

11-1993

# Multiple Query Optimization with Depth-First Branch-and-Bound


Ahmet COSAR  
*University of Minnesota*

Ee Peng LIM  
*Singapore Management University, eplim@smu.edu.sg*

Jaideep SRIVASTAVA  
*University of Minnesota*

**DOI:** <https://doi.org/10.1145/170088.170181>

Follow this and additional works at: [https://ink.library.smu.edu.sg/sis\\_research](https://ink.library.smu.edu.sg/sis_research)

 Part of the [Databases and Information Systems Commons](#), and the [Numerical Analysis and Scientific Computing Commons](#)

---

## Citation

COSAR, Ahmet; LIM, Ee Peng; and SRIVASTAVA, Jaideep. Multiple Query Optimization with Depth-First Branch-and-Bound. (1993). *CIKM '93: Proceedings of the 2nd International Conference on Information and Knowledge Management, November 1-5, 1993, Washington, DC*. 433-438. Research Collection School Of Information Systems.

**Available at:** [https://ink.library.smu.edu.sg/sis\\_research/958](https://ink.library.smu.edu.sg/sis_research/958)

This Conference Proceeding Article is brought to you for free and open access by the School of Information Systems at Institutional Knowledge at Singapore Management University. It has been accepted for inclusion in Research Collection School Of Information Systems by an authorized administrator of Institutional Knowledge at Singapore Management University. For more information, please email [libIR@smu.edu.sg](mailto:libIR@smu.edu.sg).

# Multiple Query Optimization with Depth-First Branch-and-Bound and Dynamic Query Ordering

Ahmet Cosar, Ee-Peng Lim, Jaideep Srivastava  
Department of Computer Science  
University of Minnesota  
Minneapolis, MN 55455

## Abstract

*In certain database applications such as deductive databases, batch query processing, and recursive query processing etc., a single query can be transformed into a set of closely related database queries. Great benefits can be obtained by executing a group of related queries all together in a single unified multi-plan instead of executing each query separately. In order to achieve this, Multiple Query Optimization (MQO) identifies common task(s) (e.g. common subexpressions, joins, etc.) among a set of query plans and creates a single unified plan (multi-plan) which can be executed to obtain the required outputs for all queries at once. In this paper, a new heuristic function ( $f_c$ ), dynamic query ordering heuristics, and Depth-First Branch-and-Bound (DFBB) are defined and experimentally evaluated, and compared with existing methods which use  $A^*$  and static query ordering. Our experiments show that all three of  $f_c$ , DFBB, and dynamic query ordering help to improve the performance of our MQO algorithm.*

## 1 Introduction

The objective of *multiple query optimization (MQO)* is to exploit the benefits of sharing common tasks in the access plans for a group of queries. In certain database applications, e.g. deductive query processing, batch query processing and recursive query processing, often a group of queries are submitted together to the DBMS for execution. The traditional approach of processing queries one at a time will be inefficient especially when there is a high number of queries sharing

common relations and predicates. MQO identifies common sub-expressions among queries and creates an integrated query execution plan in which common tasks are evaluated only once.

The idea of processing multiple queries has been around for almost a decade [1, 6, 2, 13, 14]. Grant et al [7] used a depth-first based approach to the problem of common sub-expression analysis. Chakravarthy and Minker [2], used an extended version of the query graph[17], called *connection graph*, to represent a set of queries. A query decomposition algorithm guided by a set of heuristics was used to evaluate all of the queries simultaneously. Chakravarthy et al [3] addressed the MQO problem at various levels of detail, depending on the cost measure used. Sellis showed the MQO problem to be NP-hard [15], and gave a state space search formulation [13, 14].  $A^*$  is used as the search algorithm with bounding functions and intelligent state expansion, based on query ordering, to eliminate states of little promise rapidly. Subsequent improvement or variation of Sellis's effort on the MQO problem has been reported in several papers[10, 4, 9]. In [4], Sellis'  $A^*$  algorithm is revised by having an improved heuristics function which prunes search space more effectively while still guaranteeing an optimal solution. Simulated annealing technique has also been experimentally analyzed to handle larger MQO problems that cannot be solved using  $A^*$  in a reasonable time with the currently available heuristics.

