Concise Papers

On the Multiple-Query Optimization Problem

TIMOS SELLIS AND SUBRATA GHOSH

Abstract—**Some recently proposed extensions to relational database systems, as well as deductive database systems, require support for multiple-query processing. In this paper, we examine the complexity of the multiple-query optimization problem in database management systems. We show that the problem is NP-hard. Then we examine the performance of a heuristic algorithm to solve the multiple-query optimization problem and suggest some heuristics for query ordering which improves the efficiency of the algorithm considerably. The paper also presents some experimental results on the performance of various heuristics.**

Index Terms—**Algorithm complexity, deductive databases, heursitic algorithms, NP-hard problem, query optimization.**

I. Introduction

There are many applications where more than one query is presented to the system in order to be processed. First, consider a database system enhanced with inference capabilities (deductive database system) [1]. A single query given to such a system may result in multiple queries that have to be run over the database. This is because a deductive database may include more than one definition for the same predicate (view in a conventional database). For example, the following three rules

$$(\mathrm{R1})\quad A \leftarrow B_1 \wedge B_2 \wedge \cdots \wedge B_l$$

$$(\mathrm{R2})\quad A \leftarrow C_1 \wedge C_2 \wedge \cdots \wedge C_m$$

$$(\mathrm{R3})\quad A \leftarrow D_1 \wedge D_2 \wedge \cdots \wedge D_n$$

define the predicate A. A query on A would have to evaluate all three queries that correspond to the right-hand sides of the above rules. The problem of multiple-query optimization also arises in an environment where queries from various users are processed in batches and the queries share some common relations. Finally, some proposals on processing recursion in database systems suggest that a recursive clause should be transformed to a set of other simpler clauses (recursive and nonrecursive). Therefore, the problem of multiple-query optimization arises in that environment as well.

In the next section, we formalize the multiple-query optimization problem, which we then prove to be an NP-hard problem in Section III. In Section IV, we briefly describe a heuristic algorithm [7]. In Section V, we suggest some improvements to the algorithm and finally, in Section VI we present some performance results and compare various heuristics for ordering queries to improve the efficiency of the algorithm.

II. Multiple-Query Optimization

Assume that a database $\boldsymbol{D}$ is given as a set of relations $\{R_1, R_2, \cdots, R_m\}$, each relation defined on a set of attributes. An access plan for a query Q is a sequence of tasks, or basic relational operations, that produces the answer to Q. For example, given two relations EMP(name, dept_name) and DEPT(dept_name, floor_no), with obvious meanings for the various fields, the following QUEL query which retrieves the names of all employees who work on the first floor

```
retrieve (EMP.name)
where DEPT.floor_no = 1 and EMP.dept_name =
  DEPT.dept_name
```

can be processed by performing the following tasks

```
(T1) TEMP1  = DEPT where floor_no = 1
(T2) TEMP2  = EMP join TEMP1
(T3) RESULT = TEMP2[name].
```

Notice that in general there may exist many possible plans to process a query.

The tasks have some cost associated with them which reflects both the CPU and I/O cost required to process them. The cost of an access plan is the cost of processing its component tasks. Assume now that a set of queries $\mathcal{Q} = \{Q_1, Q_2, \cdots, Q_n\}$ is given. A global access plan for $\mathcal{Q}$ corresponds to a plan that provides a way to compute the results of all n queries. A global access plan can be constructed by choosing one plan for each query and then merging them together. The merging process basically amounts to the identification of identical tasks. We will refer to the minimal cost plans for processing each query Q_i individually, as *locally optimal* plans. Similarly, we use the term *globally optimal* plan to refer to a global access plan with minimal cost. Due to common tasks, the union of the locally optimal plans does not necessarily give the globally optimal plan. Now, we can define the problem of multiple-query optimization problem as follows:

Given n sets of access plans $\mathcal{P}_1, \mathcal{P}_2, \cdots, \mathcal{P}_n$, with $\mathcal{P}_i = \{P_{i1}, P_{i2}, \cdots, P_{ik_i}\}$ being the set of possible plans for processing Q_i, $1 \le i \le n$,

Find a global access plan GP by selecting one plan from each $\mathcal{P}_i$ such that the cost of GP is minimal.

