PROCESSING INEQUALITY QUERIES BASED ON GENERALIZED SEMI-JOINS

Masatoshi Yoshikawa

Department of Information Science
Kyoto University
Sakyo, Kyoto 606, JAPAN

Yahiko Kambayashi

Department of Computer Science and
Communication Engineering
Kyushu University
Hakozaki, Higashi, Fukuoka 812, JAPAN

Abstract

Bernstein and Goodman showed that natural inequality (NI) queries can be processed efficiently by semi-joins, if there are no multiple inequality join edges, nor cycles with one or zero doublet. In this paper procedures to handle these cases efficiently are given. Multiple inequality join edges can be processed by multi-attribute inequality semi-joins. Two procedures based on generalized semi-joins for cyclic NI queries (with one or zero doublet) are developed.

1. Introduction

Semi-join is a useful operation to reduce the processing cost in distributed databases and database machines [BERNC8101] [BERNG8111]. Its processing power, however, is limited because not all queries can be solved using semi-joins only. When queries consisting of natural joins of relations (called NJ (Natural Join) queries) are considered, queries in the class called tree queries can be solved using semi-joins only but the rest of queries (called cyclic queries) cannot [BERNG8111].

We have introduced generalized semi-joins and developed procedures for cyclic queries using generalized semi-joins [KAMBY8206]. We have also developed several methods to transform cyclic queries into tree ones utilizing data dependencies [KAMBY8305]. Using these processing methods, cyclic queries also can be solved efficiently.

These results cannot be applied directly to natural inequality (NI) queries; queries containing inequality joins. In this paper procedures for NI queries are presented based on the extension of generalized semi-joins used for NJ queries. Bernstein and Goodman showed that a subclass of cyclic NI queries can be solved by semi-joins only as well as tree NI queries [BERNG7912] [BERNG81]. A special combination of inequality specifications is said to form a doublet. Even if there are cycles, the query is proved to be solved by semi-joins only, when two or more doublets are contained in each cycle.

An NI query which cannot be solved by semi-joins only satisfies one of the following three conditions [BERNG7912] [BERNG81].

(1) More than one inequality, which cannot be reduced to one inequality, is defined on a pair of relations (existence of multiple-inequality edges).
(2) A cycle containing exactly one doublet exists (1-doublet query).
(3) A cycle containing no doublets exists (0-doublet query).

Queries of type (1) are shown to be handled by multi-attribute inequality semi-joins to be introduced in this paper. By extending the concept of generalized semi-joins for NJ queries, any cyclic NI query is shown to be solved. The method is applied to 0-doublet queries and 1-doublet queries.

Generalized semi-joins and processing algorithms utilizing them introduced in this paper include those in [KAMBY8206] as special cases.

In Section 2, basic definitions and background are given. In Section 3, multi-attribute inequality semi-joins are defined. A query with multiple inequality edges is processed by this kind of semi-join. In Section 4, procedures for cyclic NI queries are presented. These procedures utilize generalized semi-joins and multi-attribute inequality semi-joins. Section 5 is the conclusion.

2. Basic Concepts and Background

A relation scheme and a relation consisting of attributes $A_1,A_2,\ldots,A_m$ are denoted by $\underline{R}(A_1,A_2,\ldots,A_m)$ and $R(A_1,A_2,\ldots,A_m)$, respectively. If the specification of the set of attributes is not necessary, the notations $\underline{R}$ and R are used. An attribute A in a relation R is denoted by R.A. A collection of relation schemes is called a database scheme. A collection of relations corresponding to the database scheme is called a database and is denoted by $D(R_1,R_2,\ldots,R_n)$.

Let A and B be attributes of R, and X be an attribute set such that $X \subseteq \underline{R}$. Let t be a tuple of a relation R. The following notations of the relational algebra will be used.

The projection of t on X: t[X]
The projection of R on X: $R[X] = \{t[X] | t \in R\}$
θ-restriction: $\sigma_{A\theta B}R = \{t | t[A]\,\theta\, t[B],\ t \in R\}$

(Here, θ is one of the comparison operators $=, \neq, <, >, \leq$ and $\geq$)

A query Q, which consists of a qualification q and a target attribute set TA, maps a database $D(R_1,R_2,\ldots,R_n)$ into the following relation.

$$(\sigma_q(R_1 \times R_2 \times ,\ldots, \times R_n))[TA]$$

We call $\sigma_q(R_1 \times R_2 \times ,\ldots, \times R_n)[\underline{R}_i]$ a partial solution of R_i (with repect to q). In this paper, we will develop procedures for joins which will obtain partial solutions for all relations involved in the query. Since target attribute sets are not required to be considered in our problem, we will use q to represent a query. If the complete result of the join or its projection on TA is required, our procedures to obtain the partial solutions can be used as a preprocessing step. Throughout this paper, we will assume for simplicity that all partial solutions are non-empty.

Let A and B be attributes of $\underline{R}_i$ and $\underline{R}_j$ ($i \neq j$), respectively. A qualification which is a conjunction of clauses of the form $R_i.A\,\theta\, R_j.B$ ($\theta \in \{=,<,>,\leq,\geq\}$) is called an inequality-join qualification. Note that we do not consider "$\neq$" as a comparison operator in the following discussion.

An inequality-join qualification can be expressed by a join graph $G_J(V_J,E_J)$ where

$$V_J = \{R_i.A |\ A \in \underline{R}_i,\ i=1,2,\ldots,n\}$$
$$E_J \subseteq V_J \times V_J$$

G_J is a directed graph such that an edge $\langle R_i.A, R_j.B\rangle$ represents a clause $R_i.A \geq R_j.B$ ($R_i.A > R_j.B$) in q. A clause $R_i.A = R_j.B$ is represented by a pair of edges $\langle R_i.A, R_j.B\rangle$ and $\langle R_j.B, R_i.A\rangle$. Therefore, two nodes $R_k.C$ and $R_l.D$ are in the same strongly connected component in G_J iff the clause $R_k.C = R_l.D$ can be implied by q. If two nodes from the same relation, say $R_i.A$ and $R_i.B$, are in the same strongly connected component in G_J, we can merge them by performing the restriction operation $\sigma_{A=B}R_i$ and replacing all occurrences of one attribute name (say $R_i.B$,) in q by another attribute name ($R_i.A$). Repeating these preprocessing, we can obtain a qualification such that there exists at most one node from a relation in each strongly connected component in the join graph. We will consider only such qualifications hereafter.

An inequality-join qualification satisfying the following condition is called natural:

"There exists at most one node from a relation in each weakly connected component in G_J."

The word "natural" is used because by renaming attributes properly, we can ensure that all clauses in q be represented in the form $R_i.A\,\theta\, R_j.A$.

Queries which have inequality-join qualifications (natural inequality-join qualifications, resp.) are called inequality-join queries (NI (natural inequality) queries, resp.). NI queries which consist of only clauses having "=" as comparison operators are called NJ (natural-join) queries. We will consider only NI queries in this paper, since most of inequality-join queries can be transformed into a natural one by proper renaming of attributes.

