Title
USING COMMON SUBEXPRESSIONS TO OPTIMIZE MULTIPLE QUERIES

Authors
Park, J.
Segev, A.

Publication Date
1988-02-01

LBL-23597 c.2

Lawrence Berkeley Laboratory

UNIVERSITY OF CALIFORNIA, BERKELEY

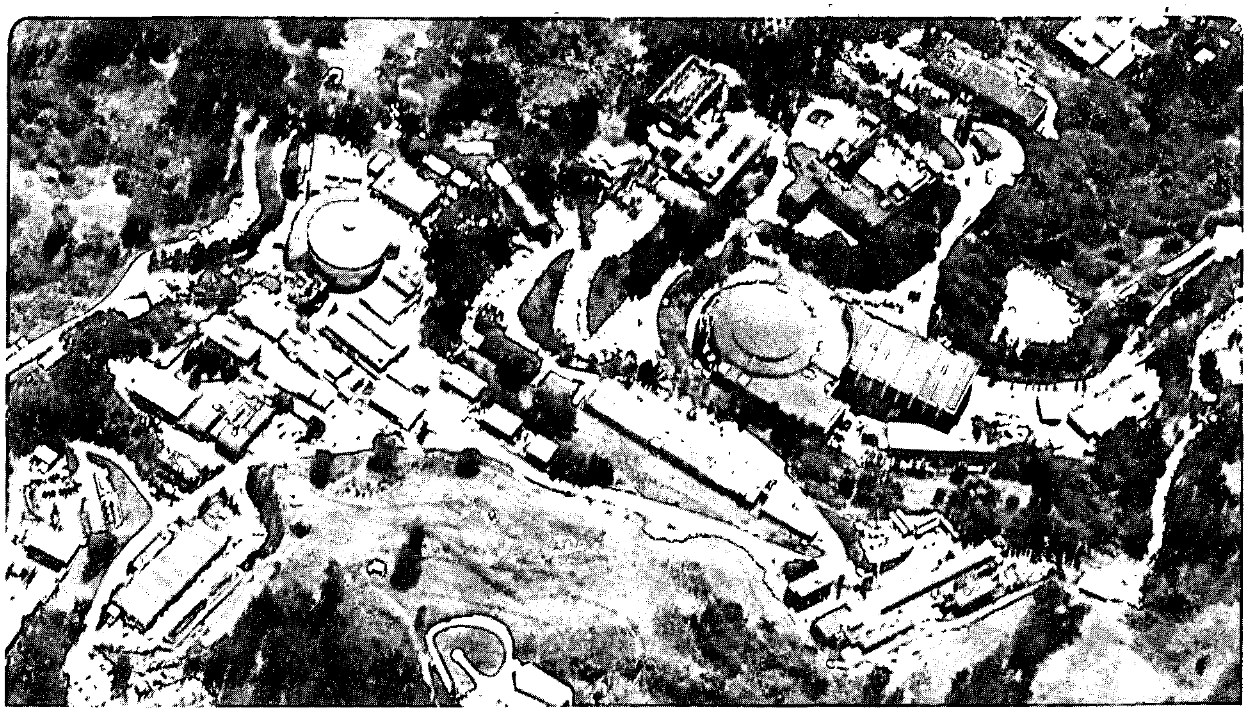
Information and Computing Sciences Division

Presented at the 4th International Conference on Data Engineering, Los Angeles, CA, February 2–5, 1988

Using Common Subexpressions to Optimize Multiple Queries

J. Park and A. Segev

February 1988



Prepared for the U.S. Department of Energy under Contract DE-AC03-76SF00098

DISCLAIMER

USING COMMON SUBEXPRESSIONS TO OPTIMIZE MULTIPLE QUERIES

Jooseok Park and Arie Segev

School of Business Administration and
Lawrence Berkeley Lab's Computer Science Research Department
The University of California
Berkeley, California 94720

Abstract

This paper deals with the problem of identifying common subexpressions and using them in the simultaneous optimization of multiple queries. In particular, we emphasize the strategy of selecting access plans for the single queries and their integration into a global access plan that takes advantage of common tasks. We present a dynamic programming algorithm for the selection of individual access plans such that the resulting global access plan is of minimum processing cost. The computational complexity of this algorithm represents a significant improvement over existing algorithms.

1. INTRODUCTION

The relational model [CODD70] allows for nonprocedural queries where the user expresses the result rather than how to get it. Consequently, an important component of a relational database management system (DBMS) is the query optimizer which transforms the user's query into a procedural access plan. These query optimizers employ algorithms such as [WONG76] and [SELL79]; see [JARK84a] for a survey of query optimization in a centralized DBMS. Query optimizers in current relational database systems minimize the cost of processing one query at a time. There are situations, however, where global optimization of multiple queries can provide substantial savings over the current single-query approach by sharing common resources in processing them.

In traditional applications, the multiple-query optimization approach is attractive when a set of queries is embedded in application programs or submitted for batch processing [KIM84]. Global optimization can reduce the processing cost significantly in on-line environments, if queries enter the system at a steady rate and can be grouped within a tolerable time-interval (e.g., a few seconds) [CHAK82, JARK84b, CHAK86]. There should be a trade-off between the reduction of processing cost and the delay in response, however [CHAK82].

In more recent applications, the multiple-query optimization approach is useful in the cases of deductive query processing [CHAK86] and integrity constraints checks [KIM84]. In relational DBMSs that are extended to provide deductive capabilities, a single user query may be translated into a set of database queries. Quite frequently this translation results in a disjunction of non-recursive queries that have to be optimized jointly. In the case of integrity checks, there is a need to simultaneously optimize a set of queries which are automatically triggered to check for possible violation of integrity rules when the user issues a data manipulation statement [KIM84]. If the integrity check consists of a conjunction of queries, they can be integrated into one query by a general integrity modification procedure [STON75], and thus be optimized by a current query optimizer. However, if the integrity constraints are represented by a disjunction of queries, this resulting processing should be optimized by a multiple-query optimization algorithm. In the above applications, queries are issued simultaneously for a single answer, so they can be grouped naturally for global optimization without a degradation response time. In fact, both response time and processing cost can be reduced significantly because these queries have a tendency to access the same data frequently.

