SQL Query Optimization: Reordering for a General Class of Queries

Piyush Goel
IBM Santa Teresa Laboratory
pgoel@vnet.ibm.com

Bala Iyer
IBM Santa Teresa Laboratory
bala@vnet.ibm.com

Abstract

The strength of commercial query optimizers like DB2 comes from their ability to select an optimal order by generating all equivalent reorderings of binary operators. However, there are no known methods to generate all equivalent reorderings for a SQL query containing joins, outer joins, and groupby aggregations. Consequently, some of the reorderings with significantly lower cost may be missed. Using hypergraph model and a set of novel identities, we propose a method to reorder a SQL query containing joins, outer joins, and groupby aggregations. While these operators are sufficient to capture the SQL semantics, it is during their reordering that we identify a powerful primitive needed for a dbms. We report our findings of a simple, yet fundamental operator, *generalized selection*, and demonstrate its power to solve the problem of reordering of SQL queries containing joins, outer joins, and groupby aggregations.

1 Introduction

1.1 Motivation

The current state-of-the-art in query optimization has few results for optimizing query expressions involving GROUP BYs, joins, outer joins, and full outer joins specified with predicates that either reference more than two relations or reference columns generated by aggregations. To the best of our knowledge, the existing algorithms are unable to exhaustively reorder SQL queries containing aggregations, complex outer join predicates and outer join predicates that reference columns generated by aggregations. Bhargava et. al. [BHAR94, BHAR95a] have proposed algorithms to reorder queries containing outer join predicates that reference two or more relations, but their work can not reorder the following:

- **Partial reordering:** Algorithms (we know) for reordering outer and inner joins require that binary operations (joins, outer joins and full outer joins) be brought adjacent to each other in the query expression [BHAR94, BHAR95a]. Gupta et. al. in [GUPT95] and Bhargava et. al. in [BHAR95d] presented algorithms to "push-up/push-down" projections, aggregations and selections in such a way that the binary operations appear adjacent to each other, without intervening unary operators. However, if an outer join predicate references a column that is generated by an aggregation operator, known algorithms do not tell us how to reorder the binary relations and assign operators to produce the same result. For example, let r_1, r_2, r_3 and r_4 be four relations such that attributes $r_1.b$ and $r_1.c$ are in relation r_1; attributes $r_2.b$ and $r_2.d$ are in relation r_2; attributes $r_3.a$ and $r_3.b$ are in relation r_3; and attribute $r_4.b$ is in relation r_4. Consider the following Query 1 in which outer join predicate ($r_3.b\ \theta_2\ V_1.c$) references column c generated by aggregation operator *count*:

View V_1: Select $r_1.c$ as a, $r_2.d$ as b, $c = count(r_1)$
From r_1, r_2
Where $r_1.b\ \theta_1\ r_2.b$
Groupby $r_1.c$, $r_2.d$

Query 1: Select $r_3.a$, $r_4.b$, $V_1.b$
From (Select * from V_1 LeftOuterJoin r_3 On $r_3.b\ \theta_2\ V_1.c$), r_4
Where $r_4.b = V_1.b$

where $\theta_1, \theta_2 \in \{=, \neq, \geq, \leq, <, >\}$. Because of column c which is generated by aggregation operator *count*, and no generally known definition of outer joins based on cartesian product, view V_1 cannot be merged with the rest of Query 1. Consequently, the existing algorithms in [BHAR95a, BHAR95d] are unable to reorder the outer and inner joins specified in the query. Specifically, the join specified in view V_1 cannot be reordered with other outer and inner

SIGMOD '96 6/96 Montreal, Canada

joins specified in Query 1 – if predicate $r_4.b = V_1.b$ is highly filtering then it may be beneficial to perform this join first, before performing the aggregation that counts the tuples obtained after combining relations r_1 and r_2. A motivating example is included next.

Example 1.1 Consider three relations 94AGG, 95DETAIL, and SUP_DETAIL, where relation 94AGG contains aggregated information about different parts supplied by different suppliers in the year 1994, relation 95DETAIL contains detailed information about every transaction recorded so far in the year 1995, and relation SUP_DETAIL contains information about each supplier. Let us assume that relation 94AGG has three attributes $SUPKEY$, $PARTKEY$ and QTY, where attribute $SUPKEY$ contains unique identifiers of suppliers, attribute $PARTKEY$ contains unique identifiers of parts, and attribute QTY contains the total quantity of a part supplied by a supplier. Also, let relation 95DETAIL contain four attributes $SUPKEY$, $PARTKEY$, $DATE$ and QTY, where attribute $DATE$ contains the date of a transcation, and the remaining attributes have similar meanings as in relation 94AGG. Let relation SUP_DETAIL contain three attributes $SUPKEY$, $SUPRATING$, and $SUPDETAIL$, where attribute $SUPKEY$ contains unique identifier for every supplier, attribute $SUPRATING$ contains the rating of a supplier, and attribute $SUPDETAIL$ contains detailed information about a supplier. Typically, relation 94AGG is relatively small in comparison to relation 95DETAIL.

Suppose a business analyst is interested in identifing those suppliers that are to be discontinued from the list of approved suppliers, either because the supplier is no longer a reliable supplier and/or the volume with the supplier is dropping. In order to generate this list, the business analyst could define the following views and pose the following query on them:

View V_2: Select a.$SUPKEY$ as $SUPKEY$,
a.QTY as QTY, a.$PARTKEY$ as $PARTKEY$,
From 94AGG a, SUP_DETAIL b
Where a.$SUPKEY$ = b.$SUPKEY$ and
b.$SUPRATING$ = "BANKRUPT";

View V_3: Select $SUPKEY$, $PARTKEY$,
95$AGGQTY$ = $COUNT(*)$
From 95DETAIL
GROUPBY $SUPKEY$, $PARTKEY$;

Query: Select $V_2.SUPKEY$, $V_2.PARTKEY$, $V_2.QTY$,
V_3.95$AGGQTY$
FROM V_2 LeftOuterJoin V_3 On
($V_2.SUPKEY = V_3.SUPKEY$ and
$V_2.PARTKEY = V_3.PARTKEY$ and
$V_2.QTY < 2 * V_3$.95$AGGQTY$);

This query, if executed as written would entail that the aggregation on the 95DETAIL table be done before it can participate in the outer join. This aggregation may potenial reduce the cardinality. However, if very few tuples are selected from 94AGG by predicate (a.$SUPKEY$ = b.$SUPKEY$ and b.$SUPRATING$ = "BANKRUPT") then it may be beneficial to first combine relations 94AGG and SUP_DETAIL with relation 95DETAIL, before performing the aggregation on relation 95DETAIL. This may generate a better plan specially if there is an index in relation 95DETAIL on attributes SUPKEY and PARTKEY. Thus, the reduction of cardinality through grouping can be used as a good alternative to the potenial reduction through join.

- **Join-aggregate queries:** The motivation, examples and issues with a class of queries called "Join Aggregate Queries" has been discussed in [GANS87, MURA92] and references therein. To the best of our knowledge, the majority of commercial RDBMS execute the *join-aggregate queries* by employing Tuple Iteration Semantics (TIS). For example, consider the following co-related query in which r_1, r_2, and r_3 denote three relations such that $r_1.a$, $r_1.b$, $r_1.c$ and $r_1.f$ are attributes in relation r_1; $r_2.c$, $r_2.d$ and $r_2.e$ are attributes in relation r_2; $r_3.e$ and $r_3.f$ are attributes in relation r_3.

