An Efficient Strategy for Beneficial Semijoins [*]

Ye-In Chang, Bor-Miin Liu, and Cheng-Huang Liao
Department of Applied Mathematics
National Sun Yat-Sen University
Kaohsiung, Taiwan, R.O.C.
{E-mail: changyi@math.nsysu.edu.tw}
{Tel: 886-7-5252000 (ext. 3819)}
{Fax: 886-7-5253809}

ABSTRACT

Semijoins have traditionally been applied for reducing the communication cost required for distributed query processing. Semijoins whose execution will reduce the amount of data transmission required to perform a join sequence are termed *beneficial semijoins* for that join sequence. Beneficial semijoins include conventional *profitable semijoins* and *gainful semijoins* that are not profitable themselves but become beneficial due to the inclusion of join reducers. In this paper, based on the values of *dynamic cumulative benefit* (DCB), we propose an efficient algorithm for finding beneficial semijoins, i.e., to interleave a sequence of join operations with semijoins to reduce the data transmission cost. In this algorithm, a *dynamic weighted graph* is constructed based on the *correlated* relationship among semijoins where two semijoins are said to be *correlated* with each other if the condition for one to be beneficial depends on execution of the other. Then, we compute the *dynamic profit* for each subgraph, which is recursively constructed. When there are N vertexes in the initial *dynamic weighted graph*, where each vertex represents a semijoin, our algorithm needs to expand (N + 1) graphs to find a solution in the best case. From our simulation study, we show that our strategy can efficiently find beneficial semijoins and requires a lower data transmission cost than does the profitable semijoin approach.

(**Key Words:** distributed databases, query optimization, relational databases, semijoins)

[*]This research was supported by the National Science Council of the Republic of China, Grant No. NSC 84-2213-E-110-023.

1 Introduction

In a wide area network, under the assumptions that each site contains one relation, that there is only one copy of each relation, and that the cost of local processing is negligible compared to the transmission cost, a query is usually processed in the following three phases [16]: (1) *the local processing phase,* which involves all local processing such as selections and projections, (2) *the reduction phase,* where a sequence of *semijoins* is used to reduce the size of relations, and (3) *the final processing phase,* in which all the resulting relations are sent to the site where final query processing is performed. Significant research efforts have been focused on the problem of reducing the amount of data transmission required for phases (2) and (3) of distributed query processing [1, 3, 13, 14]. The *semijoin* operation especially has received considerable attention and has been extensively studied in the literature [2, 5, 15, 9]. The *semijoin* operator takes the join of two relations, R and S, and then projects back out on the domains of relation R. For a semijoin to be performed, only the projection of the joining attribute need be sent. If the size of these projections is small relative to the amount by which R and S are reduced, then the preliminary semijoin will be *profitable*.

The first algorithm using semijoins for distributed query processing was implemented in SDD-1 in [3]. This SDD-1 algorithm is based on a *hill-climbing* strategy that produces efficient, but not necessarily optimal, query processing strategies. Theoretical aspects of semijoins were first studied in [2]. *Simple queries* were studied in [13]. Their algorithm for general queries was improved in [1]. It has been proved that a *tree query* can be fully reduced by using semijoins [2], and there has been much research on optimizing semijoin sequences to process certain tree queries, such as *star queries* [6] and *chain queries* [11]. However, the determination of an optimal semijoin sequence to process certain tree queries and general query graphs with cycles has been proved to be NP-hard [12]. Methods based on *dynamic programming* to get an optimal semijoin sequence for *tree queries* and *chain queries* were studied in [10] and [11], respectively.

In addition to semijoins, join operations can also be used as reducers in distributed query processing to further reduce the communication cost [4, 7, 8]. Moreover, the approach of combining join and semijoin operations as reducers can result in more beneficial semijoins due to the inclusion of joins as reducers [7]. (Note that such semijoins are referred to as *gainful semijoins.*) Both *profitable semijoins* and *gainful semijoins* are called *beneficial semijoins.*

In this paper, based on the values of the *dynamic cumulative benefit* (DCB), we propose an algorithm for finding beneficial semijoins, i.e., to interleave a sequence of given join operations with semijoins to reduce the total data transmission cost. In this algorithm, a *dynamic weighted graph* $G = (V, E)$ is constructed based on the *correlated* relationship among semijoins, where V is the set of semijoins (i.e., each vertex represents a semijoin) associated with its *dynamic cumulative benefit* (DCB) and E is the set of *correlated edges.* Note that two semijoins are said to be *correlated* with each other if the condition for one to be beneficial depends on execution of the other [8]. Based on the values of DCB, there is a *dynamic profit* associated with the whole graph. Then, the graph is transformed to another graph by a *vertex shrinking.* A *vertex shrinking* is done by eliminating a certain vertex and connecting *correlated edges.* This transformation is recursively executed and the related *dynamic profit* is updated at the same time until all vertexes are shrunken or all the values of DCB associated with vertexes are negative. When the algorithm stops, the set of semijoins which provide the maximum total *dynamic profit* is the set of beneficial semijoins. When there are N vertexes in the initial *dynamic weighted graph*, where each vertex represents a semijoin, our algorithm needs to expand (N + 1) graphs to find a solution in the best case. Moreover, since the DCB values of vertexes will be dynamically updated in the process of a *vertex shrinking*, it will speed up the execution and reduce the large space requirement of the proposed recursive algorithm in general cases. From our simulation study, we show that our strategy can efficiently find beneficial semijoins and requires a lower data transmission cost than does the profitable semijoin approach.

The rest of the paper is organized as follows. In Section 2, we give some definitions used in this paper. In Section 3, we present our proposed algorithm. Finally, in Section 4, we give a conclusion.

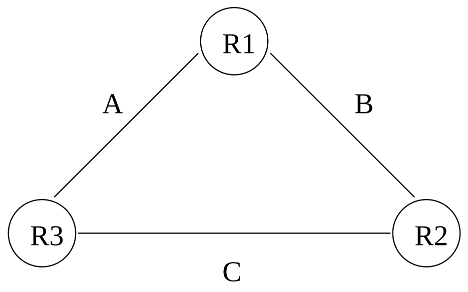

Figure 1: A query graph q_A.

2 Background

In this section, we describe assumptions and definitions used in the paper, which are mostly adapted from [7, 8] since the concept and properties of *gainful semijoins* and *beneficial semijoins* were proposed and discussed in [7, 8].

2.1 Query Graphs, Joins and Semijoins

Given a query Q with qualification q, we define its corresponding *query graph* $G_Q(V_Q, E_Q)$ as follows:

V_Q = the set of all the relation names referenced by q;

$E_Q = \{(i,\ j) \mid i \neq j$ and some clause of q references both R_i and $R_j\ \}$.

Figure 1 shows the query graph for the following query, where $R_1.B$ is the target list.

select $R_1.B$

from R_1, R_2, R_3

where $R_1.A = R_3.A$ and $R_1.B = R_2.B$ and $R_2.C = R_3.C$

We assume that we have the following information about the relations.