### Our contributions:

One of the fundamental parameters in Sellis' work is the ordering of queries so as to decrease the error in heuristic cost calculation function and also to group together queries which have a high amount of shared task(s). There are several ordering heuristics used by Sellis which are computed only once at the beginning and remain constant throughout the search. However, we have observed that an initial ordering may become ineffective, since once a plan for a query is merged with the multi-plan, the sharing between the discarded alternative plan(s) and remaining queries becomes invalid.

To account for this shortcoming of the query ordering heuristics we have adopted a set of *dynamic query*

---

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.

CIKM '93 - 11/93/D.C., USA

© 1993 ACM 0-89791-626-3/93/0011 ....\$1.50

ordering algorithms so that the order in which plans are merged with the multi-plan dynamically changes based on the current partial multi-plan to be augmented by a new plan. Experimental results show that significant gains are obtained by employing *dynamic query ordering*.

As a second contribution we have also analyzed *Depth-First Branch-and-Bound (DFBB)* as a new alternative algorithm to  $A^*$  for solving MQO problems. The *DFBB* algorithm demonstrates some preferable characteristics over the  $A^*$  algorithm. Using the  $A^*$  algorithm as a baseline, we conducted several experiments to verify the advantages of *DFBB* in the MQO domain. Improvement in the performance of the  $A^*$  algorithm and *DFBB* is dependent on (i) the heuristic function used for estimating the lower bound on the cost of a given path to an optimal plan, and (ii) a good query ordering. Sellis proposed a heuristic function ( $f_s$ ) as well as some alternative query orderings for his  $A^*$  algorithm[15]. We prove that a better heuristic function ( $f_c$ ) can be obtained, and propose *Successive Augmentation*[16] as an efficient method to calculate the initial upper bound. Lastly, we experimentally show that the heuristics of selecting the plan (and thus the query) with the largest sharing with the current partial multi-plan can be used to adjust query ordering dynamically and prune the search space more effectively. Equipped with a better heuristic function, and the *dynamic query ordering* heuristic, *DFBB* is demonstrated to perform much better than  $A^*$  algorithms. The use of *depth first search* also helps to reduce the cost of calculating the heuristic function as it reduces the number of “plan merge” operations, which is the operation used for adding a plan to the current multi-plan by considering the shared task(s).

**Paper outline:**

Our paper is structured as follows. Section 2 gives a formal definition of the MQO problem. We examine the suitability of various search algorithms for MQO in section 3. In section 4, we present our new heuristic function ( $f_c$ ) and show that it enables us to expand much less states than the previously proposed heuristic function ( $f_s$ ) when applied to the  $A^*$  algorithm. A *Successive Augmentation* algorithm is introduced for deciding the initial upper bound in section 5. We then present the new dynamic query ordering heuristics in section 5.1. We ran our heuristics on an experimental set of query plans and compared the results with those obtained by query ordering heuristics in [15], which are presented using three tables in section 6. Our conclusions are presented in section 7.

## 2 Problem Formulation

In this section, we present a formulation of the MQO problem which is due to Sellis[14, 15].

Let  $Q_1, \dots, Q_n$  be  $n$  queries to be optimized together. Query  $Q_i$  has a set of  $n_i$  alternative plans for its evaluation, namely  $P_{i,1}, P_{i,2}, \dots, P_{i,n_i}$ .

Plan  $P_{i,j}$  is a set of tasks  $\{t_{i,j}^1, t_{i,j}^2, \dots, t_{i,j}^{n_{i,j}}\}$ .

A task  $t_{i,j}^k$  has an associated cost of  $cost(t_{i,j}^k)$ .

A solution,  $S$ , to the MQO problem is a set of plans  $P_S = \{P_{1,s_1}, P_{2,s_2}, \dots, P_{n,s_n}\}$ .

Let  $T_S = \cup_{(1 \leq i \leq n)} P_{i,s_i}$  be the set of tasks in the solution  $S$ .

Now,  $cost(S) = \sum_{(t_{i,j}^k \in T_S)} cost(t_{i,j}^k)$  is the cost of the solution.