Select $r_1.a$ From r_1 Where $r_1.b$ θ_1
(Select count(*) From r_2 Where $r_2.c = r_1.c$ and
$r_2.d$ θ_2
(Select count(*) From r_3 Where $r_2.e = r_3.e$ and
$r_1.f = r_3.f$))

where $\theta_1, \theta_2 \in \{=, \neq, \geq, \leq, <, >\}$. In tuple iteration semantics, first, every tuple in r_1 is combined with tuples in r_2 by applying predicate $r_2.c = r_1.c$. Next, each selected tuple in relation r_2 (along with the tuple in r_1) is substituted in predicate $r_2.e = r_3.e$ and $r_1.f = r_3.f$ in order to select and count the selected tuples in relation r_3. This process is repeated for every tuple in R_1 and may result in a very inefficient "nested-loops" like processing strategy.

Ganski [GANS87] and Murlikrishna [MURA92] have proposed an algorithm to unnest the above query and to transform it into the following two queries that employ outer joins and do not require TIS. Let $r_1.key$, $r_2.key$ and $r_3.key$ denote key attributes of relations r_1, r_2 and r_3, respectively.

Query 2: Select into TEMP
$r_1.key$, $r_1.a$, $r_2.key$, $r_2.b$
From (Select * from r_1 LeftOuterJoin r_2 On $r_1.c = r_2.c$),
r_3 LeftOuterJoin $r_2.e = r_3.e$ and $r_1.f = r_3.f$
Groupby $r_1.key$, $r_1.a$, $r_2.key$, $r_2.b$
Having $r_2.d$ θ_2 $count(r_3.key)$

Query 3. Select $r_1.a$
From TEMP
Groupby $r_1.key$
Having $r_1.b$ θ_1 $count(r_2.key)$

However, the algorithms in [BHAR95a, BHAR95b] cannot reorder the left outer joins specified in Query 2 because of complex predicate $r_2.e = r_3.e$ and $r_1.f = r_3.f$ – if the cardinality of relation r_1 is very large then it may be beneficial to combine relations r_2 and r_3 first, before accessing relation r_1.

In this paper, we introduce a new operator, *Generalized selection (GS)*, that enables us to generate different schedules for queries similar to Query 2 and 3. For example in query 2, this allows us to generate schedules in which relations r_4 and r_1 are joined first. Consequently, if this join is very selective then this join may substantially reduce the cost of the query. Further, this new operator facilitates complete enumeration of those queries that contain only conjunctive predicates with binary operations.[1]. Through out this paper, we assume that queries have been simplified by the method presented in [BHAR95b, BHAR95c] so that they do not contain any redundant (full) outer join edges; that is, we assume queries are *simple* [GALI92a, GALI92b].

The rest of the paper is organized as follows. Section 1.2 provides some basic definitions. In Section 2, we provide the definition of the new operator. In Section 3, we provide association identities that enable us to "break-up" complex predicates. In Section 4, we provide an outline of the enumeration process. Finally, Section 5 provides the conclusion.

1.2 Basic definitions

From now on, formal algebraic syntax is used in lieu of SQL, albeit the algebraic syntax is constructed to convey SQL semantics. We restrict our attention to the more pertinent definitions – a complete set of definitions are provided in [BHAR95a], with the intent of capturing the SQL semantics. Let $r_1 = \langle R_1, V_1, E_1 \rangle$ and $r_2 = \langle R_2, V_2, E_2 \rangle$ denote two relations such that $R_1 \cap R_2 = \phi$, where R_1 and R_2 denote the schemas (sets containing column names or, equivalently, attribute names), V_1 and V_2 are virtual attributes (row identifiers is a way to conceptualize it) and E_1 and E_2 denote the *extensions* of relations r_1 and r_2, respectively. We employ $sch(r)$ to denote the schema of relation r and $sch(p)$ to denote the schema of predicate p. We assume that the predicates specified with joins reference at least one relation each from both the operand relations. If, a predicate specified with a join operator references exactly two relations then, it is referred to as a *simple predicate*; if the predicate references more than two relations then it is referred to as *complex predicate*. Through out this paper, we assume that predicates are *null in-tolerant*[2].

We represent SQL **groupby** statement, using the terminology employed in [GUPT95] (which also cites earlier work), by a *Generalized Projection* (GP), $\pi_{X,f(Y)}$, which accepts as its argument a relation r and produces a new relation according the subscripts X and $f(Y)$ [GUPT95].

1. Subscript X specifies the attributes referenced in the groupby statement. For instance, GP $\pi_X(r)$ represents SQL statements SELECT X FROM r GROUPBY X or, equivalently, SELECT DISTINCT X statement. Note that this GP $\pi_X(r)$ has no aggregate function specified with it.
2. Subscript $f(Y)$ specifies the aggregation (if present). For example, GP $\pi_{X,count(Y)}(r)$ represents the SQL statement SELECT X, count(Y) FROM r GROUPBY X.

Some aggregate functions are duplicate-insensitive, eg. **max**, **min**, **count(distinct)**, etc. If a GP either represents the SELECT DISTINCT statement or contains a duplicate-insensitive function then such a GP is termed as *duplicate-insensitive* GP and is denoted by $\delta_{X,f(Y)}$, where $f \in \{max, min, count(distinct), avg(distinct), sum(distinct)\}$. Whenever there is no confusion, we employ π to denote both δ and π.

The *union*, $r_1 \cup r_2$, of two union compatible relations r_1 and r_2 is the relation $\langle R, V, E_1 \cup E_2 \rangle$. The *outer union*, $r_1 \uplus r_2$, of relations r_1 and r_2 is the relation $\langle R_1 \cup R_2, V_1 \cup V_2, E' \rangle$, where

$$
\begin{aligned}
E' = \{t \mid & (\exists t' \in E_1)(t[R_1 V_1] = t' \wedge \\
& (\forall A \in (R_2 - R_1) \cup (V_2 - V_1))(t[A] = \mathsf{NULL})) \\
\vee \; & (\exists t'' \in E_2)(t[R_2 V_2] = t'' \wedge \\
& (\forall A \in (R_1 - R_2) \cup (V_1 - V_2))(t[A] = \mathsf{NULL}))\}.
\end{aligned}
$$

Note, rows in $r_1 \uplus r_2$ are padded with nulls for attributes that are not present either in relation r_1 or in relation r_2.

[1] We assume that predicates specified with binary operations are conjunctive, that is, in expression $r_1 \overset{p}{\odot} r_2$, predicate p is of the form $p = p_1 \wedge p_2 \wedge \ldots \wedge p_n$, where $\odot$ denotes either join or full/left/right outer join operator.

[2] A predicate p over a set of real attributes $sch(p)$, called the *schema* of p, is *null in-tolerant* in attributes $R \subseteq sch(p)$ if p evaluates to FALSE for tuples that have NULL values in any attribute in R [BHAR94].

