Improvements on a Heuristic Algorithm for Multiple-Query Optimization *

Kyuseok Shim† Timos Sellis‡ Dana Nau§

Department of Computer Science
University of Maryland
College Park, Maryland 20742
{shim,timos,nau}@cs.umd.edu

Abstract

Multiple-query processing has received a lot of attention recently. The problem arises in many areas, such as extended relational database systems and deductive database systems. In this paper we describe a heuristic search algorithm for this problem. This algorithm uses an improved heuristic function that enables it to expand only a small fraction of the nodes expanded by an algorithm that has been proposed in the past. In addition, it handles implied relationships without increasing the size of the search space or the number of nodes generated in this space. We include both theoretical analysis and experimental results to demonstrate the utility of the algorithm.

Index Terms: Multiple query optimization, relational databases, heuristic algorithms, algorithm complexity

*This research was partially sponsored by the National Science Foundation under Grants IRI-8719458, IRI-8907890, NSFD CDR-85-00108 (to the University of Maryland Systems Research Center), and IRI-9057573 (PYI Award), by DEC and Bellcore, by the Air Force Office for Scientific Research under Grant AFOSR-89-0303, and by UMIACS.

†Also supported by a Korean Government Overseas Scholarship.

‡Also with University of Maryland Institute for Advanced Computer Studies (UMIACS).

§Also with the University of Maryland Systems Research Center (SRC).

1 Introduction

There are many applications where more than one query is presented to a database system in order to be processed. First, consider a database system enhanced with inference capabilities (deductive database system). 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. For example, the following three rules

$$\begin{aligned} &(\text{R1}) \quad A \leftarrow B_1 \wedge B_2 \wedge \ldots \wedge B_l \\ &(\text{R2}) \quad A \leftarrow C_1 \wedge C_2 \wedge \ldots \wedge C_m \\ &(\text{R3}) \quad A \leftarrow D_1 \wedge D_2 \wedge \ldots \wedge D_n \end{aligned}$$

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. This can be particularly useful in distributed database systems where queries (or sub-queries) arrive to sites in order to be processed; again, devising an execution schedule that can provide answers to all queries avoiding duplicate page accesses, becomes a very crucial problem.

Sellis in [Sel88, SG90] suggests a heuristic algorithm to solve the MQO problem. The algorithm performs a search over some state space defined over access plans. A similar branch and bound algorithm has been suggested in [GM82] by Grant and Minker, while a dynamic programming algorithm was later suggested by Park and Segev [PS88]. In addition, a problem similar to the MQO problem arises in the field of Artificial Intelligence, in the problem of generating plans (i.e., sequences of actions) to achieve multiple goals. One technique that has been used successfully in some application domains (including manufacturing planning [KNY92]) has been to develop separate plans for each goal, and then to merge these plans together to produce a plan for the conjoined goal. In order to find an optimal way to merge these plans, a heuristic search algorithm has been developed [YNH89, YNH90, YNH92] that is similar in some respects to Sellis' algorithm.

In this paper we describe a new algorithm for the MQO problem. This algorithm represents an extension and improvement of both Sellis' [Sel88, SG90] algorithm for the MQO problem and the plan-merging algorithm of Yang et al [YNH89]. It uses an improved heuristic function that enables it to expand only a fraction of the nodes expanded by Sellis' algorithm. In addition, it handles implied relationships without increasing the size of the search space or the number of nodes generated in this space. To demonstrate the superiority of this algorithm, we compare its performance to Sellis' algorithm both analytically and experimentally.

In the next section we formalize the multiple-query optimization problem. In Section 3, we briefly outline the search algorithms used in this paper and describe Sellis' algorithm [Sel88, SG90] which will be used as a point of comparison to ours. In Section 4, we describe our algorithm and discuss its performance. In Section 5, we show how both Sellis' algorithm and our algorithm can be modified to handle implied relationships without increasing the size of the search space or the number of nodes generated. In Section 6, we present some experimental performance results, while Section 7 presents concluding remarks.