An optimal solution  $S^*$  is such that  $cost(S^*)$  is minimal.

Example 1 shows a sample MQO problem with two queries and five plans. Query  $Q_1$  has plans  $P_{1,1}$  and  $P_{1,2}$  while query  $Q_2$  has plans  $P_{2,1}, P_{2,2}$  and  $P_{2,3}$ . A plan is made up of a set of tasks, each with a positive cost.

**Example 1:**

Let the plans for  $Q_1$  and  $Q_2$  have the following task sets:

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

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

The task costs are:

COST TABLE

	$t_1$	$t_2$	$t_3$	$t_4$	$t_5$	$t_6$	$t_7$	$t_8$	$t_9$	$t_{10}$
Cost	40	30	5	35	20	10	5	5	10	30

Six solutions are possible, with the following costs:

$$cost(S(P_{1,1}, P_{2,1})) = cost(t_1) + cost(t_2) + cost(t_3) + cost(t_6) + cost(t_7) = 90$$

$$cost(S(P_{1,1}, P_{2,2})) = cost(t_1) + cost(t_2) + cost(t_3) + cost(t_8) + cost(t_9) = 90$$

$$cost(S(P_{1,1}, P_{2,3})) = cost(t_1) + cost(t_2) + cost(t_3) + cost(t_5) + cost(t_{10}) = 125$$

$$cost(S(P_{1,2}, P_{2,1})) = cost(t_1) + cost(t_4) + cost(t_5) + cost(t_6) + cost(t_7) = 110$$

$$cost(S(P_{1,2}, P_{2,2})) = cost(t_2) + cost(t_4) + cost(t_5) + cost(t_8) + cost(t_9) = 100$$

$$cost(S(P_{1,2}, P_{2,3})) = cost(t_4) + cost(t_5) + cost(t_{10}) = 85$$

The minimum cost plans for the queries  $Q_1$  and  $Q_2$  are  $P_{1,2}$  and  $P_{2,2}$ , with costs 55 and 45, respectively. The minimum cost multi-plan, however, is  $\{P_{1,2}, P_{2,3}\}$ . It is important to note that an optimal multi-plan is not necessarily made up of the individual optimal plans for each query. Similarly, it is not always necessary that an optimal multi-plan will include a plan having a shared task. It may well be the case that there exists a cheaper plan without any shared tasks with other plans in an optimal multi-plan. It would be an interesting research to determine sufficient conditions on the set of queries such that an optimal plan set has no sharing with a query. In that case such queries could be optimized separately.

### 3 Search Algorithms for MQO

It is well known that the  $A^*$  algorithm is optimal, over the class of best-first searches that find optimal solutions, for a given consistent non-underestimating heuristic function[5]. The optimality is measured in terms of the number of states expanded during search. Nevertheless, it is also well known that  $A^*$  requires a large storage space to store states that are yet to be expanded. As a result,  $A^*$  can easily run out of memory when the size of MQO problems gets larger. Several other alternative heuristic search algorithms have been proposed, e.g. *Iterative-Deepening- $A^*$*  ( $IDA^*$ ) and *Depth-First Branch-and-Bound* ( $DFBB$ ), which use much less memory. Rao and Kumar performed an analysis on the search efficiency of  $A^*$ ,  $IDA^*$  and  $DFBB$  algorithms, and related their efficiency to the two characteristics of search space defined below[12]:

**Definition (Solution Density):**

The solution density of a search space is the ratio of the number of solution nodes to the total number of nodes in the search space.

**Definition (Heuristic Branching Factor):**

Let  $f(n)$  be a heuristic function which is the lower bound on the cost of solutions that include the node  $n$ . Let  $V_i$  denote the set of nodes that have the same cost, such that this cost is the  $i$ th-smallest of all the distinct  $f$ -values. Thus,  $V_0, V_1, \dots, V_{N-1}$  is a sequence of sets arranged in the increasing order of cost. Assume that the sequence of sizes  $|V_i|$  is a geometric progression with ratio  $b$  where  $b$  is the *heuristic branching factor*.

Rao and Kumar showed that  $A^*$  acts poorly when the search space has high solution density or high heuristic branching factor,  $IDA^*$  performs well under low solution density and high heuristic branching factor, and  $DFBB$  performs well under high solution density and low heuristic branching factor.