The *anti join*, $r_1 \overset{p}{\triangleright} r_2$, of relations r_1 and r_2 is the relation $\langle R_1, V_1, E' \rangle$, where p is a predicate and extension E' is a set containing unmatched tuples in relation r_1 (with respect to predicate p). The *left outer join*, $r_1 \overset{p}{\rightarrow} r_2$, of relations r_1 and r_2 is the relation $\langle R_1 R_2, V_1 V_2, E' \rangle$, where extension E' is the union of relations $r_1 \overset{p}{\bowtie} r_2$ and $r_1 \overset{p}{\triangleright} r_2$. In relation $r_1 \overset{p}{\rightarrow} r_2$, tuples from relation $r_1 \overset{p}{\triangleright} r_2$ are padded with nulls for the attributes in $sch(r_2)$. Relation r_1 is called the *preserved relation* and relation r_2 is called the *null supplying relation*. The *right outer join*, $r_1 \overset{p}{\leftarrow} r_2$, can similarly be defined in which r_1 is the null supplying relation and r_2 is the preserved relation. Similarly, in the *full outer join*, $r_1 \overset{p}{\leftrightarrow} r_2$, of relations r_1 and r_2, the extension set is the union of relations $r_1 \overset{p}{\bowtie} r_2$, $r_1 \overset{p}{\triangleright} r_2$ and $r_2 \overset{p}{\triangleright} r_1$, with tuples appropriately padded with nulls.

2 Generalized selection

In this section we define a novel operator *generalized selection* (GS) which enables us to perform complete reordering of queries containing complex predicates and predicates that are specified on columns generated by aggregations. For example, consider expression $(r_1 \overset{p_{1,2}}{\rightarrow} r_2) \overset{p_{1,3} \wedge p_{2,3}}{\rightarrow} r_3$ corresponding to Query 2 in Section 1.1, where $p_{1,2} \equiv (r_1.c = r_2.c)$, $p_{1,3} \equiv (r_1.f = r_3.f)$ and $p_{2,3} \equiv (r_2.e = r_3.e)$. Since $(r_1 \overset{p_{1,2}}{\rightarrow} r_2) \overset{p_{1,3} \wedge p_{2,3}}{\rightarrow} r_3 \neq (r_1 \overset{p_{1,3}}{\rightarrow} r_3) \overset{p_{1,2} \wedge p_{2,3}}{\rightarrow} r_2$ and $(r_1 \overset{p_{1,2}}{\rightarrow} r_2) \overset{p_{1,3} \wedge p_{2,3}}{\rightarrow} r_3 \neq (r_1 \overset{p_{1,2} \wedge p_{1,3}}{\rightarrow} (r_2 \overset{p_{2,3}}{\rightarrow} r_3))$, if only joins, outer and full outer joins are employed then this query can only be executed in the manner in which it is specified. This follows from the fact that complex predicates specified with any outer join cannot be broken up and applied with some other join operator. Reordering for this query and other similar queries motivates the definition of the *generalized selection* (GS), σ^*, that enables us to break-up complex predicates. This operator facilitates generation of other plans such as $\sigma^*_{p_{2,3}}[r_1 r_2]((r_1 \overset{p_{1,3}}{\rightarrow} r_3) \overset{p_{1,2}}{\rightarrow} r_2)$ and $\sigma^*_{p_{1,3}}[r_1 r_2](r_1 \overset{p_{1,2}}{\rightarrow} (r_2 \overset{p_{2,3}}{\rightarrow} r_3))$.

Intuitively, generalized selection operator $\sigma^*_p[r_1](r)$, where $r_1 \subseteq r$, is similar to the conventional selection operator $\sigma_p(r)$ – it applies predicate p to relation r and retains those tuples in relation r that satisfy predicate p. In addition, it also retains those tuples in relation r_1 that are not selected by predicate p, appropriately padding tuples in r_1 with nulls for attributes in $r - r_1$. In general, we may specify more than one relation to be preserved in a generalized selection operator, as formally defined next.

Let $r = \langle R, V, E \rangle$, and $r_i = \langle R_i, V_i, E_i \rangle$, $1 \leq i \leq n$, be relations (not necessarily base relations) such that p denote a null-intolerant predicate in R, $R_i \subseteq R$, $R_i \cap R_j = \phi$ and $V_i \cap V_j = \phi$, where $i \neq j$, then:

Definition 2.1 The *generalized selection* (GS), $\sigma^*_p[r_1, r_2, \ldots, r_n](r)$, of relation r that preserves relations r_i, $1 \leq i \leq n$, is the relation $\langle R, V, E' \rangle$, where E' is given by:

$$E' = \sigma_p(r) \uplus_{1 \leq i \leq n} \{\pi_{R_i V_i}(r) - \pi_{R_i V_i}(\sigma_p(r))\}$$

In the above definition of *generalized selection* (GS), only those relations are preserved that are specified relations – eg. in expression $\sigma^*_{p_{2,3}}[r_1 r_2]((r_1 \overset{p_{1,3}}{\rightarrow} r_3) \overset{p_{1,2}}{\rightarrow} r_2)$ only one relation, $r_1 r_2$, is preserved. Also, note that relation $r_1 r_2$ is not a base relation. If no relation is to be preserved then the resultant relation is obtained by applying predicate p to relation r.

Example 2.1 Consider the following relations, shown here with only their real attributes:

r_1:

a	b	c	f
a_1	b_1	c_1	f_1
a_2	b_1	c_1	f_2
a_2	b_1	c_2	f_2

r_2:

c	d	e
c_1	d_1	e_1

r_3:

e	f
e_1	f_1
e_1	f_3

Consider expression $(r_1 \overset{p_{1,2}}{\rightarrow} r_2) \overset{p_{1,3} \wedge p_{2,3}}{\rightarrow} r_3$ corresponding to Query 2 in Section 1.1 with predicates as defined earlier. The following table provides the evaluation of this expression for the above tables:

T_1:

a	b	c	f	c	d	e	e	f
a_1	b_1	c_1	f_1	c_1	d_1	e_1	e_1	f_1
a_2	b_1	c_1	f_2	c_1	d_1	e_1		
a_2	b_1	c_2	f_2					

Now, consider expression $(r_1 \overset{p_{1,2}}{\rightarrow} r_2) \overset{p_{2,3}}{\rightarrow} r_3$ which evaluates to the following table:

T_2:

a	b	c	f	c	d	e	e	f
a_1	b_1	c_1	f_1	c_1	d_1	e_1	e_1	f_1
a_2	b_1	c_1	f_2	c_1	d_1	e_1	e_1	f_3
a_2	b_1	c_2	f_2					

Notice that the second tuple in table T_2 contains non-null values in attributes e and f whereas the second tuple in table T_1 contains null values in these attributes. By applying the generalized selection operator $\sigma^*_{p_{1,3}}[r_1 r_2]$ at

the top, we compensate for this difference and generate the correct result corresponding to expression $(r_1 \overset{p_{1,2}}{\rightarrow} r_2) \overset{p_{1,3} \wedge p_{2,3}}{\rightarrow} r_3$, as given in table T_1.

