example, the third query ordering heuristic uses decreasing average query costs while the costs of remaining queries for a given search states greatly changes due to selected common tasks (which have now effectively no cost) and the number of queries that can possibly share a common task. Fig.5 and Fig.6 pictorially show static and dynamic query ordering n A* proceed during optimization.

The search heuristic defined in this work uses several dynamic query ordering heuristics. For each state, dynamically, new estimated costs are calculated for each plan of queries that have not been assigned a plan yet.

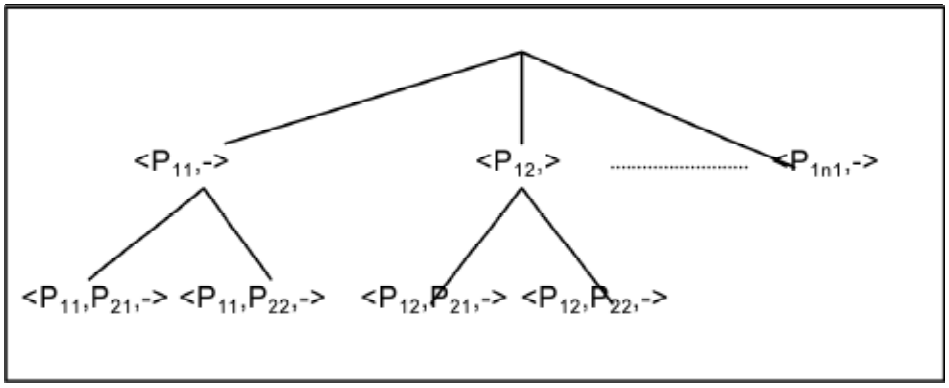


**Fig.5.** State expansion using static query ordering

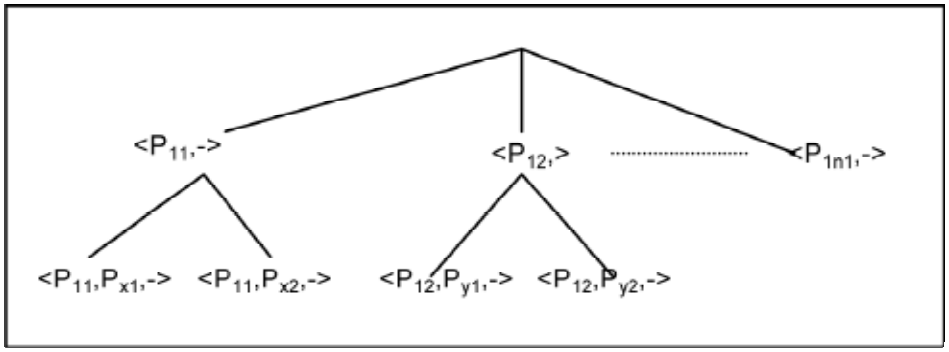


**Fig.6.** State expansion using dynamic query ordering.

*5.1. Dynamic Query Ordering Heuristics*

In this work four alternative dynamic query ordering heuristics have been defined and experimentally evaluated. Each heuristic is tried with selecting the query with the plan giving the minimum and maximum values, as well as the queries that have minimum and maximum values of average of values for all plans of a query. The results in Table 1 clearly show that the policy of choosing the most costly queries for plan assignment is the best solution.

DQO1: Current shared cost.

DQO2: Ratio of current shared plan cost to unshared plan cost.

DQO3: Difference of current shared cost from unshared plan cost.

DQO4: Estimated current plan cost($h_s$)

