Two-Step Pruning : A Distributed Query Optimization Algorithm

Hyeokman Kim[1,2], Sukho Lee[1], Hyoung-Joo Kim[1]

[1] Dept. of Computer Engineering, Seoul National University
Shinrim-Dong, Kwanak-Gu, Seoul, 151-742, Korea
[2] Korea Telecom Research Laboratories
17, Woomyon-Dong, Suhcho-Gu, Seoul, 137-792, Korea
E-mail: {hmkim, shlee}@snucom.snu.ac.kr, hjk@inm4u.snu.ac.kr

Abstract. The problem of finding an optimal global plan for a tree query in a distributed database is studied under the objective of total processing time minimization. A two-step pruning algorithm based on dynamic programming is presented. This algorithm performs a pruning step twice for each subquery by designing two separate equivalence criteria applicable to each subquery. This lessens the search work done by the optimizer considerably. Without losing optimality, the search space for finding the optimum is reduced by aggregating partial plans that always incur the same processing time into a single plan and eliminating partial plans that can never be the optimum.

1 Introduction

The importance of query optimization in centralized and distributed database systems is widely recognized. One of the key components in query optimization is the search strategy. The ability to efficiently find an optimal or best plan among all possible alternatives is indeed essential to performance.

Dynamic programming which is probably the best known standard optimization technique has been employed as a search strategy in query optimization [3, 9, 10, 12, 15]. If dynamic programming is used to solve an optimization problem, the problem must satisfy the *principle of optimality*: in optimal sequence of decisions, each subsequence must be optimal [6]. Thus, the cost function based on dynamic programming is expressed as a recursive form. When building execution plans through dynamic programming, the optimizer systematically builds all possible partial plans and compares them through their cost estimates. It then prunes costly partial plans that are equivalent to a cheaper one. This pruning reduces the optimization cost because partial plans that are not likely to be optimal are pruned as soon as possible. The advantage of dynamic programming stems from the fact that the chosen partial plan with its cost is saved, and reused rather than recomputed when this plan is employed as a subplan of another plan. This means that the saved partial plans are shared by many other subsequent plans built from them.

The pruning is performed based on *equivalence criteria*, that is, the optimal partial plan is chosen among all possible partial plans that are equivalent in some sense. In a centralized environment, equivalent plans are those which capture the same relations. However in a distributed environment, the execution site where the result of a partial plan is materialized and the delivery site where the result is transferred may have impact on the cost of future plans. Thus, the equivalence criteria for distributed plans should consider these sites. In R* optimizer [10, 11] and state transition algorithm [9], their equivalence criteria consider not both but only one of them. Thus, they apply only a single pruning step to each subquery or state.

We propose a query optimization algorithm in distributed database systems. Our objective is to achieve search efficiency without relaxing optimality. We employ dynamic programming as a search strategy. In our approach, both the execution and delivery sites are adopted as equivalence criteria. We apply a pruning step twice to each subquery by carefully designing two equivalence criteria such that the optimal partial plan chosen through the first pruning step is reused when building an optimal partial plan through the second step applied in succession. This reduces the optimization time considerably because the partial plans that are not likely to be optimal are pruned much earlier and there are more chances to share the partial plans.

1.1 Related Work

There have been large volumes of work done regarding optimization for join operations in distributed relational database and surveyed in a number of places, including [5, 13, 16]. Two basic approaches exist to determine an optimal or best join ordering in distributed database systems: join-based [9, 10, 11] and semijoin-based [1, 2, 3, 15]. The main value of a semijoin is to reduce the size of the join operands and then the communication cost. But, using semijoins might increase the local processing cost, since one of the two joining relations must be accessed twice. Furthermore, the join of two intermediate results produced by semijoins cannot exploit the indexes that were available on the base relation. Most of the semijoin approaches assume that communication cost largely dominates local processing cost. This assumption is based on very slow communication networks such as wide area networks with a bandwidth of a few kilobytes per second. However, recent advances on network communications have drastically increased the bandwidth, and distributed database environments now exist where the communication network is much faster, making the cost of local processing no longer negligible. Therefore, using semijoins may not be a good idea and more recent techniques which consider local processing costs as well as communication costs tend not to use semijoins.