Given an NI query q, let $\{c^1_{ij}, c^2_{ij},\ldots,c^k_{ij}\}$ be the set of clauses defined between R_i and R_j in q. C_{ij} is defined as follows.

$$C_{ij} = c^1_{ij} \wedge c^2_{ij} \wedge \ldots \wedge c^k_{ij}$$

A qual graph $G_q=(V, E, L)$ corresponding to an NI query q is a labeled undirected graph. V is a set of nodes, where v_i in V corresponds to relation R_i referred to in q. E is the set of edges and L is the set of labels for E. Two nodes v_i and v_j corresponding to R_i and R_j are connected by an edge iff there is a clause $R_i.A\,\theta\, R_j.B$ in q. If C_{ij} is $c^1_{ij} \wedge c^2_{ij} \wedge \ldots \wedge c^k_{ij}$, labels $l^1_{ij}, l^2_{ij},\ldots,l^k_{ij}$ are attached to the edge $\langle R_i, R_j\rangle$. Each label l^h_{ij} corresponds to c^h_{ij} ($h=1,2,\ldots,k$). Let c^h_{ij} be $R_i.A\,\theta\, R_j.B$. c^h_{ji} is used to represent the clause $R_j.B\ \theta^{-1} R_i.A$, where θ^{-1} represents the inverse of θ ("<", "≤" and "=" are the inverse of ">", "≥" and "=", respectively.). We use the following notation.

$$at(c^h_{ij}, R_i) = at(c^h_{ji}, R_i) = A$$

$$at(c^h_{ij}, R_j) = at(c^h_{ji}, R_j) = B$$

$$op(c^h_{ij}) = \theta$$

$$op(c^h_{ji}) = \theta^{-1}$$

The label l^h_{ij} is defined as follows.

$$l^h_{ij} = ((at(c^h_{ij}, R_i),\ at(c^h_{ij}, R_j)),\ op(c^h_{ij})) = ((A,B), \theta)$$

Also $(l^h_{ij})^{-1}$, the inverse of l^h_{ij} is defined as follows

$$(l^h_{ij})^{-1} = l^h_{ji} = ((at(c^h_{ij}, R_j),\ at(c^h_{ij}, R_i)),\ (op(c^h_{ij}))^{-1}) = ((B,A), \theta^{-1}),$$

L_{ij} is used to denote the label set $\{l^1_{ij}, l^2_{ij}, \ldots, l^k_{ij}\}$ corresponding to the edge $\langle R_i, R_j \rangle$.

If a qual graph is not connected, it is sufficient to process each connected component separately. Thus we will assume that a qual graph is connected.

An edge $e=\langle R_i, R_j \rangle$ of a qual graph G_q is called a <u>multiple edge</u> iff $|L_{ij}|>1$, otherwise it is called a <u>simple edge</u>. Let $e_{ij}=\langle R_i, R_j \rangle$ and $e_{jk}=\langle R_j, R_k \rangle$ be simple edges of G_q. A pair of edges e_{ij} and e_{jk} is called a <u>doublet</u> iff

(i) $at(c_{ij}, R_j) = at(c_{jk}, R_j)$, and

(ii) $op(c_{ij})$ and $op(c_{jk})$ are inequalities of opposite direction; e.g. one is "<" or "≤" and the other is ">" or "≥" (see Fig.2.1.).

Two queries are said to be <u>equivalent</u> iff both will produce the same result for any database state. Two qual graphs are <u>equivalent</u> iff the corresponding queries are equivalent.

An NJ query is called a <u>tree query</u> if it is equivalent to a query whose qual graph is circuit-free; otherwise it is called <u>cyclic</u> [BERNG8111]. There are tree queries whose qual graphs have cycles [BERNG8111]. All partial solutions for a tree query can be obtained by semi-joins only.

NI queries are also classified into tree and cyclic queries. Note that there exists a tree NI

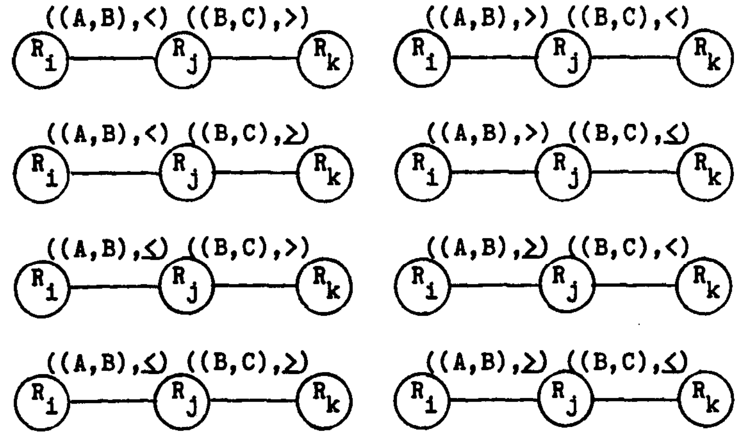


Fig.2.1 Doublets.

query for which the qual graph has cycles. For example, a query having the qual graph shown in Fig.2.2(a) is a tree NI query because it can be equivalently transformed into the one shown in Fig.2.2(b). A cycle in a qual graph is called an <u>n-doublet cycle</u> if it contains exactly n doublets. If a qual graph contains m cycles each of which is an i_j-doublet cycle (j=1,2,...,m), it is called a <u>k-doublet qual graph</u> for $k = \min\{i_1, i_2, \ldots, i_m\}$. As shown later, k-doublet cycles in a qual graph for small k are difficult to process in general. Therefore, "k-doublet qual graph" means that the most intractable cycle in the qual graph is a k-doublet one. If a cyclic NI query is equivalent to s queries each of which has a k_h-doublet qual graph (h=1,2,...,s), it is called an <u>n-doublet query</u> for $n = \max\{k_1, k_2, \ldots, k_s\}$. The maximum value is taken for n because DBMSs, in general, optimize a given query and therefore the intractability of a cyclic query can be measured by the most tractable one in the set of equivalent queries. n-doublet queries are called <u>multi-doublet queries</u> or <u>weak 1-doublet queries</u> when $n \geq 2$ or $n \leq 1$, respectively. A cyclic NI query whose qual graph consists of exactly one cycle is called a <u>simple cyclic query</u>. If it has n doublets, it is called a <u>simple n-doublet query</u>.

A tree NI query not satisfying the following condition (1) can be processed by semi-joins. Besides such tree queries, all partial solutions of a cyclic query not satisfying any one of the

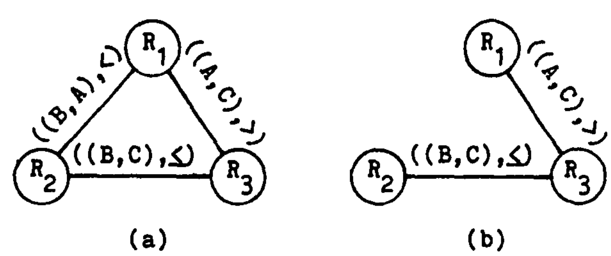


Fig.2.2 Equivalent qual graphs.

following three conditions can be obtained by semi-joins only [BERNG7912] [BERNG81].

(1) There exists a multiple edge which cannot be reduced to a simple edge.
(2) A 1-doublet query
(3) A 0-doublet query

Procedures for the case (1) are given in Section 3, and Section 4 is dedicated to the cases (2) and (3).

If two tuples $t_i (\in R_i)$ and $t_j (\in R_j)$ satisfy C_{ij} (c^h_{ij}, resp.), they are denoted by $t_i \langle C_{ij} \rangle t_j$ ($t_i \langle c^h_{ij} \rangle t_j$, resp.). Let $p = \langle R_i = R_{i0}, R_{i1}, \ldots, R_{ik-1}, R_{ik} = R_j \rangle$ be a path in a qual graph. Also let X be an attribute set such as $X \subseteq \underline{R}_j$. A tuple $t_i \in R_i$ is said to <u>join with</u> a tuple $t_j[X]$ ($t_j \in R_j$) along p iff there exist tuples $t_{ih} \in R_{ih}$ ($h=1,2,\ldots,k-1$) such that $t_{i\,h-1} \langle C_{h-1\,h} \rangle t_{ih}$ ($h=1,2,\ldots,k$). As a special case, we will say that $t_i (\in R_i)$ joins with $t_i[X]$ along the path $\langle R_i \rangle$ for X such that $X \subseteq \underline{R}_i$ and $X \neq \phi$. We will also assume that t_i does not join with any $t'_i[X]$ such that $t_i \neq t'_i$ along the path $\langle R_i \rangle$.