Multiple-query optimization algorithms consist of two conceptual parts - identifying common subexpressions, and constructing a global access plan. Some studies have focused more on how to identify common subexpressions among queries, and to check for possible benefits of their sharing [FINK82, JARK84b, CHAK86], while other studies emphasized the global access plan and taking advantage of current query optimizers [GRAN80, KIM84, SELL86]. It should be noted that the problem of identifying common subexpressions is a "hard" problem in terms of complexity theory [ROSE80, JARK84b], and that sharing of common subexpressions during execution is not always better than independent execution [GRAN80]. Therefore, the use of common subexpressions should be determined based on a cost-benefit analysis.

In this paper, we analyze the case of constructing a global access plan using candidate plans generated by a traditional optimizer, and present a dynamic programming algorithm for doing it. This algorithm has a significantly lower computational complexity than existing algorithms. In Section 2, we analyze the approach of using access plans and their tasks as the building blocks for a global access plan construction. The dynamic programming algorithm for access plan selection in the case of identical tasks is presented in Section 3. and the case of implied tasks is discussed in in Section 4. Section 5 concludes the paper with a summary and directions for future research.

2. INTEGRATION OF ACCESS PLANS

There are several approaches to identifying and using common subexpressions. A bottom-up heuristic method of using *algebraic operator trees* (expression trees) was developed to detect common subexpressions in a query [HALL74, HALL76]. The *query graph* (object graph) approach takes advantage of common intermediate results among queries, by comparing query graphs [FINK82, CHAK82, JARK84b, LARS85, CHAK86]. Unlike these approaches, the methods discussed next are based on identifying common tasks among access plans and constructing a global access plan. We will use the following definitions presented in [SELL86].

Definition 1. A task T_i *implies* task T_j ($T_i => T_j$) iff T_i is a conjunction of selection predicates on attributes $A_1, A_2, \cdots, A_k$ of some relation R, T_j is a conjunction of selection predicates on attributes $A_1, A_2, \cdots, A_l$ of the same relation with $l < k$, and the result of evaluating T_i is a subset of the result of evaluating T_j.

Definition 2. A task T_i is *identical* to task T_j ($T_i = T_j$) iff a) Selections : $T_i => T_j$ and $T_j => T_i$, or b) Joins : T_i is a conjunction of join predicates $E_1.A_1 = E_2.B_1$, $E_1.A_2 = E_2.B_2$, $\cdots$, $E_1.A_k = E_2.B_k$ and T_j is a conjunction of join predicates $E_1'.A_1 = E_2'.B_1$, $E_1'.A_2 = E_2'.B_2$, $\cdots$, $E_1'.A_k = E_2'.B_k$ where each of E_1, E_2, E_1' and E_2' is a conjunction of selections on a single relation and $E_1 = E_1'$ and $E_2 = E_2'$.

These definitions are similar to those in [FINK82, JARK84b]. However, the main difference is that the relationships are between tasks in access plans, not between nodes in query graphs.

In general, the problem to be addressed is the following. Let Q_i denote query i and let $S_i = \{P_{i1}, P_{i2}, \cdots, P_{ik}\}$ be a set of alternative access plans for Q_i. Each access plan P_{ij} consists of a set of tasks $\{T_{ij}^1, T_{ij}^2, \cdots, T_{ij}^t\}$. Then, given a set of queries $Q_1, \cdots, Q_n$ and the associated access plans and relationships among tasks, a minimum-cost global access plan has to be constructed from $\{P_{ik^*}\}$, $i = 1, \cdots, n$, where k^* is the selected access plan for query Q_i.

For this problem, a branch and bound algorithm with a depth-first-search method is presented in [GRAN80], which is limited to the case of identical relationships. This algorithm is modified in [SELL86] by using a new lower bound function and a breadth-first-search method. [SELL86] also extended his algorithm to the case of implied relationships. [SELL86] reduced the search space in a *stochastic* sense as compared to [GRAN80], but the worst-case complexity is an exhaustive search of the solution space. In the next section, we present an *access plan selection* algorithm which reduces the state-space search compared with [GRAN80, SELL86].

3. PLAN SELECTION ALGORITHM

In this section we develop an efficient dynamic programming (DP) algorithm [HORO78] to select the set $\{S_{ik^*}\}$ for a global access plan. The logic of the algorithm for the case of identical relationships among tasks will be illustrated using several examples. In section 4, we will outline the procedure for the case of implied relationships, and discuss the computational complexity of the algorithm. Due to space limitations, a complete description of the mathematical details was not feasible: we refer to the reader to [PARK88] for a complete description. We first consider an example from [GRAN80].

Example 1. Three queries Q1, Q2, and Q3 are considered. The alternative access plans are: Q1: P11,P12; Q2: P21,P22,P23; Q3: P31,P32.

Each access plan consists of a set of tasks. Some tasks are common among access plans. This example can be represented by an undirected graph G(V,E) with edge-weights S $\le$ 0 and node-weights C $>$ 0 as illustrated in Figure 1. In this graph, each node represents an access plan, and the squares in each node represent the tasks in the corresponding access plan. We will refer to the set of nodes associated with a single query as a *column* or *stage* interchangeably. An edge