In the following section, we prove that the multiple-query optimization problem is an NP-hard problem.

III. The NP-Hardness Proof

Consider the following decision problem (*Multiple-Query Processing (MQP) Problem*)

Given n sets of access plans $\mathcal{P}_1, \mathcal{P}_2, \cdots, \mathcal{P}_n$ with $\mathcal{P}_i = \{P_{i1}, P_{i2}, \cdots, P_{ik_i}\}$ being the set of possible plans for processing Q_i, $1 \le i \le n$, and a constant K

Is there a global access plan GP such that the cost of GP is less than or equal to K?

Clearly our original optimization problem will be NP-hard if the above decision problem is NP-complete. Next we prove that this latter problem is indeed NP-complete.

Proof: a) It is easy to see that MQP belongs to NP since a nondeterministic algorithm needs only guess one plan for each query and check whether the cost of the global access plan obtained by merging the guessed local access plans is less than or equal to K. Merging or combining the access plans can be easily done in polynomial time and therefore the checking step takes only polynomial time.

Manuscript received August 7, 1989; revised December 10, 1989. This work was supported in part by the National Science Foundation under the Grant IRI-8802940 and by the University of Maryland Institute for Advanced Computer Studies.

The authors are with the Department of Computer Science, University of Maryland, College Park, MD 20742.

IEEE Log Number 9035104.

b) MQP is NP-complete: We transform 3SAT [2] to MQP. Let $\mathcal{U} = \{x_1, x_2, \cdots, x_n\}$ and $\mathcal{C} = C_1 \wedge C_2 \wedge \cdots C_m$ be an instance of 3SAT. We construct a set of queries $\mathcal{Q}$ with their possible access plans and a positive number K such that $\mathcal{Q}$ has a global access plan of cost less than or equal to K if and only if $\mathcal{C}$ is satisfiable. For each variable $x_i \in \mathcal{U}$, we construct a query Q_i with two access plans: each plan has only one task corresponding to $x_i = T$ or $x_i = F$, respectively. Note that only one plan will be chosen in the global access plan ensuring that x_i will be either set to T or F. For each clause $C_i \in \mathcal{C}$, we construct a query Q_{n+i} with three access plans corresponding to the three literals in C_i. Each access plan will again have only one task corresponding to one of the literals in C_i. The cost of each task is set to unit cost (i.e., 1). Tasks corresponding to the same literal in different queries will be identical tasks. Finally, we set $\mathcal{Q} = \{Q_i \mid 1 \leq i \leq m + n\}$ and $K = n$. The following table shows an example of the MQP instance obtained when $\mathcal{U} = \{x_1, x_2, x_3\}$ and $\mathcal{C} = (x_1 \vee x_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee x_2 \vee x_3)$ (t_{ij}^l denotes the lth task of the jth plan of query Q_i).

Query	Plan	Task	Cost
Q_1	P_{11}	$t_{11}^1(x_1 = T)$	1
	P_{12}	$t_{12}^1(x_1 = F)$	1
Q_2	P_{21}	$t_{21}^1(x_2 = T)$	1
	P_{22}	$t_{22}^1(x_2 = F)$	1
Q_3	P_{31}	$t_{31}^1(x_3 = T)$	1
	P_{32}	$t_{32}^1(x_3 = F)$	1
Q_4	P_{41}	$t_{41}^1(x_1 = T)$	1
	P_{42}	$t_{42}^1(x_2 = T)$	1
	P_{43}	$t_{43}^1(x_3 = F)$	1
Q_5	P_{51}	$t_{51}^1(x_1 = F)$	1
	P_{52}	$t_{52}^1(x_2 = T)$	1
	P_{53}	$t_{53}^1(x_3 = T)$	1

The identical tasks are

$$t_{11}^1 \equiv t_{41}^1; \quad t_{12}^1 \equiv t_{51}^1; \quad t_{21}^1 \equiv t_{42}^1 \equiv t_{52}^1; \quad t_{31}^1 \equiv t_{53}^1; \quad t_{32}^1 \equiv t_{43}^1.$$

It is easy to see that the above construction can be accomplished in polynomial time.