For each relation R_i, i=1,2,...,m,

$|R_i|$: number of tuples;

w_{R_i}: size (e.g., in bytes) of R_i.

For each attribute A of relation R_i,

$|R_i(A)|$: cardinality;

$\rho_{i,A}$: selectivity;

$w_{R_i(A)}$: size (e.g., in bytes) of the data item in attribute A of relation R_i.

The *cardinality* of attribute A of relation R_i, denoted as $|A|$, is the number of distinct values in attribute A of relation R_i and the *selectivity* $\rho_{i,A}$ of attribute A is defined as the number of different values occurring in the attribute divided by the number of all possible values of the attribute.

A join clause "R_1 *joins* R_2 *on* A" is denoted by $R_1 \xleftrightarrow{A} R_2$, where R_1 and R_2 are relations, and attribute A is the joining attribute. Associated with this join are two semijoins: R_1 by R_2 on A, and R_2 by R_1 on A, denoted by $R_2 \xrightarrow{A} R_1$, and $R_1 \xrightarrow{A} R_2$, respectively. $R_1 \xrightarrow{A} R_2$ entails shipping $R_1(A)$, attribute A of R_1, to the site where R_2 resides and joining $R_1(A)$ with R_2. We denote the resulting relation by R_2' (and R_1 is unchanged). After the semijoin $R_i \xrightarrow{A} R_j$ is executed, then the parameters of relation R_j are changed in the following way:

$$|R_j| \longleftarrow |R_j| * \rho_{i,A};$$
$$\rho_{j,A} \longleftarrow \rho_{j,A} * \rho_{i,A};$$
$$|R_j(A)| \longleftarrow |R_j(A)| * \rho_{i,A}.$$

2.2 Cost and Benefit of Semijoin Reducers

We assume that the transmission cost is given by $\text{cost}(n) = c_0 + c_1 * n$, where n is the amount of data transmitted and c_0 and c_1 are constants. Let the transmission cost be one per data unit transmitted. Consider a semijoin $R_i \xrightarrow{A} R_j$ when R_i and R_j are at different sites. Then, the *cost* of the semijoin is ($size\_of\ R_i[A]$), and the *benefit* is ($size\_of\ R_j\ before\ semijoin\ -\ size\_of\ R_j\ after\ semijoin$), where the sizes of the relations are measured in bytes. A semijoin $R_i \xrightarrow{A} R_j$, is called *profitable* if its cost of sending $R_i(A)$, $w_{R_i(A)}|R_i(A)| = w_{R_i(A)}|A|\rho_{i,A}$, is less than its benefit, $w_{R_j}|R_j| - w_{R_j}|R_j|\rho_{i,A}$ $= w_{R_j}|R_j|(1-\rho_{i,A})$, where $w_{R_j}|R_j|$ and $w_{R_j}|R_j|\rho_{i,A}$ are the size of R_j before and after the semijoin, respectively. In addition, we assume that the values of attributes are uniformly distributed over all the tuples in a relation, and that the values of one attribute are independent of those in another attribute. Thus, as in most prior work [1, 2], we assume in this paper that the selectivity and the cardinality of a non-semijoin attribute remain the same after a semijoin operation to simplify our discussion.

2.3 The Effect of Join Operations

To determine the effort of a join operation specified by a query graph, the following theorem was described in [7].

Theorem 1 *Let $G_J = (V_J, E_J)$ be a join query graph. $G_B = (V_B, E_B)$ is a connected subgraph of G_J. Let $R_1, R_2, ..., R_p$ be the relations corresponding to nodes in V_B, let $A_1, A_2, ..., A_q$ be the distinct attributes associated with edges in E_B, and let m_i be the number of different nodes (relations) to which edges with attribute A_i are incident. Suppose R^* is the relation resulting from the join operations between relations in G_B, and that $N_T(G_B)$ is the expected number of tuples in R^*; then*

$$N_T(G_B) = \frac{\prod_{i=1}^{p} |R_i|}{\prod_{i=1}^{q} |A_i|^{m_i - 1}} \quad . \tag{1}$$

For the query shown in Figure 2, the expected number of tuples in the resulting relation is $(|R_1||R_2||R_3||R_4|)/(|A|^2|B||C||D|)$.

2.4 Join Reducers and Gainful Semijoins

The application of join operations as reducers may mean that more profitable semijoins will be available. Those semijoins which become profitable due to the use of join reducers are termed *gainful semijoins* [7]. Consider the query graph q_A shown in Figure 1 with its profile in Table 1 as an example. It can be verified that the semijoin $R_3 \xrightarrow{A} R_1$ is not profitable since $w_{R_3(A)}|R_3(A)| > w_{R_1}(1 - \rho_{3,A})|R_1|$. Note that although this semijoin is not profitable, it is *gainful* if we perform $R_1 \Rightarrow R_2$ and $R_2' \Rightarrow R_3$ after this semijoin operation, where $R_1 \Rightarrow R_2$ means shipping R_1 to the site where R_2 resides and joining R_1 with R_2. It can be obtained that for the total communication costs required,

$$|R_3(A)| + 2|R_1|\rho_{3,A} + 3|R_1 \ join \ R_2|\rho_{3,A} \approx 2190 < 2|R_1| + 3|R_1 \ join \ R_2| = 2542,$$

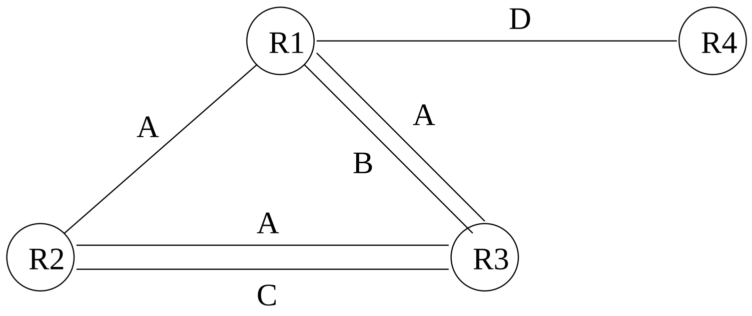

Figure 2: A query graph q_B (adapted from [8]).

meaning that it is advantageous, as far as the cost of data transmission is concerned, to perform $R_3 \xrightarrow{A} R_1$, $R_1' \Rightarrow R_2$ and then $R_2' \Rightarrow R_3$, instead of performing directly $R_1 \Rightarrow R_2$ and $R_2' \Rightarrow R_3$. Thus, it can be seen that whether a semijoin is gainful or not depends on the subsequent join operations.

Relation R_i	$\lvert R_i \rvert$	Size $w_{R_i}\lvert R_i \rvert$	Attribute X	$\lvert R_i(X) \rvert$	Selectivity $\rho_{i,x}$	W_X
R_1	620	1240	A	400	0.80	1
			B	600	0.60	1
R_2	700	1400	B	580	0.58	1
			C	450	0.75	1
R_3	778	1556	A	360	0.72	1
			C	480	0.80	1

Table 1: Profile for query q_A, where $|A| = 500$, $|B| = 1000$, and $|C| = 600$ (adapted from [7]).