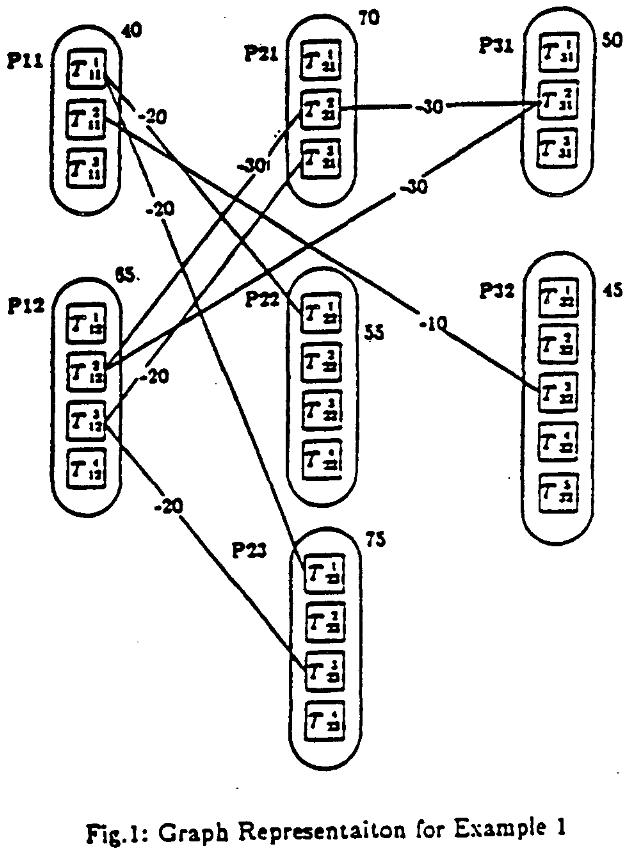


Fig.1: Graph Representaiton for Example 1

between two squares, say s and t, means that tasks s and t are identical: we will refer to such an task-edge as TE(s,t). A node-weight represents the estimated cost of the corresponding access plan, whereas the absolute value of each edge-weight represents the saving from sharing the connected common tasks. The access plan selection problem can be stated as the following graph problem: Find a set of nodes such that one and only one node is chosen from each column of the graph to minimize the sum of weights associated with the chosen nodes and the task-edges connecting squares in these nodes.

To develop an algorithm for this graph problem, the following definitions and notations are used.

Definition 3. An edge, E(X,Y), is defined between two nodes X and Y if there exists at least one task-edge connecting a task in plan X and a task in plan Y. The weight of E(X,Y), EW(X,Y), is the sum of weights of all task-edges between nodes X and Y.

Definition 4. In the graph G(V,E), the *distance* of an edge is defined as the difference between the stages of the two nodes connected by that edge. For instance, the distance of E(P11,P32) is 2.

Definition 5. In the graph G(V,E), an edge is defined as *regular* when its distance is 1. For example, E(P11,P22) is a regular edge.

Definition 6. In the graph G(V,E), an edge is defined as *distant* when its distance is greater than 1. For instance, E(P11,P32) is a distant edge. Consequently, the edges in the graph are divided into two types: regular edges and distant edges, referred to as RE(X,Y) and DE(X,Y) respectively.

Definition 7. When nodes X and Y, where the stage of X is lower than the stage of Y, are connected by an edge, we say that E(X,Y) is *incident to* node Y or is *incident from* node X.

Definition 8. A node X is *distantly-adjacent to* a node Y or X is *distantly-adjacent from* Y, if X and Y are connected by DE(X,Y) and X has a lower stage number than Y.

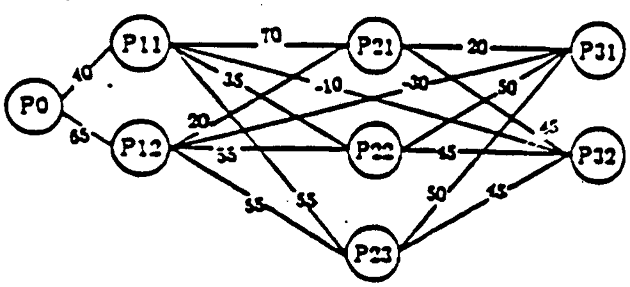


Fig. 2: A Simplified Graph for Example 1

Let us consider Example 1 again. The graph G(V,E) can be simplified into a graph G'(V,E') as shown in Figure 2. In this figure, we deleted the task identifiers and added the origin node P0 with zero weight, and connected each node with all nodes in the next stage. For computational convenience, we adjusted the weights of edges and nodes in G'(V,E') as follows. For each regular edge RE(X,Y), its adjusted weight is EW(X,Y) plus the weight of node Y in G(V,E). For example, the adjusted weight of E(P11,P21) = (-30) + 70. The weights of all distant edges remain the same as in G(V,E). All node-weights were set to zero. In addition to G'(V,E'), we have to retain the information about plans with identical tasks, whenever the number of such related plans is greater than two. This information is represented by identical list. For example, in Figure 1, plans P12, P21, and P31 have a common task: $T_{12}^2 = T_{21}^2 = T_{31}^2$. The identical-task list is {P12, P21, P31}. Finally, the access plan selection problem can be stated as the following graph problem over G'(V,E'): choose one node from each column to minimize the sum of edge-weights associated with the chosen nodes.

At a first glance it seems that this problem can be solved by a simple DP algorithm: one node is chosen for each stage (column), and there is no edge between nodes in the same stage. However, this is not so because of the existence of distant edges. The existence of such edges results in two cases; in one case the algorithm should choose a single minimum path, and in the other case it should merge two paths into one. Therefore, a straightforward application of a DP algorithm requires all past information in order to choose the next node, and thus the search space increases in terms of multiplicative complexity, rather than additive complexity.

In this paper, we devise a DP algorithm with a reduced computational complexity (as discussed in the next section). We will present several strategies to derive the logic of the algorithm. The first strategy is to modify the problem structure in order to apply a simple DP algorithm. The graph G'(V,E') can be transformed into a graph with only regular edges according to the following strategy.

Strategy 1. Each distant edge DE(X,Y) is replaced by a path between X and Y (referred to as an *artificial path*) that represents the optimal path from X to Y. We know that DE(X,Y) is a part of the optimal path between X and Y because any path between X and Y can be reduced by the weight of DE(X,Y).