#### 3.1 Analysis of MQO Search Space

For the MQO problem, a state can be defined as an  $n$ -tuple  $\langle p_{1,j_1}, p_{2,j_2}, \dots, p_{n,j_n} \rangle$  where  $p_{i,j_i} \in \{NULL\} \cup \{P_{i,1}, P_{i,2}, \dots, P_{i,n_i}\}$ . If  $p_{i,j_i} = NULL$ , it means that no plan has been selected for the query  $Q_i$ , i.e. the state is not a solution state. Therefore, we have  $\prod_{k=1}^n n_k$  solution states versus  $\prod_{k=1}^n (n_k + 1)$ , the total number of states. A typical MQO problem usually has a few queries to be optimized, but each query can have a large number of alternative plans. For example, suppose a MQO problem has 10 queries, and each query has 10 alternative plans. The ratio between the number of solution states and total number of states is  $\left(\frac{10000000000}{1111111111}\right) \simeq 0.9$ . Clearly, this indicates that MQO search space has high solution density.

Given this nature of search space, and also knowing that all solutions to MQO are at a depth of  $n$ , where

$n$  is the number of queries, it becomes unnecessary to consider  $IDA^*$ , and the analytical result by Rao and Kumar[12] suggests that depth-first branch-and-bound algorithm ( $DFBB$ ) is the most appropriate search algorithm to use.

#### 3.2 Depth-first Branch-and-Bound Algorithm

Depth-first Branch-and-Bound ( $DFBB$ ) algorithm has been extensively used by the operations research community.  $DFBB$  starts with an overestimate (upper bound) on the cost of an optimal solution, and then searches the entire space in a depth-first fashion. Whenever a solution is found, the overestimate is revised (to be the minimum of the cost of this new solution and the previous overestimate). Similarly, when a partial multi-plan is found to have a worse lower bound than the current overestimate, it is pruned.  $DFBB$  expands each node exactly once, but it can expand nodes costlier than an optimal solution, that are not expanded by the  $A^*$  algorithm.

Let  $f$  be the heuristic function that gives a lower bound estimate of any reachable solution state given a node(state). Let  $soln$  be the current best solution, and  $ubound$  be the upper bound on the solution cost. We assume that value of  $ubound$  has been appropriately set outside the  $DFBB$  algorithm. An algorithm for calculating such an initial upper bound is given in Section 5. Below is an outline of the  $DFBB$  algorithm.

```
DFBB(node)
begin
  mark(node);
  if node is a solution state then
    if  $f(node) < ubound$  then
      begin
         $soln := node$ ;
         $ubound := f(node)$ ;
      end;
  else begin
    expand(node);
    for each newly expanded node  $w$  do
      if  $w$  is unmarked then
        begin
          mark( $w$ );
          if  $f(w) < ubound$  then DFBB( $w$ );
        end;
  end;
end;
```

### 4 Search Heuristic Function

Given a partial multi-plan solution, the search heuristic function ( $f$ ) is used to estimate the lower bound on the cost of any complete multiplan solution derived by

augmenting the partial multiplan solution. In the following, we present the heuristic function proposed by Sellis[14], denoted by  $f_s$ , and another heuristic function proposed by Cosar et al[4], denoted by  $f_c$ .

**Definition ( $f_s$ ):**

Assume that the state after selecting plans for queries  $Q_1, \dots, Q_k$  is

$S_k = \langle p_{1,j_1}, \dots, p_{k,j_k}, NULL, \dots, NULL \rangle$ .

Let  $cost_{est}(t_{i,j}^k) = \frac{cost(t_{i,j}^k)}{n_{i,j}^k}$ .

Also,  $cost_{est}(p_{i,j}) = \sum_{t_{i,j}^k \in p_{i,j}} cost_{est}(t_{i,j}^k)$

Here,  $n_{i,j}^k$  is the number of queries, among the original set of  $n$  queries, with a plan containing  $t_{i,j}^k$ .

