Multiple Query Processing in Deductive Databases using Query Graphs

U. S. Chakravarthy

Computer Corporation of America
Four Cambridge Center
Cambridge, MA 02148.

J. Minker

Department of Computer Science and UMAICS
University of Maryland
College Park, MD 20740.

Abstract

Research activity on query evaluation and optimization has been centered around processing one query at a time. Query processing systems, in general, attempt to minimize the cost of processing a set of queries by minimizing the processing cost of each query separately. A separate plan is generated and executed for each query. The cost of processing (that is the CPU cost + Input/Output cost) a set of queries evaluated in the above manner is equal to the sum of the processing cost for each query. In this paper we extend the connection graph decomposition algorithm to generate a single plan for evaluating a set of queries. The approach presented in this paper is aimed at generating a single plan, exploiting the common sub-expressions that can be detected using heuristics. We assume, for the purposes of this paper, that we are answering queries over a deductive database, although the approach is equally applicable to a group of independent queries.

1. Introduction

Research activity on query evaluation and optimization has been centered around processing one query at a time. Query processing systems, in general, attempt to minimize the cost of processing a set of queries by minimizing the processing cost of each query separately. A separate plan is generated and executed for each query. The cost of processing (that is the CPU cost + I/O cost) a set of queries evaluated in the above manner is equal to the sum of the processing cost for each query.

Processing of a set of queries individually is usually appropriate if

a) the rate at which the database is queried in an interactive environment does not permit them to be grouped together without increasing the response time substantially.

b) the workspace requirements of grouping together a set of queries is not cost effective.

c) the queries are disjoint and do not have any computations that can be performed in common.

However, processing of a set of queries individually is expected to be inefficient in a batch processing environment. Also when several queries are submitted by a single user as one transaction, processing them individually could be inefficient. In an interactive environment, if the queries on a database are being generated at a steady rate, several queries may be grouped together and evaluated though they are generated independently. The response time may suffer slightly depending upon the number of queries grouped, but the evaluation cost could be decreased enormously if there exists sub-expressions that can be detected and evaluated only once. Although individual queries may be regarded as independent and possibly generated by independent sources, it is plausible that a set of queries have operations on the same relations of the database. Many queries may involve the join of the same two relations, may access the same relations and part of one query may subsume part of another query. Under these circumstances grouping a set of queries together and processing them as a unit is clearly beneficial.

Based on the output requirements, a set of queries can be classified into following categories:

1. Queries, which have been generated independently and hence a *separate answer set* for each query has to be computed. These queries are typically generated by independent users querying a database.
2. Queries, which are derived from a single query and hence there is only *one answer set* to be computed. Queries, in a deductive database environment, are of this kind where a single query may give rise to several disjunctive queries. These queries also arise when view definitions are permitted on conventional relational systems. Queries on complex objects are likely to generate multiple queries requiring an answer set.

In [Chak82], we analyzed the overlap probability (that is the probability that a single relation name occurs among a group of independently generated queries) of a set of independent queries assuming a Gamma distribution. We plotted the overlap probability with respect to the parameters of the Gamma distribution and showed that there is a reasonable overlap probability even if the number of queries grouped are under five.

In the context of deductive databases, multiple query processing is extremely useful to reduce the I/O cost when the same relation has to be accessed more than once in the process of answering a query. Identifying common sub-expressions is central to the problem of multiple query optimization.

In this paper we consolidate the ideas presented in [Chak85] to extend the connection graph decomposition algorithm to generate a common plan for evaluating a set of queries. The approach presented in this paper generates a single plan, exploiting the common sub-expressions that can be detected among the queries. Our approach modifies and extends the algorithm that generates plans for a single query. We assume that we are answering queries over a deductive database, although the approach is equally applicable to a group of independent queries.

The rest of this paper is organized as follows. Section 2 briefly describes the previous work on to multiple query processing. In section 3 we present the notion of common sub-expressions which is central to multiple query processing. Section 4 describes the use of connection graphs for representing queries. In section 5 we briefly describe the decomposition algorithm as applied to a single query. In section 6 we introduce the notion of multi-query graph as a connection graph that represents a set of queries in a non-procedural form. We then describe the decomposition algorithm that can be used on a multi-query graph to generate a single plan for the set of queries represented by the graph. We also present a set of heuristics that enable us to identify and exploit common sub-expressions. This section highlights the contributions of this paper. We briefly summarize our results in Section 7.

2. Previous Work

Kim [Kim80] recognized the potential of grouping queries and suggested the idea of processing a set of queries to reduce the total cost of executing a set of queries on the same database. His discussion involves subdividing a nested SQL query into several queries which can be executed simultaneously. He provides some rules and discusses examples of queries having only one relation. He treats transactions which have updates and deletes.

Grant and Minker [Gran80] have studied optimization for a set of disjuncts produced by intensional axioms of a deductive database. Cost formulas for a variety of physical representations are provided using which an execution plan for the set of disjuncts is generated exploiting the commonality in the set of disjuncts. A branch and bound algorithm is proposed taking into consideration indexing, join support for relations, and common sub-expressions.

Jarke [Jark84] discusses multiple query processing and common sub-expression isolation in set-oriented database systems. Recognition and evaluation of one relation sub-expressions that are common to a set of relational queries are discussed. It is indicated that multi-relation sub-expressions can only be addressed in a heuristic manner.

3. Common Sub-expressions

The objective of grouping a set of queries for simultaneous evaluation is to reduce the overall cost of processing the set of queries in comparison to the combined cost of processing each query in the set separately. This is achieved by performing computations that are common to all (or even some) of the queries, only once and use the intermediate computations for obtaining answers to all the queries. The capability to incorporate common computations into the plan being generated for a set of queries is predicated upon the ability to detect common sub-expressions among the queries grouped together. Hence, detection of common sub-expressions is central to the processing of a set of queries as a single unit. The expression tree built from the operators of the relational algebraic expression is not well suited for detecting common sub-expressions.