2 Multiple-Query Optimization

Assume that a database **D** is given as a set of relations $\{R_1, R_2, \ldots, R_m\}$, 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 SQL query which retrieves the names of all employees who work on the first floor

```
select EMP.name
from EMP, DEPT
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 exist many possible alternative plans to process a query.

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 $\mathcal{Q}=\{ Q_1, Q_2, \ldots, Q_n\}$ 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. 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 Multiple-Query Optimization (MQO) problem as follows:

> **Given** n sets of access plans $\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_n$, with $\mathcal{P}_i=\{P_{i1}, P_{i2}, \ldots, P_{ik_i}\}$ being the set of possible plans for processing $Q_i, 1 \leq i \leq n$,
>
> **Find** a global access plan GP by selecting one plan from each $\mathcal{P}_i$ such that the cost of GP is minimal.

3 State-Space Search Algorithms for MQO

The search space for the MQO problem is constructed by defining one state for each possible combination of plans among the queries [Sel88, SG90]. In particular, suppose we need to perform n queries $Q_1, Q_2, \ldots, Q_n$, and suppose that for each query Q_i, the set of possible access plans for processing Q_i is $\mathcal{P}_i = \{P_{i1}, P_{i2}, \ldots, P_{ik_i}\}$. Then the state space is constructed as follows:

1. Every state s is an n-tuple $< P_{1j_1}, P_{2j_2}, \ldots, P_{nj_n} >$, where for each i, either $P_{ij_i} \in \mathcal{P}_i$ (in which case s suggests that P_{ij_i} be used to process query Q_i), or else P_{ij_i} = NULL (in which case s does not suggest a plan for processing Q_i). The cost of s, denoted by $scost(s)$, is the total cost required for processing all of the queries in s.

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

3. Given a state $s =< P_{1j_1}, P_{2j_2}, \ldots, P_{nj_n} >$, let

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

4. Let the state s have at least one NULL entry and $m = next(s)$, then the immediate successors of s include every state $s' =< P_{1k_1}, P_{2k_2}, \ldots, P_{nk_n} >$ satisfying the following properties:

   $P_{ik_i} = P_{ij_i}$ for $1 \leq i < m$;
   $P_{mk_m} \in \mathcal{P}_m$;
   $P_{ik_i} = \text{NULL}$, for $m < i \leq n$.

   The *cost* of the transition from s to s' is the additional cost needed to process the new plan P_{mk_m}, given the (intermediate or final) results of processing the plans in s.

Note that the search space is a tree, in the sense that there is exactly one path from s_0 to each of its successors.

All of the algorithms described in this paper are versions of the following procedure, which is a special case of the $A^\star$ state-space search algorithm [Nil80]:

procedure MQO-search:
    OPEN := $\{s_0\}$
    loop
        remove from OPEN the state s that has the smallest value for $f(s)$
        (if there is more than one such node, then we select the one at the deepest level)
        if s is a final state **then** exit, returning s
        generate s's successor states, and insert them into OPEN
    repeat

The primary difference between MQO-search and $A^\star$ is that $A^\star$ has some additional coding to increase the efficiency in the case where the search space is a graph. In the case of the multiple query optimization problem, the search space is a tree, so this additional coding is unnecessary. MQO-search may equally well be considered to be a branch-and-bound procedure [NKK84].

MQO-search works by starting from the initial state s_0, and expanding states one by one (expanding a state s means generating all its immediate successors). To choose which state to expand next, it always selects for expansion the state s having the smallest value for $f(s) = scost(s) + h(s)$, where $scost(s)$ is the actual cost of getting from s_0 to s, and $h(s)$ is a *heuristic function* which approximates the cost of getting from s to a final state. The performance of $A^\star$ algorithm depends on the choice of $h(s)$. The better this function approximates the cost to a final state, the faster the algorithm. This brings up the issue of admissibility of heuristic functions.

The heuristic function $h(s)$ is *admissible* if it always returns a lower bound on the additional cost that will be incurred in getting from s to a final state. Since MQO-search is a special case of $A^\star$, it follows from theorems given in [Nil80] that if $h(s)$ is admissible,

then MQO-search is guaranteed to find 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 final states. In the worst case, MQO-search may require time exponential in the number of plans per query, but on the average the complexity depends on how closely the heuristic function estimates the actual cost [Pea84].

Given two admissible heuristic functions h_1 and h_2, h_1 is said to be *more informed* than h_2, if for every non-final node n, $h_1(n) \geq h_2(n)$. If h_1 is *more informed* than h_2, then when MQO-search is run using h_2, it is guaranteed to expand at least as many nodes as it does with h_1.

We now return to the discussion of Sellis' heuristic ($A^\star$) algorithm. Let $s =< P_{1k_1}, P_{2k_2}, \ldots, P_{nk_n} >$ be any state. The heuristic function used in [Sel88, SG90] is the function

$$h_t(s) = \Big(\sum_{i=1}^{next(s)-1} est\_cost(P_{ik_i}) + \sum_{i=next(s)}^{n} \min_{j_i} [est\_cost(P_{ij_i})] \Big) - scost(s). \tag{2}$$

The function *est_cost* is defined on plans as

$$est\_cost(P_{ij_i}) = \sum_{t \in P_{ij_i}} est\_cost(t), \tag{3}$$

and on tasks as

$$est\_cost(t) = \frac{cost(t)}{n_q}, \tag{4}$$

where n_q is the number of queries the task t occurs in, and $cost(t)$ is the cost of task t. Since $est\_cost(P_{ij_i}) \leq cost(P_{ij_i})$ for every plan, it is easy to see that h_t is admissible and MQO-search is guaranteed to find an optimal solution. Below we give an example which illustrates the heuristic.

Example 1 Suppose three queries Q_1, Q_2 and Q_3 are given along with their plans: P_{11}, P_{12}, P_{21}, P_{22}, P_{31}, P_{32}. We will use t_{ij}^k to indicate the k-th task of plan P_{ij}. The cost for each task involved in each plan is as follows.

Plan	Task	Cost	Task	Cost	Task	Cost
P_{11}	t_{11}^1	30	t_{11}^2	10	t_{11}^3	10
P_{12}	t_{12}^1	25	t_{12}^2	10		
P_{21}	t_{21}^1	30	t_{21}^2	20		
P_{22}	t_{22}^1	10	t_{22}^2	10	t_{22}^3	5
P_{31}	t_{31}^1	30	t_{31}^2	10	t_{31}^3	10
P_{32}	t_{32}^1	10	t_{32}^2	10	t_{32}^3	5

Given the actual task costs and assuming that the identical tasks are $t_{11}^1 \equiv t_{21}^1 \equiv t_{31}^1$, the estimated costs (*est_cost*) for these tasks are:

Task	t_{11}^1	t_{11}^2	t_{11}^3	t_{12}^1	t_{12}^2	t_{21}^1	t_{21}^2	t_{22}^1	t_{22}^2	t_{22}^3	t_{31}^1	t_{31}^2	t_{31}^3	t_{32}^1	t_{32}^2	t_{32}^3
Estimated cost	10	10	10	25	10	10	20	10	10	5	10	10	10	10	10	5

The estimated costs for the plans are:

Plan	P_{11}	P_{12}	P_{21}	P_{22}	P_{31}	P_{32}
Estimated cost	30	35	30	25	30	25

In the state $< P_{11}, \text{NULL}, \text{NULL} >$, since $scost(< P_{11}, \text{NULL}, \text{NULL} >) = 50$, $f(< P_{11}, \text{NULL}, \text{NULL} >)$ becomes $50 + 30 + min\{30, 25\} + min\{30, 25\} - 50 = 80$.

In the state $< P_{12}, \text{NULL}, \text{NULL} >$, since $scost(< P_{12}, \text{NULL}, \text{NULL} >) = 35$, $f(< P_{12}, \text{NULL}, \text{NULL} >)$ becomes $35 + 35 + min\{30, 25\} + min\{30, 25\} - 35 = 85$. Because $f(< P_{11}, \text{NULL}, \text{NULL} >)$ is less than $f(< P_{12}, \text{NULL}, \text{NULL} >)$, we expand the state $< P_{11}, \text{NULL}, \text{NULL} >$ which results to two new states such as $< P_{11}, P_{21}, \text{NULL} >$ and $< P_{11}, P_{22}, \text{NULL} >$. The portion of the search space explored by MQO-search using this heuristic function h_t is illustrated in Figure 1. Numbers in circles are used to indicate the order in which nodes are generated and the number on each edge represents the cost of a transition between the corresponding states. □