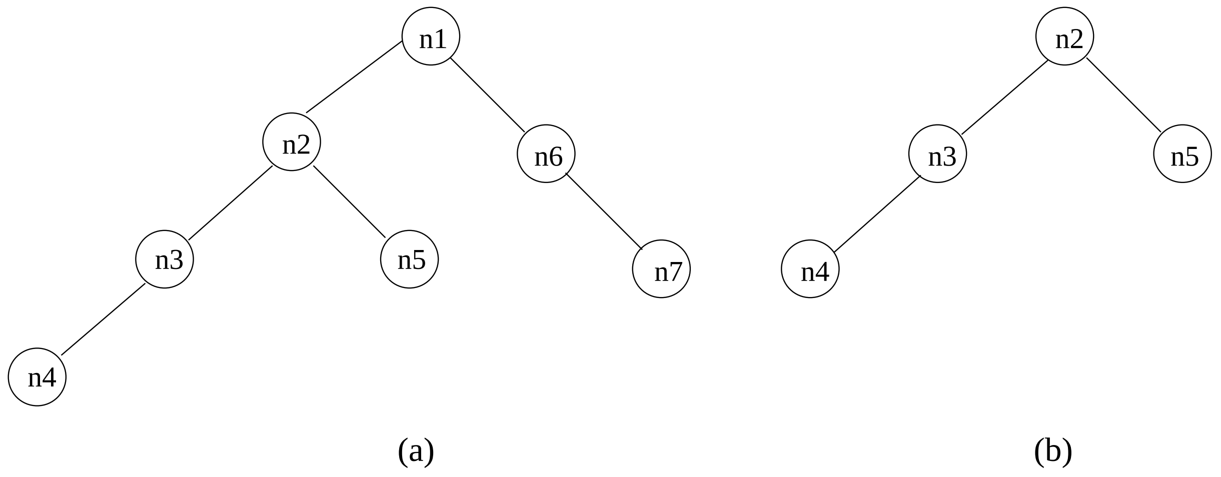

Figure 3: A rooted tree: (a) T; (b) T_{n_2} (adapted from [7]).

2.5 A Join Sequence Tree

Every edge in a tree is directed, and all the arrows in edges are away from a single node, which is called the *root* of the tree [7]. Note that a rooted tree can be viewed as a partial order set. We denote $n_i \geq n_j$, if there is a path along the arrows in the tree from n_i to n_j. In such a case, node n_j (n_i) is called an *offspring* (*ancestor*) of n_i (n_j). We use $n_i > n_j$ to mean $n_i \geq n_j$ and $n_i \neq n_j$. Let T_{n_i} denote the subtree formed by n_i and its offspring (ancestor) in a rooted tree T, and let S(T_{n_i}) be the set of nodes in T_{n_i}, i.e., $S(T_{n_i}) = \{n_j \mid n_i \geq n_j, \ n_j \in S(T)\}$. We define the *lowest common ancestor* of two nodes n_i and n_j in a rooted tree, denoted by $n_i \vee n_j$, to be the node that is an ancestor of n_i and n_j and for which none of its offspring is an ancestor of n_i and n_j [7].

For example, for a rooted tree T shown in Figure 3-(a), T_{n_2} is given in Figure 3-(b), and $S(T_{n_2}) = \{n_2, \ n_3, \ n_4, \ n_5\}$. Also, $n_4 \vee n_5 = n_2$ and $n_5 \vee n_7 = n_1$ in Figure 3-(a). In addition, when $n_i \geq n_j$ in a rooted tree T, we use $P(n_i, \ n_j)$ to denote the set of nodes that are on the path from n_i to n_j excluding n_i, i.e., $P(n_i, \ n_j) = \{n_k \mid n_i > n_k \geq n_j \ and \ i \neq k, \ \forall n_k \in S(T)\}$. In the rooted tree shown in Figure 3-(a), $P(n_2, \ n_4) = \{n_3, \ n_4\}$ and $P(n_1, \ n_5) = \{n_2, \ n_5\}$.

A join sequence tree is obtained from a join sequence [7]. Once a join sequence is determined, it can be mapped into its corresponding join sequence tree, which is defined as follows [7].

A join sequence tree is a rooted tree where each node denotes a relation and each edge implies a join between the two relations to which the edge is incident. The tree represents a sequence of join operations which are performed in such a way that each relation in a node is sent to its parent node in the tree for a join operation in the bottom-up sense.

Given a query shown in Figure 4-(a) and a join sequence $R_4 \Rightarrow R_7$, $R_5 \Rightarrow R_7$, $R_7 \Rightarrow R_6$, $R_3 \Rightarrow R_6$, $R_1 \Rightarrow R_2$, $R_6 \Rightarrow R_2$, the corresponding join sequence tree is shown in Figure 4-(b). Recall that T_{R_i} is the subtree formed by R_i and its offspring in the join sequence tree, and that $S(T_{R_i})$ is the set of nodes in T_{R_i}. The *weight* of a relation R_i in the join sequence tree, denoted by $W(R_i)$, is defined as the size of the relation resulting from joining all the relations in $S(T_{Ri})$ (and is computed by Equation 1 as described in Section 2.3). For the join sequence tree shown in Figure 4-(b), $W(R_7) = w_{R'_7}|R'_7|$ and $W(R_6) = w_{R'_6}|R'_6|$, where R'_7 is the relation resulting from joining R_4, R_5, and R_7, and R'_6 is the one resulting from joining R_3, R_4, R_5, R_6, and R_7. For convenience, the weight of the root of a join sequence tree, which corresponds to the final site, is defined to be zero. Also, to facilitate our study on the effect of semijoin operations, we define the configuration of a query, $J_Q(SMJ)$, to be the structure of the query and its profile associated after the set of semijoins SMJ has been performed. When it is necessary, we use $W(R_i, J_Q(SMJ))$, instead of $W(R_i)$, to mean the weight of R_i after the semijoins in SMJ are performed.