The following example (from [Jark84]) brings out the limitations of the use of an expression tree in detecting common sub-expressions.

Example 1:

Let the relations of the database be

employee(ename, status, ward), and
opteam(ename, day).

For the following two simple queries on the above database, the expression trees are shown in Figure 1.

a) "Which wards are members of monday operating team?", and

b) "Which doctors were on monday operating team?"

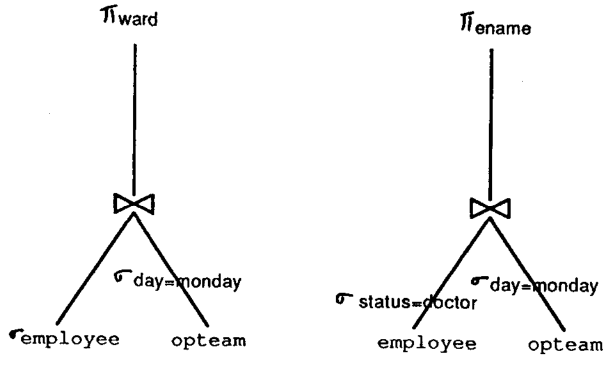


Figure 1.

In the expression trees of Figure 1, it appears as if the right sub-tree below the join operation is the only common sub-expression. In fact, one can identify a larger sub-expression if we rewrite the above expression trees as shown in Figure 2.

In figure 2, one can identify the entire sub tree below the selection $\sigma_{\text{status} = \text{doctor}}$ as common to both the trees. The above queries can be computed by accessing the relations employee and opteam and performing the selection and the join only once.

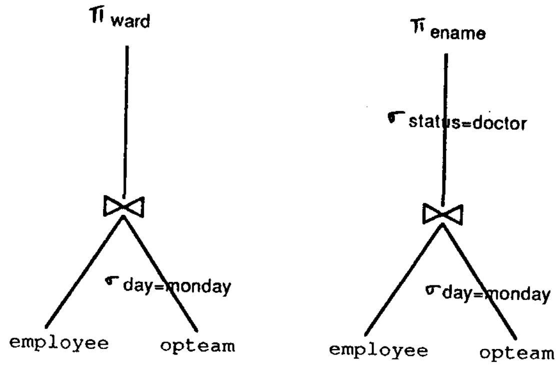


Figure 2.

4. Query Representation using a Connection Graph

A connection graph [Ullm82] is used to represent a query in a non-procedural form. A connection graph consists of nodes (representing relations) and edges representing conditions between attributes of the nodes they are connected with. An edge in a connection graph may be associated with more than two nodes making it a hypergraph. A connection graph can be decomposed (using the QUEL decomposition algorithm of [WONG76]) into a plan by using two transformations, namely, instantiation and iteration. These transformations consist of selecting a node, writing an expression for evaluating the graph in terms of the selected node (and the edges associated with the node) and rewriting the graph in terms of a simpler graph. The process described above is recursively applied till the graph consists only of isolated nodes.

A connection graph can be decomposed in several ways depending on the criterion used to select a node for the purpose of decomposition. Heuristics have been proposed to select the nodes in such a manner that the decomposition minimizes the computation cost for evaluating a query.

In this paper we propose the evaluation of a set of queries using the connection graph approach. We indicate how the non-procedural representation using a multi-query graph helps isolate common sub-expressions (using appropriate heuristics) in a set of queries and how a single plan can be generated that evaluates common sub-expressions only once.

We briefly describe the decomposition of a connection graph in the next section before presenting our extensions to multiple queries. The notation used in the remainder of the paper is borrowed from Maier [Maie83] and is extended suitably for presenting our work.

5. Decomposition of a Connection Graph

Let E be an algebraic expression

$\sigma_C\ (S_1\times, \ldots, \times S_m)$.

where $C = C_1\wedge \cdots \wedge C_k$ is in conjunctive normal form (CNF). The expression E is represented by a labeled hypergraph H_E, called the connection graph. In a hypergraph, edges may contain one or more nodes, rather than just two as in regular graphs. H_E has a node for each of $S_1, \ldots, S_m$. H_E contains an edge e_i for each conjunct C_i. Edge e_i contains nodes S_j, if $y_j.B$ appears in C_i for any attribute B, where y_j is any tuple of S_j. Edge e_i is labeled by C_i.

Though an edge in a connection graph many be associated with more than two nodes, we will assume that an edge will have at most two nodes in all subsequent discussions. A connection graph can be drawn as a regular graph under the above assumption.

Figure 2a represents a connection graph for the expression

$\Pi_{y.C,z.B}\sigma_{C_1\wedge C_2\wedge C_3\wedge C_4}\ (S_y\times S_z\times S_w)$

where C_1 is y.B = monday, C_2 is y.A = z.A, C_3 is z.B = w.A and C_4

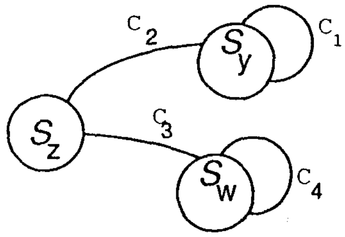


Figure 2a.

is w.B = wound. S_y, S_z and S_w are relations.

For an expression of the form

$\Pi_X\sigma_C\ (S_1\times, \ldots, \times S_m)$,

the aim is to compute the relation corresponding to the above expression by decomposing the connection graph of the expression $\sigma_C\ (S_1\times, \ldots, \times S_m)$. The result of the decomposition is a program that contains assignment, selection, projection and for loops, but no joins. The joins are performed via for loops instead. Two transformations - instantiation and iteration, are used to achieve the above goal. The transformations correspond to edge removal and node removal respectively, and the goal is to transform the connection graph for $\sigma_C\ (S_1\times, \ldots, \times S_m)$ to a graph with isolated nodes.