In order to compute the heuristic values quickly, Sellis' algorithm pre-computes an estimated cost for each plan P_{ij}, and during the search, it uses this same estimated cost regardless of what other plans it is combined with. In order to guarantee that this cost will be a lower bound to the actual cost of reaching a final state, the algorithm pro-rates the cost of each task t in P based on the largest possible number of times t can appear in some combination of plans. In states where it is possible for t actually to appear in that many plans, this will produce a tight bound for t, but in states where t cannot appear in that many plans as estimated, the bound will be more conservative than necessary.

Due to the inaccuracy of the estimated costs, the algorithm can explore a large portion of the search space, in some cases exploring more states than the branch and bound algorithm suggested by Grant and Minker [GM82]. For example, in Example 1, the algorithm explored almost every state in the search space. In the Section 4, we introduce a new heuristic function that overcomes this problem.

4 New Heuristic Function

4.1 Heuristic Function

In this section, we propose a new improved heuristic function which overcomes the inefficiencies mentioned above. The new heuristic function pro-rates the cost of common tasks assuming the best case just for the remaining queries to be processed ignoring the tasks which appear in the state.

The new heuristic function for a state $s =< P_{1j_1}, P_{2j_2}, \ldots, P_{nj_n} >$ is defined as

$$h_n(s) = \sum_{i=next(s)}^{n} \min_{j_i}[new\_est\_cost(P_{ij_i})]. \tag{5}$$

The function *new_est_cost* is defined as follows on tasks

$$new\_est\_cost(t) = \begin{cases} 0 & \text{if } t \in P_{ij_i} \text{ for } 1 \leq i < next(s) \\ \frac{cost(t)}{n_r} & \text{otherwise} \end{cases} \tag{6}$$

where n_r is the number of remaining queries to be processed which have any plan containing the task t. For a plan P_{ij_i}, we define *new_est_cost* on the plans (analogous to Equation 3)

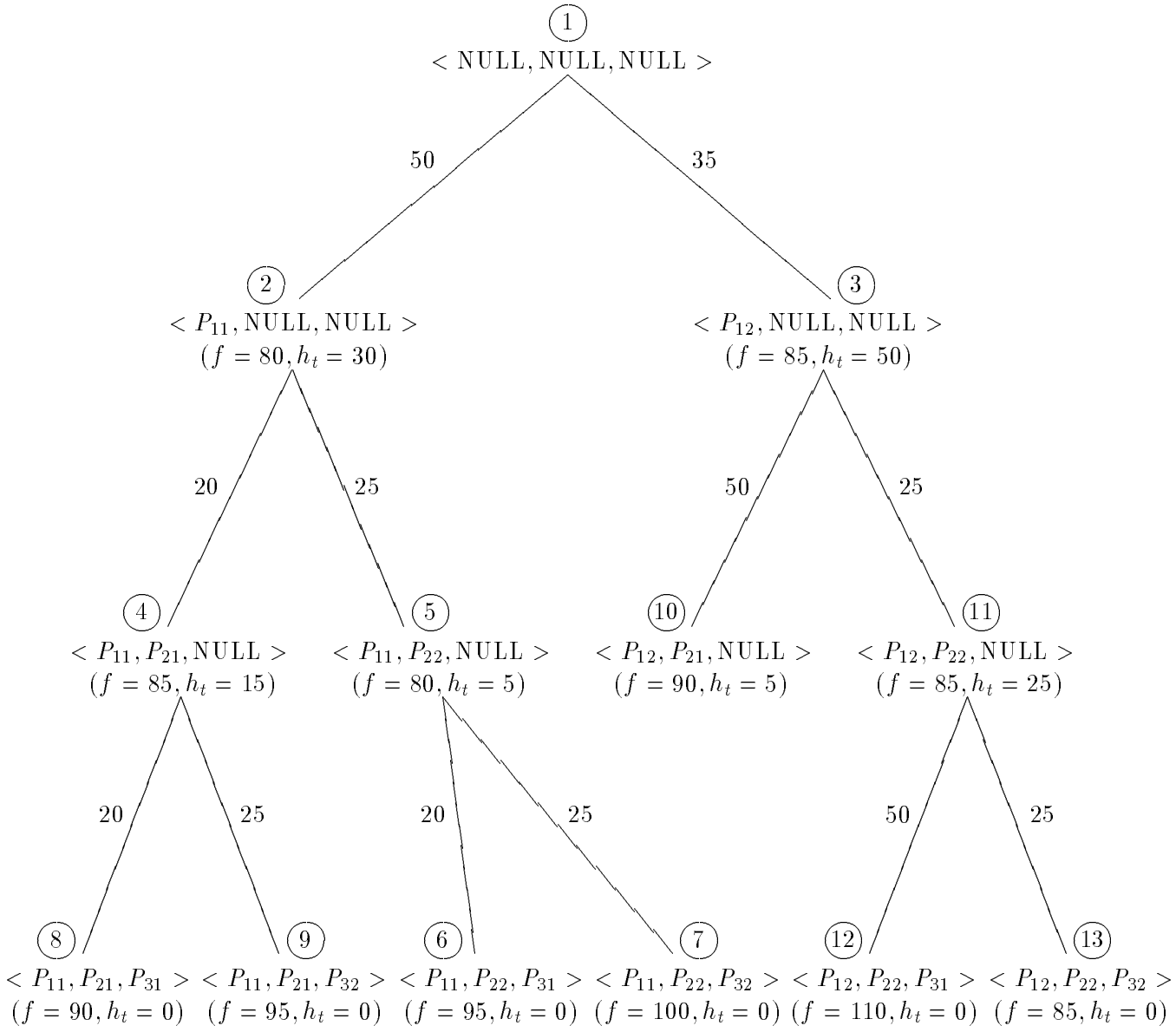

Figure 1: The portion of the search space explored by MQO-search using h_t

as

$$new\_est\_cost(P_{ij_i}) = \sum_{t \in P_{ij_i}} new\_est\_cost(t). \quad (7)$$

To illustrate the effect of h_n, we will redo Example 1. In the state $< P_{11}, \text{NULL}, \text{NULL} >$, the estimated cost of t^1_{21} and t^1_{31} are considered as zero since they are the identical tasks with t^1_{11} and so, appeared in plan P_{11}. For other tasks that are in the plans of remaining queries Q_2 and Q_3, there is no common tasks and so, their estimated cost are considered the same as the actual cost. Since $scost(< P_{11}, \text{NULL}, \text{NULL} >) = 50$, $f(< P_{11}, \text{NULL}, \text{NULL} >)$ becomes $50 + min\{20, 25\} + min\{20, 25\} = 90$.

In the state $< P_{12}, \text{NULL}, \text{NULL} >$, there are no tasks for the plans in remaining queries Q_2 and Q_3 that appear in plan P_{12}. However, there are common tasks which appear only in the plans of the remaining queries. The identical tasks are

$$t^1_{21} \equiv t^1_{31}$$

and so, their estimated cost is considered as the half of the actual cost. The estimated costs for the tasks in the plans of remaining queries are as follows:

Task	t^1_{21}	t^2_{21}	t^1_{22}	t^2_{22}	t^3_{22}	t^1_{31}	t^2_{31}	t^3_{31}	t^1_{32}	t^2_{32}	t^3_{32}
Estimated cost	15	20	10	10	5	15	10	10	10	10	5

Therefore, the estimated costs for the plans in the remaining queries are

Plan	P_{21}	P_{22}	P_{31}	P_{32}
Estimated cost	35	25	35	25

Since $scost(< P_{12}, \text{NULL}, \text{NULL} >) = 35$, $f(< P_{12}, \text{NULL}, \text{NULL} >$ becomes $35 + min\{35, 25\} + min\{35, 25\} = 85$. Because $f(< P_{12}, \text{NULL}, \text{NULL} >)$ is less than $f(< P_{11}, \text{NULL}, \text{NULL} >)$, we expand the state $< P_{12}, \text{NULL}, \text{NULL} >$ which results to two new states such as $< P_{12}, P_{21}, \text{NULL} >$ and $< P_{12}, P_{22}, \text{NULL} >$. The search space for the new heuristic function is shown in Figure 2. Note that 13 states are generated using the heuristic h_t while only 7 states are generated with the new heuristic h_n.

As it can be seen from the above, with h_n we estimated the cost of getting to a final state by taking into account only the remaining queries to be processed ignoring those tasks that appeared in a state s. Therefore, the new heuristic function gives a more tight bound than the previous heuristic function and it makes MQO-search to have faster convergence to the optimal solution. In the next section we study the properties of the new heuristic function h_n and formally prove that it is a better heuristic.

4.2 Proof of Admissibility

Assume that the algorithm visits a state s. The estimated cost of a task is zero, if it appears in a plan already selected in state s. Hence, for a task already selected in the state s, the admissibility criterion is satisfied. Otherwise, the estimated cost of a task t will be calculated as $\frac{cost(t)}{n_r}$, where n_r is the number of remaining queries $Q_{next(s)}$, $Q_{next(s)+1}$, ..., Q_n task t occurs in. For tasks which are not shared among remaining queries, the cost to

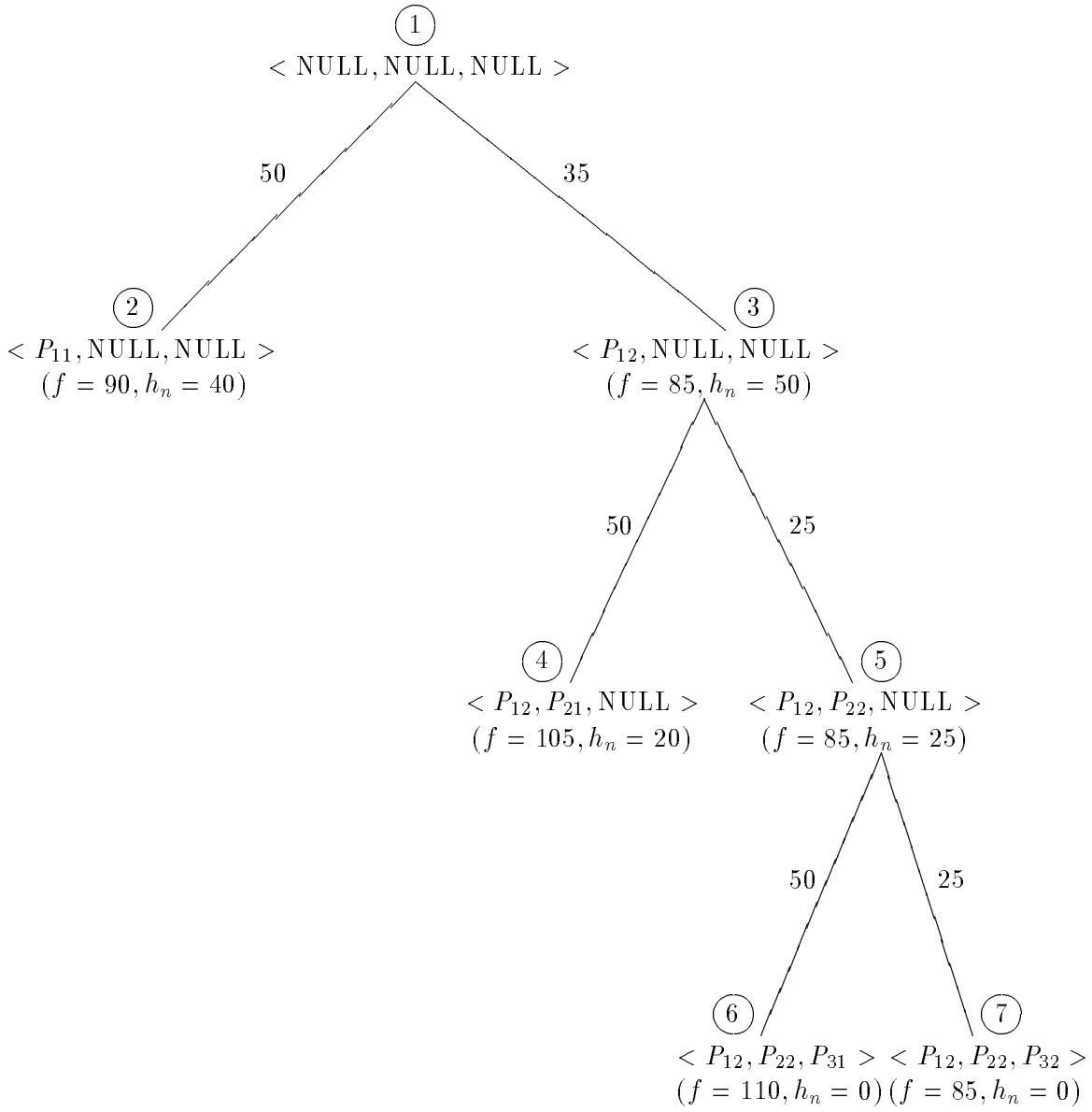

Figure 2: The portion of the search space explored by MQO-search using $h_n(s)$

be considered when calculating h_n will be the tasks' actual cost and therefore it is clear that the criterion for admissibility is also satisfied. However, for tasks which are shared but not chosen yet, and since we add their estimated cost, $\frac{cost(t)}{n_r}$, whenever they are selected, the total cost to be considered in $h_n(s)$ will be at most the same as the actual cost (i.e. $\frac{cost(t)}{n_r} * n_r = cost(t)$). Therefore, the criterion is satisfied for those cases as well, and h_n is admissible.

4.3 Time Complexity

We now compare the complexity of our algorithm with that of Sellis [Sel88, SG90]. First we examine the number of states produced by each algorithm, and then we examine the complexity of computing the heuristic function values at each node.

4.3.1 Number of Nodes Expanded

Let N_i denote the number of candidate access plans for query Q_i. Regardless of whether h_t or h_n is used, the best-case complexity of the search algorithm is the same: $(\sum_{i=1}^{n} N_i) + 1$.

Consider the worst case for h_t. It is easy to see (for example, this is alluded to in [PS88]) that in the worst case, h_t will generate every node in the search space. Thus, in the worst case, the number of nodes generated will be

$$(\sum_{i=1}^{n} \prod_{k=1}^{i} N_k) + 1 = \Theta(N_1 N_2 \ldots N_n).$$

Now, consider the new heuristic h_n. Suppose we are running MQO-search and let s be a state at level $n-1$. With h_n, we estimate the cost of identical tasks, which are already present in a given state $s =< P_{1k_1}, P_{2k_2}, \ldots, P_{(n-1)k_{(n-1)}}, \text{NULL} >$, as zero and estimate the cost of other tasks in the plans of Q_n as their original cost. Note that, excluding those tasks which appear in the state s, either the task in the plans of Q_n has no identical task or it has identical tasks but they exist only among the plans of Q_n and so n_r in Equation 6 becomes 1 in both cases. Therefore, for every state s at level $n-1$, $h_n(s) = h^*(s)$ where $h^*(s)$ is the cost of the minimal cost path from s to a goal state.