**Table 1.** Comparison of dynamic query ordering heuristics.

| | **#of Qs** | 5 | 6 | 7 | 8 | 9 | 10 | 11 | AVG. RANK |
|---|---|---|---|---|---|---|---|---|---|
| DQO1 | Min | 31 | 23 | 79 | 114 | 910 | 534 | 7796 | 9,46 |
| | Min+Avg | 33 | 37 | 88 | 118 | 886 | 744 | 7055 | 11,00 |
| | Max | 17 | 17 | 35 | 48 | 577 | 206 | 3336 | 3,38 |
| | Max+Avg | 14 | 17 | 30 | 58 | 232 | 195 | 1631 | 1,46 |
| DQO2 | Min | 28 | 25 | 87 | 122 | 960 | 512 | 5599 | 9,54 |
| | Min+Avg | 29 | 33 | 82 | 125 | 910 | 590 | 3897 | 9,92 |
| | Max | 20 | 16 | 41 | 49 | 502 | 226 | 4627 | 3,69 |
| | Max+Avg | 14 | 17 | 32 | 36 | 222 | 197 | 1792 | 1,23 |
| DQO3 | Min | 26 | 32 | 52 | 67 | 823 | 384 | 5880 | 7,92 |
| | Min+Avg | 24 | 31 | 49 | 65 | 860 | 361 | 5501 | 6,85 |
| | Max | 28 | 23 | 61 | 69 | 592 | 499 | 5774 | 7,38 |
| | Max+Avg | 14 | 27 | 34 | 64 | 575 | 402 | 2711 | 4,08 |
| DQO4 | Min | 28 | 23 | 59 | 69 | 593 | 504 | 5770 | 7,38 |
| | Min+Avg | 14 | 27 | 32 | 64 | 689 | 436 | 2949 | 4,69 |
| | Max | 25 | 36 | 52 | 67 | 732 | 389 | 4925 | 7,46 |
| | Max+Avg | 25 | 30 | 50 | 69 | 748 | 361 | 4800 | 6,92 |

## 6. Experimental Comparison of $h_s$ and $h_t$

In order to compare the performances of $h_s$ and $h_t$ in the presence of static and dynamic query ordering several query sets were generated randomly. The parameters used for this random query-task-plan generation were as follows:

*The number of queries*: The size of query sets was varied from 5 to 15 so that the performance of heuristics could be observed as the MQO problem got larger.

*The number of plans per query*: Each query was assigned between a minimum of 3 to a maximum of 5 plans, randomly.

*The number of tasks per plan*: This parameter was used to control the amount of task sharing between queries. The tasks of plan were selected from a fixed set, so that as the number of tasks per plan increased so did the possibility of having a common task. This number was randomly assigned between a minimum of three to a maximum of 6. The total number of plans in the query set was increased in proportion to the number of queries in the query group.

*The cost of a task*: This parameter was also varied randomly between a minimum of 10 to a maximum of 100. Later, the task costs were fixed at 1 to make the MQO even more difficult by forcing the optimization algorithm to try every shared task alternative.

Table 2 shows the number of states expanded by hs and ht and also how dynamic query ordering affects the performance. From these experiments it became clear that the number of states expanded by $h_s$ is much smaller than $h_t$ and execution time is about 60 times faster for 15 queries, and increases linearly with the size of MQO problem.

**Table 2.** Experimental comparison of $h_s$ and $h_t$.

| # of queries | # of states expanded | | MQO Time(sec) | |
|---|---|---|---|---|
| | $h_t$ | $h_s$ | $h_t$ | $h_s$ |
| 5 | 14 | 14 | 0.1 | 0.2 |
| 6 | 56 | 10 | 0.1 | 0.2 |
| 7 | 137 | 15 | 0.2 | 0.3 |
| 8 | 527 | 15 | 0.5 | 0.3 |
| 9 | 2,929 | 37 | 2.2 | 0.5 |
| 10 | 19,464 | 185 | 14.5 | 1.6 |
| 11 | 24,780 | 239 | 17.5 | 1.7 |
| 12 | 67,875 | 156 | 50.0 | 1.5 |
| 13 | 222,558 | 267 | 175.9 | 2.5 |
| 14 | 598,117 | 520 | 477.3 | 5.3 |
| 15 | 593,997 | 691 | 492.4 | 7.3 |

The experimental results given in Table 3 show clearly that dynamic query ordering is much better than static query ordering. Each result given in Table 3 is an average of 20 repeated runs; therefore the reliability of these experimental results is high.

**Table 3.** Experimental comparison of static and dynamic query ordering.

| | # of expanded states | |
|---|---|---|
| # of Queries | $A^*$ with $h_s$ and Static query ordering | $A^*$ with $h_s$ and DQO1 |
| 15 | 179 | 51 |
| 16 | 289 | 178 |
| 17 | 228 | 142 |
| 18 | 286 | 132 |
| 19 | 119 | 58 |
| 20 | 1000 | 442 |

In Table 4 we give a comparison of DQO1, DQO2, and DQO3. DQO4 is eliminated because the results in Table 1 show that DQO4 is the worst algorithm. In fact, DQO4 was run on some problem instances but its execution time was so long that it was decided to be kept outside the experiments reported in Table 4 which take the most time.