2.6 Properties of Beneficial Semijoins

A relation is said to be *reducible* by a semijoin SJ_i if the size of the relation in the join sequence tree is affected by the execution of the semijoin. Then, the set of reducible relations of a semijoin under a join sequence tree can be determined by the following theorems [7]:

Theorem 2 *Given a join sequence tree T, the set of reducible relations of a semijoin $R_i \longrightarrow R_j$, denoted by $Rd(R_i \longrightarrow R_j)$, is $P(R_i \vee R_j, R_j)$.*

For example, suppose Figure 4-(b) is the join sequence tree derived from Figure 4-(a); then, $Rd(R_1 \longrightarrow R_4) = \{R_4, R_6, R_7\}$, $Rd(R_4 \longrightarrow R_3) = \{R_3\}$ and $Rd(R_2 \longrightarrow R_3) =$

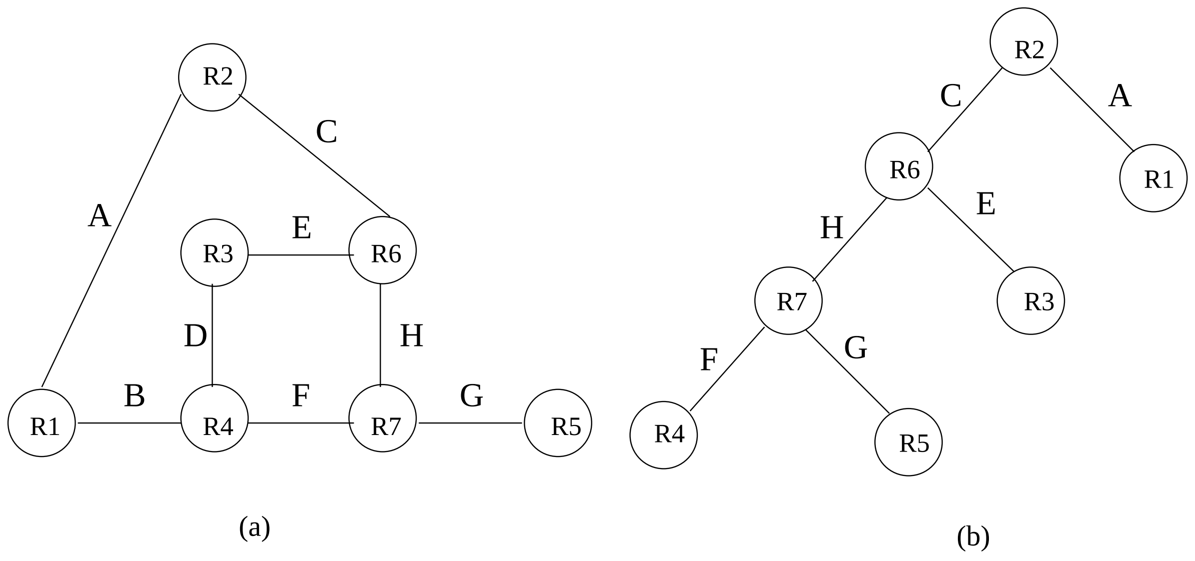

Figure 4: An example: (a) a query graph G_{EX1}; (b) the related join sequence tree (adapted from [7]).

$\{R_3, R_6\}$.

Theorem 3 *A semijoin SJ_k, $R_i \xrightarrow{A} R_j$ in the configuration $J_Q(SMJ)$ is beneficial if and only if $w_{R_i(A)}|R_i(A)| \leq (1-\rho_{i,A})\sum_{R_p \in Rd(SJ_k)} W(R_p, J_Q(SMJ))$.*

Corollary 1 *Suppose that R_i and R_j are two relations in a join sequence tree T and $R_i \geq R_j$. Then, $R_j \longrightarrow R_i$ is not a beneficial semijoin for T.*

Two semijoins are said to be *correlated* with each other if the condition for one to be beneficial depends on execution of the other. Thus, using **Theorem 3**, we can determine by the following corollary [7] whether two semijoins are correlated with each other in a join sequence tree.

Corollary 2 *In a join sequence tree, two semijoins, SJ_i and SJ_k, are correlated with each other if and only if $Rd(SJ_i) \cap Rd(SJ_k) \neq \emptyset$.*

3 The Algorithm for Beneficial Semijoins

Given a join sequence, we can map the join reducer sequence into the corresponding join sequence tree. According to **Theorem 2**, we can derive reducible relations from the join sequence tree. Based on the relationship among these reducible relations, we propose a new algorithm for interleaving a sequence of join operations with semijoins, i.e., for locating beneficial semijoins. The proposed algorithm is based on a value, called the *dynamic cumulative benefit*, which will be defined later.

Let SM_T be the set of possible semijoins in the given join sequence tree T except those semijoins $R_j \longrightarrow R_i$ which occur between two relations R_i and R_j in the join sequence tree T and $R_i \geq R_j$ (i.e., R_i is an ancestor of R_j). (Note that, based on Corollary 1, we do not want to include these non-beneficial semijoins in SM_T.) Let SMJ be the set of beneficial semijoins identified so far. Initially, SMJ is an empty set. We define the *dynamic cumulative benefit* of a semijoin SJ_i ($R_k \xrightarrow{A} R_j$), denoted by $DCB(SJ_i)$, as the amount of reduction minus the cost of semijoin SJ_i if this semijoin is applied to the execution of a given join sequence and the profile of the semijoin is the one resulting from the semijoin executions SJ_k, for $SJ_k \in SMJ$. That is, $DCB(SJ_i) = DCB(R_k \xrightarrow{A} R_j) = (1 - \rho_{k,A}) \sum_{R_p \in Rd(SJ_i)} W(R_p, J_Q(SMJ)) - Cost_i$. Given the query graph shown in Figure 4-(a) with its profile shown in Table 2, Table 3 shows SM_T and the value of DCB for each semijoin.

For all semijoins in SM_T, the sets of reducible relations of semijoins can be further classified according to whether the semijoin is correlated with some other semijoins or not based on **Corollary 2**. We assign those semijoins which are correlated to the same group. For example, given a query graph shown in Figure 4-(a), Figure 4-(b) shows the related join sequence tree, and Tables 4-(a), (b), and (c) shows three groups of semijoins which belong to SM_T and are correlated in the same group. Moreover, some semijoins in SM_T may not be correlated with some other semijoins. If such a semijoin exists and it is beneficial (i.e., $DCB > 0$), we add such a semijoin into SMJ.

For those semijoins which are in the same group, we want to find a good combination of beneficial semijoins such that we can obtain the largest profit and then add them into SMJ. For each group GP_i of semijoins SJ_k with $DCB(SJ_k) > 0$, we construct a *dynamic weighted graph*. Given a GP_i, a *dynamic weighted graph* $G = (V_{GP_i},\ E_{GP_i})$ is constructed based on the *correlated* relationship among the semijoins, where V_{GP_i} is the set of semijoins with positive values of DCB (i.e., each vertex represents a semijoin) associated with its

Relation R_i	$\|R_i\|$	Size of relation	Attribute X	$\|R_i(X)\|$	Selectivity	W_X
R_1	1150	2300	A	420	0.70	1
			B	325	0.65	1
R_2	1200	2400	A	300	0.50	1
			C	385	0.55	1
R_3	850	1700	D	585	0.65	1
			E	550	0.55	1
R_4	1100	3300	B	300	0.60	1
			D	405	0.45	1
			F	770	0.55	1
R_5	900	900	G	585	0.45	1
R_6	900	2700	C	490	0.70	1
			E	400	0.40	1
			H	525	0.50	1
R_7	1000	3000	F	630	0.45	1
			G	650	0.50	1
			H	630	0.60	1

Table 2: Profile for query graph G_{EX1}, where $|A| = 600$, $|B| = 500$, $|C| = 700$, $|D| = 900$, $|E| = 1000$, $|F| = 1400$, $|G| = 1300$, and $|H| = 1050$.