Let s be the first state selected for expansion at level $n-1$, and let u be any other state in the OPEN list. Since MQO-search selected s for expansion rather than u, this means that $f(s) \leq f(u)$. Thus for every goal state u' which is a descendant of u, $f(u) \leq scost(u')$. Next, MQO-search will expand s. Since $h_n(s) = h^*(s)$, this means that s has at least one successor s' such that $scost(s') = scost(s) + h_n(s) = f(s)$. Since s' is a goal state, it follows that $h_n(s') = 0$ and thus $f(s') = scost(s') = f(s) \leq f(u)$, so s' is the next state that MQO-search will select for expansion. At this point, MQO-search will realize that s' is the optimal goal, and will terminate, returning s'.

¿From the above argument, it follows that even in the worst case, MQO-search will expand only one state at level $n-1$, and thus will generate only N_n states at level n. Thus, the total number of nodes generated in the worst case will be no greater than

$$(\sum_{i=1}^{n-1} \prod_{k=1}^{i} N_k) + N_n + 1 = \Theta(N_1 N_2 \ldots N_{n-1}).$$

Thus, in the worst case, Sellis' algorithm will generate $\Theta(N_n)$ times as many states as ours.

We can prove that in all cases, the number of nodes expanded by using h_n will never be more than the number of nodes expanded by using h_t. To do this, we prove that h_n is more informed than h_t. Let us assume that the set of identical tasks is $\{c_1, c_2, \ldots, c_g\}$ and the number of queries among $Q_1, Q_2, \ldots, Q_n$ in which the task c_i occurs is n_{q_i}. Let us also assume for a state s that the number of plans in which task c_i occurs is n_{p_i} and P_{ij_i} are the plans selected in state s. Then, the cost of getting from the initial state s_0 to the current state s is

$$scost(s) = \sum_{i=1}^{next(s)-1} est\_cost(P_{ij_i}) + \sum_{i=1}^{g}(n_{q_i} - n_{p_i})est\_cost(c_i)$$

Therefore, according to Equation 2,

$$\begin{aligned} h_t(s) &= \sum_{i=1}^{next(s)-1} est\_cost(P_{ij_i}) + \sum_{i=next(s)}^{n} \min_{j_i}[est\_cost(P_{ij_i})] - scost(s) \\ &= \sum_{i=next(s)}^{n} \min_{j_i}[est\_cost(P_{ij_i})] - \sum_{i=1}^{g}(n_{q_i} - n_{p_i})est\_cost(c_i) \\ &\leq \sum_{i=next(s)}^{n} \min_{j_i}[new\_est\_cost(P_{ij_i})] = h_n(s). \end{aligned}$$

4.3.2 Complexity of Computing h_t and h_n

We turn next to discuss the complexity of computing the h_t and h_n functions on a given state. The computation of Sellis' heuristic h_t is very fast. Once we find the cheapest plan for each query according to *est_cost*, we just add their *est_cost* values and subtract $scost(s)$ (cf. Equation 2). Hence, assuming that we pre-compute the estimated cost values before starting the algorithm, the complexity of computing the heuristic h_t is just $O(1)$ in each state. However, in our improved heuristic, we have to re-calculate the estimated cost in each state of the search space. Thus, it follows that for each node, we spend more time computing h_n compared to computing h_t. However the amount of time spent for MQO-search is not determined solely by the complexity of computing h_t or h_n at each node. Instead, in order to expand any state, we have some execution time overhead to deal with stack management for recursion, to calculate $scost(s)$, and to maintain sorted list of OPEN in decreasing f value. Although we spend more time at each node, we generate sufficiently fewer states that we would expect our search algorithm to have a lower time complexity. This hypothesis has been confirmed experimentally, as shown in Section 6. The details of computing efficiently the heuristic function appear in Section 6.2.

5 Extending the Algorithms to Handle Implications

In this section, we extend both Sellis' algorithm and ours to cover implied relationships among tasks.

A task t_i *implies* task t_j ($t_i \Rightarrow t_j$) iff t_i is a conjunction of selection predicates on attributes $A_1, A_2, \ldots, A_l$ of some relation R, and t_j is a conjunction of selection predicates on the same relation R and on attributes $A_1, A_2, \ldots, A_k$ with $l \leq k$, and it is the case that for any instance of the relation R the result of evaluating t_i is a subset of the result of evaluating t_j. Two tasks t_i and t_j are *identical* ($t_i \equiv t_j$) if $t_i \Rightarrow t_j$ and $t_j \Rightarrow t_i$.

For a task t_i, let t_j be the tasks such that $t_i \Rightarrow t_j$ holds but $t_j \Rightarrow t_i$ does not hold. Then the result of t_j can be used as the input for computing the result of task t_i. This results in savings in terms of the time needed to compute t_i. Of course, to get this savings t_j has to be executed before t_i. To express this, we use the notation $t_j \rightarrow t_i$.

Let R be a relation and t_i and t_j be two tasks on R such that $t_i \rightarrow t_j$ or $t_i \equiv t_j$. Then, t_j can be processed using the result of t_i instead of R. Let C_R be the cost of accessing R to evaluate t_i and C_{t_i} be the cost of accessing the result of t_i. We assume that the results of t_i and t_j are stored for later use (temporary results). Then, the cost of processing t_i is C_R(to read the data) + C_{t_i}(to write the result). With sharing, the savings that can be achieved are

$$savings(t_j) = \begin{cases} C_R - C_{t_i} & \text{if } t_i \rightarrow t_j \\ C_R + C_{t_i} & \text{if } t_i \equiv t_j \end{cases} \quad (8)$$

For the first case, we incur a savings since instead of accessing R we access the result of t_i. In the second case, more savings is achieved because not only R needs not be accessed (since the result of t_j is identical to that of t_i), but the temporary result of t_i can also be used as the result of t_j. Therefore, there is no need to write the result of t_j in a separate temporary relation.

Sellis [Sel88] suggested the following way to handle implied relationships. Consider two queries, Q_1 and Q_2 such that Q_1 has a more restrictive selection than Q_2. Clearly it would be better to consider executing Q_2 first since, in that case, the result of Q_2 can be used to answer Q_1, the opposite being impossible. Therefore arbitrary ordering for the queries in MQO-search would be ineffective. Sellis suggested as a possible solution a new way to fill the next available NULL slots in a state vector s. Instead of using the function $next(s)$ of Section 3 to identify the next query to be considered, the algorithm would be allowed to replace any available (NULL) position of s. This results in a larger fanout for each state and clearly more processing for MQO-search. The worst case complexity of the states explored with this modification for implied relationships is $(\sum_{i=1}^{n} \prod_{j=1}^{i} \sum_{k=1}^{n} N_k) + 1$, while it is $(\sum_{i=1}^{n} \prod_{k=1}^{i} N_k) + 1$ for identical relationships only. Clearly this adds a significant overhead to the algorithm.

In the rest of this section we extend the heuristic algorithms to cover implied relationships.

5.1 Merging Tasks

After a global optimizer picks the appropriate plans among several possible query plans, a sequence of tasks must be produced to indicate the order in which these tasks need to be processed. For example, consider the two access plans, P_1 and P_2, of Figure 3. The task ordering for the global access plan is shown in Figure 4. To form such a global plan, the implication relationships as well as identical relationships need to be taken under account.

There are two kinds of interactions among the tasks in the access plans for MQO problem.

1. An *identical-task* interaction is an interaction when a task in one plan is identical to a task in one of the other plans.

2. A *mergeable-task precedence* interaction is an interaction which requires that a task

a in some plan P_{ik_i} is executed before a task b in some other plan P_{jk_j} in order to achieve savings by using a's result. We denote this interaction as $a \rightarrow b$.