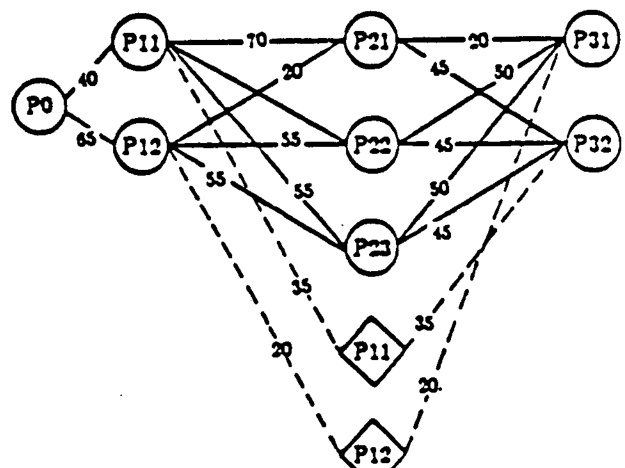


Fig. 3: A Modified Problem Structure for DP

Applying strategy 1 to Example 1 results in the modified structure shown in Figure 3. An artificial path consists of *artificial nodes* and *artificial edges*. In Figure 3, the broken lines represent artificial edges and the diamonds represents artificial nodes. We will denote an artificial edge between X and Y by AE(X,Y) and its weight by AW(X,Y). For computational convenience, the weights of all regular edges incident to Y were reduced by the weight of DE(X,Y) in the modification procedure. Then, when constructing an artificial path between X and Y, the weights of the regular edges in the optimal path were assigned to the weights of the corresponding edges in the artificial path. The detailed modification procedure for Example 1 follows. Construction of the artificial path between P11 and P32 for DE(P11,P32): i) Delete DE(P11,P32) from the graph. ii) Adjust the weights of the edges incident to P32 by the weight of DE(P11,P32): EW(P21,P32) ← 45 - 10, EW(P22,P32) ← 45 - 10, and EW(P23,P32) ← 45 - 10. iii) Find the optimal path from P11 to P32 by a simple DP algorithm: min [70+35,35+35,55+35] = 70. iv) Connect P11 and P32 using a chain of artificial edges.
v) Assign to the artificial edges the following weights: AW(P11,P32) ← EW(P22,P32) and AW(P11,P11) ← EW(P11,P22).

Construction of the artificial path between P21 and P31 for DE(P12,P31): i) Delete DE(P12,P31). ii) EW(P21,P31) ← 20 - 0†, EW(P22,P31) ← 50 - 30, and EW(P23,P31) ← 50 - 30. iii) Optimal path from P12 to P31 is min {20+20,55+20,55+20} = 40. iv) Connect P12 and P31 using a chain of artificial edges. v) AW(P12,P31) ← EW(P21,P31) and AW(P12,P12) ← EW(P12,P21).

The resulting modified problem can be solved by a simple DP algorithm. The following strategy makes the problem's modification more efficient.

Strategy 2. For each distant edge incident from node X, construct the artificial path starting from X, Path(X), in a single scan. Path(X) connects X with all nodes which were distantly-adjacent from node X in the original graph. At each stage to be scanned, we keep the the values of the optimal paths from node X to nodes in this stage.

The following example illustrates strategy 2. Consider the graph of Figure 4‡. This graph has four distant edges incident from P12: DE(P12,P41), DE(P12,P51), DE(P12,P52), and DE(P12,P71). This graph representation is modified as shown in Figure 5 using the following notations. Let From(P_{ij},k), $i < k$, be the set of the values of the optimal paths from P_{ij} to all nodes in stage k. Let Last(X,Y) be the set of possible values for the weight of the last artificial edge AE(X,Y), which are the weights of all regular edges incident to Y reduced by the weight of DE(X,Y). These two sets will be used to find an optimal path for each distant edge. In Figure 5, the values in parenthesis on the k-th AE(X,X) from X represent From(X,k) (e.g., {7,8} on the first AE(P12,P12)), and the values in parenthesis on AE(X,Y) represent Last(X,Y) (e.g., {10,12} on AE(P12,P41)). In Figure 5, Path(P12) was constructed instead of four distant edges in the following procedure: **Stage 1.** From(P12,1) = {7,8}. **Stage 2.** From(P12,2) = {min[7+9,8+11], min[7+10,8+12]} = {16,17}. **Stage 3.** Existence of DE(P12,P41): Last(P12,P41) = {13-3,15-3} From(P12,3) = {min[16+10,17+12], min[16+14,17+16]} = {26,30}. **Stage 4.** Existence of DE(P12,P51): Last(P12,P51) = {17-6,19-6} Existence of DE(P12,P52): Last(P12,P52) = {18-12,20-12} From(P12,4) = {min[26+11,30+13], min[26+6,30+8]} = {37,32}. **Stage 5.** From(P12,5) = {min[37+21,32+23], min[37+22,32+24]} = {55,56}. **Stage 6.** Existence of DE(P12,P71): Last(P12,P71) = {25-9,27-9} From(P12,6) = {min[55+16,56+18], min[55+26,56+28]} = {71,81}.

Therefore, a modified problem structure is obtained by a single scan from P12 to nodes in column 7.

We should generalize strategy 2 to consider the case of interactions among distant edges. The following definitions are needed.

Definition 9. In the graph G'(V,E'), a distant edge DE(X,Y) *contains* another distant edge DE(Z,W) if node X has a smaller stage number than node Z and node Y has a larger stage number than node W.

Definition 10. In the graph G'(V,E'), a distant edge DE(X,Y) *overlaps* another distant edge DE(Z,W) if node X has a smaller stage number than node Z but node Y has a larger stage number than node Z and a smaller stage number than node W.

Suppose DE(X,Y) overlaps or contains DE(Z,W). The graph problem modified by strategy 2 is not always equivalent to the original problem because the artificial path between X and Y may not dominate all possible paths between X and Y. We will use two examples to show how the problem is overcome. The first example is for the case where one distant edge contains another. The simplified graph representation for this example is given in Figure 6. Figure 6 shows that DE(P12,P71) contains DE(P32,P61). First transform DE(P32,P61) into the artificial path between P32 and P61 based on strategy 1. This artificial path dominates all possible paths between P32 and P61. On the other hand, DE(P12,P71) cannot be transformed into the corresponding artificial path according to strategy 1, because the cost of path P12 → P32 → P61 → P71 is reduced by the weights of DE(P12,P71) and DE(P32,P61) while the cost of the other paths are reduced by the weight of DE(P12,P71) only. Therefore, when constructing the artificial path between P12 and P71, the algorithm should consider all possible paths including the artificial path between P32 and P61. The resulting modified problem structure is given in Figure 7.

† The weight of E(P21,P31) is not reduced by the weight of DE(P12,P31) since the edges are identical. The details will be discussed in strategy 4.

‡ Fig. 4 - 11 at end of paper.