Then,  $f_s = cost_{est}(S_k) = \sum_{1 \leq i \leq k} cost_{est}(p_{i,j_i}) + \sum_{k < i \leq n} \min(cost_{est}(p_{i,1}), \dots, cost_{est}(p_{i,n_i}))$ .

**Definition ( $f_c$ ):**

Assume that the state after selecting plans for queries  $Q_1, \dots, Q_k$  is  $S_k$ .

Let  $t_{sel} = \cup_{(1 \leq i \leq k)} p_{i,j_k}$  be the set of tasks of the selected plans.

Let  $cost_{est}(Q_i) = \min\{\sum_{t_{i,j}^k \in P_{i,j}} cost_{est}(t_{i,j}^k)\}$  for  $1 \leq j \leq n_i$

Also,  $cost_{est}(t_{i,j}^k) = 0$ , if  $t_{i,j}^k \in t_{sel}$ , and  $cost_{est}(t_{i,j}^k) = \frac{cost(t_{i,j}^k)}{m_{i,j}^k}$ , if  $t_{i,j}^k \notin t_{sel}$ .

Here  $m_{i,j}^k$  is the number of queries, among those not assigned a plan yet, with a plan containing  $t_{i,j}^k$ .

Now,  $f_c = cost_{est}(S_k) = \sum_{t_x \in t_{sel}} cost(t_x) + \sum_{k < i \leq n} cost_{est}(Q_i)$ .

**Lemma 1:**  $f_c$  is at least as informed as  $f_s$ .

**Proof:**

In  $f_s$ , the first term corresponds to the plans already selected and a lower bound is calculated for their total cost. In  $f_c$ , the real shared cost for this set of plans is used. Thus, the first term of  $f_c$  is at least as large as the first term of  $f_s$ . The second term of  $f_c$  uses  $m_{i,j}^k$  instead of  $n_{i,j}^k$  as in  $f_s$ . Since  $m_{i,j}^k \leq n_{i,j}^k$ , it is guaranteed that the second term of  $f_c$  is at least as large as the second term of  $f_s$ . Hence,  $f_c \geq f_s$ , and is thus more informed[11]. The cost estimation of  $f_c$  thus provides at least as good (and often better) lower bound as that of  $f_s$ , and its convergence will be at least as good as, and in most cases better than,  $f_s$ .  $\square$

**Definition ( $f_*$ ):** The *perfect cost estimation function* can be defined as follows:

Assume, the state after selecting plans for queries  $Q_1, Q_2, \dots, Q_k$  is  $S_k$ .

$S_k = \langle p_{1,j_1}, \dots, p_{k,j_k}, NULL, \dots, NULL \rangle$

Let  $t_{sel} = \cup_{(1 \leq i \leq k)} p_{i,j_k}$  be the set of tasks of the selected plans.

Let  $cost_{real}(Q_i) = \min\{\sum_{t_{i,j}^k \in p_{i,j_i}} cost_{real}(t_{i,j}^k)\}$   $1 \leq j \leq n_i$

Also,  $cost_{real}(t_{i,j}^k) = 0$ , if  $t_{i,j}^k \in t_{sel}$  and  $cost_{real}(t_{i,j}^k) \geq$

$\frac{cost(t_{i,j}^k)}{m_{i,j}^k}$ , if  $t_{i,j}^k \notin t_{sel}$ .

A key observation here is the inequality above, which in the case of  $f_c$  was an equality. The reason for this being that the sharing estimate may be overly optimistic.

Now,  $f_* = cost_{real}(S_k) = \sum_{t_x \in t_{sel}} cost(t_x) + \sum_{k < i \leq n} cost_{real}(Q_i)$ .

**Lemma 2:** Heuristic cost estimate  $f_c$  is admissible, i.e.  $f_c \leq f_*$ .

**Proof:**

By comparing the definitions of the two cost estimates, and the inequality of  $f_*$  compared to the equality of  $f_c$ , we see that  $f_c \leq f_*$ . Thus,  $f_c$  is admissible.  $\square$

**Theorem 1:** An  $A^*$  algorithm for the multiple query optimization problem using  $f_c$  will expand no more states than one using  $f_s$ .