By appropriately selecting the preserved relations in a generalized selection operator, the inner, outer and full outer join operators can be defined as a generalized selection on the cartesian product as follows:

$$\begin{aligned} r_1 \overset{p}{\bowtie} r_2 &= \sigma^*_p[](r_1 \times r_2) \\ r_1 \overset{p}{\rightarrow} r_2 &= \sigma^*_p[r_1](r_1 \times r_2) \\ r_1 \overset{p}{\leftrightarrow} r_2 &= \sigma^*_p[r_1, r_2](r_1 \times r_2) \end{aligned}$$

3 Deferred application of complex predicates

The reordering of queries that contain aggregations and predicates referencing aggregated columns requires moving aggregations and associated predicates to the top. This push-up of aggregation may require breaking up of a complex predicate, as shown in the following example.

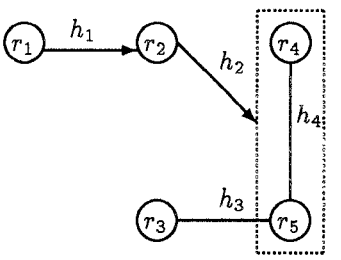
Example 3.1 Consider the following expression:

$$r = \pi_{r_1' r_2', c=count(r_1)}(r_1 \overset{p_{1,2}}{\rightarrow} r_2) \overset{p_{1,3} \wedge p_{2,3}}{\rightarrow} r_3$$

where $r_1' \subseteq r_1$ and $r_2' \subseteq r_2$. Let column c only be referenced in predicate $p_{1,3}$ and not in predicate $p_{2,3}$. Expression r cannot be reordered due to aggregation *count*. In order to reorder expression r, we can move the aggregation to the top and break-up the complex predicate $p_{1,3} \wedge p_{2,3}$. Since, column c is generated by aggregation and not in the base relations, predicate $p_{1,3}$ can no longer be applied with the outer join. The modified expression is $\pi_{V_3 r_3 r_1' r_2', c=count(r_1)}((r_1 \overset{p_{1,2}}{\rightarrow} r_2) \overset{p_{2,3}}{\rightarrow} r_3)$ which is completely reorderable by the algorithms presented in [BHAR95a, GALI92a, GALI92b]. For each reordering of the modified expression, we can generate the correct result by applying the remaining predicate $p_{1,3}$ at the top – eg. $\sigma'_{p_{1,3}}[r_1 r_2](\pi_{V_3 r_3 r_1' r_2', r1\ c=count(r_1)}((r_1 \overset{p_{1,2}}{\rightarrow} r_2) \overset{p_{2,3}}{\rightarrow} r_3))$, as shown in this section.

Further, even if a query does not contain predicates that reference aggregated columns, it is desirable to break-up the complex predicate in order to generate a complete set of reorderings. In order to show this, we employ the *hypergraph* model presented earlier in [BHAR95a] and *association trees*. Intuitively, the set of nodes of the hypergraph correspond to relations referenced in the query, and a *hyperedge* represents an (inner, outer, or full outer) join operation involving a predicate between the sets of relations in its two *hypernodes*, as follows.

Definition 3.1 A *hypergraph* H is defined to be the pair (V, E), where V is a non-empty set of *nodes* and



$$H = \langle \{r_1, r_2, r_3, r_4, r_5\}, \{h_1, h_2, h_3, h_4\} \rangle$$

Figure 1: Hypergraph for Example 3.2.

E is the set of *hyperedges*, such that E is a mapping on non-empty subsets of V (i.e., $E : 2^V \rightarrow 2^V$). For hyperedge $e = \langle V_1, V_2 \rangle$, where $V_1, V_2 \in 2^V$, we call V_1 and V_2 *hypernodes*.

If $e = \langle V_1, V_2 \rangle$ and the cardinalities of both V_1 and V_2 are 1, we refer to hyperedge e as simply an *edge* and the hypernodes V_1 and V_2 as simply *nodes*. In the context of query hypergraphs, we say a hyperedge is *directed* if it represents an outer join operation in the query. Further, we say a hyperedge is *bi-directed* if it represents a full outer join operation in the query. Note, predicates specified with inner joins always generate undirected edges.

Example 3.2 Consider the following query Q_4:

$$Q_4 = r_1 \overset{p_{1,2}}{\rightarrow} (r_2 \overset{p_{2,4} \wedge p_{2,5}}{\rightarrow} ((r_4 \overset{p_{4,5}}{\bowtie} r_5) \overset{p_{3,5}}{\bowtie} r_3))$$

where $r_i = \langle R_i, V_i, E_i \rangle$, for $1 \leq i \leq 5$, is a relation and p_{ij} is a predicate between r_i and r_j, for $1 \leq i, j \leq 5$. Figure 1 shows the hypergraph for this query. Note that this hypergraph has no cycles (even though there are 3 paths[3] — $r_1 h_1 r_3 h_3 r_5$, $r_1 h_1 r_2 h_2 r_5$, and $r_1 h_1 r_2 h_2 r_4 h_4 r_5$ — between nodes r_1 and r_5). Note, for directed hyperedge h_2, the hypernodes are $\{r_2\}$ and $\{r_4, r_5\}$. If we break complex predicate $p_{2,4} \wedge p_{2,5}$ then we obtain new expressions $Q_4^1 = (r_1 \overset{p_{1,2}}{\rightarrow} r_2) \overset{p_{2,5}}{\rightarrow} ((r_4 \overset{p_{4,5}}{\bowtie} r_5) \overset{p_{3,5}}{\bowtie} r_3))$ and $Q_4^2 = (r_1 \overset{p_{1,2}}{\rightarrow} r_2) \overset{p_{2,4}}{\rightarrow} ((r_4 \overset{p_{4,5}}{\bowtie} r_5) \overset{p_{3,5}}{\bowtie} r_3))$. We intend to show that expression Q_4 is equivalent to $\sigma^*_{p_{2,4}}[r_1 r_2](Q_4^1)$ and $\sigma^*_{p_{2,4}}[r_1 r_2](Q_4^2)$.

A query hypergraph is a useful abstraction for representing predicate connectivity, as well as the semantic information regarding outer join predicates that reference more than two relations and, therefore, *cannot* be broken up into smaller components using the known operators and identities [BHAR95a, GALI92a, GALI92b]. For a given query, we can construct its hypergraph and

[3] Please refer to [BHAR95a] for the precise definition.

then determine those reorderings of the operations that satisfy the connectivity criterion, as defined in the following. Intuitively, an association tree of a hypergraph specifies the order in which relations are combined – eg. $(r_1.r_2).((r_4.r_5).r_3))$ is an association tree for Q_4, $(r_1.((r_2.r_5).(r_4.r_3)))$ is an association tree for Q_4^1, and $(r_1.((r_2.r_4).(r_5.r_3)))$ is an association tree for Q_4^2. Note, the definition in [BHAR95a] requires that relation r_2 should be combined only after combining relations r_4 and r_5[4] – eg. $(r_1.r_2).((r_4.r_5).r_3))$. The following definition allows more association trees by permitting association trees for Q_4^1 and Q_4^2 as valid association trees for Q_4.

Definition 3.2 For a query hypergraph $H = (V, E)$, we define an *association tree* T for H as a binary tree in which.