Semijoin SJ_i	in SM_T	$DCB(SJ_i)$
$R_1 \rightarrow R_2$	N	-
$R_1 \rightarrow R_4$	Y	2753
$R_2 \rightarrow R_1$	Y	850
$R_2 \rightarrow R_6$	Y	863
$R_3 \rightarrow R_4$	Y	1522
$R_3 \rightarrow R_6$	N	-
$R_4 \rightarrow R_1$	Y	620
$R_4 \rightarrow R_3$	Y	530
$R_4 \rightarrow R_7$	N	-
$R_5 \rightarrow R_7$	N	-
$R_6 \rightarrow R_2$	N	-
$R_6 \rightarrow R_3$	Y	620
$R_6 \rightarrow R_7$	Y	835
$R_7 \rightarrow R_4$	Y	1185
$R_7 \rightarrow R_5$	Y	-200
$R_7 \rightarrow R_6$	N	-

Table 3: SM_T for query graph G_{EX1}.

No.	**Semijoin** (SJ_i)	**Reducible relations**
1	$R_1 \longrightarrow R_4$	$\{ R_4, R_6, R_7 \}$
2	$R_2 \longrightarrow R_6$	$\{ R_6 \}$
3	$R_3 \longrightarrow R_4$	$\{ R_4, R_7 \}$
4	$R_6 \longrightarrow R_7$	$\{ R_7 \}$
5	$R_7 \longrightarrow R_4$	$\{ R_4 \}$

(a)

No.	**Semijoin** (SJ_i)	**Reducible relations**
1	$R_6 \longrightarrow R_3$	$\{ R_3 \}$
2	$R_4 \longrightarrow R_3$	$\{ R_3 \}$

(b)

No.	**Semijoin** (SJ_i)	**Reducible relations**
1	$R_4 \longrightarrow R_1$	$\{ R_1 \}$
2	$R_2 \longrightarrow R_1$	$\{ R_1 \}$

(c)

Table 4: Four groups in SM_T: (a) GP_1; (b) GP_2; (c) GP_3.

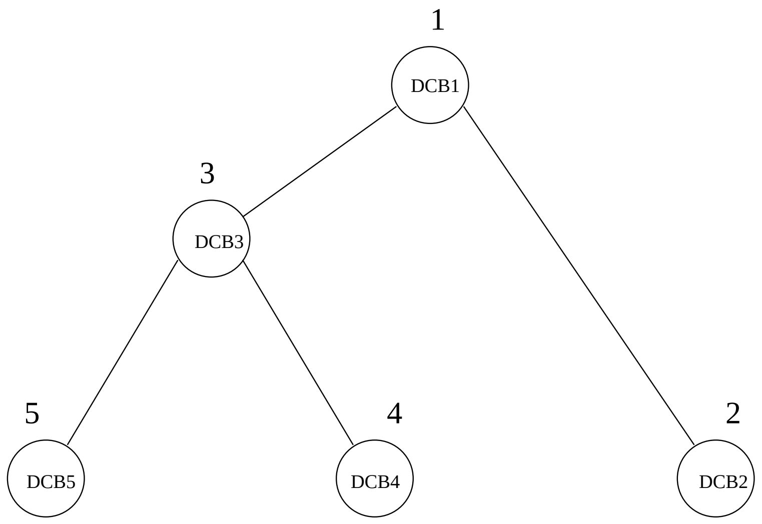

Figure 5: A dynamic weighted graph for GP_1.

dynamic cumulative benefit (DCB), and E_{GP_i} is the set of *correlated edges*. *A correlated edge* exists between two vertexes v_i (denoting SJ_i) and v_j (denoting SJ_j) such that the intersection of the sets of reducible relations of these two semijoins represented by v_i and v_j is not empty. As an example, Figure 5 shows the dynamic weighted graph for group GP_1 shown in Table 4-(a).

Given the dynamic weighted graph G, G_p is a graph constructed from G by eliminating vertex p and connecting correlated edges (p, x) and (p, y), where the intersection of the sets of reducible relations of these two vertexes x and y (representing two semijoins) is not empty. The step of constructing G_p from G is called a *vertex shrinking*. Figure 6 shows two examples of vertex shrinking for the dynamic weighted graph shown in Figure 5. After the process of *vertex shrinking*, the DCB values of those vertexes which are correlated with the shrunken vertex in the group are also updated. (Note that the DCB values of the vertexes will be the same or be smaller in each update.) We define the *dynamic profit* (denoted as $DP(G)$) of the most profitable set of semijoins for a given dynamic weighted graph G as follows:

$$DP(G) \;=\; \max_{p\in G}\{DP(G_p) \;+\; DCB_p\},$$

where we always consider positive values of DCB only. (Note that when the DCB value

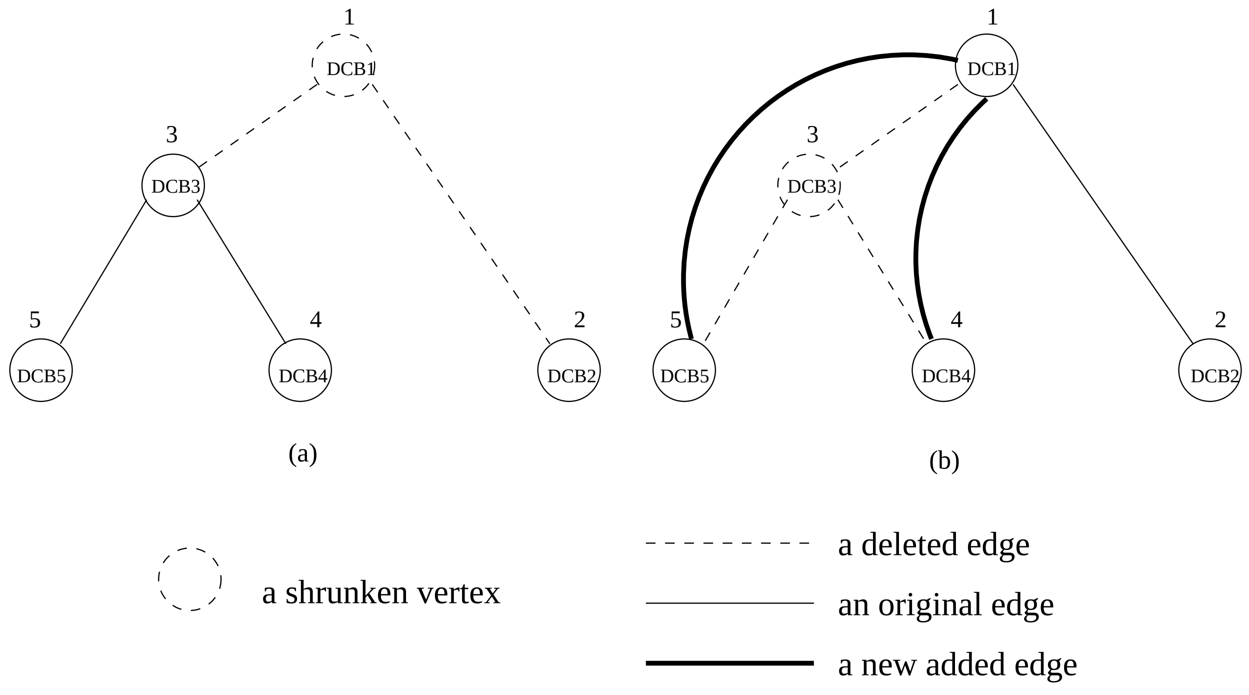

Figure 6: A vertex shrinking: (1) vertex 1; (2) vertex 2.