Let C be a conjunction of clauses defined between the relations R_i and R_j. The <u>join</u> of R_i and R_j on C is denoted by $R_i \underset{C}{\bowtie} R_j$ and defined as follows.

$$R_i \underset{C}{\bowtie} R_j = \{t_i t_j \mid t_i \in R_i,\ t_j \in R_j,\ t_i \langle C \rangle t_j\}$$

Let c (for example, $R_i.A \,\theta\, R_j.B$) be a clause defined between the relations R_i and R_j. A <u>(single-attribute) semi-join</u> of R_i by R_j on c is denoted by $R_i \underset{c}{\ltimes} R_j$ (or $R_i \underset{A\theta B}{\ltimes} R_j$) and defined as

$$R_i \underset{c}{\ltimes} R_j = (R_i \underset{c}{\bowtie} R_j)[\underline{R}_i]$$

A sequence of semi-joins is called a <u>semi-join program</u>.

If no further application of semi-join changes the contents of a database, that database is called <u>semi-join reduced</u>.

In [BERNG7912], a (single-attribute) semi-join program for processing a tree query or a multi-doublet query with no multiple edge is shown. The program is summarized in Procedure 2.1. Before giving the procedure, the Acyclicity Property must be defined.

[Acyclicity Property]

Let G_q be a qual graph. G_q is said to satisfy the Acyclicity Property iff there exists a proper assignment of directions on edges in G_q so that the resulting digraph, designated by G_{DOWN}, satisfies the following two conditions.

(i) G_{DOWN} is acyclic.
(ii) A pair of edges having the same destination in G_{DOWN} is a doublet.

<u>Procedure 2.1</u>: Processing tree queries and multi-doublet queries with no multiple edge [BERNG7912] [BERNG81]

Let G_q be a tree qual graph (multi-doublet qual graph, resp.) which is equivalent to a given tree (multi-doublet, resp.) query. For G_q, there exists a directed acyclic graph G_{DOWN} since G_q satisfies the Acyclicity Property (G_{DOWN} is obtained by applying a modified depth first search algorithm to G_q). The directed edge $\langle R_i, R_j \rangle$ of G_{DOWN} with label $((A,B),\theta)$ is interpreted as a semi-join operation

$$R_j \underset{B\theta^{-1}A}{\ltimes} R_i .$$

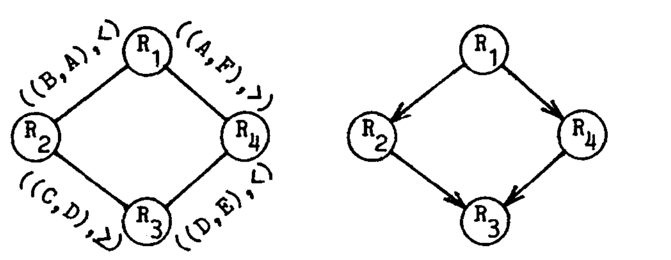
G_{DOWN} as a whole is interpreted as a semi-join program consisting of semi-joins represented by its edges, ordered by any topological sort. This semi-join program is called DOWN. G_{UP} is defined analogously with all edges of G_{DOWN} reversed, and represents a semi-join program UP. By performing UP·DOWN, the semi-join program UP followed by DOWN, all the partial solutions w.r.t. q are obtained.

[Example 2.1] Consider a multi-doublet query whose qual graph is shown in Fig.2.3(a). Selecting a directed acyclic graph G_{DOWN} as shown in Fig.2.3(b), a semi-join program UP is obtained as

$$R_2 \underset{C \geq D}{\ltimes} R_3,\ R_1 \underset{A>B}{\ltimes} R_2,\ R_4 \underset{E>D}{\ltimes} R_3,\ R_1 \underset{A>F}{\ltimes} R_4$$

Also a semi-join program DOWN is as follows.

$$R_2 \underset{B<A}{\ltimes} R_1,\ R_3 \underset{D \leq C}{\ltimes} R_2,\ R_4 \underset{F<A}{\ltimes} R_1,\ R_3 \underset{D<E}{\ltimes} R_4$$



(a) a qual graph (b) G_{DOWN}

Fig.2.3 An example of multi-doublet query.

Query structure / Type of edges	Tree	Cyclic		Semi-join procedure
		Multi-doublet	Weak 1-doublet	
Simple edges only	○	○	△	Single-attribute inequality semi-join
Multiple edges permitted	△	△	△	Multi-attribute inequality semi-join (Section 3)
Query processing prcedure	All partial solutions can be obtained by semi-joins.		All partial solutions can be obtained by generalized semi-joins (Section 4).	

o: The results in [BERNG7912] [BERNG81]
△: The results by this paper

Fig.2.4 Problems for NI query processing.

The following proposition holds directly from the definition of partial solution.

[Proposition 2.1] Let q be a simple cyclic query consisting of n relations $R_1, R_2, \ldots, R_n$ (Let us assume that the qual graph G_q consists of a cycle $R_1\text{-}R_2\text{-}\ldots\text{-}R_{n-1}\text{-}R_n\text{-}R_1$.). A tuple $t_i(\in R_i)$ is contained in the partial solution of R_i w.r.t. q iff there exists a relation $R_j(i \neq j)$ and a tuple $t_j(\in R_j)$ such that

(i) t_i joins with t_j along the path $\langle R_i, R_{i\oplus 1}, \ldots, R_{j\ominus 1}, R_j \rangle$ in G_q, and

(ii) t_i joins with t_j along the path $\langle R_i, R_{i\ominus 1}, \ldots, R_{j\oplus 1}, R_j \rangle$ in G_q.

Here $\oplus$ and $\ominus$ represent addition and subtraction of modulo n, respectively.

In the case of Example 2.1, applying the semi-join program UP•DOWN, a semi-join reduced database is obtained. In that database, any tuple in R_i (i=1,2,4) joins with the tuple t_3 in R_3 such that $t_3[D] = \min(R_3[D])$ along both of the two paths between R_i and R_3. Also, any tuple in R_j (j=2,3,4) joins with the tuple t_1 in R_1 such that $t_1[A] = \max(R_1[A])$ along both of the two paths between R_j and R_1. Thus, from Proposition 2.1 partial solutions of all relations are obtained.

The reason why all partial solutions of a multi-doublet query with no multiple edge can be obtained by semi-joins only is explained intuitively because the above discussion holds for every cycle in a multi-doublet qual graph.

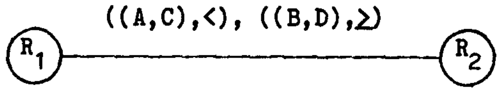
Fig.2.4 summarizes the results by Bernstein and Goodman [BERNG7912] [BERNG81] (denoted by o) and the results by this paper (denoted by △). Bernstein and Goodman handled queries which can be processed by single-attribute inequality semi-joins. We generalized the result to utilize multi-attribute semi-joins and the case (1) above is solved. For weak 1-doublet queries an extension is made to the definition of generalized semi-joins in [KAMBY8206].

3. Multi-attribute Inequality Semi-Joins and Processing Multiple Edges

In this section, we will introduce multi-attribute semi-joins, which are a natural extension of single-attribute ones, for processing multiple edges in a qual graph. The notion of inequality projection is also introduced to clarify the idea of multi-attribute semi-joins.

First we will give an example to illustrate the situation that a multiple edge in a qual graph cannot be processed by single-attribute semi-joins only. Let us consider a multiple-edge shown in Fig.3.1(a). The database shown in Fig.3.1(b) can be easily verified to be (single-attribute) semi-join reduced; therefore further applications of any single-attribute semi-join do not change the database. The relation R_1, however, is not the partial solution since the tuple (2,2) does not join with any tuple in R_2. Such a situation arises because single-attribute semi-joins can check whether a tuple in one relation joins with tuples in other relations only with respect to one join attribute each time. Therefore, the multiple edge shown in Fig.3.1(a) can be considered to be equivalent to the 0-doublet cycle shown in Fig.3.2 if only

(a)



(b) R_1 R_2

A	B
0	4
2	2

C	D
1	1
3	3

Fig.3.1 A multiple edge which cannot be processed by single-attribute semi-joins only.