1. $leaves(T) = V$, and no relation appears in more than one leaf[5].
2. for any subtree T_s of T, $H\,|_{leaves(T_s)}$[6] is connected.
3. for any subtree $T_s = (T_l.T_r)$ of T, let E_{T_s} denotes the set of all hyperedges in E that connect leaves of T_l with leaves of T_r, and let E'_{T_s} denote all the hyperedges used in subtree T_s in order to combine subtrees T_l and T_r. Then $\forall e = \langle V_1, V_2\rangle \in E'_{T_s}$, $V_1 \subseteq leaves(T_l)$ and $V_2 \subseteq leaves(T_r)$, or $V_2 \subseteq leaves(T_l)$ and $V_1 \subseteq leaves(T_r)$; and either of the following is true:
   - $e \in E_{T_s}$; or
   - $\exists \langle V_1', V_2'\rangle \in E_{T_s}$ such that either $V_1 \subseteq V_1'$ and $V_2 \subseteq V_2'$ or $V_1 \subseteq V_2'$ and $V_2 \subseteq V_1'$.

The key difference between the above definition and the definition in [BHAR95a] is Item 3, which states that a hyperedge can be broken-up in such way that only the subsets of hypernodes (corresponding to the original hyperedge) need be combined, before the two subsets are combined. For example, hyperedge $h_2 = \langle\{r_2\}, \{r_4, r_5\}\rangle$ in query Q_4 can be broken-up so that relation r_2 can be combined with either relation r_4 or r_5. That is, the above definition allows association trees for Q_4^1 and Q_4^2 as valid association trees for query Q_4 – some of the valid association trees for Q_4 are: $((r_1.r_2).((r_4.r_5).r_3))$, $((r_1.r_2).((r_4.(r_5.r_3))$, $(r_1.((r_2.r_4).(r_5.r_3)))$, $(r_1.((r_2.r_5).(r_4.r_3)))$, etc.

[4] Definition 2.3 in [BHAR95a].

[5] As in SQL and other optimization literature, we assume that relations occurring more than once in a query expression have been renamed to have unique names.

[6] We use $H\,|_{leaves(T_s)}$ to denote the induced sub-hypergraph $(leaves(T_s), E')$, where $E' = \{\langle V_1, V_2\rangle | V_1 \subseteq leaves(T_s) \wedge V_2 \subseteq leaves(T_s) \wedge (\langle V_1, V_2\rangle \in E \vee (\exists \langle V_1', V_2'\rangle \in E\ V_1 \subseteq V_1' \wedge V_2 \subseteq V_2'))\}$.

The association trees only determine the order of execution for different operations; they do not contain operators. Given association trees, the next step is to assign operators to the internal nodes in order to generate expression trees. Then, the optimal expression tree with the minimal cost can be selected. We also use the process of assigning operators as a mean to establish the equivalence of the association trees. First, we consider expression trees that contain at most one complex predicate, and then generalize our results to expression trees containing any number of complex predicates.

3.1 Splitting complex predicates

Initially, assume that a given expression contains at most one complex predicate. Our approach consists of transforming the given expression into an equivalent expression in which the complex predicate appears either at the root or with the child or grand child of the root. If the complex predicate appears at the root or with the child or grand child of the root, then the following identities can be used to break-up the complex predicate. Let $r_i = \langle R_i, V_i, E_i\rangle$, where $1 \le i \le 4$, be four relations, $p_{i,j}$ denote the conjunctive predicate between relations r_i and r_j, then[7]:

$$r_1 \overset{p^1_{1,2} \wedge p^2_{1,2}}{\rightarrow} r_2 \quad = \sigma^*_{p^1_{1,2}}[r_1](r_1 \overset{p^2_{1,2}}{\rightarrow} r_2) \tag{1}$$

$$r_1 \overset{p^1_{1,2} \wedge p^2_{1,2}}{\leftrightarrow} r_2 \quad = \sigma^*_{p^1_{1,2}}[r_1, r_2](r_1 \overset{p^2_{1,2}}{\leftrightarrow} r_2) \tag{2}$$

$$(r_1 \overset{p_{1,2}}{\odot} r_2) \overset{p_{1,3} \wedge p_{2,3}}{\rightarrow} r_3 = \sigma^*_{p_{1,3}}[r_1 r_2]((r_1 \overset{p_{1,2}}{\odot} r_2) \overset{p_{2,3}}{\rightarrow} r_3) \tag{3}$$

$$(r_1 \overset{p_{1,2}}{\odot} r_2) \overset{p_{1,3} \wedge p_{2,3}}{\leftrightarrow} r_3 = \sigma^*_{p_{1,3}}[r_1 r_2, r_3]((r_1 \overset{p_{1,2}}{\odot} r_2) \overset{p_{2,3}}{\leftrightarrow} r_3) \tag{4}$$

$$r_1 \overset{p_{1,2}}{\rightarrow} (r_2 \overset{p^1_{2,3} \wedge p^2_{2,3}}{\bowtie} r_3) = \sigma^*_{p^1_{2,3}}[r_1](r_1 \overset{p_{1,2}}{\rightarrow} (r_2 \overset{p^2_{2,3}}{\bowtie} r_3)) \tag{5}$$

$$r_1 \overset{p_{1,2}}{\leftrightarrow} (r_2 \overset{p^1_{2,3} \wedge p^2_{2,3}}{\bowtie} r_3) = \sigma^*_{p^1_{2,3}}[r_1, r_2 r_3](r_1 \overset{p_{1,2}}{\leftrightarrow} (r_2 \overset{p^2_{2,3}}{\bowtie} r_3)) \tag{6}$$

$$r_1 \overset{p_{1,2}}{\leftrightarrow} (r_2 \overset{p^1_{2,3} \wedge p^2_{2,3}}{\leftarrow} r_3) = \sigma^*_{p^1_{2,3}}[r_1, r_3](r_1 \overset{p_{1,2}}{\leftrightarrow} (r_2 \overset{p^2_{2,3}}{\leftarrow} r_3)) \tag{7}$$

$$r_1 \overset{p_{1,2}}{\leftrightarrow} ((r_2 \overset{p^1_{2,3} \wedge p^2_{2,3}}{\bowtie} r_3) \overset{p_{2,4}}{\leftarrow} r_4) = \sigma^*_{p^1_{2,3}}[r_1, r_4](r_1 \overset{p_{1,2}}{\leftrightarrow} ((r_2 \overset{p^2_{2,3}}{\bowtie} r_3) \overset{p_{2,4}}{\leftarrow} r_4)) \tag{8}$$

[7] In general, predicate $p^1_{i,j} = \wedge_{x=1}^{s} p^x_{i,j}$, $p^2_{i,j} = \wedge_{y=s+1}^{m} p^y_{i,j}$ and $p_{i,j} = p^1_{i,j} \wedge p^2_{i,j}$.

where $\odot \in \{\bowtie, \rightarrow, \leftarrow, \leftrightarrow\}$. Note, the last four identities are not associative[8] (eg. $r_1 \overset{p_{1,3}}{\rightarrow} (r_2 \overset{p_{1,3}}{\bowtie} r_3) \neq (r_1 \overset{p_{1,3}}{\rightarrow} r_2) \overset{p_{1,3}}{\bowtie} r_3$), and the complex predicate is specified with an operator that is not the root.