of a vertex is negative, we will give up the vertex shrinking for this vertex in this G_p.) For group GP_1, according to the above formula, we illustrate the process of finding the largest profit $DP(G)$ from Figure 7-(a) to Figure 7-(m).

$$\begin{aligned} DP(G) &= \max\{DP(G_1)+2753,\ DP(G_2)+863,\ DP(G_3)+1522, \\ &\quad DP(G_4)+835,\ DP(G_5)+1185\}; \\ DP(G_2) &= \max\{DP(G_{2_1})+2317,\ DP(G_{2_3})+1522,\ DP(G_{2_4})+835, \\ &\quad DP(G_{2_5})+1185\}; \\ DP(G_{2_1}) &= \max\{DP(G_{2_{1_3}})+785,\ DP(G_{2_{1_4}})+359,\ DP(G_{2_{1_5}})+550\}; \\ DP(G_{2_{1_3}}) &= 137;\ DP(G_{2_{1_4}}) = 550;\ DP(G_{2_{1_5}}) = 32. \end{aligned}$$

Thus

$$DP(G_{2_1}) = 922, \text{ and } DP(G_2) = 3239.$$

Consequently,

$$DP(G) = \max\{\ 3229,\ 3239,\ 3051,\ 2960,\ 2081\ \} = 3239.$$

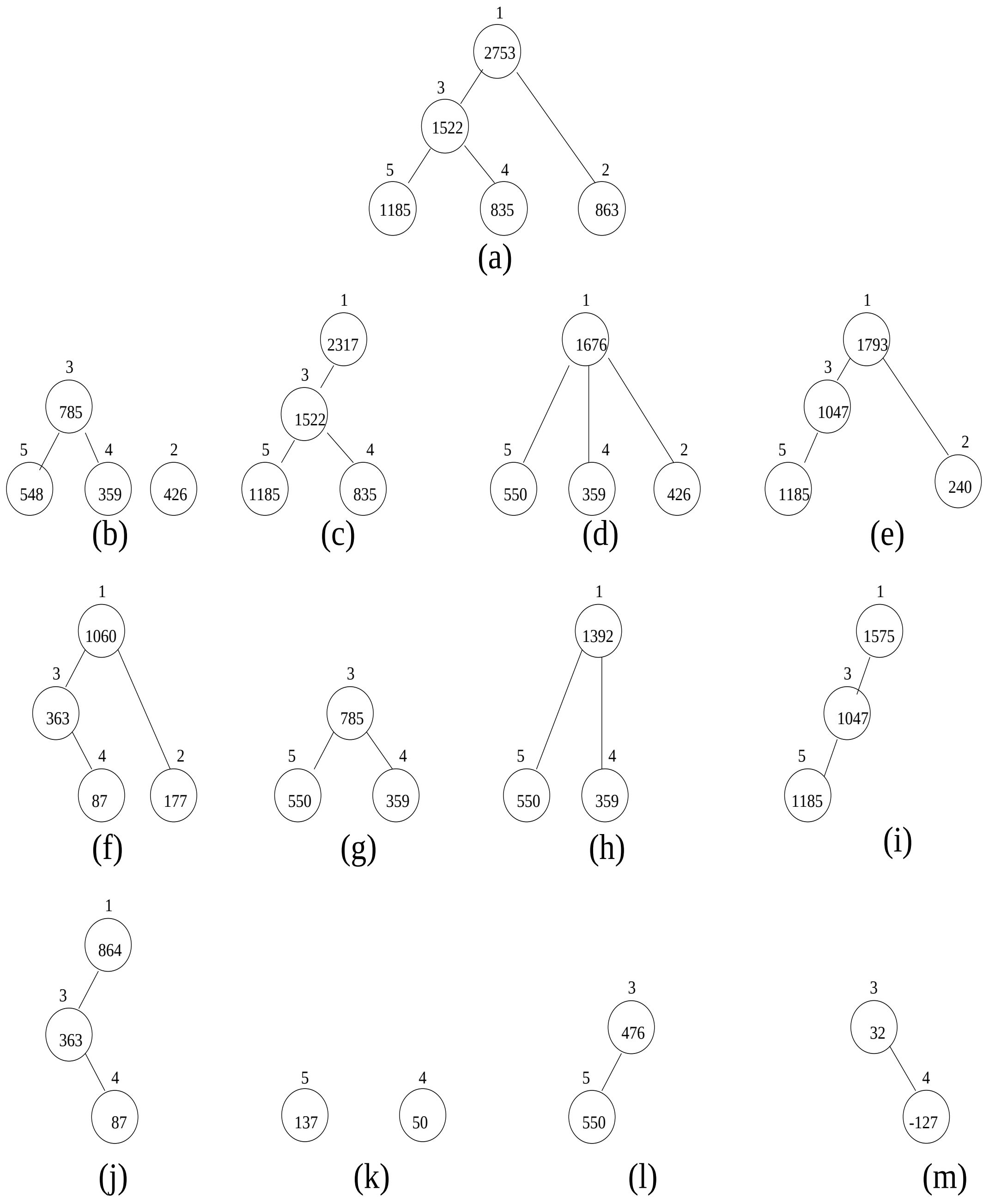

Figure 7: A dynamic weighted graph and its subgraphs: (a) G; (b) G_1; (c) G_2; (d) G_3; (e) G_4; (f) G_5; (g) G_{2_1}; (h) G_{2_3}; (i) G_{2_4}; (j) G_{2_5} (k) $G_{2_{1_3}}$; (l) $G_{2_{1_4}}$; (m) $G_{2_{1_5}}$.

That is,

$$\begin{aligned} DP(G) &= DP(G_2) + DCB_2 \\ &= DP(G_{2_1}) + DCB_1 + DCB_2 \\ &= DP(G_{2_{1_3}}) + DCB_3 + DCB_1 + DCB_2 \\ &= DCB_5 + DCB_3 + DCB_1 + DCB_2 \end{aligned}$$

Therefore, for group GP_1 shown in Table 4-(a), the most profitable set of semijoins (i.e., SMJ_1) is { $R_1 \rightarrow R_4$, $R_2 \rightarrow R_6$, $R_3 \rightarrow R_4$, $R_7 \rightarrow R_4$ } (= $\{SJ_1, SJ_2, SJ_3, SJ_5\}$). Similarly, for the groups shown in Table 4-(b), Table 4-(c), the most profitable sets of semijoins are { $R_6 \rightarrow R_3$ } (= SMJ_2), { $R_2 \rightarrow R_1$ } (= SMJ_3), respectively. Therefore, $SMJ = SMJ_1 \cup SMJ_2 \cup SMJ_3$. Finally, we interleave those semijoins in SMJ in the given join sequence such that every semijoin $R_i \rightarrow R_j$ precedes every related join $R_j \Rightarrow R_k$.