Note that the precedence graph of the tasks in the access plans is acyclic, and thus, it is always possible to merge a set of plans into a global access plan.

Suppose we are given the following:

1. A set of plans $\mathcal{P} = \{P_{1j_1}, P_{2j_2}, \ldots, P_{nj_n}\}$ containing one plan P_{ij_i} for each query Q_i. Let N be the total number of tasks in $\mathcal{P}$.

2. A list of interactions among the tasks in the plans such as identical-task interactions and mergeable-task precedence interactions. Let m be the total number of interactions in this list. Then $m = O(N^2)$.

The global access plan is the set of tasks in $\mathcal{P}$, with additional ordering constraints imposed upon the tasks to handle the interactions. This *merged* plan is denoted by $merge(\mathcal{P})$ and to generate it, we use the following procedure which is a topological sorting algorithm with a simple extension to handle the identical tasks. We call it the 'Merge Algorithm' and it is shown in Figure 5.

The Merge Algorithm works as follows. First, we initialize the list of global ordering to be empty and mark all tasks in $\mathcal{P}$ as unvisited. Then, we check each task in $\mathcal{P}$ in turn. When an unvisited task is found, we visit it using procedure DFS which is a Depth-First-Search algorithm with an extension to handle the identical tasks.

In each call to DFS(t), we mark the task t and its identical tasks in $\mathcal{P}$ as visited and we call them *equivalence_group*(t). Then, we examine each task u adjacent to t (denoted by $Adj[t]$ above) and recursively visit u if it is unvisited. Here, adjacent tasks are those tasks in $\mathcal{P}$ which should be executed later than t due to a mergeable-task precedence interaction. After this, for each task u in equivalence_group(t), we examine each task which comes later than u in the same plan and recursively visit it, if it is unvisited. After we finish these steps, we insert the task t into the front of the list of global ordering. This merging can be done in $O(N^2)$ time and savings in cost can be calculated based on the above result order.

In the next subsection, we extend the heuristic algorithm of Section 3 to handle the case of implied relationships.

5.2 Extension to the Heuristic Algorithm

As we mentioned in the beginning of this section, the worst-case complexity of the states explored with the heuristic algorithm for implied relationships by Sellis in [Sel88] is $(\sum_{i=1}^{n} \prod_{j=1}^{i} \sum_{k=1}^{n} N_k) + 1$, since we have to consider not only one query but all remaining queries in each state. However, if we use the merge algorithm of the previous section, we can still consider one query in each state and the size of the search space becomes the same as without implied relationships, i.e. $(\sum_{i=1}^{n} \prod_{k=1}^{i} N_k) + 1$.

Now we show how the heuristic function h_t in [Sel88, SG90] can be extended to deal with implied relationships. To model the existence of both identical and implied relationships, we define equivalence groups among tasks. An equivalence group for a task t contains all tasks which are identical to the task t, including t itself. Suppose we have a mergeable-task precedence interaction such that $g_1 \rightarrow g_2 \rightarrow \ldots \rightarrow g_n$, and g_i has m_i identical tasks. Let

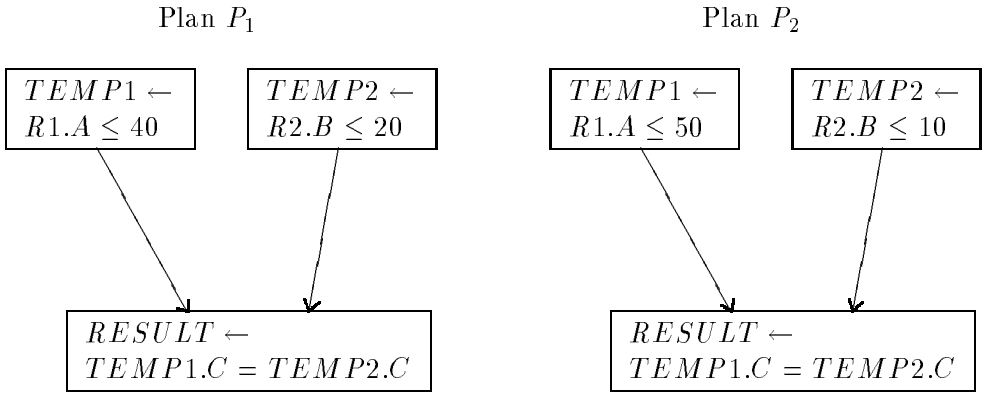

Figure 3: Two access plans P_1 and P_2

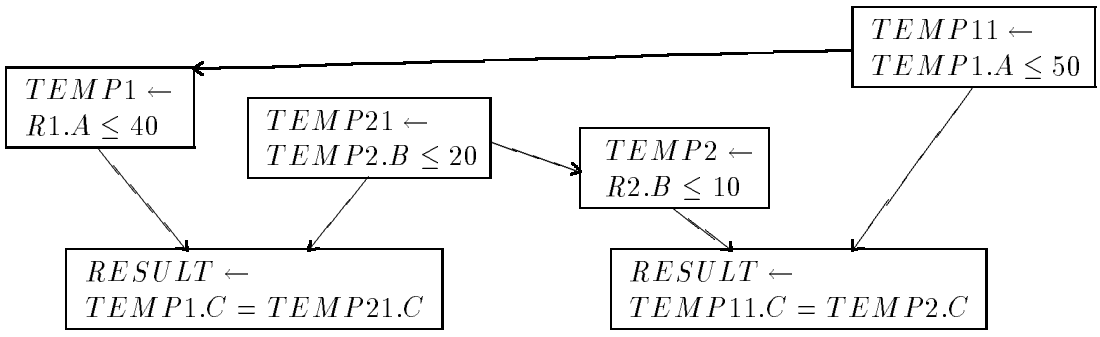

Figure 4: Task ordering in the global access plan for P_1 and P_2

procedure Merge:
    Global_Order := $\emptyset$
    for each task $t \in \mathcal{P}$ **do**
        visited[t] := **false**
    for each task $t \in \mathcal{P}$ **do**
        if not visited[t] **then** DFS(t)
    return Global_Order

procedure DFS(t):
    for each task $u \in$ equivalence_group(t) **do**
        visited[u] := **true**
    for each task $u \in$ Adj[t] **do**
        if not visited[u] **then** DFS(u)
    for each task $u \in$ equivalence_group(t) **do**
        for each task v which comes
            later than u in the same plan **do**
            if not visited[v] **then** DFS(v)
    Insert the task t onto the front of Global_Order

Figure 5: Merge Algorithm

$gcost(g_i)$ be $C_{g_{i-1}}$(to read the data from g_{i-1}) + C_{g_i} (to write the result) and $C_{g_0} = C_R$. Then, the estimated cost of task g_i in an equivalence group and having a mergeable-task precedence interaction is defined as

$$est\_cost\_group(g_i) = gcost(g_i) \tag{9}$$

while the estimated cost of other tasks which do not occur in a mergeable-task precedence interaction is the cost of the tasks themselves. Then, the estimated cost of a task t is

$$est\_cost(t) = \frac{est\_cost\_group(t)}{n_q} \tag{10}$$

where n_q is the number of queries the task t occurs in with identical-task interaction.

The estimated cost of a plan and the heuristic function is the same as in Section 3, i.e.

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

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

Since $est\_cost(P_{ij_i}) \leq cost(P_{ij_i})$ for every plan, it is easy to see that h_t is admissible and MQO-search is guaranteed to find an optimal solution. Let us give an example to illustrate how this heuristic function works.

Example 2 Suppose three queries Q_1, Q_2 and Q_3 are given along with their plans: P_{11}, P_{12}, P_{21}, P_{22}, P_{31}, P_{32}. We will again use t_{ij}^k to indicate the k-th task of plan P_{ij}. The cost for each task involved in each plan is as follows.

Plan	Task	Cost	Task	Cost	Task	Cost
P_{11}	t_{11}^1	35	t_{11}^2	10	t_{11}^3	10
P_{12}	t_{12}^1	25	t_{12}^2	20		
P_{21}	t_{21}^1	30	t_{21}^2	20		
P_{22}	t_{22}^1	10	t_{22}^2	40		
P_{31}	t_{31}^1	42	t_{31}^2	10		
P_{32}	t_{32}^1	10	t_{32}^2	30	t_{32}^3	10

Let us assume that the mergeable-task precedence interaction list has information of the form $t_{31}^1 \rightarrow t_{11}^1 \rightarrow t_{21}^1$ where $C_R = 25$, $C_{t_{11}^1} = 10$, $C_{t_{21}^1} = 5$ and $C_{t_{31}^1} = 17$. $C_{t_{ij}^k}$ represents the cost to write the result of task t_{ij}^k (or to read the output file of task t_{ij}^k). C_R is the cost to read the input relations. Therefore, in the table above for the actual costs, the cost for a task t in the implied relationships is calculated as $C_R + C_t$; e.g., the cost of $t_{11}^1 = 25 + 10 = 35$, $t_{21}^1 = 25 + 5 = 30$ and $t_{31}^1 = 25 + 17 = 42$. Since there is no identical-task interaction, the estimated cost of each equivalence group for a task and estimated cost for the task is the same. For the tasks t_{11}^1, t_{21}^1 and t_{31}^1, we get

$$est\_cost(t_{11}^1) = 17 + 10 = 27 \quad est\_cost(t_{21}^1) = 10 + 5 = 15 \quad est\_cost(t_{31}^1) = 25 + 17 = 42.$$

Given the information above, the estimated costs (est_cost) for the tasks are:

Task	t_{11}^1	t_{11}^2	t_{11}^3	t_{12}^1	t_{12}^2	t_{21}^1	t_{21}^2	t_{22}^1	t_{22}^2	t_{31}^1	t_{31}^2	t_{32}^1	t_{32}^2	t_{32}^3
Estimated cost	27	10	10	25	20	15	20	10	40	42	10	10	30	10

and the estimated costs for the plans are:

Plan	P_{11}	P_{12}	P_{21}	P_{22}	P_{31}	P_{32}
Estimated cost	47	45	35	50	52	50

The search space generated by the heuristic algorithm for Example 2 is shown in Figure 6.
□

5.3 Extension to the New Heuristic Algorithm

In this section, we show how we can extend our new heuristic algorithm of Section 4 to accommodate implied relationships.

Let us assume that there is a mergeable-task precedence interaction $g_1 \rightarrow g_2 \rightarrow \ldots \rightarrow g_n$ and the tasks, which cannot be processed later because the query in which they exist has been already chosen but the plan in which they occur has not been chosen, are removed from this interaction list.