Instantiation is analogous to pushing selections down an expression tree. Consider the statement

$r \leftarrow \Pi_X\sigma_C\ (S_1\times, \ldots, \times S_m)$,

where $C = C_1\wedge \cdots \wedge C_k$ is a CNF selection condition. Let E denote the expression $\sigma_C\ (S_1\times, \ldots, \times S_m)$. For the purpose of instantiation choose a relation, say S_1. Let $e_1, \ldots, e_q$ be the edges in H_E that consists only of node S_1. The transformed program for this instantiation has two statements

$r_1 \leftarrow \Pi_Y\sigma_{C_1\wedge \cdots \wedge C_q}\ (S_1)$

$r \leftarrow \Pi_X\sigma_{C_{q+1}\wedge \cdots \wedge C_k}\ (r_1\times, \ldots, \times S_m)$

Y is the set of attributes in S_1 that are mentioned in either $C_{q+1}\wedge \cdots \wedge C_k$ or contained in X. Relation r_1 is a temporary relation to hold the intermediate result. Note that instantiation can be applied to a set of nodes (relations) of the graph H_E.

The connection graph is modified as follows. Substitute r_1 in place of S_1 and remove all edges $e_1, \ldots, e_q$. Create a new graph with node S_1 and the edges $e_1, \ldots, e_q$.

Instantiation of the relation S_y in the Figure 2a produces the following program. The modified connection graph is shown in Figure 2b.

$r_1 \leftarrow \Pi_{y.A,y.C}\ (\sigma_{y.B\ =\text{monday}}\ (S_y))$

$r \leftarrow \Pi_{y.C,z.B}\ \sigma_{C_2\wedge C_3\wedge C_4}\ (r_1\times S_z\times S_w)$

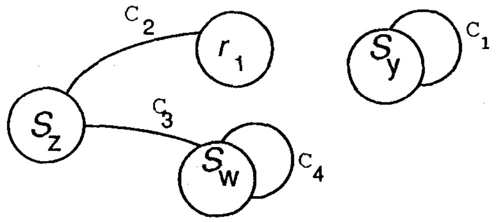


Figure 2b.

Iteration is analogous to the notion of tuple substitution. Given a single statement

$r \leftarrow \Pi_X\sigma_C\ (S_1\times, \ldots, \times S_m)$,

and its corresponding connection graph, choose a relation, say S_1. The tuples of S_1 are iterated giving rise to the program

```
r ← ∅
for each tuple t in $S_1$ do
begin
  $r_1 \leftarrow \Pi_Y\sigma_{C(t)}\ (S_2\times, \ldots, \times S_m)$
  add $r_1$ to r with appropriate padding
end
```

Here C(t) means C with every occurrence of an attribute $y_1.A$ replaced by the value $t(y_1.A)$. Y contains those attributes in X that are not in the schema of S_1. The appropriate padding for r_1 is t(X-Y). That is r_1 is extended by the portion of t that is included in X.

The corresponding changes in the connection graph is to remove the node S_1. Any edges that were incident on S_1 become loops.

Iterating on the tuples of the relation S_z in Figure 2b, we obtain the following program. The transformation of the connection graph of Figure 2b is shown in Figure 2c.

```
$r_1 \leftarrow \Pi_{y.A,y.C}\ (\sigma_{y.B\ =\text{monday}}\ (S_y))$;
r ← ∅;
for each tuple t in $S_z$ do
begin
  $\Pi_{y.C}\ r_2 \leftarrow \sigma_{C_2(t)\wedge C_3(t)\wedge C_4}\ (r_1\times S_w)$;
  add $r_2 \times <t(z.B)>$ to r;
end.
```

The query decomposition algorithm carries out the above two transformations on a given connection graph until the graph reduces to a set of isolated nodes. For details regarding the query decomposition algorithm see Maier [Maie83].

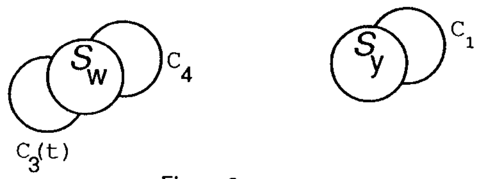


Figure 2c.

6. Multi-query Graph and its Decomposition

The basic idea behind a multi-query graph is to create a single connection graph which represents a set of queries in a non-procedural form. A multi-query graph facilitates the detection of common sub-expressions among the queries represented by the graph. In this section we start with the definition of a multi-query graph. We then extend the definition of the two transformations, namely, instantiation and iteration with respect to a multi-query graph and describe the new decomposition algorithm for a multi-query graph. A multi-query graph, when decomposed, computes the relations corresponding to the queries represented by the multi-query graph. We propose a set of heuristics for the decomposition of a multi-query graph using which common sub-expressions are easily identified.

Let $E_1, \ldots, E_n$ be n algebraic expressions as shown below:

$$\sigma_{C^1}(S_1^1 \times \cdots \times S_{m_1}^1)$$
$$\sigma_{C^2}(S_1^2 \times \cdots \times S_{m_2}^2)$$
$$\vdots$$
$$\sigma_{C^n}(S_1^n \times \cdots \times S_{m_n}^n)$$

where $C^i = C_1^i \wedge \cdots \wedge C_{k_i}^i$ is in conjunctive normal form. We shall write the expressions $E_1, \ldots, E_n$ in an alternative form as

$$\sigma_{C^1}: \cdots :\sigma_{C^n}(S_1 \times \cdots \times S_m)$$

where each S_i, for $i = 1$ to m is a member of the set of relations making up the cartesian product of any of the expression E_j, for $j=1$ to n. The separator ':' separates the selection conditions of the queries. Note that the expression $(S_1 \times \cdots \times S_m)$ may contain relations that are not referenced in some of the conditional expressions. In that case the cartesian product with respect to a relation whose attributes are not referenced in the selection expression need not be computed. For the sake of simplicity, we will assume in the remainder of the discussion that every condition C_j^i is atomic in nature.