The second example is for the case where a distant edge overlaps the other. The simplified graph representation for this example is given in Figure 8. In this figure, DE(P12,P51) overlaps DE(P22,P71). If two artificial paths are constructed for DE(P12,P51) and DE(P22,P71) by strategy 1, they dominate all possible paths between P12 and P51 and between P22 and P71 respectively. Then let us consider the dominant path between P12 and P71. Three paths are possible: path P12 → P51 → P71, path P12 → P22 → P71, and path P12 → P22 → P51 → P71. The cost of the first path is affected by the weight of DE(P12,P51), and that of the second path is affected by the weight of DE(P22,P71). The cost of the last path, however, is affected by the weights of both DE(P12,P51) and DE(P22,P71). If strategy 1 is applied to this overlapping case, the last path cannot be considered. In order to consider the last path, the artificial path starting from P12 is constructed as follows: i) When finding the optimal path, all possible paths including the artificial path between P22 and P71 are considered. ii) An artificial edge, AE(P12,P22), is introduced to connect the two artificial paths: the last artificial node in the first artificial path is adjacent to the artificial node with the next stage number in the second path. The resulting modified problem structure for the example is given in Figure 9. The following strategy is proposed to generalize the ideas from the previous two examples.

Strategy 3. As the construction order of Path(X), start the distant edge(s) incident from the node with the largest stage number in the original graph, and continue until transforming the distant edges incident from the node with the smallest stage number. When constructing a Path(X), find an optimal path by applying a DP algorithm to the currently modified problem structure.

So far, we have discussed only the case where no more than two plans had a common identical task. Now, let us consider the case where more than two plans have the common identical task(s). Let us look at plans P12, P21, and P31 in Figure 1 again. To analyze this case more easily, let us consider three task-edges only: TE(T^2_{12}, T^2_{21}), TE(T^2_{21}, T^2_{31}), and TE(T^2_{12}, T^2_{31}). If these three plans are chosen, the total saving is not the sum of the weights of all three task-edges, because one task should be executed and its result used by the other two tasks. Hence, the total saving is 60 and not 90. Consequently, the calculation method should handle the case of two plans sharing a task differently than the case of three or more plans. The following propositions are used to reduce the complexity of identifying the cases.

Proposition 1. Given the graph G'(V,E') with N stages, suppose $n \le N$ access plans have a common identical task, the subgraph (nodes and edges) representing these n plans is always a *complete graph*.

Proof) In the graph G(V,E), one plan should be connected to the other n-1 plans by an identical task-edge representing the common task. For any n plans, it is alway true. Therefore, in the graph G'(V,E'), the nodes representing these n plans are completely connected to each other.

Proposition 2. Given that the complete subgraph of proposition 1 results from only one identical task, say task T, then the resulting saving from these n plans is the sum of the weights of n-1 edges, not of n(n-1)/2 edges.

Proof) Task T is shared by all n access plans. Only one plan has to execute task T. For the global plan, there is one edge connecting the node with the executed task T to each of the n-1 nodes with the unexecuted task T. Therefore, the saving is the sum of the weights of n-1 edges incident to the node with the executed task T.

Prop. 1 implies that if more than two plans have a common identical task†, then there exists at least one distant edge among the corresponding nodes since they form a complete graph. Hence, in order to identify the case of a task being shared by more than two plans, we have to check only for the existence of a distant edge. Prop. 1 also indicates that in order to detect how many plans have common task(s) with a given plan, we check only the nodes adjacent to that node in G'(V,E'). If a distant edge overlaps or contains another distant edge, they do not have a common identical task. Prop. 2 implies that when detecting several identical task-edges incident to a node, we use the only one of them to calculate the saving from the sharing.

The afore-mentioned ideas for are incorporated into the procedure in the following way. According to strategy 2, the last artificial edge in the construction of each artificial path reflects the saving associated with a distant edge. In strategy 3, the case of three or more plans sharing a task is a special case of containment where several distant edges are incident to the same node. Therefore, the modification procedure of strategy 3 is revised as follows: the algorithm is to find an optimal path using Last(X,Y) based on the following strategy.

Strategy 4. Given three nodes X, W, and Y (in ascending order of stage numbers) and the values in Last(W,Y), we need to find the values of Last(X,Y). Last(X,Y) represents not only the adjusted weights of the regular edges adjacent to Y but also the adjusted weights of the artificial edges adjacent to Y in the currently obtained artificial paths. If X, W, and Y are a part of an identical-task list for some task, then the values of Last(W,Y) appear in in Last(X,Y) are adjusted by subtracting from them the weight of DE(X,Y) and adding to them their common weights of the identical lists.

Let us consider an example to illustrate the modification procedure implied by strategy 4. The simplified graph representation for the example is given

† If the plans are located in two adjacent columns, this case is exactly the same as the case of only two plans with the same common task.

in Figure 10. In this graph, the identical-task list is {P12,P42,P71}: DE(P12,P42), DE(P42,P71), and DE(P12,P71) has a common task and their common weight is 10. The modification procedure for this graph is: i) Applying strategy 2: From(P42,2) = {32,33} and Last(P42,P71) = {34-15,32-15}. ii) According to strategies 3 and 4, From(P12,5) = {37,38,40,41} and Last(P12,P71) = {34-25,32-25, 19-(25-10),17-(25-10)}. Then, the value of the optimal path between P12 and P71 is min[37+9,38+7,40+4,41+2] = 43. It should be noticed that the last two elements of Last(P12,P71) were reduced by 15, not 25. The resulting modified problem structure is given in Figure 11.

In this section we have described the logic of the algorithm using examples. We have discussed five possible relationships between two distant edges. 1) Independent relationship (See Figure 2). 2) Incidence-from relationship (Figure 4). 3) Containment relationship (Figure 6). 4) Overlapping Relationship (Figure 8). 5) Incidence-to relationship: (Figure 10). After modifying the graph as demonstrated in this section, the optimal solution can be achieved by applying a standard DP procedure to the modified graph. However the DP algorithm described in [PARK88] performs the modifications at each stage during its process. The minimum value in From(P0,*N*), where *N* is the last stage number, gives an optimal solution to the access plan selection problem.

4. IMPLIED RELATIONSHIPS

In this section, the DP algorithm is extended to the case of implied relationships among tasks. We present here an informal description of the procedure; a formal analysis is given in [PARK88]. Consider the access plans, P1 and P2, as shown in Figure 12.