For a general expression containing a single complex predicate, our approach consists of transforming the given expression into an equivalent expression in which the complex predicate appears either at the root or with the child or grand child of the root and then apply the appropriate identity. For example, expression Q_4 in Example 3.2 contains only one complex predicate, and there is no ancestor operator that requires the application of this complex predicate, before the application of the ancestor. By results in [BHAR95a], we can transform expression Q_4 into an equivalent expression $(r_1 \overset{p_{1,2}}{\rightarrow} r_2) \overset{p_{2,4} \wedge p_{2,5}}{\rightarrow} ((r_4 \overset{p_{4,5}}{\bowtie} r_5) \overset{p_{3,5}}{\bowtie} r_3))$ in which the complex predicate appears at the root. On the other hand, in expressions such as $r_1 \overset{p_{1,3}}{\leftrightarrow} (r_2 \overset{p_{1,3} \wedge p_{1,3}}{\leftarrow} r_3)$, due to the lack of associativity, we can not push the operator containing the complex predicate to the root. In this case, by employing the results in [BHAR95a], we transform the given expression into an equivalent expression in which the complex predicate appears with either child or grand child of the root. Thus we conclude:

Lemma 1 *A query Q containing a single complex predicate p can be transformed into an equivalent query Q' in which complex predicate p appears with either the root, or a child or a grand child of the root.*

Now, applying the appropriate identity ((1) - (8)), we can break-up the complex predicate. For example, by Identity 3, the complex predicate in expression $(r_1 \overset{p_{1,2}}{\rightarrow} r_2) \overset{p_{2,4} \wedge p_{2,5}}{\rightarrow} ((r_4 \overset{p_{4,5}}{\bowtie} r_5) \overset{p_{3,5}}{\bowtie} r_3))$ can be broken-up to generate equivalent expressions $\sigma^*_{p_{2,4}}[r_1 r_2](Q_4^1)$ and $\sigma^*_{p_{2,5}}[r_1 r_2](Q_4^2)$. Thus, any expression equivalent to expression Q_4^1 (Q_4^2) can be made equivalent to expression Q_4 by applying $\sigma^*_{p_{2,4}}[r_1 r_2]$ ($\sigma^*_{p_{2,5}}[r_1 r_2]$) at the root. We generate different equivalent expressions for expressions Q_4^1 and Q_4^2 by generating their association trees (or, equivalently, association trees for Q_4 according to Definition 3.2) and then assigning operators to the interior nodes.

In general, our approach consists of two steps – a) assign operators to the internal nodes by a method similar to the method presented in [BHAR95a] (for detail, please refer to Section 4 in [BHAR95a]); and b) then apply unused predicates with the novel compensation operator, generalized selection. This requires determining the preserved relations for the generalized selection, as described next.

The preserved relations are computed by constructing the *preserved sets* of (full) outer joins. Intuitively, a preserved set of (full) outer join hyperedge contains those relations that are preserved by it. Formally, for a directed hyperedge $\overset{h}{\rightarrow}$ in hypergraph H, its preserved set are the relations "to the left" of the hyperedge:

$$pres(h) = \{r \mid \text{there is a path } r \sim \ldots \overset{h}{\rightarrow} h \in H\}$$

For example, preserved set for hyperedge h_2 is $\{r_1, r_2\}$ in query Q_4. We generate expression $(r_1 \overset{p_{1,2}}{\rightarrow} ((r_2 \overset{p_{2,4}}{\rightarrow} r_4) MGOJ[r_1 r_2, p_{4,5}](r_5 \overset{p_{3,5}}{\bowtie} r_3)))$ for association tree $(r_1.((r_2.r_4).(r_5.r_3)))$ in step a) of our approach. In step b), $\sigma^*_{p_{2,5}}[r_1, r_2]$ is applied at the top of expression $(r_1 \overset{h_1}{\rightarrow} ((r_2 \overset{h_2}{\rightarrow} r_4) MGOJ[r_1 r_2, h_4](r_5 \overset{h_3}{\bowtie} r_3)))$ to obtain expression:

$$Q_4' = \sigma^*_{p_{2,5}}[r_1, r_2]((r_1 \overset{h_1}{\rightarrow} ((r_2 \overset{h_2}{\rightarrow} r_4) MGOJ[r_1 r_2, h_4](r_5 \overset{h_3}{\bowtie} r_3)))$$

The above expression is equivalent to the original query Q_4. In order to show this equivalence, we apply the identities in [BHAR95a, GALI92a, GALI92b] to transform Q_4' to expression $\sigma^*_{p_{2,5}}[r_1, r_2](Q_4'')$ where expression Q_4'' is identical to expression $Q^1_{2,4}$. The equivalence of $\sigma^*_{p_{2,5}}(Q^1_{2,4})$ and the original query Q_4 follows from Identity 3.

Following the arguments similar to the above arguments, it can be shown that if query Q contains a complex predicate $p_1 \wedge p_2$ with a bi-directed edge h, then query Q is equivalent to query $\sigma^*_{p_1}[r_i \in pres_1(h), r_j \in pres_2(h)](Q')$, where query Q' does not contain predicate p_1, the association tree of query Q' is also an association tree for query Q, and preserved sets $pres_1(h)$ and $pres_2(h)$ are determined as follows. Let bi-directed hyperedge h disconnect hypergraph H into two components $H_1 = (V_1, E_1)$ and $H_2 = (V_2, E_2)$[9], then $pres_1(h) = V_1$ and $pres_2(h) = V_2$.

For those expressions in which we do not push the complex predicate to the root (eg. Identities (5)-(8)), we require the concepts of *closest conflicting outer joins* and *conflict set* [BHAR95a]. Intuitively, if hyperedge h_1 belongs to $conf(h_0)$, then the operator corresponding to h_1 cannot be a descendant of the operator for h_0 in expression trees that employ inner, (full) outer joins only[10]. For a join edge h_0, the set of *closest conflicting*

[8] With out using the MGOJ operator in [BHAR95a].

[9] From Lemma 1 in [BHAR95a], every (bi) directed hyperedge disconnects hypergraph H into two connected components.

[10] These expressions can be reorder by employing $MGOJ$ described in [BHAR95a].

outer joins, denoted $ccoj(h_0)$, is the set $ccoj(h_0) = \{h \mid R_k \xrightarrow{e} R_l \overset{h_1}{\bowtie} \cdots \overset{h_n}{\bowtie} R_i \overset{h_0}{\bowtie} R_j$ is a path in $H\}$. Note, there can be at most one closest conflicting outer join hyperedge in $ccoj(h_0)$. Also, if removing a set of join edges $h_0, h_1, \cdots, h_n$ from H disconnects H into exactly two connected hypergraphs, then all these join edges have the same closest conflicting outer join hyperedge.

Definition 3.3 The *hypergraph conflict set* for a hyperedge, h_0, denoted by $conf(h_0)$, is defined as follows: $conf(h_0) =$

$$\begin{cases} \phi & \cdots \text{ if } h_0 \text{ is bi-directed} \\ \{h \mid R_i \xrightarrow{h_0} R_j \overset{h_1}{\sim} \cdots \overset{h_n}{\sim} R_k \overset{h}{\leftrightarrow} R_l \text{ is a path in } H\} & \cdots \text{ if } h_0 \text{ is directed} \\ \{h \mid R_i \overset{h_0}{\bowtie} R_j \overset{h_1}{\sim} \cdots \overset{h_n}{\sim} R_k \overset{h}{\leftrightarrow} R_l \text{ is a path in } H\} & \cdots \text{ if } h_0 \text{ is undirected and } ccoj(h_0) = \phi \\ \{h\} \cup conf(h) & \cdots \text{ if } h_0 \text{ is undirected and } ccoj(h_0) = \{h\} \end{cases}$$

where $\overset{h_i}{\sim}$ is a join or a left/right outer join.