Example 2:

Two expressions E_1 and E_2 are shown below.

$$\sigma_{C_1^1 \wedge C_2^1 \wedge C_3^1}(S_x \times S_y) \text{ and}$$
$$\sigma_{C_1^2 \wedge C_2^2 \wedge C_3^2 \wedge C_4^2}(S_y \times S_z \times S_w)$$

The two expressions E_1 and E_2 can be alternatively written as

$$\sigma_{C_1^1 \wedge C_2^1 \wedge C_3^1} : \sigma_{C_1^2 \wedge C_2^2 \wedge C_3^2 \wedge C_4^2}(S_x \times S_y \times S_z \times S_w)$$

We represent the expressions $E_1, \ldots, E_n$ by a hypergraph $H_{E_1, \ldots, E_n}$, called the multi-query graph. For an arbitrary expression

$$\sigma_{C^1}: \cdots :\sigma_{C^n}(S_1 \times \cdots \times S_m)$$

a multi-query graph is defined as follows:

a) There is a node corresponding to each of the relations $S_1, \ldots, S_m$ in the above expression.

b) There is an edge for each of the conditions in the selection expressions C^1 through C^n. Each edge bears the condition number as well as the query number it belongs to.

Henceforth, We shall simply refer to a multi-query graph as a query graph.

Example 3:

The multi-query graph corresponding to the expression of Example 2 is shown in Figure 3. The conditions correspond to the attributes of the participating nodes (or relations). It is straightforward to construct a multi-query graph using the algorithm for constructing a connection graph. The transformations on the multi-query graph are explained below.

6.1 Instantiation

The purpose of instantiation is to push the selections towards the leaves of an expression tree. In a query graph several selections on a relation can be performed depending on the number of edges from different queries that are incident on that node. Later we will indicate how the similarity of these conditions can be used to

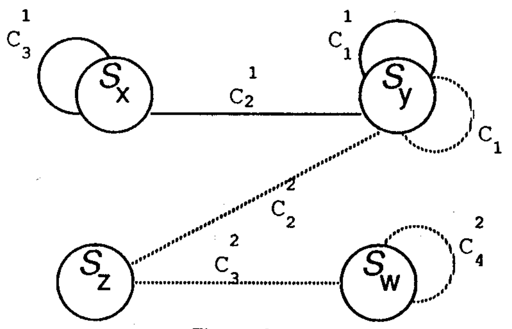


Figure 3.

compute temporary relations for the instantiation operation.

We start with the statement

$$r^1 : \cdots :r^n \leftarrow \Pi_{X^1}:\ldots:\Pi_{X^n}\sigma_{C^1}: \cdots :\sigma_{C^n}(S_1 \times \cdots \times S_m)$$

Instantiation starts with the choice of a relation to instantiate. Let S_1 be the relation (node) chosen. Let $e_1, \ldots, e_q$ be the only edges of the query graph that consists of node S_1. Recall that each edge e_i is labeled with a conjunct C_i^j where j denotes the query number to which the conjunct C_i belongs. Let us partition the edges $e_1, \ldots, e_q$ into n sets, each corresponding to a query, such that each partition contains edges that belongs to that query. If there are no edges in a partition, then an identity condition (which is satisfied by every tuple in a relation) is assumed for that partition. Let k partitions numbered 1 through k contain non-identity conditions.

The transformed program for this instantiation is as follows:

$$r_1^1 :\ldots :r_1^k \leftarrow \Pi_{Y^1} :\ldots :\Pi_{Y^k}(\sigma_{C_{inst}^1, \ldots, C_{inst}^k}(S_1))$$
$$r^1 \leftarrow \Pi_{X^1}\sigma_{C_{rest}^1}(r_1^1 \times \cdots \times S_m)$$
$$\vdots$$
$$r^k \leftarrow \Pi_{X^k}\sigma_{C_{rest}^k}(r_1^k \times \cdots \times S_m)$$
$$r^{k+1} \leftarrow \Pi_{X^{k+1}}\sigma_{C^{k+1}}(S_1 \times \cdots \times S_m)$$
$$\vdots$$
$$r^n \leftarrow \Pi_{X^n}\sigma_{C^n}(S_1 \times \cdots \times S_m)$$

where r_1^p, $p = 1$ to k represents different relations computed for different queries using the selection condition C_{inst}^p (the subscript inst denotes the conditions that are relevant to the instantiation of the query p; the subscript rest denotes the rest of the conditions (C_{rest}^i) for the query p. That is $C_{rest}^p = (C^p - C_{inst}^p)$) on S_1. For a query u whose edges are not incident on S_1 (that is C_i^u is the identity condition) no selection is applied and the temporary relation is not computed. Y^i for i = 1 to k represents the set of attributes that are in S_1 and are not mentioned in the rest of the conditions in the graph corresponding to the query i or contained in X^i. Relations r_1^i are temporary relations to hold the intermediate result.

The changes to the query graph, corresponding to an instantiation, are as follows. Replace the node S_1 by a set of nodes $r_1^1, \ldots, r_1^k$. Each edge e^i, belonging to the query i that is incident on S_1 and does not belong to the set $e_1, \ldots, e_q$ is associated with the node r_1^i, if i is in the range 1 to k or with the node S_1. Also the node S_1 is represented k times associating the appropriate edges from the set $e_1, \ldots, e_q$. The resulting graph has at least k+1 components.