Now, all we need to show is that $\mathcal{C}$ is satisfiable if and only if $\mathcal{Q}$ has a global access plan GP of cost less than or equal to K. Suppose first that $t: \mathcal{U} \rightarrow \{T, F\}$ is a satisfying truth assignment for $\mathcal{C}$. Then we construct a global access plan GP of cost n as follows. For each query Q_i, $1 \leq i \leq n$, choose the access plan P_{i1} if $t(x_i) = T$, and P_{i2} if $t(x_i) = F$. For each query Q_i, $n < i \leq m + n$, choose the access plan corresponding to the literal set true by the assignment t. Since $\mathcal{C}$ is satisfiable at least one literal in each clause will be set true by t. In case multiple literals in a clause become true, choose any one of the corresponding plans arbitrarily. Clearly the cost of the first n access plans is equal to n (since they share no tasks) and no additional cost is needed to process the rest of the m queries (since the task in each of their access plans is identical to one of the tasks in the first n access plans).

Now, suppose there exists a global access plan GP of cost less than or equal to n. Then we have selected one access plan for each query Q_i, $1 \leq i \leq m + n$. As above, processing the first n plans costs n units. Then we are left with m queries corresponding to the m clauses without any additional processing cost. Therefore, the plan chosen for each of these queries must have tasks identical to the tasks in the first n plans. Now the satisfying assignment for $\mathcal{C}$ can be easily constructed from the plans chosen as follows. If the ith plan, $1 \leq i \leq n$, selected is P_{i1} then x_i is set to T, while if the ith plan chosen is P_{i2} then x_i is set to F.

The above completes the proof that the multiple-query optimization problem is NP-hard. In the next section, we briefly describe a heuristic algorithm of [7] that can be used to find a global access plan for a set of queries.

IV. A Heuristic Algorithm

[7] suggests a heuristic algorithm which performs a search over some state space defined over access plans. A similar branch and bound algorithm has been suggested by Grant and Minker in the past [3], while a dynamic programming algorithm was recently suggested by Park and Segev [5]. The search space in [7] is constructed by defining one state for each possible combination of plans among the queries. The initial state is empty, then each query is considered in turn, adding one of its possible plans into the state description. This way, transitions between states are performed. The cost of a transition is the additional cost to process the newly added query, given that the tasks of the queries already encountered have been processed and may be shared. At the end, when all queries have been examined, a final state is reached, i.e., a state that has one plan for each query. The heuristic algorithm traverses this state space to find the cheapest way to go from the initial to a final state. In the following, we give a brief formal description of the state space and the algorithm of [7].

Definition 1: A *state* s is an n-tuple $< P_{1j_1}, P_{2j_2}, \cdots, P_{nj_n} >$, where $P_{ij_i} \in \{\text{NULL}\} \cup \mathcal{P}_i$. If $P_{ij_i} = \text{NULL}$, it is assumed that state s suggests no plan for processing query Q_i. We denote by $\mathcal{S}$ the set of all possible states.

Definition 2: Given a state $s = < P_{1j_1}, P_{2j_2}, \cdots, P_{nj_n} >$, we define a function $next: \mathcal{S} \rightarrow \mathcal{Z}$ ($\mathcal{Z}$ is the set of integers) as follows

$$next(s) = \begin{cases} \min \{i \mid P_{ij_i} = \text{NULL}\}, & \text{if } \{i \mid P_{ij_i} = \text{NULL}\} \neq \emptyset \\ n + 1 & \text{otherwise.} \end{cases}$$

Let $s_1 = < P_{1j_1}, P_{2j_2}, \cdots, P_{nj_n} >$ and $s_2 = < P_{1k_1}, P_{2k_2}, \cdots, P_{nk_n} >$ be the two states such that s_1 has at least one NULL entry. Also let $m = next(s_1)$. A *transition* $T(s_1, s_2)$ from state s_1 to state s_2 exists iff $P_{ik_i} = P_{ij_i}$, for $1 \leq i < m$, $P_{mk_m} \in \mathcal{P}_m$ and $P_{ik_i} = \text{NULL}$, for $m < i \leq n$.