R* optimizer is a representative join-based approach [10, 11]. Conceptually, it can be viewed as an exhaustive search among all the enumerated alternative plans. The optimizer reduces the number of alternatives and search work by building partial plans of subquery and pruning them except the cheapest one through dynamic programming. R* optimizer adopts a single equivalence criteria which includes the relations captured by the subquery and the execution site. As a consequence, the optimizer performs a single pruning step for each subquery. We call the optimizer a *one-step pruning (OSP)* algorithm.

In OSP algorithm, a join between relations at different sites is accomplished by transferring one relation to the site of the other or both of them to a third site. In the last case, the subsequent join between the intermediate result of the join executed at this third site and the relation stored here may then be executed without transmissions. We denote T_i to be a subquery of a given query where i represents the number of relations captured by the subquery. A subquery T_i is represented as a set of captured relations. T_1 is a subquery containing only one relation. We denote y to be the resident site of a relation captured by T_1. Let $CostTJ_t(T_i)$ be the minimum cost to execute the join between the results of subquery T_{i-1} and T_1 at site t, $trans_{xt}(T_i)$ be the cost for transferring the result of T_i from site x to site t, and $join_t(T_{i-1}, T_1)$ be the join cost between the results of T_{i-1} and T_1 at site t. $CostTJ_t(T_i)$ is computed according to the following dynamic programming equation.

$$CostTJ_t(T_i) = \min_{\forall pair\ s.t.\ T_i=T_{i-1}\cup T_1\ and\ \forall x\in\{resident\ sites\ of\ relations\ in\ T_i\}} \{CostTJ_x(T_{i-1}) + trans_{xt}(T_{i-1}) + trans_{yt}(T_1) + join_t(T_{i-1}, T_1)\} \quad (1)$$

If $CostTJ_t(T_i)$ is computed for all possible execution site $t \in$ {resident sites of relations in T_i and a third site} and these computations are repeatedly applied to each subquery T_i varying i from 2 to n, the cost of an optimal plan is finally obtained. OSP algorithm uses the heuristic to restrict the search space, that is, either operand of joins must be a base relation. This reduces the optimization overhead. Though the heuristic takes advantages of indexes on base relations, this algorithm does not exploit the huge portion of feasible plans. Thus, OSP algorithm produces a best plan, not an optimal one.

In state transition algorithm [9], the relations and intermediate results are modeled into a state represented as a s-component vector where s is the number of sites related. The state says where the relations and intermediate results are, that is, the delivery sites. When a join is executed (one-step transition), one state transits to another state. The state transitions are equivalent if the resulting states have the same components, that is, the same delivery sites of all subqueries. The algorithm constructs a state space which includes all possible states and their transition relationships. It then chooses an optimal path (trajectory) from the initial state to a final one. The state space X is divided into n disjoint subsets, $X(i)$'s, such that $X = X(0) \cup X(1) \cup \cdots \cup X(n-1)$ where i is the number of joins performed. The initial state x_0 and the final state x_f belong to $X(0)$ and $X(n-1)$ respectively.

When one state $x \in X(i-1)$ transits to another state $y \in X(i)$ by executing a single join, there are at most three alternative strategies which are a sequence of a join operation and additional transmission operations. Let γ be a strategy for doing a one-step transition from state x to state y, Γ be the set of all feasible γ and $cost(x, y)$ be the minimum cost of doing the one-step transition from x to y. Then, the minimum transition cost is

$$cost(x, y) = \min_{\forall \gamma \in \Gamma} \{the\ sum\ of\ the\ costs\ of\ operations\ in\ \gamma\} \tag{2}$$

The cost of state x means the total join and transmission costs required to go from the initial state to state x. The cost of state y which can be reached from state x in a one-step transition is the sum of the cost of state x and the cost of the transition. Let $CostS_{i-1}(x)$ be the minimum cost to go from the initial state to state x in $X(i-1)$ and $T^{-}(y)$ be the set of states that can reach y in a one-step transition. $CostS_i(y)$ is then computed according to the following dynamic programming equation.

$$CostS_i(y) = \min_{\forall x \in T^{-}(y)} \{CostS_{i-1}(x) + cost(x, y)\} \tag{3}$$

If $cost(x, y)$ and $CostS_i(y)$ are computed for each state y in $X(i)$ varying i from 1 to $n-1$, the cost of an optimal plan, $CostS_{n-1}(x_f)$, is eventually computed. As opposed to OSP algorithm, the state transition algorithm performs two pruning steps for each state. That is, each of Equations (2) and (3) prunes all feasible transition strategies and trajectories except the optimal ones. However, Equation (2) is not a dynamic programming equation. Thus, we call the state transition algorithm a *semi-two-step pruning (Semi-TSP)* algorithm.