In addition, we slightly modify the definition of the preserved set: for a bi-directed or directed hyperedge h, the set of relations preserved by h away from edge h_1 is the following set:

$$pres_{h_1}(h) = \begin{cases} \{r \mid \text{ no path } r \sim \ldots \overset{h}{\leftrightarrow} \in H \\ \quad \text{includes } h_1\}, \text{ for } \overset{h}{\leftrightarrow} \\ pres(h), \text{ for } \overset{h}{\leftarrow} \end{cases}$$

Finally, we state the main result of this section.

Theorem 1 *Let the hypergraph for query* $Q = Q_1 \overset{p_1 \wedge p_2}{\odot} Q_2$ *containing a single complex predicate* $p_1 \wedge p_2$ *be* $H = (V, E)$. *Let the hyperedge corresponding to the complex predicate be* $h = \langle V_1, V_2 \rangle$, *where* $|V_1| \geq 1$ *or* $|V_2| \geq 1$, T' *be an association tree for hypergraph* H, *and* Q' *be the expression tree for association tree* T' *such that predicate* $p_1 \notin Q'$. *Then:*

- *If* $\odot = \leftrightarrow$ *then* $Q = \sigma^*_{p_1}[pres_1(h), pres_2(h)](Q')$.
- *If* $\odot = \rightarrow$ *then:*
  - *If* $conf(h) = \phi$, *then* $Q = \sigma^*_{p_1}[pres(h)](Q')$.
  - *If* $conf(h) = \{h_1, \ldots, h_n\}$, *then:* $Q = \sigma^*_{p_1}[pres_h(h_1), \ldots, pres_h(h_n), pres(h)](Q')$
- *If* $\odot = \bowtie$ *then:*
  - *If* $conf(h) = \phi$, *then* $Q = \sigma^*_{p_1}[](Q')$.
  - *If* $conf(h) = \{h_1, \ldots, h_n\}$, *then:* $Q = \sigma^*_{p_1}[pres_h(h_1), \ldots, pres_h(h_n)](Q')$.

Note, the preserved sets and conflict sets are computed only once from the original hypergraph.

Next, by means of examples, we explain how our results can be extended to expressions containing any number of complex predicates. Our approach consists of traversing the given expressions in a top-down manner and recursively breaking-up the complex predicates. The expressions trees that correspond to broken-up complex predicates can be shown to be equivalent.

For example, consider $Q_5 = (r_1 \overset{p_{1,2} \wedge p_{1,3}}{\leftrightarrow} (r_2 \overset{p_{2,3}}{\rightarrow} r_3)) \overset{p_{2,4}}{\rightarrow} (r_4 \overset{p_{4,5} \wedge p_{4,6}}{\rightarrow} (r_5 \overset{p_{5,6}}{\bowtie} r_6))$ which contains two complex predicates $P_1 = p_{1,2} \wedge p_{1,3}$ and $P_2 = p_{4,5} \wedge p_{4,6}$. Note that predicate P_1 does not require the application of predicate P_2, before applying predicates P_1; and vice-versa. Therefore, expression Q_5 can be transformed to into the following equivalent expressions in which complex predicates appear at the root:

$$Q_5^1 = r_1 \overset{p_{1,2} \wedge p_{1,3}}{\leftrightarrow} ((r_2 \overset{p_{2,3}}{\rightarrow} r_3)) \overset{p_{2,4}}{\rightarrow} (r_4 \overset{p_{4,5} \wedge p_{4,6}}{\rightarrow} (r_5 \overset{p_{5,6}}{\bowtie} r_6))$$

$$Q_5^2 = (r_1 \overset{p_{1,2} \wedge p_{1,3}}{\leftrightarrow} (r_2 \overset{p_{2,3}}{\rightarrow} r_3)) \overset{p_{2,4}}{\rightarrow} r_4) \overset{p_{4,5} \wedge p_{4,6}}{\rightarrow} (r_5 \overset{p_{5,6}}{\bowtie} r_6)$$

By identity (4):

$$Q_5^1 = \begin{cases} \sigma^*_{p_{1,2}}[r_1, r_j](r_1 \overset{p_{1,3}}{\leftrightarrow} ((r_2 \overset{p_{2,3}}{\rightarrow} r_3)) \overset{p_{2,4}}{\rightarrow} \\ \quad (r_4 \overset{p_{4,5} \wedge p_{4,6}}{\rightarrow} (r_5 \overset{p_{5,6}}{\bowtie} r_6)), \quad 2 \leq j \leq 6 \\ \sigma^*_{p_{1,3}}[r_1, r_j](r_1 \overset{p_{1,2}}{\leftrightarrow} ((r_2 \overset{p_{2,3}}{\rightarrow} r_3)) \overset{p_{2,4}}{\rightarrow} \\ \quad (r_4 \overset{p_{4,5} \wedge p_{4,6}}{\rightarrow} (r_5 \overset{p_{5,6}}{\bowtie} r_6)), \quad 2 \leq j \leq 6 \end{cases}$$

By pushing complex predicate $p_{4,5} \wedge p_{4,6}$ to the root and applying identity (3), we generate new equivalent expressions. For example, the above last expression generates the following equivalent expressions:

$$Q_5^1 = \begin{cases} \sigma^*_{p_{1,3}}[r_1, r_j]\sigma^*_{p_{4,5}}[r_i](r_1 \overset{p_{1,2}}{\leftrightarrow} ((r_2 \overset{p_{2,3}}{\rightarrow} r_3)) \overset{p_{2,4}}{\rightarrow} \\ \quad (r_4 \overset{p_{4,6}}{\rightarrow} (r_5 \overset{p_{5,6}}{\bowtie} r_6)), \ 1 \leq i \leq 4, \ 2 \leq j \leq 6 \\ \sigma^*_{p_{1,3}}[r_1, r_j]\sigma^*_{p_{4,6}}[r_i](r_1 \overset{p_{1,2}}{\leftrightarrow} ((r_2 \overset{p_{2,3}}{\rightarrow} r_3)) \overset{p_{2,4}}{\rightarrow} \\ \quad (r_4 \overset{p_{4,5}}{\rightarrow} (r_5 \overset{p_{5,6}}{\bowtie} r_6)), \ 1 \leq i \leq 4, \ 2 \leq j \leq 6 \end{cases}$$

Similar equivalent expressions can be generated from expression Q_5^2. Note, expression trees that are generated by breaking-up either complex predicate P_1 or P_2 are equivalent to either Q_5^1 or Q_5^2.

In expressions in which one complex predicate requires the application of some other predicate(s) (dependent predicate(s)), our approach consists of breaking the independent predicate first, followed by the breaking of its dependent predicate(s).