Example 4:

For the query graph of the previous example, if we instantiate on

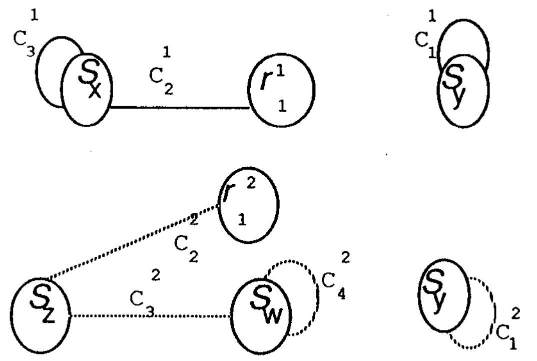


Figure 4.

the node S_y, we would obtain the graph shown in Figure 4.

Note that the instantiation transformation as stated above splits a node into k nodes (two in this example) which is determined by the edges that are incident on the instantiated node from other nodes in the query graph. Note also that if the conditions C_1^1 and C_1^2 were to be identical, then the node S_y need not be split into two nodes. In other words, r_1^1 and r_1^2 would be identical. It is also possible to choose a set of nodes for the purpose of instantiation.

The transformation described above on a query graph computes r_1^i for each i by scanning the relation S_1 only once.

6.2 Iteration

Iteration performs one or more joins using a for loop. This transformation is also referred to as *tuple substitution* or *dissection* in the literature.

We start with a query graph corresponding to the statement

$$r^1 ; \cdots ; r^n \leftarrow \Pi_{X^1} ; \ldots ; \Pi_{X^n} \sigma_{C^1} ; \cdots ; \sigma_{C^n} (S_1 \times \cdots \times S_m)$$

One of the relations, say S_1 is chosen for iteration. The transformed program is

```
$r^1; \ldots ; r^k \leftarrow \emptyset$
for each tuple t in $S_1$ do
begin
  $r_1^1; \ldots ; r_1^n \leftarrow \Pi_{Y^1}; \ldots ; \Pi_{Y^n} \sigma_{C^1(t)}; \cdots ; \sigma_{C^n(t)} (S_2 \times \cdots \times S_m)$
  for i = 1 to k do
  begin

    add $r_1^i$ to $r^i$ with appropriate padding.
  end;
end.
```

Here C^i (t) means C^i with every occurrence of an attribute $Y_1^i.A$ is replaced by the value $t(Y_1^i.A)$. Y^i contains those attributes in X^i that are not in the scheme of S_1. The appropriate padding for r_1^i is $t(X^i - Y^i)$. That is r^i is extended by that portion of t that is included in X^i.

The query graph is transformed as follows. The node S_1 is removed from the query graph. Any edges that were incident on S_1 become loops.

Example 5:

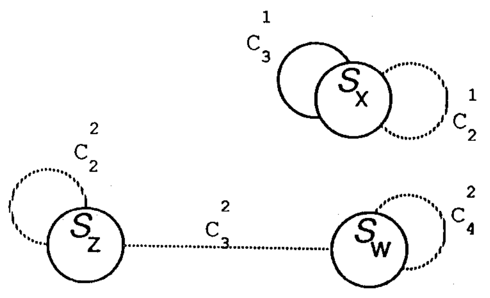
Iterating on S_y, of Figure 3, gives the following transformed program.

```
$r^1 : r^2 \leftarrow \emptyset$;
for each tuple t in $S_y$ do
begin
  $r_1^1 : r_1^2 \leftarrow \Pi_{X^1} : \Pi_{X^2}(\sigma_{C^1(t)} : \sigma_{C^2(t)} (S_x \times S_z \times S_w))$
  add $r_1^1$ to $r^1$ with appropriate padding
  add $r_1^2$ to $r^2$ with appropriate padding
end.
```

where $C^1(t)$ is $C_1^1(t) \wedge C_3^1(t) \wedge C_2^1(t)$ and
$C^2(t)$ is $C_1^2(t) \wedge C_2^2(t) \wedge C_3^2(t) \wedge C_4^2(t)$.

Note that $C_3^1(t)$ and $(C_3^2(t) \wedge C_4^2(t))$ do not contain any attributes of S_y. Hence the tuple t does not participate in the evaluation of these conditions.

The modified query graph after the iteration is shown in Figure



5.

Figure 5.

6.3 Heuristics for Detecting Common Sub-expressions

So far we have extended the transformations, namely, instantiation and iteration in order that they can be applied to decompose a multi-query graph. Using the decomposition algorithm proposed in the next section, one can obtain a single program which computes answers for a set of queries. However, the transformations do not identify and make use of the presence of a common sub-expression in the evaluation of a set of queries. The detection and exploitation of common sub-expressions have to be performed using heuristics to select a node that is appropriate for that purpose. In this section we express the conditions under which common sub-expressions can be detected and present heuristics that can be incorporated into the decomposition algorithm.

We enumerate below conditions that can be used to detect common sub-expressions.

1. A node chosen for instantiation has several edges from different queries incident on it and the conditions associated with all the edges are identical.

2. A node chosen for instantiation has several edges from different queries incident on it and the edges can be partitioned into two groups (each having identical conditions with in the group) such that the condition of one group subsumes (intuitively more general) the condition of the other group.

3. A node chosen for iteration has several edges from different queries incident on it and these edges can be partitioned into sets, where each set consists of edges from the same subset of queries.

4. Whenever, a node chosen for instantiation satisfies 3) and the number of partitions is small (either 1 or 2), then perform the iteration immediately followed by instantiation.

In the above, the first heuristic handles the case where all selections conditions are identical and hence a single temporary relation is computed. The second heuristic groups conditions for instantiation in such a way that only a single temporary relation corresponding to the weaker of the two conditions is computed and later the tuples are separated using a selection using the subsumed condition.

The third heuristic tries to perform the iteration operation for several queries once. The fourth heuristic is important to postpone the selection in favor of iteration to capture a larger sub-expression.

6.4 The Decomposition Algorithm

We describe below an algorithm to decompose a multi-query graph. The algorithm incorporates the heuristics described in the previous section.