Since semi-TSP algorithm does not employ dynamic programming in the first pruning step, there is no way to reuse the computed results of the first pruning step: if $y \in X(i)$ and $y' \in X(i+1) \cup \cdots \cup X(n-1)$ and the subqueries related with the one-step transitions from x to y and from x' to y' are the same, $cost(x', y')$ must be recomputed though it is equal to $cost(x, y)$. For example, let's consider a chain query with relations A,B,C,D located at sites 1,2,3,4 respectively. A small portion of the state space for the query is given in Figure 1. There are three strategies for one-step transition from state x_0 to x_1, that is, A and B may be joined at one of their resident sites and the result is then transferred to site 3 or both of them may be transferred to site 3 and then joined. Since the transition strategies from x_2 to x_4 are equal to those from x_0 to x_1, $cost(x_0, x_1)$ and $cost(x_2, x_4)$ are the same. Semi-TSP algorithm has to do this redundant computation. Furthermore, the extra work in constructing the state space is too expensive, though the algorithm produces an optimal plan.

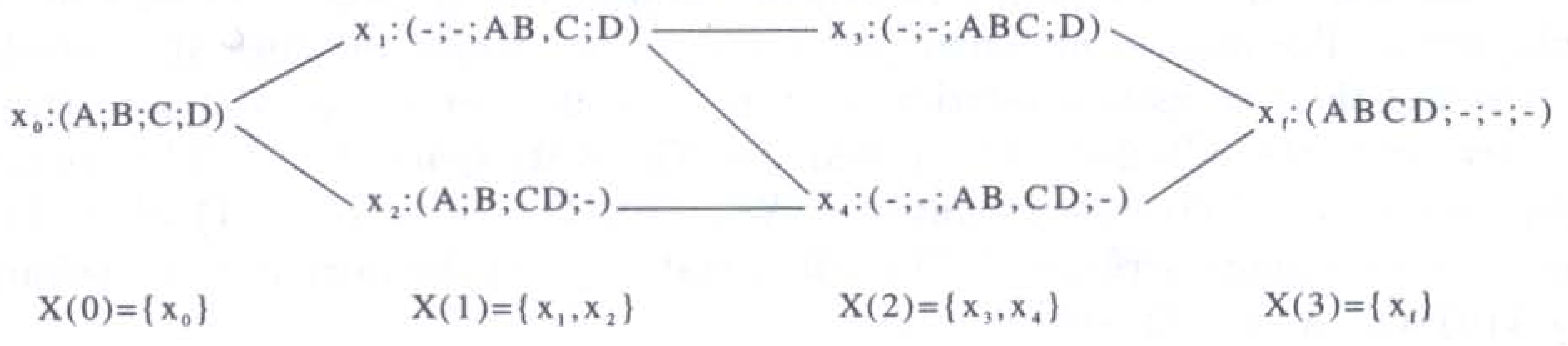


Fig. 1. State space

In a distributed database, both execution and delivery sites have influence on the cost of plan. OSP and semi-TSP algorithms include either execution or delivery site in their equivalence criteria. Our algorithm considers both sites in its criteria. We design two equivalence criteria such that one include an execution site and the other a delivery site, and the pruning steps based on these criteria obey the principle of optimality. Thus, our algorithm applies the pruning step twice to each subquery instead of a single step from the viewpoint of dynamic programming. We call our algorithm a *two-step pruning (TSP)* algorithm. Compared with OSP algorithm, TSP algorithm prunes much earlier the partial plans that are not likely to be optimal. In contrast to semi-TSP algorithm, the costs of partial plans chosen by the first pruning step as well as by the second step are reused every time another partial plan is built from those plan. Our TSP algorithm avoids the redundant computations done in semi-TSP algorithm.