Definition 3: The *cost* of a transition $T(s_1, s_2)$ is defined as the *additional cost* needed to process the new plan P_{mk_m} introduced (according to Definition 2), given the (intermediate or final) results of processing the plans of s_1.

Finally, we define the initial and the final states of the algorithm. The state $s_0 = <$ NULL, NULL, $\cdots$, NULL $>$ is the initial state of the algorithm and the states $s_F = < P_{1j_1}, P_{2j_2}, \cdots, P_{nj_n} >$ with $P_{ij_i} \neq \text{NULL}$, for all i, are the final states.

The A^* algorithm starts from the initial state s_0 and finds a final state s_F such that the cost of getting from s_0 to s_F is minimal among all paths leading from s_0 to any final state. The cost of such a path is the total cost required for processing all n queries. Given a state s, we will denote by $scost(s)$ the cost of getting from initial state s_0 to s.

In order for the A^* algorithm to have a fast convergence, a *lower bound* function h is defined on the states [4]. This function is used to prune down the size of the search space that will be explored. We use the following function $h: \mathcal{S} \rightarrow \mathcal{Z}$ on a given state $s = < P_{ik_1}, P_{2k_2}, \cdots, P_{nk_n} >$

$$h(s) = \sum_{i=1}^{next(s)-1} est\_cost(P_{ik_i}) + \sum_{i=next(s)}^{n} \min_{j_i} [est\_cost(P_{ij_i})] - scost(s).$$

The function est_cost is defined as follows

$$est\_cost(t) = \frac{cost(t)}{n_q}$$

where n_q is the number of queries the task t occurs in and $cost(t)$ is the cost of t. For a plan P_{ij_i}, it is assumed that

$$est\_cost(P_{ij_i}) = \sum_{t \in P_{ij_i}} est\_cost(t).$$

In the worst case, the above algorithm may require time exponential on the number of plans per query but on the average the complexity depends on how closely the *lower bound* function estimates the actual cost. In the next section, we present some improvements to the above algorithm and in Section VI we give some experimental results.

V. Improvements

A. Upper Bound

The efficiency of the above search algorithm can be further improved by computing an upper bound on the cost of the optimal global access plan. For each $\mathcal{P}_i$, let P_{ij_i} be the locally optimal plan. Then the upper bound is

$$\text{UB} = \text{Cost of the global access plan obtained by merging}$$
$$P_{1j_1}, P_{2j_2}, \cdots, P_{nj_n}.$$

The merging can be done as follows:

a) Sort the tasks in $P_{1j_1}, P_{2j_2}, \cdots, P_{nj_n}$ in increasing order of their cost.

b) Delete the duplicates (same cost tasks) if they are identical by checking the identical tasks list.

Let m be the total number of tasks in $P_{1j_1}, P^i_{2j_2}, \cdots, P_{nj_n}$, and i be the length of the list of identical tasks (note that $i = O(m^2)$). Step a) takes $O(m \log m)$ time and step b) takes $O(mi)$ time and therefore the total time taken is $O(m \log m + mi) = O(m^3)$. However, since UB needs to be computed only once at the beginning of the search, it does not increase the overhead of the algorithm much. During the search, any state s such that $h(s)$ is greater than UB $-$ $s\,cost(s)$ is discarded. This further cuts down the number of states expanded by the search algorithm and increases the efficiency of the algorithm.