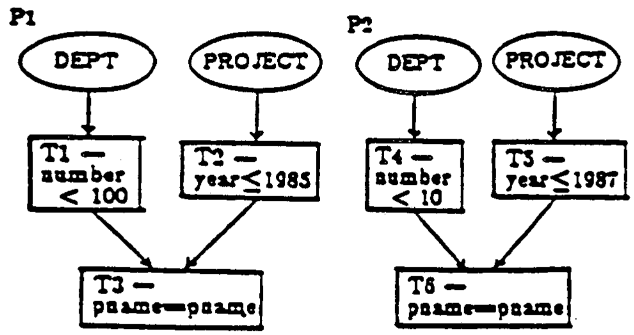


Fig. 12: Access Plans P1 and P2

Assume that task T4 implies task T1, and that task T2 implies task T5. The implied relationship between tasks T1 and T4 illustrates the difference of this case from the case of identical relationships: i) The result of the implied task T1 can be used for the execution of the implying task T4, but the reverse is not true. ii) The savings from sharing T1 and T2 is dependent on the cost of T1, not on the cost of T4.

The savings is calculated by a joint consideration of the two implied relationships, T4 => T1 and T2 => T5, because the global access plan for P1 and P2 is given as in Figure 13.

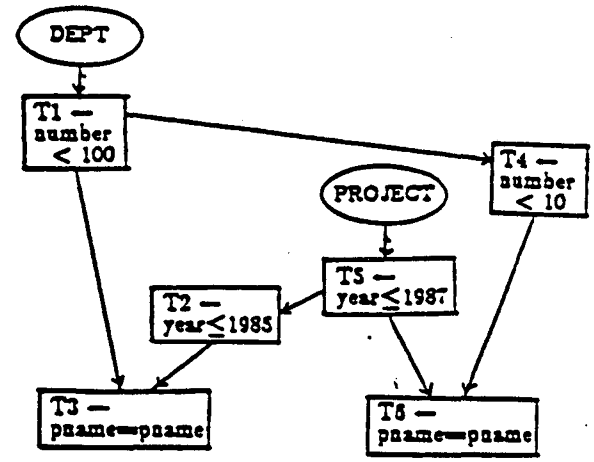


Fig. 13: Global Access Plan for P1 and P2

Let us consider N access plans to generalize the idea of the previous example. The plans and their relationships can be represented by a directed graph G(V,A) with arc-weights S ≤ 0 and node-weights C > 0. Assume that for each plan *Pi*, there exists only one plan *Pj*, *j*≠*i*, such that a task of *Pj* has an implied or identical relationship with a task in plan *Pi*. Then the total savings from using the N plans is the sum of the weights of all arcs connecting tasks in the N plans in G(V,A), regardless of the direction of the arcs. Under the above assumption, using all arcs does not make the graph cyclic according to [SELL86]; so, the maximum savings results from the use of all arcs. In this case, the saving depends on the chosen plans and not on their order. Therefore, the graph G(V,A) can be simplified into G'(V,E'), which is the same as G'(V,E') in the previous section.

Now, we discuss the case of relating more than two plans. Consider three access plans as shown in Figure 14. This graph shows that there is a task in each plan having an implied relationship with tasks in the other two plans.

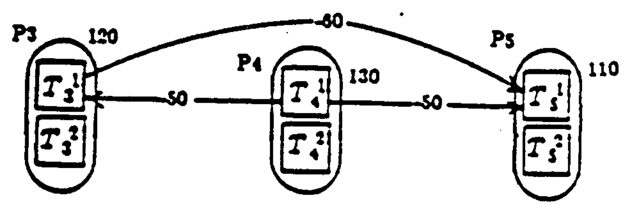


Fig. 14: G(V,A) for Plans P3, P4, and P5

IF plans P3, P4, and P5 are chosen by the access plan selection algorithm, the total savings is not the sum of the weights of all arcs, because we can use the results of either P3 or P4 (but not both) for the execution of P5. Hence, the maximum saving is the sum of the weights of Arc(P4,P3) and Arc(P3,P5). the same direction.

Let us consider the graph G(V,A) with N stages, each of which has only one plan. The following observations in G(V,A) and G'(V,E') can be derived from the

previous example ($n \leq N$):
i) If a task in a given plan is related to (is identical to or implies) tasks in n-1 plans respectively, then in G(V,A), the subgraph (nodes and arcs) representing these n tasks is always a *complete digraph*. Moreover, in G'(V,E') the subgraph representing the n plans with these tasks is always a *complete graph*. ii) For each complete sub-digraph Gi(Vi,Ai) in G(V,A), its maximum saving can be obtained by solving the directed spanning forest problem [LAWL76] over Gi(Vi,Ai). iii) The maximum savings for each Gi(Vi,Ai) is equal to the optimal value of the maximal spanning tree problem [LAWL76]; it results in the total weight of N-1 arcs chosen in descending order of their weights.

The above two cases show that the calculation of the savings depends on how many plans have common relationships, but not on the order of the chosen plans. Therefore, for the access plan selection problem, G(V,A) can be simplified into G'(V,E') plus the sets of identical-task and implied-task lists. It can be concluded that the strategies proposed in the previous section, except strategy 4, can be applied directly to the case of implied relationships. Strategy 4 is modified by choosing the edge with the maximum weight as the distant edge used to reduce the weights in Last(X,Y).

We now consider the computational complexity of the DP algorithm. The worst-case complexity of the DP algorithm occurs where each node in a stage is completely connected with all nodes in all other stages (which is impossible in a real situation). Let N_i denote the number of candidate access plans for query Q_i. The maximum number of nodes searched by our algorithm is: $\sum_{i=2}^{N-1} (\prod_{k=1}^{i-1} N_k) + 3$. This complexity is for the case of general predicates. Moreover, we believe that in most situations the number of distant edges are not large and the complexity is much less than this worst case. The worst-case complexity of [SELL86] for the case of identical relationships is $\prod_{i=1}^{N} N_i$, and it is $\prod_{i=1}^{N} (\sum_{k=i}^{N} N_k)$ in the case of implied relationships. In the case of implied relationships, our algorithm presents better worst-case performance with no qualification. In the case of identical relationships, we require that $N_N N_{N-1} > N$. This is a reasonable assumption because we are free to permute the order of the stages, and have the last two be the ones with the maximum number of candidate plans. To appreciate the complexity improvement of our algorithm, consider the case of 6 queries, each with 5 candidate plans. The maximum number of nodes for [SELL86] is 15,625 for the case of identical relationships and 11,250,000 for implied relationships. In the case of our algorithm, the number is 783 for the two cases

5. CONCLUSIONS

The problem of multiple-query optimization consists of two conceptual parts - identifying common subexpressions (or tasks) and constructing a global access plan for a set of queries such that their processing cost is minimized. In this paper we focused on the second part, and proposed a DP algorithm. The following considerations guided our work: i) As expert database systems and extended database systems are developed, the number of rules or queries considered at one time can become quite large. Therefore, the computational complexity of the access plan selection algorithm is a very important factor in the design of such systems. ii) The construction of a global plan should be based on a cost-benefit analysis in order to achieve a satisfying performance of processing several queries. iii) The savings from the sharing of several plans depends on the chosen access plans but not on their order.