**Proof:**

Follows from Lemmas 1 and 2, and properties of  $A^*$ [11].  $\square$

## 5 Initial Upper Bound

We know that the total number of states expanded by  $A^*$  is independent of the initial upper bound. However, the upper bound greatly affects the memory requirements since the better it is the more states are pruned at earlier stages of the search. Sellis showed that the performance of  $A^*$  can be improved by having an upper bound to discard all states with  $f_s$  value higher than the upper bound and used the cost of the multi-plan constructed by merging locally optimal single query plans as the initial upper bound. The construction of such a multiplan takes  $O(m^3)$  time (though it can be done in  $O(m)$  time and  $O(t)$  space where  $t$  is the total number of distinct tasks in all plans), where  $m$  is the total number of tasks in the set of locally optimal plans.

*Successive Augmentation* has been successfully used for solving many optimization problems, including single query optimization[16, 8]. Therefore we decided to use it for obtaining a good initial upper bound, since such an initial upper bound would cut down the search space and reduce memory requirements. We will now present our *Successive Augmentation* algorithm as an efficient method to construct a good initial upper bound;

$solution[1 \dots n] = 0$ ; ( $n$  is the number of queries)

$Q = \{Q_1, \dots, Q_n\}$ ;

for ( $i = 1$ ;  $i \leq n$ ;  $i++$ ) {

$next = choose\_query(solution, Q)$ ;

$solution[next] = choose\_plan(solution, Q_{next})$ ;

$Q := Q - \{Q_{next}\}$ ;

The function *choose\_query* is used to determine the order in which queries are considered. For this we find the query with highest sharing with already selected query plans in the sense that the difference (*original\_cost* - *shared\_cost*) is the largest. Here *original\_cost* is the cost of a plan obtained by adding together the costs of

all tasks in it, and *shared\_cost* is the remaining cost of a plan by ignoring its tasks which are already merged into the multi-plan.

### 5.1 Dynamic Query Orderings

As seen from Sellis' results the query ordering used for selecting the order in which queries will be merged into the multi-plan affects the total number of states expanded throughout the search greatly. However, the fact that Sellis' ordering heuristics are static and are calculated without considering the current partial multi-plan affects their benefit to the search negatively. Our *dynamic query ordering* is done at run time by analyzing a given partial multi-plan. For this, all alternative plans of queries which are not assigned a plan in this multi-plan are checked to find out the one with maximum sharing with the tasks already in the multi-plan. The amount of sharing can be calculated by any formula, we preferred to use the difference (*original\_cost* - *shared\_cost*), but the ratio (*original\_cost*/*shared\_cost*) also performs well.

## 6 Experimental Results

We ran our heuristics on query sets consisting of 10-20 queries and obtained results for  $f_c$  with both  $A^*$  and  $DFBB$ . The queries were assigned between 3 to 5 plans which were again generated randomly. The number of queries were increased from 10 to 20 and a random query set was generated for each of them and then the various heuristics were run on the same queries to perform an experimental comparison between them. Note that we preferred to perform only one experiment for each query set size (rather than providing averages as done in [14]) as this provides us better insight into the performance of each heuristic function in comparison to the others.

The results in Table 1 show that  $f_c$  performs substantially better than  $f_s$ , ( we had to stop at 12 queries for  $f_s$  as it took too long to run the experiments). The use of *dynamic query ordering* also improves the performance of the search algorithm which was verified by our experiments given in Table 2 as  $A^*$  with  $f_c$  performed better with *dynamic query ordering*.  $DFBB$  has also been found to be promising since it reduces the amount of work involved in calculating our heuristic function,  $f_c$ . In fact our results in Table 3 show that even though  $A^*$  expands fewer states than  $DFBB$ , most of the time  $DFBB$  executes much faster. For example, as seen in Table 3, for 18 queries both  $A^*$  and  $DFBB$  expanded 132 states but  $DFBB$  performed only 786 "plan merge" operations ("plan merge" is the operation used for calculating  $f_c$ ) while  $A^*$  performed 1096 "plan merge" operations. We expect  $DFBB$  to be a good alternative to  $A^*$  both because of its low memory requirements during search and also its efficiency in calculating  $f_c$ .