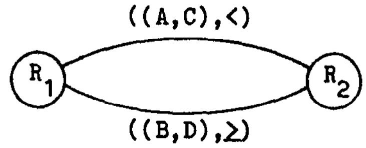


Fig.3.2 A 0-doublet cycle which is equivalent to the multiple edge in Fig.3.1(a).

single-attribute semi-joins are considered. To check the joinability of tuples on more than one join attributes simultaneously, multi-attribute semi-joins defined below are required.

Let C be a conjunction of join clauses $c_1, c_2, \ldots, c_k$ defined between relations R_i and R_j. A (multi-attribute) semi-join of R_i by R_j on C is denoted by $R_i \underset{C}{\ltimes} R_j$ and is defined as follows.

$$R_i \underset{C}{\ltimes} R_j = (R_i \underset{C}{\bowtie} R_j)[\underline{R_i}]$$

To realize a single attribute semi-join, say $R_i \underset{A\geq B}{\ltimes} R_j$, not all join attribute values of R_j are necessary but it is sufficient to know the information only about the minimum value of R_j's join attribute (i.e. $\min(R_j[B])$). For multi-attribute semi-joins, we only need to consider minimal tuples in the partial order defined by the projection qualification defined below.

Let $A_1, A_2, \ldots, A_k$ be a collection of attributes, and $\theta_1, \theta_2, \ldots, \theta_k$ be a corresponding collection of comparison operators. We call $(A_1\theta_1, A_2\theta_2, \ldots, A_k\theta_k)$ a projection qualification. Let t and t' be tuples defined on the set of attributes including $A_1, A_2, \ldots, A_k$. We say that t is $(A_1\theta_1, A_2\theta_2, \ldots, A_k\theta_k)$-smaller than t' under the partial order*) defined by the projection qualification iff

$$\bigwedge_{i=1}^{k} (t[A_i]\ \theta_i\ t'[A_i])$$

holds. $(A_1\theta_1, A_2\theta_2, \ldots, A_k\theta_k)$-minimal tuples can also be defined in the partial order. Using this notion we will define inequality projection, which is an extension of the ordinary projection operation, as follows.

$$R[A_1\theta_1, A_2\theta_2, \ldots, A_k\theta_k]$$
$$= \{t \in R[A_1, A_2, \ldots, A_k] \mid t \text{ is } (A_1\theta_1, A_2\theta_2, \ldots, A_k\theta_k)\text{-minimal in } R[A_1, R_2, \ldots, A_k]\}$$

$R[A_1\theta_1, A_2\theta_2, \ldots, A_k\theta_k]$ is called an inequality projection of R on $A_1\theta_1, A_2\theta_2, \ldots, A_k\theta_k$.

[Example 3.1] Consider a relation R shown in Fig.3.3(a). An inequality projection R[A=,C>] can be obtained by first taking a GROUP-BY[A] operation to the ordinary projection of R on AC (see Fig.3.3(b)), and by picking up only maximum C-values in each group. (see Fig.3.3(c).) R[B>,C≤], which is another example of an inequality projection of R, is a set of (B>,C≤)-minimal tuples in R (Fig.3.3(d) shows the Hasse diagram of the partial order, where ● shows a minimal tuple. The projected relation is given in Fig.3.3(e).).

R

A	B	C
a	2	10
a	3	10
a	3	25
a	1	30
a	3	30
b	1	20
b	3	20
b	4	20
c	2	15
c	4	15
c	5	30

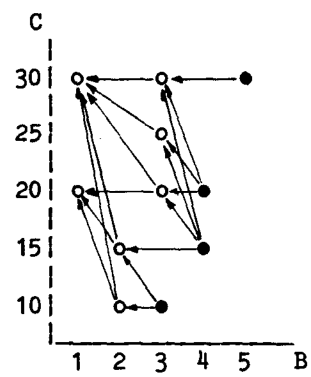
(a)

A	C
a	10
	25
	30
b	20
c	15
	30

(b)

A	C
a	30
b	20
c	30

(c)



(d)

B	C
3	10
4	15
4	20
5	30

(e)

Fig.3.3 Inequality projections.

*) Strictly speaking, if "<" or ">" exists in $\{\theta_1, \theta_2, \ldots, \theta_k\}$, the binary relation "is $(A_1\theta_1, A_2\theta_2, \ldots, A_k\theta_k)$-smaller than" is not a partial order, since it does not satisfy reflexive law. However, even in those cases we will call the binary relation as partial order and will use the word "minimal" tuple to represent the tuple t such that for any tuple t' in R "t' is $(A_1\theta_1, A_2\theta_2, \ldots, A_k\theta_k)$-smaller than t" does not hold.

Let c_h be a join clause of the form $R_i.A_h\theta_h R_j.B_h$. Let C be a conjunction of join clauses $c_1,c_2,\ldots,c_k$ (i.e. $C=c_1 \wedge c_2 \wedge \ldots \wedge c_k$). A projection qualification can be represented using a join qualification as follows.

$$R_i[C] = R_i[A_1\theta_1, A_2\theta_2, \ldots, A_k\theta_k]$$

$$R_j[C] = R_j[B_1\theta_1^{-1}, B_2\theta_2^{-1}, \ldots, B_k\theta_k^{-1}]$$

Using these notions, the multi-attribute semi-join $R_i \ltimes_C R_j$ can be represented as follows in general, which implies that only the inequality projection of R on C is necessary to realize the multi-attribute semi-join $R_i \ltimes_C R_j$.

$$R_i \ltimes_C R_j = R_i \ltimes_C R_j[C]$$

The extension of semi-joins stated above allows the existence of multiple edges in qual graphs of tree queries or multi-doublet queries. Therefore, Procedure 2.1 can be extended to the cases where a qual graph contains multiple edges, and all partial solutions of a tree query or multi-doublet query can be obtained by applying multi-doublet semi-joins only.

4. Generalized Semi-Joins and Processing Weak 1-doublet Queries

In this section, we will introduce the notion of generalized semi-joins, and then utilizing them we will formalize the weak 1-doublet query processing algorithms. Generalized semi-joins presented in this section include the one introduced in [KAMBY8206] as a special case since only NJ queries are considered in [KAMBY8206].

To process a cycle in qual graphs of NI queries, we need to test for each tuple t_i in each relation R_i in the cycle whether t_i joins with itself along the cycle or not. Let us consider the cycle $R_1-R_2-\ldots-R_{n-1}-R_n-R_1$. There are the following two basic methods for the test.

(1) [Comparison of Join Attribute Values of Two Adjacent Relations]
Test whether or not there exists the tuple $t_1(\in R_1)$ ($t_n(\in R_n)$, resp.) with which t_i joins along the path $R_i-R_{i-1}-\ldots-R_2-R_1$ ($R_i-R_{i+1}-\ldots-R_{n-1}-R_n$, resp.), and t_1 and t_n satisfy the join clause defined between R_1 and R_n (see Fig.4.1(a)).

(2) [Comparison of Join Attribute Values of One Relation]
Test whether or not there exists the tuple $t_j(\in R_j)$ ($i \neq j$) with which t_i joins along the two paths between R_i and R_j in the cycle (see Fig.4.1(b)).