Future research is concerned with the analysis of the average performance of the algorithms, and the use of fathoming techniques such as in [SELL86]. We would also like to incorporate the algorithm for identifying common subexpressions as an integral part of our access plan selection algorithm.

ACKNOWLEDGMENT

This research was supported by the Applied Mathematics Sciences Research Program of the Office of Energy Research U.S. Department of Energy under contract DE-AC03-76SF00098.

REFERENCES

Astrahan, M. et al. "System R: A Relational Approach to Database Management", **ACM Transactions on Database Systems,** (1), 2, June 1976.

Chakravarthy, U.S. and Minker, J., "Processing Multiple Queries in Database Systems", **Database Engineering,** (1), 1982.

Codd, E.F., "A Relational Model for Large Shared Data Banks", **Communication of the ACM,** (13), 6, June 1970.

Chakravarthy, U.S. and Minker, J., "Multiple Query Processing in Deductive Databases using Query Graphs", **Proceedings of the 12th International Conference on Very Large Data Bases,** Kyoto, August, 1986.

Finkelstein, S., "Common Expression Analysis in Database Applications", **Proceedings of the ACM-SIGMOD International Conference on Management of Data,** Orlando, FL, June 1982.

Grant, J. and Minker, J., "On Optimizing the Evaluation f a Set of Expressions", **University of Maryland,** Technical Report TR-916, College Park, MD, July 1980.

Grant, J. and Minker, J., "Optimization in Deductive and Conventional Relational Database Systems", in **Advanced in Database Theory,** vol. 1, H. Gallarie, J. Minker and J.M. Nicolas, Eds., Plenum Press, New York, 1981.

Hall, P.V., "Common Subexpression Identification in General Algebraic Systems", **IBM United Kingdom Scientific Center,** Technical Report UKSC 0060, November 1974.

Hall, P.V., "Optimization of a Single Relational Expression in a Relational Database System", IBM

Journal of Research and Development, (20), 3, May 1976.

Horowitz, E., Sahni, S., "Fundamentals of Computer Algorithms", **Computer Science Press, Inc.**, 1978.

Jarke, M., and Koch J., "Range nesting: a fast method to evaluate quantified queries", **Proceedings of the ACM-SIGMOD Conference on Management of Data**, San Jose, 1983.

Jarke, M. and Koch, J., "Query Optimization in Database Systems", **ACM Computing Survey**, (16), 2, June, 1984a.

Jarke, M., "Common Subexpression Isolation in Multiple Query Optimization", in **Query Processing in Database Systems**, W. Kim, D. Reiner and D. Batory, Eds., Springer-Verlag, New York, 1984b.

Kim, W., "Global Optimization of Relational Queries: A First Step", in **Query Processing in Database Systems**, W. Kim, D. Reiner and D. Batory, Eds., Springer-Verlag, New York, 1984.

Larson, P. and Yang, H., "Computing Queries from Derived Relations", **Proceedings of the 11th International Conference on Very Large Data bases**, Stockholm, August 1985.

Lawler, E., "Combinatorial Optimization: Networks and Matroids", **Holt, Rinehart and Winston Inc.**, 1976.

Park, J. and Segev, A., "Common Subexpression Optimization in Database Systems", **Working Paper**, 1988.

Rosenkrantz, D.J., and Hunt, H.B., "Processing Conjunctive Predicates and Queries", **Proceedings of the 6th International Conference on Very Large Data Bases**, Montreal, October 1980.

Roussopoulos, N., "View Indexing in Relational Databases", **ACM Transactions on Database Systems**, (7), 2, June 1982.

Sellinger, P.G., "Access Path Selection in a Relational Database System", **Proceedings of the ACM-SIGMOD International Conference on Management of Data**, May 1979.

Sellis, T., "Global Query Optimization", **Proceedings of the ACM-SIGMOD International Conference on Management of Data**, Washington, DC, May 1986. (An extended version is forthcoming in ACM Transactions on Database Systems)

Shneiderman, D.W. and Goodman V, "Batched searching of sequential and tree structured files", **ACM Transactions on Database Systems**, (1), 3, September 1976.

Stonebraker, M., "Implementation of Integrity Constraints and Views by Query Modification", **Proceedings of the ACM-SIGMOD International Conference on Management of Data**, San Jose, May, 1975.

Stonebraker, M.R., Wong, E., Kreps, P., Held, G., "The Design and Implementation of INGRES", **ACM Transactions on Database Systems**, (15), 4, September 1976.

Wong, E. and Youssefi, K., "Decomposition - a Strategy for Query Processing", **ACM Transactions on Database Systems**, (1), 3, September 1976.