Note that when the dynamic weighted graph does not have any correlated edge or when the values of DCB of all the vertexes are negative, we stop the execution of $DP(G)$. Moreover, when a vertex shrinking is executed, those related DCB of semijoins will be updated.

Let's consider the following three cases to compare our strategy (Case 3) with other strategies for distributed joins, where we use the query graph shown in Figure 4 with its profile shown in Table 2.

- **Case 1:** Using profitable semijoins.
  Execute profitable semijoins $R_2 \rightarrow R_1$, $R_4 \rightarrow R_1$, $R_4 \rightarrow R_3$, $R_6 \rightarrow R_3$, $R_1 \rightarrow R_4$, $R_3 \rightarrow R_4$, $R_7 \rightarrow R_4$, $R_2 \rightarrow R_6$, $R_6 \rightarrow R_7$ based on the strategy proposed in [3]. Then, send relations R_1, R_3, R_4, R_5, R_6, R_7 to the site containing R_2. The total transmission cost is 300 + 300 + 405 + 400 + 325 + 585 + 630 + 385 + 525 + 690 + 306 + 626 + 900 + 1485 + 1500 = 9362.

- **Case 2:** Applying join operations as reducers without semijoins [7, 8].
  Execute the join sequence $R_4 \Rightarrow R_7$, $R_5 \Rightarrow R_7$, $R_7' \Rightarrow R_6$, $R_3 \Rightarrow R_6$, $R_6' \Rightarrow R_2$, $R_1 \Rightarrow R_2$. The total transmission cost is 3300 + 900 + 2720 + 1700 + 2772 + 2300 = 13692.

- **Case 3:** Interleaving a join sequence with semijoins.
  Execute $R_1 \rightarrow R_4$, $R_3 \rightarrow R_4$, $R_7 \rightarrow R_4$, $R_4' \Rightarrow R_7$, $R_5 \Rightarrow R_7$, $R_7' \Rightarrow R_6$, $R_6 \rightarrow R_3$, $R_3' \Rightarrow R_6$, $R_2 \rightarrow R_6$, $R_6' \Rightarrow R_2$, $R_2 \rightarrow R_1$, $R_1' \Rightarrow R_2$. The total transmission cost is 325 + 585 + 630 + 627 + 900 + 1150 + 400 + 680 + 385 + 259 + 300 + 1150 = 7391.

(Note that in cases 2 and 3, we use the join sequences derived from the given join sequence tree as shown in Figure 4-(b).) From the above comparison, we show that the data transmission cost by using our strategy can be less than that by using profitable semijoins only or by using join reducers only. In general, in our algorithm, when there are N vertexes in the initial *dynamic weighted graph*, where each vertex represents a semijoin, our strategy needs to expand $1+N!*(1/(N-1)!+1/(N-2)!+1/(N-3)!+\cdots+1/2!+1/1!)$ graphs to find the solution in the worst case and (N + 1) graphs in the best case. Since in the worst cast, there is no vertex with a negative value of DCB, it results in (1 + N + (N * (N - 1)) + (N * (N - 1) * (N - 2)) + .. + N!) graphs, where there is one initial graph in level 0 and there are (N! / (N - i)!) graphs in the ith recursive execution, $1 \leq i \leq$ N. While in the best case, after executing a vertex shrinking for each vertex in the initial graph, each resulting graph contains vertexes with all negative values. Therefore, (1 + N) graphs are created. (Note that only those semijoins which have positive values of DCB will be included as a vertex in the initial graph.) For the initial state shown in Figure 4 with its profile shown in Table 2, we need to expand 75 graphs to find the solution where 206 graphs are needed in the worst case with some other profile.

Table 6 shows a comparison of the data transmission cost using five different profiles, where Profile C is the one shown in Table 2. For Profiles A and B, we increase the selectivity in all the attributes of Profile C to 0.2 and 0.1, respectively. For Profiles C and D, we decrease the selectivity in all the attributes of Profile C to 0.1 and 0.2, respectively. From the above comparison, we can see that the data transmission cost using our strategy can be less than that using profitable semijoins only or using join reducers only. Table 7 shows a comparison of the number of states created in one case of our algorithm, which was chosen randomly, the worst case of our algorithm and exhaustive search. From this table, we can see that our strategy is very efficient.

In [7], Chen et al. have proposed a heuristic algorithm to interleave a sequence of join operation with semijoins. In their algorithm, they define the *cumulative benefit* of a semijoin SJ_i ($R_k \xrightarrow{A} R_j$), denoted by $CB(SJ_i)$, as the amount of reduction if it is applied individually prior to the execution of a given join sequence, i.e., $CB(SJ_i) = (1 - \rho_{k,A}) \sum_{R_p \in Rd(SJ_i)} W(R_p, J_Q(\emptyset))$. In this algorithm, they use the *cumulative benefit* (CB) as a heuristic to determine the set of semijoins to be interleaved into a given join reducer sequence. However, in their algorithm, CB depends on a static profile of semijoins; while in our approach, DCB depends on a dynamic profile of semijoins. Based on those dynamic values of DCB, we can have fewer computation steps than Chen et al.'s algorithm [7] if there exists vertexes with negative values of DCB. Moreover, our algorithm can find better solution than their algorithm. For example, for the same query graph shown in Figure 4 with its profile shown in Table 2, their algorithm will execute $R_1 \rightarrow R_4$, $R_3 \rightarrow R_4$, $R'_4 \Rightarrow R_7$, $R_5 \Rightarrow R_7$, $R'_7 \Rightarrow R_6$, $R_6 \rightarrow R_3$, $R'_3 \Rightarrow R_6$, $R_2 \rightarrow R_6$, $R'_6 \Rightarrow R_2$, $R_2 \rightarrow R_1$, $R'_1 \Rightarrow R_2$. The total transmission cost is $325 + 585 + 1395 + 900 + 1150 + 400 + 680 + 385 + 259 + 300 + 1150 = 7529$, which is larger than our solution.