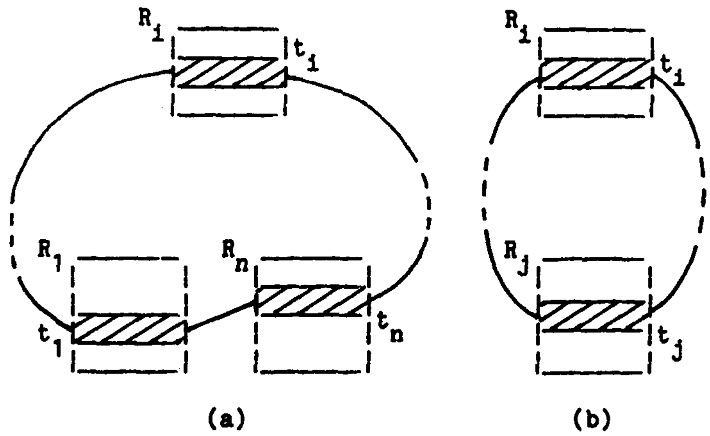


Fig.4.1 Processing of a cycle.

The following proposition, which can be proved directly from Proposition 2.1, is a basis of method (1). Method (2) is based on Proposition 2.1.

[Proposition 4.1] Let q be a simple cyclic query consisting of n relations $R_1,R_2,\ldots,R_n$ (Let us assume that the qual graph G_q consists of a cycle $R_1-R_2-\ldots-R_{n-1}-R_n-R_1$.). A tuple $t_i(\in R_i)$ is contained in the partial solution of R_i w.r.t. q iff there exist tuples $t_1(\in R_1)$ and $t_n(\in R_n)$ such that

(i) t_i joins with t_1 along the path $\langle R_i,R_{i-1},\ldots,R_2,R_1\rangle$ in G_q,
(ii) t_i joins with t_n along the path $\langle R_i,R_{i+1},\ldots,R_{n-1},R_n\rangle$ in G_q, and
(iii) $t_1 \langle C_{1n} \rangle t_n$, where C_{1n} is the conjunction of join clauses defined between R_1 and R_n in q.

Section 4.1 explains the basic idea of method (1) using simple 0-doublet queries as examples. The definition of generalized semi-joins is given in Section 4.2. Query processing strategies based on methods (1) and (2) for simple weak 1-doublet queries are formally given in Sections 4.3 and 4.4, respectively. The extension for general weak 1-doublet queries is discussed in Section 4.5.

4.1 Basic Consideration on Processing 0-doublet Queries

Consider a simple 0-doublet query whose qual graph is shown in Fig.4.2. The following procedure gives the partial solution of R_i.

[Procedure 4.1] A Basic Procedure to Compute the Partial Solution of R_i

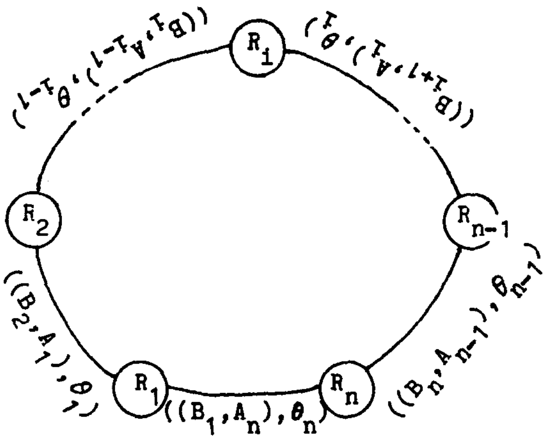


Fig.4.2 A simple 0-doublet query.

I. Computation of R_i', which shows the correspondence between tuples in R_i and B_1-values.

(1-1) $R_1' = R_1$.

(1-2) Repeat the step (1-3) for j=1,2,...,i-1.

(1-3) $R_{j+1}' = R_{j+1} \underset{B_{j+1}\theta_j A_j}{\bowtie} R_j'[A_j B_1]$.

II. Computation of R_{i+1}'', which shows the correspondence between tuples in R_{i+1} and A_n-values.

(2-1) $R_n'' = R_n$.

(2-2) Repeat the step (2-3) for h=n,n-1,...,i+2.

(2-3) $R_{h-1}'' = R_h''[A_n B_h] \underset{B_h\theta_{h-1}A_{h-1}}{\bowtie} R_{h-1}$.

III. Computation of R_i''', which shows the correspondence between tuples in R_i and A_n-values.

(3-1) $R_i''' = R_{i+1}''[A_n B_{i+1}] \underset{B_{i+1}\theta_i A_i}{\bowtie} R_i'$.

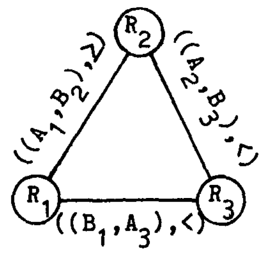
IV. Computation of the partial solution.

(4-1) $(\sigma_{B_1\theta_n A_n}(R_i'''))[\underline{R_i}]$

Let us denote the set of tuples t_1 in R_1 (t_n in R_n, resp.) which satisfy the condition (i) ((ii), resp.) in Proposition 4.1 for a given tuple $t_i(\in R_i)$ as $R_1(t_i)$ ($R_n(t_i)$, resp.). Executing the steps I, II and III of the procedure shown above, we can obtain R_i''', which is the set of tuples t_i together with the corresponding $(R_1(t_i))[B_1]$ and $(R_n(t_i))[A_n]$ value sets. Comparing the both value sets under the condition of $R_1.B_1\theta_n R_n.A_n$ at the step IV, the condition (iii) of Proposition 4.1 is tested and the partial solution of R_i can be obtained.

If θ_n is an inequality operator, we can reduce the cost of operations without changing the final result. The cost of time-consuming operations appeared at (1-3), (2-3) and (3-1) will be reduced by removing tuples which will not affect the final result. For example, we will assume that θ_n is "<". Let $b_m^1(t_i)$ be representing the minimum value in $(R_1(t_i))[B_1]$. Also let $a_M^n(t_i)$ be representing the maximum value in $(R_n(t_i))[A_n]$. A tuple $t_i(\in R_i)$ is contained in the partial solution of R_i iff $b_m^1 < a_M^n$ from Propositon 4.1. So, in the steps (1-3), (2-3) and (3-1), it is sufficient to obtain the tuples which is necessary for R_i'' to contain both $b_m^1(t_i)$ and $a_M^n(t_i)$ for each t_i. The following example illustrates this property.

[Example 4.1] Consider a 0-doublet query, say q_0, of which qual graph is shown in Fig.4.3(a). If the database is in the state as shown in Fig.4.3(b), it is semi-join reduced but any relation is not its partial solution, since tuples marked with x are not in the partial



(a)

R_2

	B_2	A_2
	1	45
	4	35
x	4	45
	5	10

R_1

	A_1	B_1
	5	300
	4	300
	3	100
x	2	200

R_3

	A_3	B_3
	500	40
	400	30
	200	50
x	200	20

(b)

Fig.4.3 A simple 0-doublet query and a example of databases.

solution. A simple method to obtain the partial solutions of, for example, R_2 is to perform the join of three relations R_1, R_2 and R_3 under the qualification q_0 and to project the resultant relation on $\underline{R}_2$. However, not all tuples in R_1 (or R_3) are necessary in the join as discussed above. For example, let us consider tuples $t_1 = (3,100)$ and $t_1' = (2,200)$ in R_1. Since the join clause between R_1 and R_2 is $R_1.A_1 \geq R_2.B_2$, if a tuple t_2 in R_2 joins with t_1', t_2 also joins with t_1 along the path R_2-R_1. Moreover, the join clause $R_1.B_1 < R_3.A_3$ implies that if $t_1' < C_{13} > t_3$ holds for a tuple t_3 in R_3, then $t_1 < C_{13} > t_3$ also holds. These facts mean that whenever t_1' satisfies the conditions (i) and (iii) of Proposition 4.1, t_1 also satisfies these two conditions. Thus there is no need to consider the tuple $t_1' = (2,200)$ in the join of three relations. Similarly the tuple (4,300) becomes unnecessary to be considered due to the tuple (5,300). The same discussion applies to R_3, and only two tuples (500,40) and (200,50) in R_3 are necessary to be considered.