TABLE 1: THE COMPARISON OF  $f_s$  AND  $f_c$ .

no. of queries	$A^*+f_s$	$A^*+f_c$
10	61635	51
11	57694	50
12	96290	103

TABLE 2: THE COMPARISON OF DYNAMIC QUERY ORDERING WITH STATIC QUERY ORDERING.

no. of queries	$A^*+f_c$	$A^*+f_c+DQO$
15	179	51
16	289	178
17	228	142
18	286	132
19	119	58
20	1000	442

TABLE 3: THE COMPARISON OF  $DFBB$  WITH  $A^*$ .

no. of queries	$A^*+f_c+DQO$	$DFBB+f_c+DQO$
	no. of states/ no. of planmerges	
15	51/646	91/716
16	178/1195	186/857
17	142/1065	142/717
18	132/1096	132/786
19	58/713	58/610
20	442/3059	563/2164

## 7 Conclusions

From these experimental results we have realized that  $f_c$  performs quite well using *dynamic query ordering*.  $DFBB$  also helps to improve the overall performance of the search since it reduces the number of times  $f_c$  needs to be calculated. The low memory requirements for  $DFBB$  makes it a good candidate for smaller systems, as well. Another desirable property of  $DFBB$  is that unlike  $A^*$ , a complete multi-plan is found in the shortest possible time and the optimizer need not wait for "the optimum" to be found if the time allowed for optimization is limited.

## References

- [1] U.S. Chakravarthy and J. Minker. Processing multiple queries in database systems. In *Database Engineering 5.3*, pages 38-44, 1982.
- [2] U.S. Chakravarthy and J. Minker. Multiple query processing in deductive database using query graphs. In *Proc. of the VLDB Conf.*, pages 384-391, 1986.
- [3] U.S. Chakravarthy and A. Rosenthal. Anatomy of a modular multiple query optimizer. In *Proc. of the VLDB Conf.*, pages 230-239, 1988.

- [4] A. Cosar, J. Srivastava, and S. Shekhar. On the multiple pattern multiple object (mpmo) match problem. In *Int'l Conf. on Management of Data*, 1991.
- [5] R. Dechter and J. Pearl. *Network-based Heuristics for Constraint Satisfaction Problems*. Springer-Verlag, 1988.
- [6] S. Finkelstein. Common expression analysis in database applications. In *Proc. of the ACM-SIGMOD Int'l Conf. on the Management of Data*, pages 235–245, 1982.
- [7] J. Grant and J. Minker. On optimizing the evaluation of a set of expressions. *Int'l J. Comput. Inform. Sci.*, 11, March 1982.
- [8] Y.E. Ioannidis and Y.C. Kang. Randomized algorithms for optimizing large join queries. In *Proc. of the ACM-SIGMOD Int'l Conf. on the Management of Data*, pages 312–321, 1990.
- [9] E-P. Lim, J Srivastava, and A. Cosar. An extensive search for optimal multiple query plans. In *Int'l Conf. on Management of Data*, 1992.
- [10] J. Park and A. Segev. Using common subexpressions to optimize multiple queries. In *Proceedings of the Int'l Conf. on Data Engineering*, pages 311–319, 1988.
- [11] J. Pearl. *Heuristics*. Reading, MA: Addison-Wesley, 1984.
- [12] V.N. Rao and V. Kumar. Analysis of heuristic search algorithms. Technical Report TR 90-40, Dept. of Computer Science, Univ. of Minnesota, July 1990.
- [13] T. Sellis. Global query optimization. In *Proc. of the ACM-SIGMOD Int'l Conf. on the Management of Data*, 1986.
- [14] T. Sellis. Multiple query optimization. *ACM Transactions on Database Systems*, 13(1):23–52, 1988.
- [15] T. Sellis and S. Ghosh. On the multiple query optimization problem. *IEEE Trans. on Knowledge and Data Engineering*, 2(2):262–266, 1990.
- [16] A. Swami and A. Gupta. Optimization of large join queries. In *Proc. of the ACM-SIGMOD Int'l Conf. on the Management of Data*, pages 8–17, 1988.
- [17] E. Wong and K. Youssefi. Decomposition: A strategy for query processing. *ACM Transactions on Database Systems*, 1(3):223–241, 1976.