Let us also assume that the queries processed at state s were Q_i and Q_j, and that the task g_i in Q_i and g_j in Q_j were chosen in this state where $1 \leq i < j \leq n$. The cost of producing g_i is $C_R + C_{g_i}$ while the cost for producing g_j is $C_{g_i} + C_{g_j}$.

First, let us consider the tasks from g_1 to g_{i-1}. If any task g_k among these tasks is executed in the future, the cost added to the cost of $merge(s)$ is only $2C_{g_k}$ since we considered the cost of task g_i as $C_R + C_{g_i}$. So, we estimate the cost of each equivalence group for the tasks among g_1,...,g_{i-1} as follows

$$new\_est\_cost\_group(g_k) = 2C_{g_k} \tag{11}$$

where $k = 1, \ldots, i-1$. For the task g_i, the estimated cost becomes zero since it is already processed at the state s. For the tasks $g_{i+1}, \ldots, g_j$, the same idea for estimating the cost still holds. However, for the tasks $g_{j+1}, \ldots, g_n$, we have to calculate their estimated cost differently. Let $gcost(g_p)$ be $C_{g_{p-1}} + C_{g_p}$. Then, we define the estimated cost of each equivalence group for the tasks $g_{j+1}, \ldots, g_n$ as follows

$$new\_est\_cost\_group(g_k) = gcost(g_p) \tag{12}$$

where $k = j+1, \ldots, n$. Again, the estimated cost of the equivalence group for other tasks which do not occur in mergeable-task precedence interaction is the cost of the tasks themselves. Then the estimated cost of a task t is defined as

$$new\_est\_cost(t) = \frac{new\_est\_cost\_group(g_t)}{n_q} \tag{13}$$

where n_q is the number of queries the task t occurs in with the identical-task interaction.

The heuristic function for state s is the same as in Section 4, i.e.,

$$h_n(s) = \sum_{i=next(s)}^{n} \min_{j_i} [new\_est\_cost(P_{ij_i})]$$

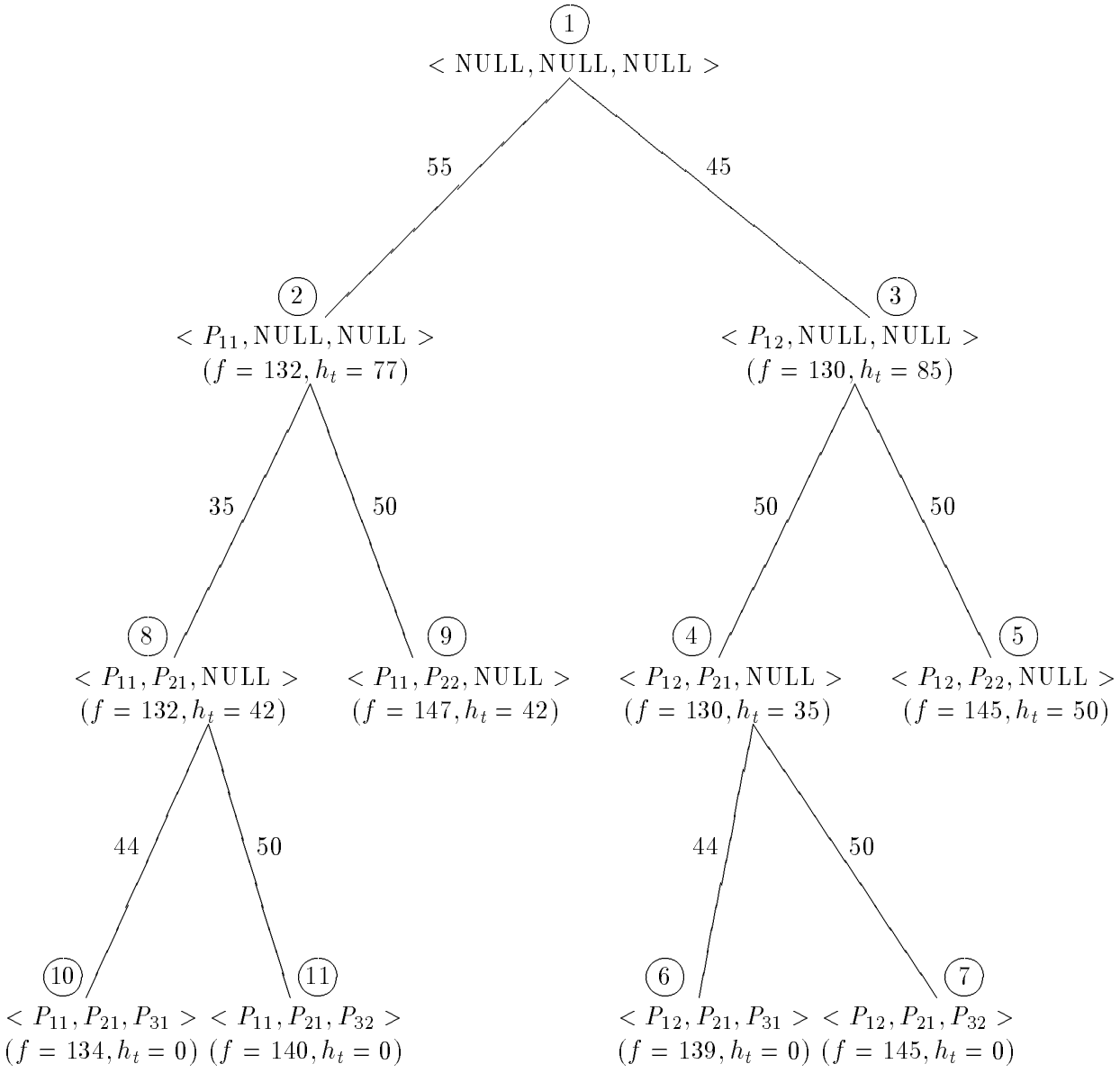

Figure 6: The portion of the search space explored by MQO-search using $h_t(s)$

where, for a plan P_{ij_i}, we define (analogously to Equation 3)

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

Again, since $new\_est\_cost(P_{ij_i}) \leq cost(P_{ij_i})$ for every plan, it is easy to see that h_n is admissible and MQO-search is guaranteed to find an optimal solution. In Section 6.2, we present in detail how the new heuristic function can be calculated efficiently.

To illustrate the effect of h_n, we will redo Example 2. Note that since there is no identical-task interaction, the estimated cost of each equivalence group for a task is the same as the estimated cost for the task. From the initial state, we consider two states, $< P_{11}, \text{NULL}, \text{NULL} >$ and $< P_{12}, \text{NULL}, \text{NULL} >$.

For the state $< P_{11}, \text{NULL}, \text{NULL} >$, the mergeable-task precedence interaction list is $t^1_{31} \rightarrow t^1_{11} \rightarrow t^1_{21}$; t^1_{11} is the task we already processed. Based on the formulas given above, the estimated cost of the task t^1_{31} becomes $2C_{t^1_{31}} = 34$ and that for the task t^1_{21} is $C_{t^1_{11}} + C_{t^1_{21}} = 15$. Using these estimated costs of tasks, the estimated cost of each plan will be

Plan	P_{21}	P_{22}	P_{31}	P_{32}
Estimated cost	35	50	44	50

Since $cost(merge(< P_{11}, \text{NULL}, \text{NULL} >)) = 55$, $f(< P_{11}, \text{NULL}, \text{NULL} >)$ becomes $55 + \min\{35, 50\} + \min\{44, 50\} = 134$.

We now turn to examine the state $< P_{12}, \text{NULL}, \text{NULL} >$. Since we have chosen P_{12}, task t^1_{11} cannot be processed in the subtree of this node. In this case the mergeable-task precedence interaction list becomes $t^1_{31} \rightarrow t^1_{21}$. Therefore the estimated costs of the task t^1_{31} and task $t^{t^1_{21}}$ becomes

$$new\_est\_cost(t^1_{21}) = 17 + 5 = 22 \quad new\_est\_cost(t^1_{31}) = 25 + 17 = 42.$$

Given those estimated costs for tasks, the estimated cost of each plan becomes as follows

Plan	P_{21}	P_{22}	P_{31}	P_{32}
Estimated cost	42	50	52	50

Since $cost(merge(< P_{12}, \text{NULL}, \text{NULL} >)) = 45$, $f(< P_{12}, \text{NULL}, \text{NULL} >)$ becomes $45 + \min\{42, 50\} + \min\{52, 50\} = 137$. Therefore we will expand the state $< P_{11}, \text{NULL}, \text{NULL} >$ at this point and we get two states, $< P_{11}, P_{21}, \text{NULL} >$ and $< P_{11}, P_{22}, \text{NULL} >$.

At the state $< P_{11}, P_{21}, \text{NULL} >$, the mergeable-task precedence interaction list is $t^1_{31} \rightarrow t^1_{11} \rightarrow t^1_{21}$, with t^1_{11} and t^1_{21} being the tasks we have already processed. The estimated cost of task t^1_{31} becomes $2C_{t^1_{31}} = 30$, and the estimated costs of plans to be processed will be $P_{31} = 44$ and $P_{32} = 50$. Since $cost(merge(< P_{11}, P_{21}, \text{NULL} >)) = 90$, $f(< P_{11}, P_{21}, \text{NULL} >)$ becomes $90 + \min\{44, 50\} = 134$.

Finally, at the state $< P_{11}, P_{22}, \text{NULL} >$, the interaction list is $t^1_{31} \rightarrow t^1_{11}$ (since t^1_{21} cannot be considered). The estimated cost of task t^1_{31} becomes $2C_{t^1_{31}} = 30$ and the estimated costs of the plans to be processed will be $P_{31} = 44$ and $P_{32} = 50$. Since $cost(merge(< P_{11}, P_{22}, \text{NULL} >)) = 105$, $f(< P_{11}, P_{22}, \text{NULL} >)$ becomes $105 + \min\{44, 50\} = 149$. Therefore we will expand the state $< P_{11}, P_{21}, \text{NULL} >$. The search space of this new heuristic algorithm for Example 2 is shown in Figure 7. Note that 11 states are generated with the heuristic h_t while only 7 states are generated with the new heuristic h_n.

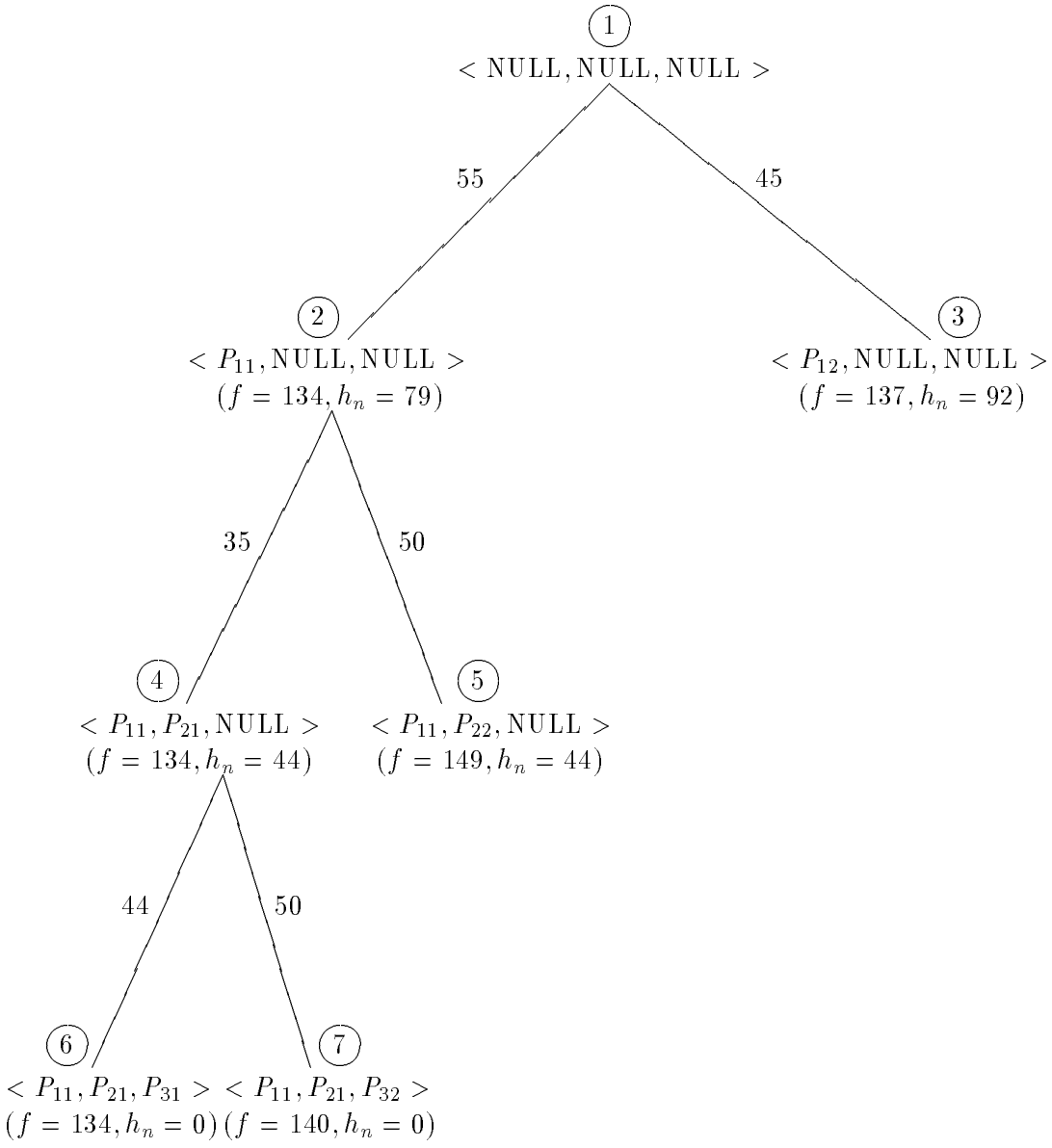

Figure 7: The portion of the search space explored by using MQO-search using $h_n(s)$

Query Set	No of Queries	No of Plans	Range of No of Tasks	Range of Task cost	Sharability
$QSET1$	10	[3-6]	[3-6]	[5-100]	10%
$QSET2$	10	[3-6]	[3-6]	[5-100]	20%
$QSET3$	10	[3-6]	[3-6]	[5-100]	30%
$QSET4$	10	[3-6]	[3-6]	[5-100]	40%
$QSET5$	10	[3-6]	[3-6]	[5-100]	50%
$QSET6$	10	[3-6]	[3-6]	[5-100]	60%
$QSET7$	10	[3-6]	[3-6]	[5-100]	70%
$QSET8$	10	[3-6]	[3-6]	[5-100]	80%
$QSET9$	10	[3-6]	[3-6]	[5-100]	90%
$QSET10$	10	[3-6]	[3-6]	[5-100]	100%
$QSET11$	10	2	[4-6]	[5-100]	30%
$QSET12$	15	2	[4-6]	[5-100]	30%
$QSET13$	10	4	[4-6]	[5-100]	30%
$QSET14$	15	4	[4-6]	[5-100]	30%
$QSET15$	20	4	[4-6]	[5-100]	30%
$QSET16$	10	2	[4-6]	[5-100]	30%

Table 1: Query sets used in the Experiment

6 Experimental Results and Comparisons

In this section, we present simulation results for the improved heuristic algorithm and compare them with Sellis' previous heuristic algorithm in [Sel88, SG90].

6.1 Experimental Testbed

Our experiments were performed on a DECStation 5000/200 and the simulation software was written in C. The experiments were run over randomly generated query sets. The queries are randomly generated as a set of plans, which are sets of tasks. The costs of individual tasks are also randomly generated. For experiments involving implied relationships, the costs of deriving the implications among tasks are included in the cost model. 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 tasks in the interaction list among all tasks of the queries.

To study the performance of the two heuristic functions, we generated 16 query data sets QSET1-QSET16. Each set consists of 10 queries with a fixed sharability varying from 10% to 100%. Except query set QSET16, all query sets have identical tasks only. Table 1 shows the parameters used to generate query data sets QSET1-QSET16.

6.2 Implementation Details for Computing the New Heuristic Function

To calculate the heuristic function in each step of the MQO search algorithm, we can compute repeatedly the estimated cost of tasks and plans in the remaining queries. However, this approach blindly computes the estimated costs of tasks and plans, even if the estimated cost of the tasks having no commonalities does not change. To speed up the computation overhead of the heuristic function, we use the following method in our implementation.