In [4], based on the heuristic values of DCB, we have proposed a variant of the A^* algorithm to find beneficial semijoins. In the variant of the A^* algorithm, which is a well-known heuristic search method, the search is controlled by a heuristic function $f(x)(= g(x) + h(x))$. The state x chosen for expansion is the one which has the *largest* value of $f(x)$ among all the generated states that have not been expanded so far. In this strategy, a state is defined as a set of correlated semijoins; i.e., it is represented by a dynamic weighted graph. When a state x is expanded, all the states that can be reached from state x by a vertex shrinking are generated. (Note that vertex shrinking can be executed only when the vertex has a positive value of DCB.) Among those leaf nodes x which have not been expanded thus far, the one with the largest $f(x)$ value will be chosen for future expansion. This procedure is repeated until the goal state is reached. The initial state is the given dynamic weighted graph, and the goal state is a dynamic weighted graph which does not have any correlated edge or in which the values of DCB of all the vertexes are negative. Let $SMJ_k(x)$ be the set of beneficial semijoins applied to the initial state to reach a state x in GP_k, and let $C(SMJ_k(x))$ be the total DCB values for the semijoin operation in

—	Profile A	Profile B	Profile C	Profile D	Profile E
Case 1	13891	11519	9362	7383	5327
Case 2	13692	13692	13692	13692	13692
Case 3	12005	9210	7391	4604	3510

Table 5: A comparison.

the number of vertexes in the initial state (N)	3	4	5	6
the states created in one case of our strategy	4	18	75	614
the states created in the worst case of our strategy	10	41	206	1237

Table 6: The number of states.

$SMJ_k(x)$. Then, at a state x, we let $g(x) = C(SMJ_k(x))$. Let $h(x)$ be the largest profit of future semijoins obtained in reaching the goal state from x. Let $DW(G)$ be the *dynamic weight* of the set of semijoins for a given *dynamic weighted graph* G, which is defined as follows:

$$DW(G) \;=\; \{DW(G_p) \;+\; DCB_p\},$$

where $p \in G$, DCB_p is the largest value among all the current DCB_i in G, and where we always consider positive values of DCB_p only. Then, at a state x, we let $h(x) = DW(x)$. For the same query graph shown in Figure 4 with its profile shown in Table 2, the set of beneficial semijoins found by this heuristic strategy [4] is the same as Chen et al.'s [7]. As compared to the performance of our previous proposed heuristic strategy which is also based on the values of DCB, although the heuristic strategy will expand fewer states than the new proposed one, our new proposed strategy can find better solution than the heuristic one.

4 Conclusion

In this paper, based on the values of *dynamic cumulative benefit* (DCB), we have proposed an algorithm for finding beneficial semijoins, i.e., to interleave a sequence of join operations with semijoins to reduce the data transmission cost. In this algorithm, a *dynamic weighted graph* is constructed based on the *correlated* relationship among semijoins. When there are N vertexes in the initial *dynamic weighted graph*, where each vertex represents a semijoin, our algorithm needs to expand (N + 1) graphs to find a solution in the best case. Moreover, since the DCB values of vertexes will be dynamically updated in the process of a *vertex shrinking*, it will speed up the execution and reduce the large space requirement of the proposed recursive algorithm in general cases. From our simulation study, we have shown that our strategy can efficiently find beneficial semijoins and require a lower data transmission cost than does the profitable semijoin approach. In addition, this interleaving a join sequence with semijoins approach can reduce the communication cost further by taking advantage of the removability of *pure join attributes*, where *pure join attributes* are those which are used in join predicates but not part of the output attributes.

References

[1] Peter M. G. Apers, Alan R. Hevner and S. Bing Yao, "Optimization Algorithms for Distributed Queries," *IEEE Trans. on Software Eng.,* Vol. SE-9, No. 1, pp. 57-68, Jan. 1983.

[2] Philip A. Bernstein and Dah-Ming W. Chiu, "Using Semi-Joins to Solve Relational Queries," *Journal of ACM,* Vol. 28, No. 1, pp. 25-40, Jan. 1981.

[3] Philip A. Bernstein, Nathan Goodman, Eugene Wong, Christopher L. Reeve and James B. Rothnie, "Query Processing in a System for Distributed Databases (SDD-1)," *ACM Trans. on Database Syst.,* Vol. 6, No. 4, pp. 602-625, Dec. 1981.

[4] Ye-In Chang, Bor-Miin Liu and Cheng-Huang Liao, "A Dynamic-Cumulative-Benefit-based Algorithm for Beneficial Semijoins," *Proceedings of National Science Council: Part A – Physical Science and Engineering,* Vol. 20, No. 5, pp. 507-518, Sept. 1996.

[5] Arbee L. P. Chen and Victor O. K. Li, "Improvement Algorithm for Semijoin Query Processing Programs in Distributed Database Systems," *IEEE Trans. on Computer,* Vol. C-33. No. 11, pp. 959-967, Nov. 1984.

[6] Arbee L. P. Chen and Victor O. K. Li, "An Optimal Algorithm for Processing Distributed Star Queries," *IEEE Trans. on Software Eng.,* Vol. SE-11, No. 10, pp. 1097-1107, Oct. 1985.

[7] Ming-Syan Chen and Philip S. Yu, "Interleaving a Join Sequence with Semijoins in Distributed Query Processing,"*IEEE Trans. on Parallel and Distributed Syst.,* Vol. 3, No. 5, pp. 661-621, Sept. 1992.

[8] Ming-Syan Chen and Philip S. Yu, "Combining Join and Semi-Join Operations for Distributed Query Processing," *IEEE Trans. on Knowledge and Data Eng.,* Vol. 5, No. 3, pp. 534-542, June 1993.

[9] C. Wang and Ming-Syan Chen, " On the Complexity of Distributed Query Optimization," *IEEE Trans. on Knowledge and Data Engineering,* Vol. 8, No. 4, pp. 650-662, Aug. 1996.

[10] D. -M. Chiu and Y. -C. Ho, "A Methodology for Interpreting Tree Queries into Optimal Semijoin Expressions," *Proc. of 1980 ACM-SIGMOD Int. Conf. on Management of Data*, pp. 169-178, 1980.

[11] D. -M. Chiu, P. A. Bernstein and Y. -C. Ho, "Optimizing Chain Queries in a Distributed Database System," *SIAM J. Comput.*, Vol. 13, pp. 116-134, Feb. 1984.

[12] A. Hevner, "Query Optimization in Distributed Database Systems," Ph.D. Dissertation, Univ. Minnesita, 1979.

[13] Alan R. Hevner and S. Bing Yao, "Query Processing in Distributed Database Systems," *IEEE Trans. on Software Eng.,* Vol. SE-5, No. 3, pp. 177-187, May 1979.

[14] Sakti Pramanik and David Ittner, "Optimizing Join Queries in Distributed Databases," *IEEE Trans. on Software Eng.,* Vol. 14, No. 9,pp. 1319-1326, Sept. 1988.

[15] Hyuck Yoo and Stephane Lafortune, "An Intelligent Search Method for Query Optimization by Semijoins," *IEEE Trans. on Knowledge and Data Eng.,* Vol. 1, No. 2, pp. 226-237, June 1989.

[16] C. T. Yu and C. C. Chang, "Distributed Query Processing," *ACM Computing Surveys,* Vol. 16, No. 4, pp. 388-433, Dec. 1984.