B. Query Ordering

The heuristic algorithm of [7] fills in the state vector s, in increasing order of the query index. Since the *lower bound* function used in the algorithm is partly based on the query order, we derived some heuristics to order the queries in a way such that on the average, lesser number of states are expanded by the heuristic algorithm (a measure of the complexity of the heuristic algorithm). Different query orders generated by our heuristics were tested on the same data set (generated randomly) and compared. On the average, some heuristics seemed to do better than the others and all of them did better than a random query order, like the increasing query index. The following heuristics did particularly well.

1) Order Queries in Increasing Numnber of Plans in Each Query: From the definition of a state, it is clear that the number of plans for the ith query Q_i, that is $|\mathcal{P}_i|$, determines the number of immediate successors of a state at the $(i - 1)$th level in the search space. In fact, according to our definition of state transition, these two quantities are equal. For example, the root state $<$ NULL, NULL, $\cdots$, NULL $>$ has $|\mathcal{P}_1|$ immediate successors, namely $< P_{11}$, NULL, $\cdots$, NULL $>$, $< P_{12}$, NULL, $\cdots$, NULL $>$, $\cdots$, $< P_{1m}$, NULL, $\cdots$, NULL $>$, $m = |\mathcal{P}_1|$. This order of queries where $|\mathcal{P}_1| \leq |\mathcal{P}_2| \leq \cdots \leq |\mathcal{P}_n|$ maximizes the size of the search space below a state at level i, for all i, $1 \leq i \leq n$. This is because the size of the search space rooted at a state at level i is

$$|\mathcal{P}_{i+1}| \times |\mathcal{P}_{i+2}| \times \cdots \times |\mathcal{P}_n|.$$

By sorting $\mathcal{P}_1, \mathcal{P}_2, \cdots, \mathcal{P}_n$ we clearly maximize the above expression for all i, $1 \leq i \leq n$. As a result of this, for any state s at level i not expanded by the algorithm, the total number of states pruned down is maximized.

2) Order Queries in Decreasing Average Query Cost: The average query cost of a query Q_i is given by

$$\frac{\sum_{P_{ij_i} \in \mathcal{P}_i} cost(P_{ij_i})}{|\mathcal{P}_i|}$$

where

$$cost(P_{ij_i}) = \sum_{t \in P_{ij_i}} cost(t).$$

The average query cost does not consider the sharing among the queries. We consider average query cost along with average estimated query cost (in the next heuristic) because estimated query cost is often much less than the real average cost in any global access plan. This is due to the fact that many simultaneous sharings are infeasible [7]. By ordering the queries in decreasing average query cost, we force the algorithm to expand costly queries in the beginning. Since many times, the error in the *lower bound* function is proportional to its actual value, this heuristic tries to minimize the error in the cost of any path.

3) Order Queries in Decreasing Average Estimated Query Cost: The average estimated query cost of a query Q_i is given by

$$\frac{\sum_{P_{ij_i} \in \mathcal{P}_i} est\_cost(P_{ij_i})}{|\mathcal{P}_i|}$$

where $est\_cost(P_{ij_i})$ is as defined above (see Section IV). The intuition behind this heuristic is the same as in that of (2). By assuming that the error in the *lower bound* function is proportional to its actual value, this tries to minimize the error in the cost of any path.

4) Order Queries in Decreasing Average Query Cost per Number of Plans for the Query: This heuristic is a combination of 1) and 2). By dividing the average query cost by the number of plans and then ordering in decreasing value, this heuristic incorporates the effects of both heuristics 1) and 2).

5) Order Queries in Decreasing Average Estimated Query Cost per Number of Plans for the Query: Similarly, this heuristic is a combination of 1) and 3).

VI. Experimental Results

Using the implementation of the heuristic algorithm of [8], we experimented with its performance. The experiments were run over randomly generated query sets. The following quantities were used as input parameters by the random query generator while generating a query set

1) Minimum and Maximum number of queries in a query set.
2) Minimum and Maximum number of plans for a query.
3) Minimum and Maximum number of tasks in a plan.
4) Minimum and Maximum cost of a task.
5) Sharability: This factor represents the percentage of common tasks among the queries.