4.2 Generalized Semi-Joins

Let c_h be a join caluse of the form $R_i.A_h \theta_h R_j.B_h$. For a conjunction C of join clauses $c_1, c_2, \ldots, c_k$ (i.e. $C = c_1 \wedge c_2 \wedge \ldots \wedge c_k$), we use the following notation.

$$at(C,R_i) \triangleq \bigcup_{h=1}^{k} \{at(c_h,R_i)\} = \{A_1,A_2,\ldots,A_k\}$$

$$at(C,R_j) \triangleq \bigcup_{h=1}^{k} \{at(c_h,R_j)\} = \{B_1,B_2,\ldots,B_k\}$$

Let C be a conjunction of join clauses such that $at(C,R_j) \subseteq \underline{R}_j$. A generalized semi-join of R_i by R_j on C is denoted by $R_i \ltimes_C R_j$ and defined as

$$R_i \ltimes_C R_j = R_i \bowtie_{C'} R_j[C]$$

where C' is the conjunction of the clauses c_h in C such that $at(c_h,R_i) \subseteq \underline{R}_i$. Note that the relation scheme of $R_i \ltimes_C R_j$ is $\underline{R}_i \cup at(C,R_j)$. In processing queries, it is required to replace R_i by $R_i \ltimes_C R_j$. This operation is denoted by

$$R_i \longleftarrow R_i \ltimes_C R_j.$$

Next we will present processing algorithms for weak 1-doublet queries utilizing generalized semi-joins. We will describe our procedure by means of modification of qual graphs.

4.3 Comparison of Join Attribute Values of Two Adjacent Relations

Consider a query of which qual graph is shown in Fig.4.4(a). To make the notation be succinct, we will use C_i and L_i as the join clause corresponding to the edge $\langle R_{i\ominus 1}, R_i \rangle$ (i.e. $C_{i\ominus 1\ i}$) and its label set (i.e. $L_{i\ominus 1\ i}$), respectively ($i=1,2,\ldots,n$). As illustrated in Example 4.1, in this method join operations are repeated using only tuples which are sufficient for the testing of the condition (iii) in Proposition 4.1. In Section 4.1, we have shown procedures to obtain the partial solution of only one relation in a cycle. To obtain the partial solutions of all relations, we need to repeat the procedures for every relation in a cycle. However, as shown in Procedure 4.1 in Section 4.1, R_{j+1}' is obtained using R_j' for $j=1,2,\ldots,n-1$ and R_{h-1}'' is obtained using R_h'' for $h=n,n-1,\ldots,2$. Therefore, procedures for obtaining all partial solutions can be "piggybacked". The method is formally described as follows. First the qual graph is transformed to a tree by embedding an edge, say $\langle R_n, R_1 \rangle$, into all other edges as shown in Fig.4.4(b). Then, label sets L_i and L_n^{-1} are merged as shown in Fig.4.4(c). L_i' is defined as

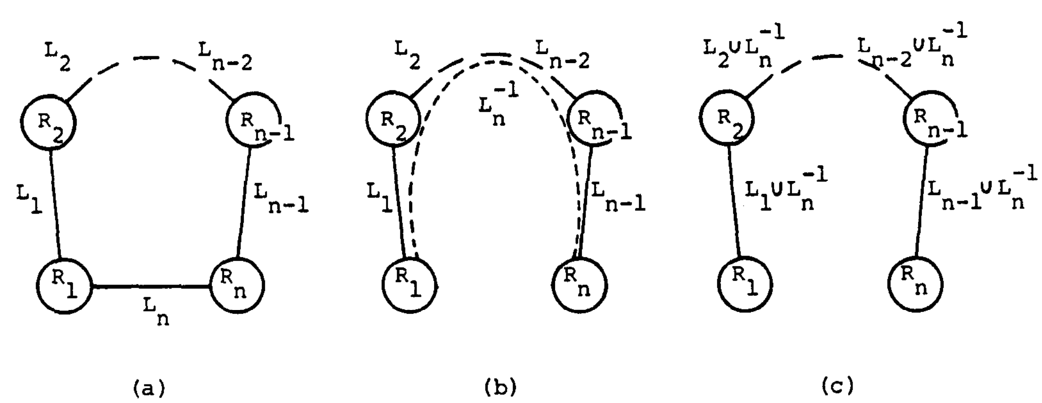


Fig.4.4 Elimination of an edge by embedding into other edges.

follows.

$$L_i' \longleftarrow L_i \cup L_n^{-1} \quad (i=1,2,\ldots,n-1)$$

Also let C_i' be the join qualification corresponding to L_i'. C_i' can be obtained as follows. We assume that $C_n = c_n^1 \wedge c_n^2 \wedge \cdots \wedge c_n^k$ where c_n^h is a join clause $R_1.B_1^h \theta_n^h R_n.A_n^h$ for $h=1,2,\ldots,k$.

(i) $C_i' \longleftarrow C_i$

(ii) for each $l_n^h = ((B_1^h, A_n^h), \theta_n^h)$ in L_n, repeat the following step (iii).

(iii) $C_i' \longleftarrow C_i' \wedge (R_{i+1}.A_n^h \, (\theta_n^h)^{-1} \, R_i.B_1^h)$

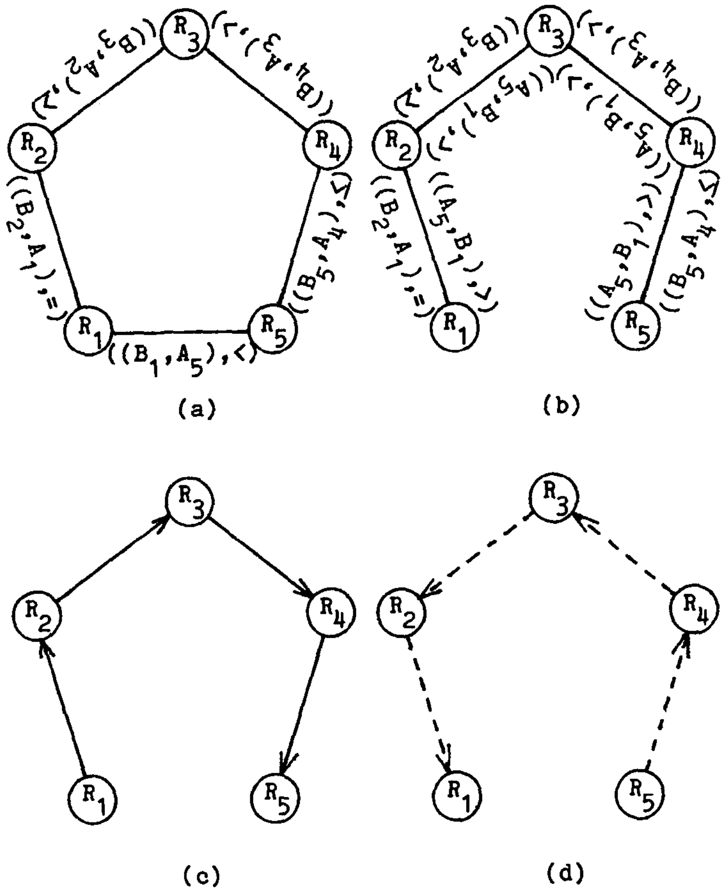
Performing the following generalized semi-join programs along the transformed qual graph, we can obtain the partial solutions of all relations.

$$R_2 \longleftarrow R_2 \ltimes_{C_1'} R_1,\quad R_3 \longleftarrow R_3 \ltimes_{C_2'} R_2,\quad \ldots,\quad R_n \longleftarrow R_n \ltimes_{C_{n-1}'} R_{n-1} \qquad \text{(G1)}$$

$$R_{n-1} \longleftarrow R_{n-1} \ltimes_{C_{n-1}'} R_n,\quad R_{n-2} \longleftarrow R_{n-2} \ltimes_{C_{n-2}'} R_{n-1},\quad \ldots,\quad R_1 \longleftarrow R_1 \ltimes_{C_1'} R_2 \qquad \text{(G2)}$$

By performing the generalized semi-join program (G1), we can obtain a subset of $R_1[C_n]$ with which a tuple $t_i(\in R_i)$ can join along the path $\langle R_i, R_{i-1}, \ldots, R_1\rangle$. Also, by performing the generalized semi-join program (G2), we can obtain a subset of $R_n[C_n]$ with which a tuple $t_i(\in R_i)$ can join along the path $\langle R_i, R_{i+1}, \ldots, R_n\rangle$. We can perform any shuffle of the programs (G1) and (G2) in practice.