Algorithm MQD /* multiple query decomposition */

INPUT: A multi-query graph and a statement corresponding to the multi-query graph.
OUTPUT: A program which computes the relations for the expressions in the multi-query graph.

```
begin
  Repeat
    Choose a node satisfying the lowest numbered option from the
    following set of options.

      1) Choose a node for iteration, if all the edges going to
      other nodes are identical and the edges associated with
      that node itself are not in the option 2). If edges incident
      only on that node satisfy 2), then perform instantiation
      immediately following the iteration operation.

      2) Instantiate a relation that is contained in several one
      node edges, if

        a) the conditions for all edges are identical
        b) the conditions can be grouped into two sets such
        that one set subsumes the conditions of the other set
      If all the edges are simple (an edge is simple, if the condi-
      tion associated with the edge is an equality condition), then
      label the newly generated relations as small. In the above,
      preference is given to a node that has edges incident on it
      from different queries rather than the same query.

      3) When a sub-graph (or a node) has edges belonging to
      only one query, then use the heuristics associated with the
      connection graph decomposition algorithm.

      4) Iterate a "small" relation. Prefer the one having only
      simple edges.

      5) Iterate a relation. Prefer one on simple edges.

    Either instantiate or iterate on the chosen node. Obtain a new
    multi-query graph as well as the corresponding program.
  until (the query graph is a set of isolated nodes)
end
```

6.5 Examples

We illustrate the above algorithm on the following examples to indicate the computation of common sub-expressions. The examples are drawn from Jarke [Jark84].

Example 6:

Let the database consist of the following base relations:
isolated(pname, organism, site, qty)
observed(pname, symptom)
relevant(symptom, site)
surgery(pname, day)

Let the intensional axioms (view definitions) be
infected(pname, site) ← isolated(pname, organism, site, qty)
infected(pname, site) ← observed(pname, symptom), relevant(symptom, site)

Let the query to be evaluated on the above deductive database be

"What patients with monday surgery have wound infections?"

The above query produces the following two disjunctive queries:

Q_1: $\Pi_{y.A}\, \sigma_{C_1^1 \wedge C_2^1 \wedge C_3^1} (S_x \times S_y)$

where C_3^1 is x.C = wound, C_2^1 is y.A = x.A and C_1^1 is y.B = monday and S_x represents the relation isolated and S_y represents the relation surgery. The attributes in each relation are denoted by upper case alphabets such as A, B etc.

Q_2: $\Pi_{y.A}\, \sigma_{C_1^2 \wedge C_2^2 \wedge C_3^2 \wedge C_4^2} (S_y \times S_z \times S_w)$

where C_1^2 is y.B = monday, C_2^2 is y.A = z.A, C_3^2 is z.B = w.A and C_4^2 is w.B = wound. Here S_y represents the relation surgery, S_z represents the relation observed and S_w represents the relation relevant.

The multi-query graph corresponding to the above disjunctive queries is as shown in Figure 3. The statements corresponding to the queries are

$$r^1;r^2 \leftarrow \Pi_{y.A}:\Pi_{y.A}\sigma_{C_1^1 \wedge C_2^1 \wedge C_3^1} : \sigma_{C_1^2 \wedge C_2^2 \wedge C_3^2 \wedge C_4^2} (S_x \times S_y \times S_z \times S_w)$$
$$r \leftarrow r^1 \cup r^2.$$

Selecting the node S_y for instantiation (using option 2), we obtain the following program:

$$r_1^{1,2} \leftarrow \Pi_{y.A} (\sigma_{y.B = \text{monday}} (S_y))$$
$$r^1 ;r^2 \leftarrow \Pi_{y.A} :\Pi_{y.A}\, \sigma_{C_2^1 \wedge C_3^1} : \sigma_{C_2^2 \wedge C_3^2 \wedge C_4^2} (S_x \times r_1^{1,2} \times S_z \times S_w)$$
$$r \leftarrow r^1 \cup r^2.$$

$r_1^{1,2}$ is labeled as a simple relation since all the conditions of instantiation are equality conditions. The transformed graph is shown in Figure 6.

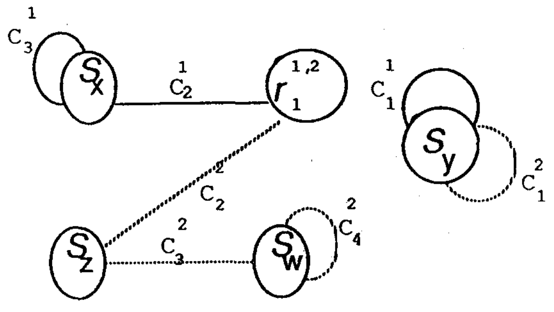


Figure 6.

Iterating on the node $r_1^{1,2}$ (using option 1), we obtain the following program.

```
$r_1^{1,2} \leftarrow \Pi_{y.A}(\sigma_{y.B = \text{monday}}(S_y))$;
$r^1 ; r^2 \leftarrow \emptyset$;
for each tuple t in $r_1^{1,2}$ do

 begin
  $r_2^1 ; r_2^2 \leftarrow \sigma_{C_2^1(t) \wedge C_3^1} : \sigma_{C_2^2(t) \wedge C_3^2 \wedge C_4^2}(S_x \times S_z \times S_w)$;
  add $r_2^1 \times$ <t (y.A)> to $r^1$;
  add $r_2^2 \times$ <t (y.A)> to $r^2$;
 end.
 $r \leftarrow r^1 \cup r^2$.
```

where C_2^1 (t) is x.A = t(y.A) and C_2^2 (t) is z.A = t(y.A). The

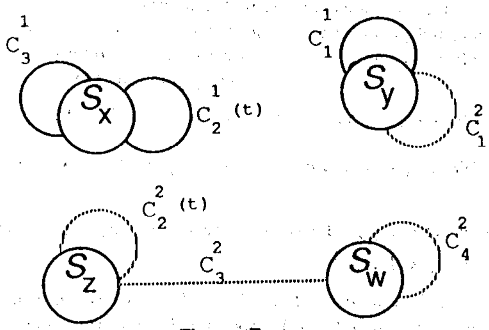


Figure 7.

transformed graph is shown in Figure 7.