Before we start the execution of the algorithm, we first calculate the cost of all plans using the actual costs of the tasks ignoring any possible commonalities. Let us call the cost computed in this manner, the *individual cost* of the plan. Then, whenever we calculate the estimated cost of the plan, we compute it from the *individual cost* of the plan taking into account only the tasks which have common tasks. Then, for a state $s =< P_{1k_1}, P_{2k_2}, ..., P_{gk_g}, NULL, NULL >$, the heuristic function is computed in the following way.

1. Copy the *individual costs* of the plans that appear in the remaining queries $Q_{g+1}, Q_{g+2}, ..., Q_n$ into the array that keeps the estimated costs of the plans. (Note that the *individual costs* were calculated at the initial step)

2. Scan each identical task interaction list to calculate the estimated cost of the task in each list. We first check whether there is any task from the interaction list that appears in $P_{1k_1}, P_{2k_2}, ..., P_{gk_g}$. If such a task does exist, we exit immediately and set the estimated costs of the tasks of the list to zero; this is natural as we can ignore (cost-wise) those tasks that appear on the path from the start node to the current state. However, if none of the tasks in the interaction list has been encountered in the plans that have been processed so far, we count the number of queries among $Q_{g+1}, Q_{g+2}, ..., Q_n$ in which the identical tasks of the list appear. In terms of implementation, we organize the lists of tasks based on the queries containing the tasks in an identical task interaction list, so that we can compute the number of queries, in which a task occurs, by just counting the number of head nodes (of a list) whose content is non-null. We next compute the estimated cost of the tasks of that list by dividing the actual cost of any task in the list by the number of queries just derived. Then we update the estimated costs of plans in which tasks of the current list appear by scanning the list once again. For each task t_{ij}^k $(i = g+1, ..., n)$ in the list, we update the $est\_cost(P_{ij})$ to be $est\_cost(P_{ij}) - cost(t_{ij}^k) + estcost(t_{ij}^k)$.

3. Find the minimal estimated cost of the plan for each query among $Q_{g+1}, Q_{g+2}, ..., Q_n$ and add them all.

When we have a mergeable task precedence interaction list, we have to modify step 2 above. We first process the mergeable task precedence interaction list to compute the estimated cost of the task in each group and traverse the identical task interaction list of each group to change the estimated cost of the task of the group. We then process in the manner described above the identical task interaction lists that do not appear in mergeable task precedence interaction lists.

6.3 Experiment 1: Identical Relationships

To study the performance of the previous and new heuristic functions, we first tested with 10 query data sets QSET1-QSET10 which have identical relationships only, each set consisting of 10 queries and with sharability varying from 10% to 100%. Table 1 shows the parameters used in QSET1-QSET10. For each query set, the heuristic search algorithm was run using the following six query orders for filling in the state vectors (i.e. the order in which queries are examined by the algorithm) as they were suggested in [SG90]:

Order 1 : the original order, i.e., increasing query index.

Order 2 : increasing number of plans.
This order of queries maximizes the size of the search space below a state at level i, 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.

Order 3 : decreasing average query cost.
Since many times, 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.

Order 4 : decreasing average estimated query cost.
By assuming that the error in the *lower bound* function is proportional to its estimated value, this tries to minimize the error in the cost of any path.

Order 5 : decreasing average query cost per number of plans.
This heuristic is a combination of Order 2 and Order 3.

Order 6 : decreasing average estimated query cost per number of plans.
Similarly, this heuristic is a combination of Order 2 and Order 4.

For each query ordering, we recorded the number of states generated, the total number of states in the search space, the percentage of the total number of states that were generated by the heuristic algorithm, and the CPU time. Table 2 summarizes these figures; the numbers shown are averages of 10 runs. MQO1 is the heuristic algorithm with our new heuristic function h_n and MQO2 is the one using the previous heuristic function h_t. Figure 8 illustrates the effect that the sharability has on the ratio of states generated versus the total number of states, for both MQO1 and MQO2, and in particular when ordering method 1 (i.e. original ordering) is used. Because Order 3 is the best for both MQO1 and MQO2, we also show the result of using this ordering method as well. As it can be seen from this figure, the ratio of states generated versus the total number of states by MQO1 increases slightly as the sharability becomes large. However, the same ratio for MQO2 reaches a maximum when sharability becomes 40% with Order 1 and it decreases as the sharability becomes large. The worst value for the ratio of states generated versus the total number of states in MQO1 was 0.02436% while it was 0.69561% for MQO2. Note that the ratio with MQO1 is about 3.5% of the ratio with MQO2. From the experimental results above, it is evident that the new heuristic algorithm generates only a small percentage of the total number of states in the search space for various sharability figures. On the other hand, the more expensive computation required for h_n compared to h_t, was well offset by the gains in limiting the number of states generated. Most of the times, the CPU cost for MQO1 was much less than 1 second, while the CPU time for MQO2 was as big as 1041 seconds. This is due to exploring more search space, and therefore manipulating a larger number of states.

MQO Algorithm		MQO1			MQO2		
Query Set	Query Order	States Generated	% Generated	CPU Time(sec)	States Generated	% Generated	CPU Time(sec)
	1	84	0.00251	0.03000	2950	0.10215	2.89667
	2	140	0.00275	0.06000	2958	0.15220	3.83000
$QSET1$	3	127	0.00453	0.05000	2628	0.12713	2.88833
	4	160	0.00528	0.06500	2716	0.12692	2.90833
	5	101	0.00328	0.04000	2539	0.14981	3.43000
	6	153	0.00602	0.06000	2066	0.08875	1.52333
	1	215	0.00569	0.11333	23633	0.58595	296.538
	2	228	0.00607	0.11833	16257	0.52748	125.342
$QSET2$	3	250	0.01088	0.13333	11275	0.28472	51.6933
	4	423	0.01404	0.23833	12716	0.30599	71.7567
	5	175	0.00645	0.09000	13032	0.44198	84.0383
	6	338	0.00995	0.19167	10843	0.26243	56.3117
	1	398	0.00808	0.27000	35119	0.63433	1041.62
	2	400	0.00832	0.26500	21999	0.45096	362.443
$QSET3$	3	290	0.00731	0.16500	13755	0.24694	120.118
	4	583	0.01233	0.40000	15792	0.28487	162.690
	5	274	0.00790	0.15833	14868	0.33141	134.153
	6	454	0.01018	0.29000	14223	0.24953	137.887
	1	384	0.00802	0.25167	30689	0.69561	544.92007
	2	470	0.00981	0.35833	15998	0.35715	160.42667
$QSET4$	3	348	0.00734	0.23167	9973	0.18990	46.36333
	4	592	0.01276	0.41833	10432	0.22320	52.15333
	5	387	0.00994	0.24833	10778	0.25330	56.05166
	6	544	0.01174	0.39000	9516	0.19542	43.14833
	1	452	0.00872	0.32333	28167	0.58325	490.005
	2	443	0.00914	0.31333	12701	0.27885	98.8350
$QSET5$	3	339	0.00615	0.23833	7812	0.15402	27.4633
	4	553	0.01144	0.40667	8073	0.19048	37.9617
	5	381	0.00735	0.26500	7813	0.18170	30.5300
	6	487	0.00934	0.35000	7273	0.14820	30.5217
	1	960	0.01683	0.83333	20188	0.36907	246.0717
	2	681	0.01351	0.53333	14221	0.24834	251.7967
$QSET6$	3	420	0.00809	0.29000	5806	0.10578	17.9517
	4	738	0.01254	0.61500	5766	0.10053	16.5800