[Example 4.2] For the qual graph shown in Fig.4.5(a), we can obtain the qual graph in Fig.4.5(b) by embedding the edge $\langle R_5, R_1\rangle$ into all other edges. Generalized semi-join programs corresponding to (G1) and (G2) above is visualized as Fig.4.5(c) and (d), respectively. Edge labels in Fig.4.5(c) and (d) can also be considered to represent projection qualification. The labels of the edge $\langle R_1, R_2\rangle$ in Fig.4.5(c) (resp. Fig.4.5(d)), for example, imply that $R_1[A_1{=}, B_1{<}]$ (resp. $R_2[B_2{=}, A_5{>}]$) is necessary in the corresponding generalized semi-join in (G1) (resp. (G2)).



(Labels of (c), (d) are same as (b), and are omitted.)

Fig.4.5 Example of the embedding of an edge.

4.4 Comparison of Join Attribute Values of One Relation

In this section, we will describe the method to compare two different sets of join attribute values of one relation with which a tuples in another relation in a cycle joins along the two paths. From Proposition 2.1, the following proposition holds.

[Proposition 4.2] Let q be a simple cyclic query consisting of n relations $R_1, R_2, \ldots, R_n$ (Let us assume that the qual graph G_q consists of a cycle R_1-R_2-...-R_{n-1}-R_n-R_1.). A tuple $t_i(\in R_i)$ is contained in the partial solution of R_i w.r.t. q iff there exist tuples t_n and t_n' both in R_n such that

(i) t_i joins with t_n along the path $\langle R_i, R_{i-1}, \ldots, R_1, R_n\rangle$ in G_q,

(ii) t_i joins with t_n' along the path $\langle R_i, R_{i+1}, \ldots, R_{n-1}, R_n\rangle$ in G_q, and

(iii) for an attribute set X such that $at(C_{n1}, R_n) \subseteq X \subseteq \underline{R}_n$, $t_n[X] = t_n'[X]$ holds.

(Proof) From the conditions (i) and (iii), t_i

joins with also t'_n along the path $<R_i, R_{i-1}, \ldots, R_1, R_n>$ since X includes the join attribute of R_n. This fact and the condition (ii) prove the proposition from Proposition 2.1.

Consider again a query whose qual graph is shown in Fig.4.5(a). Let the join attribute X be A_5. To test whether or not a tuple t_i in R_i is in the partial solution for i=1,2,3,4 and 5 it is sufficient to obtain the set of A_5-values with which t_i joins along the path $<R_i, R_{i-1}, \ldots, R_1, R_5>$ and the set of A_5-values with which t_i joins along the path $<R_i, R_{i+1}, \ldots, R_5>$. The tuple t_i is contained in the partial solution of R_i iff the intersection of these two sets are non-empty. Therefore, adding a label $((A_5, A_5), =)$ to all the edges in the qual graph (see Fig.4.6) and performing generalized semi-joins along the transformed qual graph, all the partial solutions are obtained. Thus, it is sufficient to perform the any shuffle of the following two generalized semi-join programs (G3) and (G4).

$$R_1 \longleftarrow R_1 \ltimes_{C'_5} R_5,\quad R_2 \longleftarrow R_2 \ltimes_{C'_1} R_1,\quad R_3 \longleftarrow R_3 \ltimes_{C'_2} R_2,$$
$$R_4 \longleftarrow R_4 \ltimes_{C'_3} R_3,\quad R_5 \longleftarrow R_5 \ltimes_{C'_4} R_4 \qquad \text{(G3)}$$

$$R_4 \longleftarrow R_4 \ltimes_{C'_4} R_5,\quad R_3 \longleftarrow R_3 \ltimes_{C'_3} R_4,\quad R_2 \longleftarrow R_2 \ltimes_{C'_2} R_3,$$
$$R_1 \longleftarrow R_1 \ltimes_{C'_1} R_2 \qquad \text{(G4)}$$

where
$$C'_1: (R_2.B_2 = R_1.A_1) \wedge (R_2.A_5 = R_1.A_5)$$
$$C'_2: (R_3.B_3 \geq R_2.A_2) \wedge (R_3.A_5 = R_2.A_5)$$
$$C'_3: (R_4.B_4 > R_3.A_3) \wedge (R_4.A_5 = R_3.A_5)$$
$$C'_4: (R_5.B_5 \leq R_4.A_4) \wedge (R_5.A_5 = R_4.A_5)$$
$$C'_5: (R_1.B_1 < R_5.A_5) \wedge (R_1.A_5 = R_5.A_5)$$

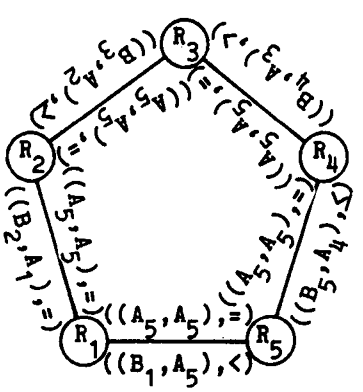


Fig.4.6 Comparison of A_5-values by equality condition.

In the above example, since the comparison operator corresponding to the edge $<R_1, R_5>$ is "<", if a tuple t_i in R_i joins with a tuple $t_5 (\in R_5)$ such that $t_5[A_5] = a_0$ along the path $<R_i, R_{i-1}, \ldots, R_1, R_5>$, t_i joins also with any tuple $t'_5 (\in R_5)$ such that $t'_5[A_5] \geq a_0$. Thus to know the set of A_5-values with which t_i joins along the path $<R_i, R_{i-1}, \ldots, R_1, R_5>$, the information on the minimum value a_m in that set is sufficient. Also, let a_M represent the maximum value in A_5 with which t_i joins along the path $<R_i, R_{i+1}, \ldots, R_5>$. We can say that t_i is included in the partial solution iff $a_m \leq a_M$ (see Fig.4.7.). Therefore any shuffle of the following two generalized semi-join programs (G5) and (G6) can be used in place of (G3) and (G4).

$$R_1 \longleftarrow R_1 \ltimes_{C''_5} R_5,\quad R_2 \longleftarrow R_2 \ltimes_{C''_1} R_1,\quad R_3 \longleftarrow R_3 \ltimes_{C''_2} R_2,$$
$$R_4 \longleftarrow R_4 \ltimes_{C''_3} R_3,\quad R_5 \longleftarrow R_5 \ltimes_{C''_4} R_4 \qquad \text{(G5)}$$

$$R_4 \longleftarrow R_4 \ltimes_{C''_4} R_5,\quad R_3 \longleftarrow R_3 \ltimes_{C''_3} R_4,\quad R_2 \longleftarrow R_2 \ltimes_{C''_2} R_3,$$
$$R_1 \longleftarrow R_1 \ltimes_{C''_1} R_2 \qquad \text{(G6)}$$

where
$$C''_1: (R_2.B_2 = R_1.A_1) \wedge (R_2.A_5^M \geq R_1.A_5^m)$$
$$C''_2: (R_3.B_3 \geq R_2.A_2) \wedge (R_3.A_5^M \geq R_2.A_5^m)$$
$$C''_3: (R_4.B_4 > R_3.A_3) \wedge (R_4.A_5^M \geq R_3.A_5^m)$$
$$C''_4: (R_5.B_5 \leq R_4.A_4) \wedge (R_5.A_5^M \geq R_4.A_5^m)$$