Selecting S_x for instantiation (using option 2). we obtain the program

```
$r_1^{1,2} \leftarrow \Pi_{y.A}(\sigma_{y.B = \text{monday}}(S_y))$;
$r^1 ; r^2 \leftarrow \emptyset$;
for each tuple t in $r_1^{1,2}$ do
begin
  $r_3^1 \leftarrow \sigma_{C_2^1(t) \wedge C_3^1}(S_x)$;
  $r_2^1 \leftarrow (r_3^1 \times S_z \times S_w)$;
  $r_2^2 \leftarrow \sigma_{C_2^2(t) \wedge C_3^2 \wedge C_4^2}(S_x \times S_z \times S_w)$;
  add $r_2^1 \times$ <t(y.A)> to $r^1$;
  add $r_2^2 \times$ <t (y.A)> to $r^2$;
end.
$r \leftarrow r^1 \cup r^2$.
```

Note that since there are no conditions associated with the computation of r_2^1, the statement can be removed and the relation r_3^1 can be renamed as r_2^1. We effect this change when we write the next version of this program. The corresponding query graph is shown in Figure 8. Note that r_3^1 is labeled as simple.

Instantiating on S_w (using option 2a), we obtain the following program.

```
$r_1^{1,2} \leftarrow \Pi_{y.A}(\sigma_{y.B = \text{monday}}(S_y))$;
$r^1 ; r^2 \leftarrow \emptyset$;
for each tuple t in $r_1^{1,2}$ do
begin
  $r_2^1 \leftarrow \sigma_{C_2^1(t) \wedge C_3^1}(S_x)$;
  $r_3^2 \leftarrow \Pi_{w.A}(\sigma_{C_4^2}(S_w))$;
  $r_2^2 \leftarrow \sigma_{C_2^2(t) \wedge C_3^2}(S_x \times S_z \times r_3^2)$;
  add $r_2^1 \times$ <t (y.A)> to $r^1$;
  add $r_2^2 \times$ <t (y.A)> to $r^2$;
end.
$r \leftarrow r^1 \cup r^2$.
```

The corresponding query graph is shown in Figure 9.

Iterating on r_3^2, we obtain the following program.

```
$r_1^{1,2} \leftarrow \Pi_{y.A}(\sigma_{y.B = \text{monday}}(S_y))$;
$r^1 ; r^2 \leftarrow \emptyset$;
```

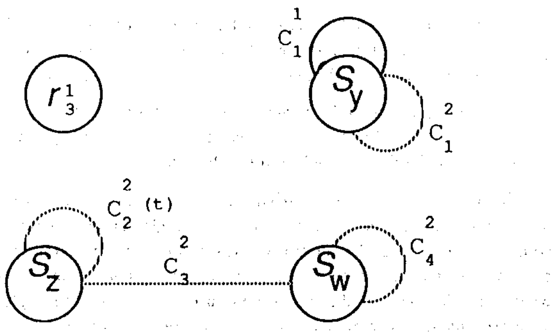


Figure 8.

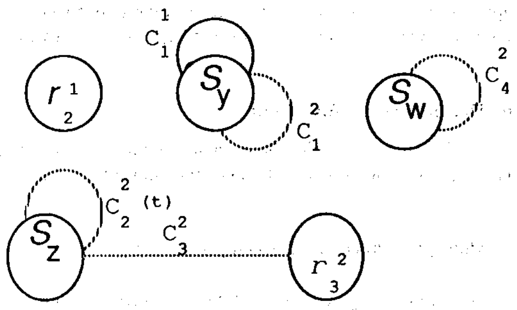


Figure 9.

```
for each tuple t in $r_1^{1,2}$ do
begin
  $r_2^1 \leftarrow \sigma_{C_2^1(t) \wedge C_3^1}(S_x)$;
  $r_3^2 \leftarrow \Pi_{w.A}(\sigma_{C_4^2}(S_w))$;
  $r_2^2 \leftarrow \emptyset$;
  for each tuple u in $r_3^2$ do
  begin
    $r_4^2 \leftarrow \sigma_{C_2^2(t) \wedge C_3^2(u)}(S_x \times S_z)$;
    add $r_4^2$ to $r_2^2$;
  end;
  add $r_2^1 \times$ <t (y.A)> to $r^1$;
  add $r_2^2 \times$ <t (y.A)> to $r^2$;
end.
$r \leftarrow r^1 \cup r^2$.
```

where C_3^2 (u) is z.B = u(w.A). The query graph is shown in Figure 10.

Finally, instantiating on the relation S_z, we obtain the following program.

```
$r_1^{1,2} \leftarrow \Pi_{y.A}(\sigma_{y.B = \text{monday}}(S_y))$;
$r^1 ; r^2 \leftarrow \emptyset$;
for each tuple t in $r_1^{1,2}$ do
begin
  $r_2^1 \leftarrow \sigma_{C_2^1(t) \wedge C_3^1}(S_x)$;
  $r_3^2 \leftarrow \Pi_{w.A}(\sigma_{C_4^2}(S_w))$;
  $r_2^2 \leftarrow \emptyset$;
  for each tuple u in $r_3^2$ do
  begin
    $r_5^2 \leftarrow \sigma_{C_2^2(t) \wedge C_3^2(u)}(S_z)$;
    $r_4^2 \leftarrow (S_x \times r_5^2)$;
    add $r_4^2$ to $r_2^2$;
  end;
  add $r_2^1 \times$ <t (y.A)> to $r^1$;
  add $r_2^2 \times$ <t (y.A)> to $r^2$;
end.
$r \leftarrow r^1 \cup r^2$.
```