	5	445	0.00911	0.30833	7255	0.14421	38.3933
	6	686	0.01227	0.53333	6243	0.10485	22.2033
	1	824	0.02155	0.70000	9018	0.21862	32.2050
	2	488	0.01378	0.36667	7120	0.16605	31.0600
$QSET7$	3	389	0.00892	0.29000	2956	0.06719	3.14000
	4	582	0.01203	0.48333	3775	0.08156	5.86167
	5	368	0.01006	0.26667	3590	0.09781	5.42167
	6	466	0.01027	0.35167	3531	0.07668	5.14500
	1	858	0.02208	0.69667	4943	0.12477	9.66667
	2	593	0.01787	0.46500	3695	0.10142	6.30167
$QSET8$	3	444	0.01019	0.32833	2108	0.04905	1.69000
	4	559	0.01343	0.44833	2544	0.06255	3.73333
	5	480	0.01353	0.36000	2550	0.07405	3.24000
	6	467	0.01264	0.34833	2151	0.05754	2.02667
	1	663	0.01648	0.54667	3155	0.07952	3.67500
	2	496	0.01263	0.39500	1912	0.05279	1.32500
$QSET9$	3	362	0.00829	0.27333	1060	0.02550	0.50500
	4	416	0.00916	0.32667	1427	0.03345	0.85333
	5	379	0.00933	0.29167	1355	0.03824	0.79667
	6	380	0.00932	0.28833	1281	0.03473	0.67333
	1	863	0.02436	0.75833	3065	0.08209	3.86667
	2	507	0.01577	0.40667	1412	0.04560	0.78000
$QSET10$	3	353	0.00881	0.27667	885	0.02320	0.39333
	4	391	0.00921	0.31833	1122	0.02753	0.63667
	5	356	0.01136	0.27500	961	0.03342	0.43667
	6	370	0.01070	0.28500	1092	0.03771	0.51000

Table 2: Results of various query ordering strategies with heuristics.

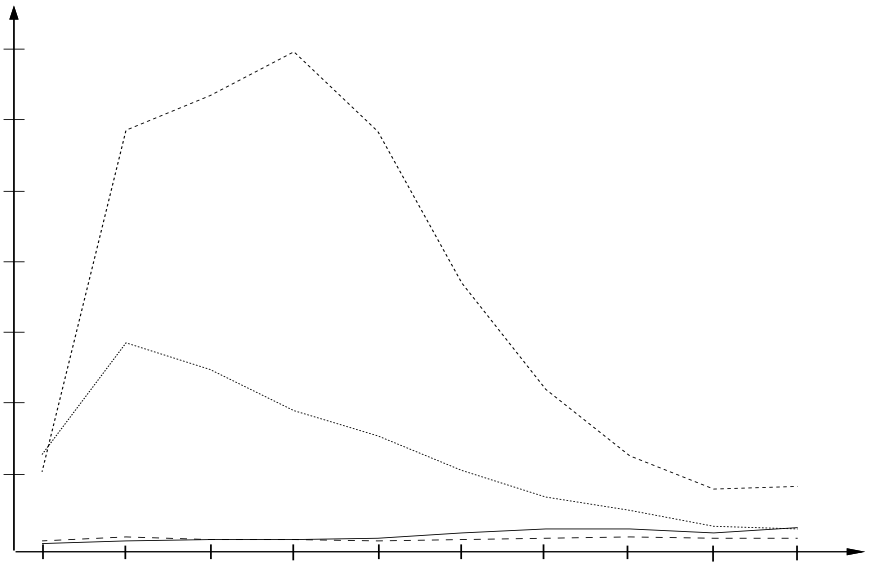

Figure 8: Percentage of states generated versus total number of states

In order to see how the percentage of states generated changes as the search space increases we tested with five new query sets QSET11-QSET15 (also shown in Table 1). The results of running these query sets are summarized in Table 3. It is clear that as the number of states in the search space increases, the percentage of states generated becomes smaller and smaller. This is an important characteristic of our heuristic algorithm as it also shows that the number of states generated is consistently kept to a low number.

6.4 Experiment 2: Identical and Implied Relationships

Finally, a last set, QSET16, was tested. This query set is used to illustrate the performance of algorithm in the presence of implied relationships. The detailed parameters for QSET16 are in Table 1. Note that the parameters in QSET11 and QSET16 are the same except that in QSET16 there are tasks with implied relationships. The result of our experiments are summarized in Table 4. In this case, MQO1 and MQO2 are both the algorithm proposed in Section 5, with the heuristic functions of sections 5.3 and 5.2 respectively. As it can be seen, the percentage of states generated in QSET16 with MQO1 is even smaller than that of QSET11 with MQO1, which means that the heuristic function also performs well with implied relationships.

In summary, our experimental results show that the new heuristic algorithm significantly improves the search space and time complexity compared to the heuristic algorithm of [Sel88]. Furthermore, it is very efficient for both identical and implied relationships.

7 Summary

In this paper, we first studied the performance of existing heuristic algorithms for the multiple-query optimization problem [Sel88, SG90] and pointed out some deficiencies. A new heuristic function was proposed to offset these deficiencies. We proved that the new

Query Set	Total States	States Generated	% Generated	CPU Time(sec)
$QSET11$	2047	33	1.62188	0.01833
$QSET12$	65535	70	0.10682	0.03667
$QSET13$	1398101	125	0.00894	0.06833
$QSET14$	1431655680	1715	0.00012	2.06667
$QSET15$	1466015416320	2599	0.0000001772	4.64000

Table 3: Results of MQO1 increasing search space.

MQO Algorithm	Total States	States Generated	% Generated	CPU Time(sec)
$MQO1$	2047	26	1.26038	0.05000
$MQO2$	2047	124	6.07719	0.49667

Table 4: Results with QSET16 (implied relationships).

heuristic function is more informed than the one proposed by Sellis, and studied the search space and time complexity of MQO-search.

We then extended the new improved heuristic algorithm to handle implied relationships, something that the algorithm of [Sel88] could not handle efficiently. Again, we gave formulas for the worst case complexity of the extended heuristic algorithm. Finally, we simulated the new and old algorithms for various query sets, with identical and implied relationships among queries. The results obtained strengthened our belief that the performance of the new improved heuristic algorithm is superior to the previous one and the complexity is much less than the worst case one in most situations.

As interesting issues of future research, we view the following: First, we are trying to find other strategies of query ordering so that the algorithm can perform even better with implied relationships. For example, ordering the queries according to the *mergeable-task precedence* can be a good strategy because the error in estimating the real costs can be minimized. The second interesting issue is comparing the the dynamic programming approach of Park and Segev [PS88] to our heuristic algorithm, especially with respect to search space and time complexity. A final issue of interest is the average case performance analysis of our heuristic algorithms. In this paper only best-case and worst-case performance has been studied. We plan to use the ideas in [Pea84] to study the average performance of the algorithms assuming a very simple model.

References

[GM78] H. Gallaire and J. Minker. *Logic and Databases.* Plenum Press, New York, 1978.

[GM82] J. Grant and J. Minker. On optimizing the evaluation of a set of expressions. *Intern. J. Comput. Inform. Science*, March 1982.

[KNY92] R. Karinthi, D. Nau, and Q. Yang. Handling feature interactions in process planning. *Applied Artificial Intelligence*, 6(4):389–415, October-December 1992. Special issue on AI for manufacturing.

[MKGP90] A. Mahanti, R. Karinthi, S. Ghosh, and A. Pal. AI search for minimum-cost set cover and multiple-goal plan optimization problems: Applications to process planning. Technical Report CS-TR-2540, Dept. of Computer Science, Univ. of Maryland, College Park, 1990.

[Nil80] N. J. Nilsson. *Principles of Artificial Intelligence*. Springer-Verlag, New York, 1980.

[NKK84] D. Nau, V. Kumar, and L. N. Kanal. General branch and bound, and its relation to A* and AO*. *Artificial Intelligence*, 23:29–58, 1984. (Also available as Tech. Report TR-1170, Computer Science Dept., Univ. of Maryland, 1982.).

[Pea84] J. Pearl. *Heuristics*. Addision-Wesley, Reading, MA, 1984.

[PS88] J. Park and A. Segev. Using common subexpressions to optimize multiple queries. *in Proc. 4th Intern. Conf. on Data Engineering*, pages 311–319, February 1988.

[Sel88] T. Sellis. Multiple-query optimization. *ACM Trans. on Database Systems*, pages 23–53, March 1988.

[SG90] T. Sellis and S. Ghosh. On the multiple-query optimization problem. *IEEE Trans. on Knowledge and Data Engineering*, pages 262–266, June 1990.

[SP89] A. Segev and J. Park. Identifying common tasks in multiple access plans. Technical Report LBL-27877, School of Business Administration and Lawrence Berkeley Lab's Computer Science Research Department, University of California, Berkeley, December 1989.

[SW88] T. Sellis and Y. C. Wong. The implementation of a heuristic algorithm for the multiple-query optimization problem. Unpublished manuscript, March 1988.

[YNH89] Q. Yang, D. Nau, and J. Hendler. Planning for multiple goals with limited interactions. *In Proc. 5th IEEE Conf. on AI Applications*, 1989.

[YNH90] Q. Yang, D. Nau, and J. Hendler. Optimization of multiple-goal plans with limited interaction. Technical Report CS-TR-2411, Dept. of Computer Science, Univ. of Maryland, College Park, February1990.

[YNH92] Q. Yang, D. Nau, and J. Hendler. Merging separately generated plans with restricted interactions. *Computational Intelligence*, 8(2):648–676, February 1992.