The actual values of the number of queries, number of plans, number of tasks, and the task cost were generated randomly from the input range. To show the performance of the algorithm and compare the relative merit of various query ordering strategies, we present ten data sets *QSET*0–*QSET*9 (we ran our experiments over a much larger number of data sets). We fixed the number of queries in each query set to ten and varied the sharability from 5% to 50% to study its effect on the performance of the algorithm. Table I shows the parameters used in *QSET*0–*QSET*9. Finally, we computed the savings obtained by doing multiple-query optimization in each case, using the formula

$$Savings = \frac{\sum_{i=1}^{n} best\_cost(Q_i) - cost(\text{GP})}{\sum_{i=1}^{n} best\_cost(Q_i)} \, 100$$

where $best\_cost(Q_i)$ is the cost of the locally optimal plan for processing Q_i and $cost(\text{GP})$ is the cost of the global access plan found by the heuristic algorithm.

For each query set the search algorithm was run over the following six query orders

Order 1: the original order, i.e., increasing query index.
Order 2: increasing number of plans.
Order 3: decreasing average query cost.

TABLE I
QUERY SETS USED IN THE EXPERIMENT

Query Set	No of Plans	Range of No of Tasks	Range of Task cost	Sharability
$QSET0$	[3-6]	[3-6]	[5-100]	5%
$QSET1$	[3-6]	[3-6]	[5-100]	5%
$QSET2$	[3-6]	[3-6]	[5-100]	10%
$QSET3$	[2-6]	[3-5]	[5-100]	10%
$QSET4$	[2-6]	[3-5]	[5-100]	20%
$QSET5$	[2-6]	[3-5]	[5-100]	20%
$QSET6$	[2-6]	[2-4]	[5-100]	30%
$QSET7$	[2-6]	[2-4]	[5-100]	30%
$QSET8$	[2-6]	[2-4]	[5-100]	50%
$QSET9$	[2-6]	[2-4]	[5-100]	50%

TABLE II
SIZE OF THE STATE AND SAVINGS FOR EACH QUERY SET

Query Set	Total no of States	Savings
$QSET0$	8890560	8.06
$QSET1$	4410000	6.74
$QSET2$	19361664	14.53
$QSET3$	7560000	22.60
$QSET4$	3402000	36.20
$QSET5$	2257920	47.40
$QSET6$	691200	38.89
$QSET7$	17781120	35.59
$QSET8$	66679200	53.51
$QSET9$	5443200	50.43

TABLE III
RESULTS OF VARIOUS QUERY ORDERING STRATEGIES

Query Set	Query Order	States Explored	% Expanded		Query Set	Query Order	States Explored	% Expanded
	1	228	0.002565			1	326	0.007392
	2	104	0.001170			2	212	0.004807
$QSET0$	3	177	0.001991		$QSET1$	3	199	0.004512
	4	158	0.001777			4	209	0.004739
	5	104	0.001170			5	262	0.005941
	6	104	0.001170			6	262	0.005941
	1	1162	0.006002			1	1136	0.015027
	2	105	0.000542			2	598	0.007910
$QSET2$	3	150	0.000775		$QSET3$	3	523	0.006918
	4	92	0.000475			4	429	0.005675
	5	123	0.000635			5	266	0.003519
	6	102	0.000527			6	203	0.002685
	1	625	0.018372			1	740	0.032774
	2	42	0.001235			2	69	0.003056
$QSET4$	3	65	0.001911		$QSET5$	3	143	0.006333
	4	156	0.004586			4	125	0.005536
	5	41	0.001205			5	77	0.003410
	6	39	0.001146			6	64	0.002834
	1	1151	0.166522			1	2281	0.012828
	2	81	0.011719			2	113	0.000636
$QSET6$	3	93	0.013455		$QSET7$	3	324	0.001822
	4	92	0.013310			4	236	0.001327
	5	68	0.009838			5	97	0.000546
	6	55	0.007957			6	102	0.000574
	1	106	0.000159			1	484	0.008892
	2	104	0.000156			2	126	0.002315
$QSET8$	3	107	0.000160		$QSET9$	3	101	0.001856
	4	96	0.000144			4	320	0.005879
	5	39	0.000058			5	142	0.002609
	6	111	0.000166			6	166	0.003050

Order 4: decreasing average estimated query cost.