For example, consider $Q_6 = r_1 \overset{p_{1,2} \wedge p_{1,4}}{\leftrightarrow} (r_2 \overset{p_{2,3} \wedge p_{2,4}}{\rightarrow} (r_3 \overset{p_{3,4}}{\rightarrow} r_4))$, where $P_1 = p_{1,2} \wedge p_{1,4}$ and $P_2 = p_{2,3} \wedge p_{2,4}$, in which complex predicate P_1 requires the application of complex predicate P_2. In this case, we first break complex predicate P_1:

$$Q_6 = \begin{cases} \sigma^*_{p_{1,2}}[r_1, r_i](r_1 \overset{p_{1,4}}{\leftrightarrow} (r_2 \overset{p_{2,3} \wedge p_{2,4}}{\rightarrow} (r_3 \overset{p_{3,4}}{\rightarrow} r_4))), \\ \qquad 2 \leq i \leq 4 \\ \sigma^*_{p_{1,4}}[r_1, r_i](r_1 \overset{p_{1,2}}{\leftrightarrow} (r_2 \overset{p_{2,3} \wedge p_{2,4}}{\rightarrow} (r_3 \overset{p_{3,4}}{\rightarrow} r_4))), \\ \qquad 2 \leq i \leq 4 \end{cases}$$

and then break complex predicate P_2:

$$Q_6 = \begin{cases} \sigma^*_{P_{1,2}}[r_1, r_i]\sigma^*_{P_{2,3}}[r_1 r_2](r_1 \overset{p_{1,4}}{\leftrightarrow} (r_2 \overset{p_{2,4}}{\rightarrow} (r_3 \overset{p_{3,4}}{\rightarrow} r_4))), \\ \qquad\qquad 2 \leq i \leq 4 \\ \sigma^*_{P_{1,2}}[r_1, r_i]\sigma^*_{P_{2,4}}[r_1 r_2](r_1 \overset{p_{1,4}}{\leftrightarrow} (r_2 \overset{p_{2,3}}{\rightarrow} (r_3 \overset{p_{3,4}}{\rightarrow} r_4))), \\ \qquad\qquad 2 \leq i \leq 4 \\ \sigma^*_{P_{1,4}}[r_1, r_i]\sigma^*_{P_{2,3}}[r_1 r_2](r_1 \overset{p_{1,2}}{\leftrightarrow} (r_2 \overset{p_{2,4}}{\rightarrow} (r_3 \overset{p_{3,4}}{\rightarrow} r_4))), \\ \qquad\qquad 2 \leq i \leq 4 \\ \sigma^*_{P_{1,4}}[r_1, r_i]\sigma^*_{P_{2,4}}[r_1 r_2](r_1 \overset{p_{1,2}}{\leftrightarrow} (r_2 \overset{p_{2,3}}{\rightarrow} (r_3 \overset{p_{3,4}}{\rightarrow} r_4))), \\ \qquad\qquad 2 \leq i \leq 4 \end{cases}$$

Note, expression trees that are generated by breaking-up either complex predicate P_1 or P_2 are equivalent to one of the above six expressions.

4 Generation of the optimal schedule

In this section, we present an outline of the enumeration process that selects the schedule with the minimum cost. Our approach consists of two steps – a) push aggregations to the root by the method presented in [BHAR95b]; if any of aggregations generate a column that is referenced in join predicate(s) or the join predicate references more than two base relation columns, then the join predicate(s) is broken-up and pushed to the root by the method presented in the last section, eg. Example 3.1; and b) then generate all possible equivalent expression trees by generating the association trees.

It is easy to envision an enumeration algorithm for association trees, constructed bottom up from the given query hypergraph. The first step is to create one leaf trees. Then, at each subsequent step, two subtrees are combined to obtain a larger tree, only if all the conditions specified in the definition of association trees are satisfied. This check is easily included in, say, the dynamic programming approach of existing RDBMS optimizers. [BHAR95a, GALI92a] presents details of such extensions to the dynamic programming enumeration technique and also explains the enumeration of aggregations along with joins. Details of operator assignments are found in [BHAR95b]. Note that the enumeration technique has to be extended so that it considers the cost of the generalized selection operator. The cost of of the generalized selection operator is very similar to the cost of MGOJ operator in [BHAR95a] or GOJ operator in [GALI92a].

5 Conclusion

New results are presented in this paper that can be employed for exhaustively reordering of SQL queries containing joins, (full) outer joins, and Groupby aggregations. We employ the query hypergraph model in order to capture all the inter operator dependencies that are computed only once for the given query. Once these dependencies are generated, slight modifications to existing enumeration algorithms generate all possible reorderings and assign operators. For this, we propose and employ a simple yet very powerful novel operator, generalized selection. From an implementation point of view, this operator is similar to the generalized outer join operator. We can now reorder queries that contain predicates referencing more than two relations or aggregated columns. Since the outer join operation is used to to traverse parent child hierarchies, other applications like hierarchical data applications, "object" data applications and "object" extended relational dbms systems may benefit from our findings.

References

[BHAR94] Bhargava, G., Goel, P. and Iyer, B., "Reordering of complex queries involving joins and outer joins," *IBM Technical Report TR03.567*, July 1994.

[BHAR95a] Bhargava, G., Goel, P. and Iyer, B., "Hypergraph based reorderings of outer join queries with complex predicates," *SIGMOD*, 1995, pp. 304-315.

[BHAR95b] Bhargava, G., Goel, P. and Iyer, B., "No Regression Algorithm for the Enumeration of Projections in SQL Queries with Joins and Outer Joins," *CASCON*, pp. 87-99, 1995.

[BHAR95c] Bhargava, G., Goel, P. and Iyer, B., "Simplification of outer joins," *CASCON*, pp. 63-75, 1995.

[BHAR95d] Bhargava, G., Goel, P. and Iyer, B., "Efficient processing of outer joins and aggregate functions", *To appear in Proceedings of Data Engineering*, 1996.

[GALI92a] Galindo-Legaria, C., and Rosenthal, A., "How to extend a conventional optimizer to handle one- and two-sided outer join," *Proceedings of Data Engineering*, pp. 402-409, 1992.

[GALI92b] Galindo-Legaria, C. A., "Algebraic optimization of outer join queries," Ph.D. dissertation, Dept. of Applied Science, Harvard University, Cambridge, 1992.

[GALI94] Galindo-Legaria, C. A., "Outer joins as disjunctions," *SIGMOD*, pp. 348-358, 1994.

[GANS87] Ganski, R. A., and Wong, H. K. T., "Optimization of nested SQL Queries Revisited", *Proc. SIGMOD*, pp. 23-33, May 1987.

[GUPT95] Gupta, A., Harinarayan, V. and Quass, D., "Generalized Projections: A powerful approach to aggregation", *To appear in VLDB*, 1995.

[MURA92] Muralikrishna, M., "Improved Unnesting Algorithms for Join Aggregate SQL Queries", *Proceedings of 18th VLDB Con.*, pp 91-102, Vancouver, British Columbia, Canada, 1992.

[PIRA92] Pirahesh, H., Hellerstein, J. M. and Hasan, W., "Extensible/rule based query rewrite optimization in Starburst," *SIGMOD*, pp. 39-48, San Diego, Ca., June 1992.

[ROSE90] Rosenthal, A. and Galindo-Legaria, C., "Query graphs, implementing trees, and freely-reorderable outer joins," *SIGMOD*, pp. 291-299, 1990.

[SELI79] Selinger, P. G., Astrahan, M. M., Chamberlin, D. D., Lorie, R. A. and Price, T. G., "Access path selection in a relational database management system," *SIGMOD*, pp. 23-34, 1979.