**Table 4.** Experimental comparison of static and dynamic query ordering.

| | # of expanded states | | | |
|---|---|---|---|---|
| # of queries | $A^*$ with $h_s$ and Static query ordering | $A^*$ with $h_s$ and Dynamic query ordering | | |
| | | DQO1 | DQO2 | DQO3 |
| 5 | 66 | 30 | 23 | 22 |
| 6 | 22 | 40 | 49 | 18 |
| 7 | 59 | 60 | 51 | 29 |
| 8 | 1126 | 328 | 358 | 128 |
| 9 | 1045 | 1591 | 2382 | 380 |
| 10 | 1954 | 1025 | 983 | 396 |
| 11 | 2420 | | 16999 | 3021 |

## 7. Conclusions And Future Research Directions

DQO3 is clearly the best optimal algorithm and it should be preferred when an optimal solution must be found in the shortest possible time. Our experimental results clearly show that MQO execution times have been reduced to acceptable levels for use in on-line query processing environments. It needs to be investigated what would be an acceptable query grouping size where the overhead of MQO could be overcome by its savings and without noticeably delaying processing of queries while forming them into query groups. For critical applications where even a minimal overhead for MQO cannot be tolerated, approximate algorithms could be employed provided that they have acceptable performance in terms of multi-plan quality. For materialized view selection and maintenance environments it remains to be seen how feasible it is to apply current algorithms to optimization problems where hundreds of relations and queries are involved, and it could be necessary to develop new approximate algorithms in order to be able to handle such large problems.

## References


1. T. Sellis. “Multiple query optimization,” ACM Transactions on Database Systems, 13(1), pp. 23-52 (1988).
2. K. Shim, T. Sellis, D. Nau. “Improvements on a heuristic algorithm for multiple-query optimization,” pp. 1-26 (1994).
3. A. Cosar, J. Srivastava, S. Shekhar. “On the multiple pattern multiple object (mpmo) match problem,” Int. Conf. On Man. of Data, India (1991).
4. F. Polat, A. Coşar, R.Alhajj. “Semantic information-based alternative plan generation for multiple query optimization,” Information Sciences, vol. 137, pp. 103-133 (2001).
5. E.F. Codd, “Relational completeness of data base sublanguages,” in R.J. Rustin(ed.), Data Base Systems. Prentice-Hall (1972).
6. U.S. Chakravarthy and A. Rosenthal. “Anatomy of a modular multiple query optimizer,” In Proc. Of the VLDB Conf., pp. 230-239 (1988).
7. S. Chakravarthy, “Divide and conquer: A basis for augmenting a conventional query optimizer with multiple query-processing capabilities,” in Proc. 7th Int. Conf. Data Eng., Kobe, Japan, , pp. 482-490 (1991).
8. A. Cosar. Design and experimental evaluation of a multiple query optimizer. PhD. Thesis, CS Dept, Univ. of Minnesota, Minneapolis (1996).
9. P. Roy, S. Seshadri, S.Sudarshan, S. Bhobe. “Efficient and extensible algorithms for multi-query optimization,” SIGMOD Conference, pp. 249-260 (2000).

10. H. Mistry, P. Roy, S. Sudarshan, K. Ramamritham. "Materialized view selection and maintenance using multi-query optimization," SIGMOD Conference (2001).
11. A.Y.Halevy. "Answering queries using views: A survey," VLDB journal, vol.10(4), pp.270-294 (2001).
12. C. Liu, A. Ursu. "A framework for global optimization of aggregate queries," Conference on Information and Knowledge Management(CIKM), pp.262-269 (1997).
13. U. Herzog, J. Schlosser, "Global optimization and parallelization of integrity constraint checks," Int. Conf. on Man. of Data, pp.186-205 (1995).
14. N Trigoni, Y. Yao, J. Gehrke, R. Rajaraman, and A. Demers. "Multi-query optimization for sensor networks, Multi-query optimization for sensor networks," in DCOSS (2005).
15. I.H. Toroslu, A Cosar. "Dynamic programming solution for multiple query optimization problem," Information Processing Letters (2004).
16. M.A. Bayir, I. H. Toroslu, and A. Cosar. "Genetic Algorithms for the multiple Query Optimization problem," IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews, vol 37, no.1 (2007).
17. M. Astrahan et al., "System R: A relational approach to database management," ACM Trans. Database Syst., vol. 1, no. 2, pp. 97–137 (1976).
18. R. Elmasri, S.B. Navathe. Fundamentals of Database Systems, 3ed. (1999).
19. J. Pearl. Heuristics. Reading, MA: Addison-Wesley (1984).