Order 5: decreasing average query cost per number of plans.

Order 6: decreasing average estimated query cost per number of plans.

For each order we recorded the number of states explored, total number of states in the search space, the percentage of the total number of states that were explored, and the savings achieved in each case. The results of the experiments are summarized in Tables II and III.

From the results of our experiment it is clear that the search algorithm expands only a small percentage of the total number of states in the search space. The worst value we have is 0.17% and this was for *QSET*0 with order 1 (original order). Our results indicate another important feature: the relative merit of the various query ordering heuristics. It is clear that in most cases orders 2–6 perform better than the originally given order. In particular, order 6 seems to be very good. This is not surprising because this heuristic takes the number of plans, the cost of the plans, and the sharability, all into account. We also observed another intuitive result, namely that the savings increases with sharability.

VII. Conclusion

In this paper, we first studied the complexity of the multiple-query optimization problem. By reducing 3SAT to MQP (multiple-query processing problem) we proved that the multiple-query optimization problem is NP-hard. We have also presented a heuristic algorithm that can be used to solve the problem. Although the complexity of the algorithm may be exponential in the number of plans per query, we control its performance with a careful choice of the *lower bound* function [4]. Our experiment with the algorithm strengthened our belief that in most situations the complexity of the algorithm is much less than the worst case. We also suggested some improvements to the algorithm by computing an upper bound on the cost of the optimal global access plan and ordering queries intelligently. Experimental results clearly indicate the potential of various ordering strategies in improving the complexity of the algorithm.

As interesting future work we view two major issues. First, we are working on extending the algorithm so that it does not consider identical tasks only, but implications as well. For example, two tasks that perform selections on a relation and the condition of the one is implied by the condition of the other, obviously account for sharing of intermediate results. The second interesting issue is the average case performance analysis of the algorithm. Pearl [6] gives some analysis of average performance of heuristic algorithms assuming a very simple model. Currently we are exploring ways to analytically model a multiple-query optimization environment which is simple enough for analysis yet captures real-life applications. We would also like to compare the average performance of our algorithm to the dynamic programming algorithm of [5].

Acknowledgment

We would like to thank M. Yannakakis of AT&T Bell Laboratories for pointing out the 3SAT to MQO reduction, and the anonymous referees for their constructive criticism that greatly improved the presentation of this paper.

References

[1] H. Gallaire and J. Minker, *Logic and Databases*. New York: Plenum, 1978.
[2] M. R. Garey and D. S. Johnson, *Computers and Intractability*. San Francisco, CA: Freeman, 1979.
[3] J. Grant and J. Minker, "On optimizing the evaluation of a set of expressions," *Int. J. Comput. Inform. Sci.*, vol. 11, Mar. 1982.
[4] N. J. Nillson, *Principles of Artificial Intelligence*. Palo Alto, CA: Tioga, 1980.
[5] J. Park and A. Segev, "Using common subexpressions to optimize multiple queries," in *Proc. 4th Int. Conf. Management Data Eng.*, Feb. 1988, pp. 311-319.
[6] J. Pearl, *Heuristics*. Reading, MA: Addison-Wesley, 1984.
[7] T. Sellis, "Multiple-query optimization," *ACM Trans. Database Syst.*, vol. 13, pp. 23-52, Mar. 1988.
[8] T. Sellis and Y. C. Wong, "The implementation of a heuristic algorithm for the multiple-query optimization problem," unpublished manuscript, Dep. Comput. Sci., Univ. Maryland, College Park, Mar. 1988.