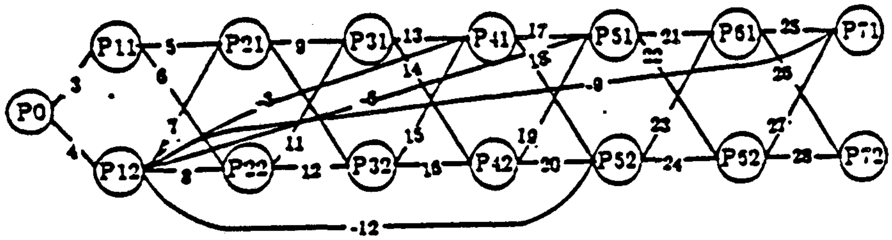


Fig. 4: Example with Distant Edges Incident from the Same Node

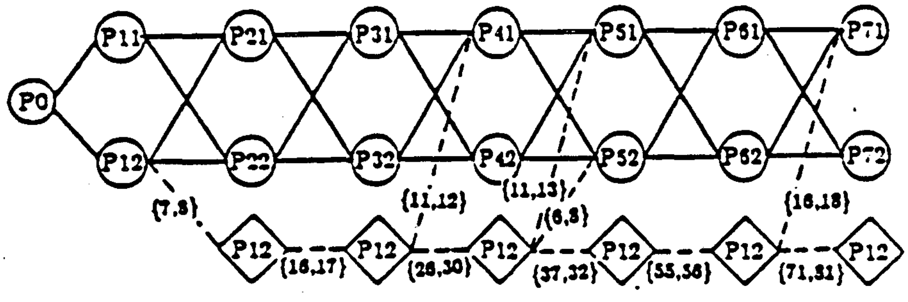


Fig. 5: Modified Problem Structure for Figure 4

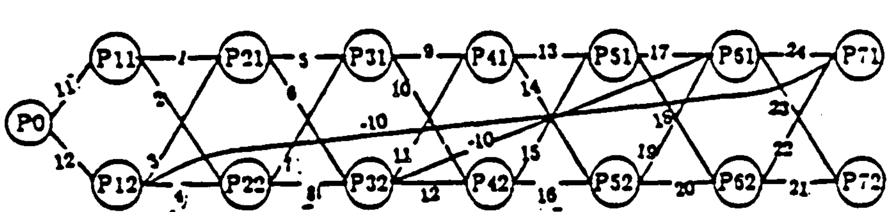


Fig. 6: Example for the Containment Case

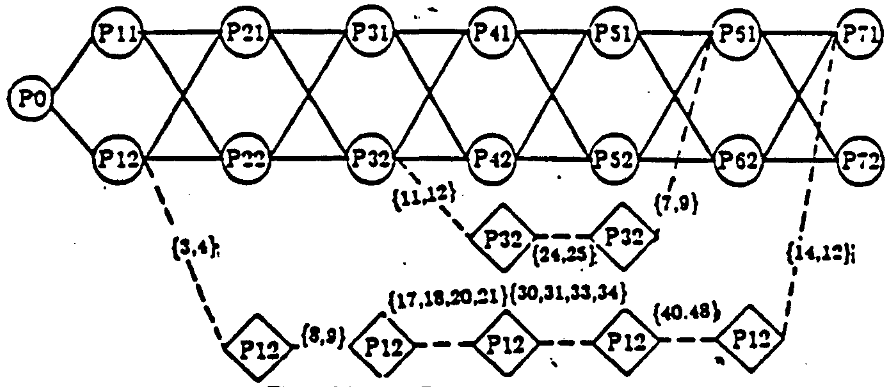


Fig. 7: Modified Problem Structure for Figure 6

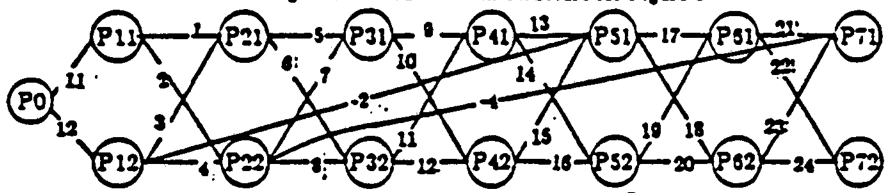


Fig. 8: Example fo the Overlapping Case

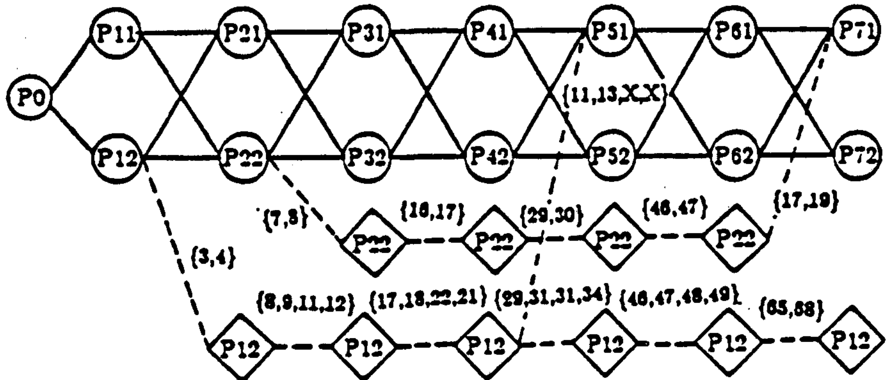


Fig. 9: Modified Problem Structure for Figure 8.

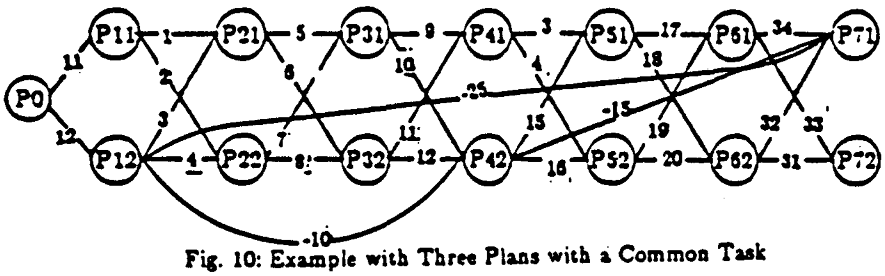


Fig. 10: Example with Three Plans with a Common Task

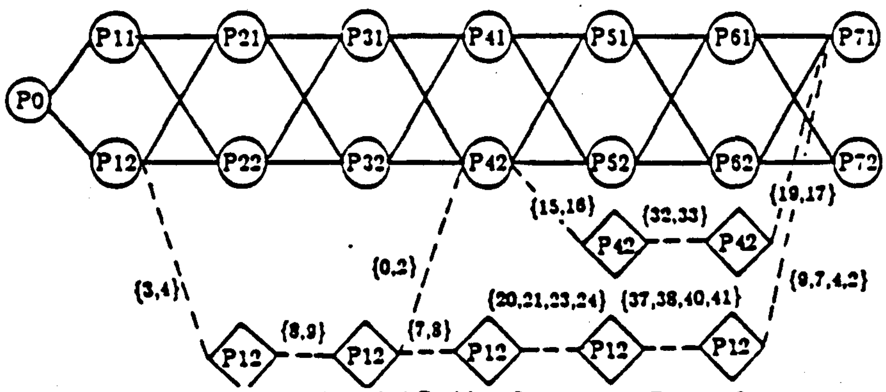


Fig. 11: Modified Problem Structure for Figure 10