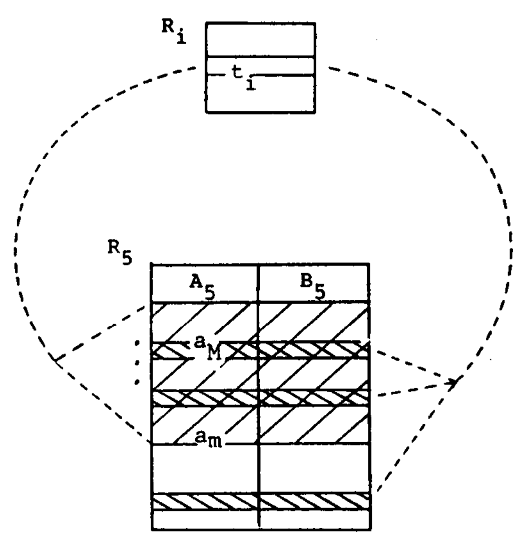


Fig.4.7 Tuples in R_5 which join with t_i.

C_5'': $(R_1.B_1 < R_5.A_5) \wedge (R_1.A_5^M \geq R_5.A_5^m)$

We assume that the relation R_5 is augmented before query processing to include attributes A_5^m and A_5^M of which values are same as A_5. In general, given a qual graph shown in Fig.4.4(a), first transform a label l_n^h in L_n into a label $l_n^{h'}$ as shown in Table 4.1. Then add the label L_n', which is the set of $l_n^{h'}$, to all the label sets in the qual graph as shown below.

$$L_i \longleftarrow L_i \cup L_n' \ (i=1,2,\ldots,n)$$

After these transformations, executing generalized semi-join programs along the two directed paths $\langle R_n, R_1, R_2, \ldots, R_n \rangle$ and $\langle R_n, R_{n-1}, \ldots, R_1 \rangle$ all the partial solutions can be obtained.

l_n^h	$l_n^{h'}$
$((B_1, A_n), =)$	$((A_n^M, A_n^m), =)$
$((B_1, A_n), \leq)$	$((A_n^M, A_n^m), \geq)$
$((B_1, A_n), <)$	$((A_n^M, A_n^m), \geq)$
$((B_1, A_n), \geq)$	$((A_n^M, A_n^m), \leq)$
$((B_1, A_n), >)$	$((A_n^M, A_n^m), \leq)$

Table 4.1 Labels to be added.

4.5 A Strategy for Processing General Weak 1-doublet Queries

In this section, we will give a strategy for processing general weak 1-doublet queries using an example. The processing strategy given here is based on the same idea as presented in [KAMBY8206]. Given a cyclic qual graph G_q, first a spanning tree T is chosen. Next each edge e in G_q-T is "embedded" in the edges $e_1, e_2, \ldots, e_k$ in T such that the addition of e to T yields a cycle consisting of $e, e_1, e_2, \ldots, e_k$. Then a node in G_q is selected as a root of T. By performing generalized semi-joins along the resultant T first from leaf to root and then from root to leaf, partial solutions of all relaions are obtained. For example, consider a query whose qual graph is shown in Fig.4.8(a). By selecting the spanning tree T shown by the bold-line edges in Fig.4.8(b), and embedding edges e($\in G_q$ - T) in T, a tree qual graph shown in Fig.4.8(c) is obtained. Performing generalized semi-joins along this tree just like the semi-join program UP•DOWN, all the partial solutions are obtained.

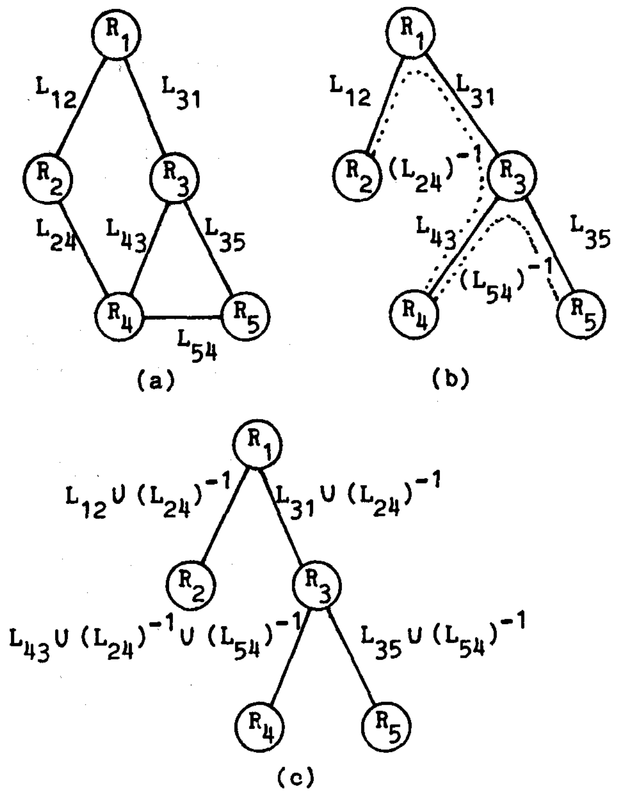


Fig.4.8 Embedding edges in a sapnning tree.

5. Conclusion

In this paper, we have presented processing algorithms for NI queries whose qual graphs contain
(1) Multiple edges,
(2) 1-doublet cycles, or
(3) 0-doublet cycles.
To handle the case (1), we have introduced multi-doublet inequality semi-joins which are natural extension of ordinary (single-attribute) inequality semi-joins. Although the NJ queries processed by multi-attribute natural semi-joins are strictly characterized by tree queries [BERNG8111], the characterization of the class of queries processed by multi-attribute inequality semi-joins are not known. Since the power of multi-attribute semi-joins are strictly stronger than that of single-attribute semi-joins, some weak 1-doublet queries can be solved using multi-attribute semi-joins only.

We have obtained the following results related to this paper.
(1) Sufficient conditions for qual graphs to be processed using multi-attribute semi-joins only.
(2) Efficient procedures utilizing the property of doublets for processing 1-doublet cycles.
(3) Effective data compression methods which reduce the data transmission cost significantly in distributed databases or database machines.

Acknowledgments

The authors are grateful to Professor Shuzo Yajima and Dr. Hiroto Yasuura of Kyoto University and other members of Yajima Lab. for helpful discussions. Their heartful thanks are also due to Professor Tim Merrett of McGill University for useful comments for the presentation of this paper.

References

[BERNC8101] Bernstein,P.A. and Chiu,D.M., "Using Semi-Joins to Solve Relational Queries", JACM, Vol.28, No.1, pp,25-40, Jan. 1981.

[BERNG7912] Bernstein,P.A. and Goodman,N., "Inequality Semi-Joins", CCA Report, No.CCA-79-28, Dec.15, 1979.

[BERNG81] Bernstein,P.A. and Goodman,N., "The Power of Inequality Semijoins", Information Systems, Vol.6, No.4, pp.255-265, 1981.

[BERNG8111] Bernstein,P.A. and Goodman,N., "Power of Natural Semijoins", SIAM J. Comput., Vol.10, No.4, pp.751-771, Nov. 1981.

[KAMBY8206] Kambayashi,Y., Yoshikawa,M. and Yajima,S., "Query Processing for Distributed Databases Using Generalized Semi-Joins.", Proc. ACM SIGMOD Intl. Conf. on Management of Data, pp.151-160, June 1982.

[KAMBY8305] Kambayashi,Y. and Yoshikawa,M., "Query Processing Utilizing Dependencies and Horizontal Decomposition", Proc. ACM SIGMOD Intl. Conf. on Management of Data, pp.55-67, May 1983.