The above program can be simplified by removing relations from the cartesian products for which there are no conditions. The final

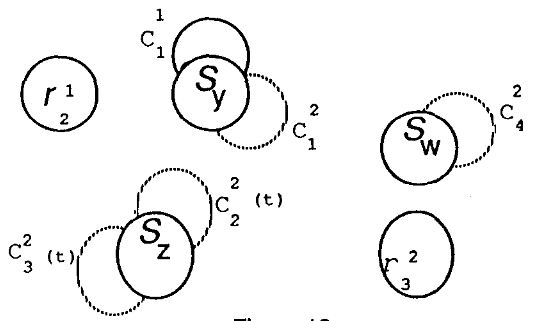


Figure 10.

versions of the program is as follows:

```
$r_1^{1.2} \leftarrow \Pi_{y.A} (\sigma_{y.B = \text{monday}} (S_y))$;
$r^1 ; r^2 \leftarrow \emptyset$;
for each tuple t in $r_1^{1.2}$ do
begin
  $r_2^1 \leftarrow \sigma_{C_2^1(t) \wedge C_3^1} (S_x)$;
  $r_3^2 \leftarrow \Pi_{w.A} (\sigma_{C_4^2} (S_w))$;
  $r_2^2 \leftarrow \emptyset$;
  for each tuple u in $r_3^2$ do
  begin
     $r_5^2 \leftarrow \sigma_{C_2^2(t) \wedge C_3^2(u)} (S_z)$;
     $r_4^2 \leftarrow (r_5^2)$;
     add $r_4^2$ to $r_2^2$;

  end;
  add $r_2^1 \times <t (y.A)>$ to $r^1$;
  add $r_2^2 \times <t (y.A)>$ to $r^2$;
end.
$r \leftarrow r^1 \cup r^2$.
```

In the above program, several computations are performed in common. The relation S_y is accessed only once and the selection on that relation is performed only once. Also, the intermediate relation $r_1^{1.2}$ is scanned only once to compute join with respect two relations S_x and S_z corresponding to different queries.

Apart from the heuristics specifically characterizing the detection and exploitation of common computations, heuristics associated with the decomposition of a connection graph can also be utilized.

Example 7:

Without going into the details, the final program obtained for the the queries of Example 1, using the heuristics described above, is:

```
$r_1^{1.2} \leftarrow \Pi_{y.A} (\sigma_{y.B = \text{monday}} (S_x))$
$r^1 ; r^2 \leftarrow \emptyset$;
for each tuple t in $S_x$ do
begin
   $r_2^1 \leftarrow \Pi_{x.C} (\sigma_{C_2^1(t)} (r_1^{1.2}))$ .
   $r_2^2 \leftarrow \Pi_{x.A} ( \sigma_{C_2^2(t)} (r_1^{1.2}))$ .
   add $r_2^1$ to $r^1$
   add $r_2^2$ times $<t(x.B)>$ to $r^2$
end;
$r^2 \leftarrow \Pi_{x.A} (\sigma_{C_3^2} (r^2)))$
```

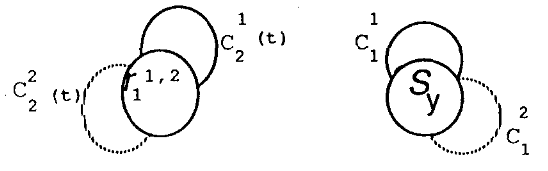


Figure 11.

The transformed graph is shown in Figure 11.

In the above program the larger common sub-expression has been identified and computed. The selection on employee has been postponed to compute the common join.

7. Summary

In this paper we defined a multi-query graph as a means to represent a set of queries in a non-procedural form. We defined instantiation and iteration operations for a multi-query graph. We proposed a set of heuristics and indicated through several examples how common sub-expressions can be detected and exploited using the proposed heuristics. We described an algorithm to generate a single plan from a multi-query graph using a set of heuristics. We believe that the approach presented in this paper can be extended to include a cost model based of the physical representation of relations.

Acknowledgements

We gratefully appreciate support received under NSF grant MCS-83-05992 for U. S. Chakravarthy while he was at the University of Maryland and J. Minker which made this work possible. We wish to thank Dr. John Grant for his comments and valuable suggestions, which helped improve the readability of this paper.

References

[Chak82] U. S. Chakravarthy and J. Minker, Processing Multiple Queries in Database Systems, *database Engineering* **5**, 3, pp. 38-44, Sept 1982.

[Chak85] U. S. Chakravarthy and J. Minker, Multiple Query Processing in Deductive Databases, Tech. Report 1554, University of Maryland, College Park, Aug 1985.

[Gran80] J. Grant and J. Minker, Optimization in Deductive and Conventional Relational Database Systems, pp. 195-234 in *Advances in Database theory, Vol. 1*, eds. H. Gallaire, J. Minker and J. M. Nicolas, Plenum Press, 1980.

[Jark84] M. Jarke, Common Subexpression isolation in Multiple Query Optimization, pp. 65-85 in *Artificial intelligence Applications for Business*, ed. W. Reitman, Ablex, Norwood, 1984.

[Kim80] W. Kim, Query Optimization for Relational Database Systems, Ph.D Thesis, Univ. of Illinois at Urbana-Champaign, 1980.

[Maie83] D. Maier, *The Theory of Relational databases*, Computer Science Press, 1983.

[Ullm82] J. D. Ullman, *Principles of database Systems, 2nd Edition*, Computer Science Press, 1982.

[Wong76] E. Wong and K. Youssefi, Decomposition - A Strategy for Query Processing, *ACM TODS 1:3*, September 1976, pp 223-241.