This paper is organized as follows. In Section 2, the problem statement, the global execution plan, and the notion and assumption required are given. In Section 3, we first describe the cost model to compute the cost of execution plan according to the two pruning steps. Then, we show that this cost model obeys the principle of optimality. Based on these, we propose TSP algorithm and compute the complexity of the algorithm. Finally, in Section 4, we draw some conclusions and suggest future research directions.

2 Preliminaries

2.1 Problem Statement

Our goal is to find optimal join and transmission order of a given query under the objective of minimizing total processing time which includes local processing and communication costs. We are given a query T referencing n distinct base relations $R_i's$ distributed among s sites and an *initial distribution* representing the resident sites of n relations in database. We call the *query graph* of T the graph with n nodes, where each join clause in T is indicated as a link between corresponding nodes. Each node has a label which specifies the resident site of the corresponding relation. An example query graph is represented in Figure 2. In this paper, we focus on *tree query* whose query graph is tree. The site where a user gives a query and returns its result is called the query site.

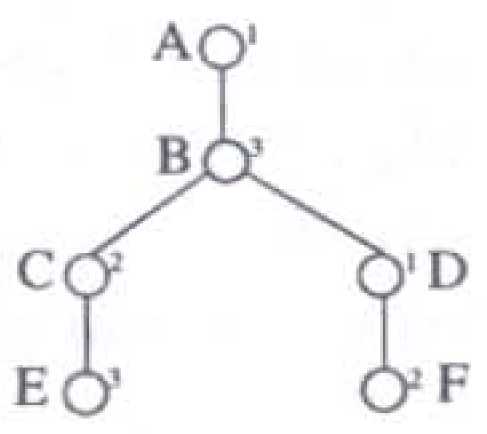


Fig. 2. Query graph

2.2 Global Execution Tree

Distributed query optimizer can be defined as an algorithm to choose an optimal global processing strategy for a given query. The design of such optimizers may be divided into three components: execution space which is the set of the execution plans searched by the optimizer, cost model which predicts the cost of an execution plan, and search strategy to obtain the minimum cost plan. In this subsection, we describe the execution space. The others are described in the following section. Query executions are represented as execution plans which transform a nonprocedural query into a sequence of operations. An execution plan can be syntactically represented as a join processing tree [7] or a dataflow graph [2]. We extend these representations into a *global execution tree (GET)* to express execution plans processed in distributed database environment.

GET is a labelled tree where the indegree of each nodes must not exceed two. It represents the flow of data from leaves to root. The leaf nodes are base relations and each non-leaf node is an intermediate result from joining or transmission: a node with indegree 2 (*join node*) is an intermediate result from joining its children and a node with indegree 1 (*transmission node*) is an intermediate result from the transmission of its child. Each non-leaf node is stored in a temporary file. In GET, the leaf, join and transmission nodes are graphically represented by circle, closed square and open square respectively.

Each node has a site label which represents different meaning according to the kinds of nodes. The site label for a leaf node represents the resident site of its corresponding relation. The site label for a transmission node represents the delivery site to which its child is transferred. If the site labels for a transmission node and its child are the same, no transmission occurs. The site label for a join node represents a join execution site. Relations or intermediate results to be joined must be at the same site. Thus, the site labels for a join node and its children must be the same. We say that GET is *complete* if it captures all the relations of the given query and the site label of the root is equal to the query site. Otherwise, it is said to be *partial*.

GET can be distinguished into *deep* or *bushy* trees [7]. If all join nodes of a GET have as a child at least one leaf node or transmission node which also has a leaf node as a child, the tree is called deep. Otherwise, it is called bushy. Figure 3 gives examples of deep and bushy GETs generated from the query in Figure 2. In Figure 3, the number annotated on each node is the site label. If the execution space does not include all feasible plans for a query, the optimizer may produce a suboptimal plan not an optimal one. For example, OSP algorithm in general produces a suboptimal plan because it has a restricted execution space that includes only deep trees. We do not restrict the execution space of our optimizer. Our optimizer searches execution plans which may be deep or bushy trees.

Execution plans represented by GET specify the order of joins and transmissions to be executed. The order is determined by traversing GET in a postorder sequence. Based on the costs of the individual join and transmission specified in GET, we can estimate the cost of the corresponding execution. This cost is defined as the cost of GET.

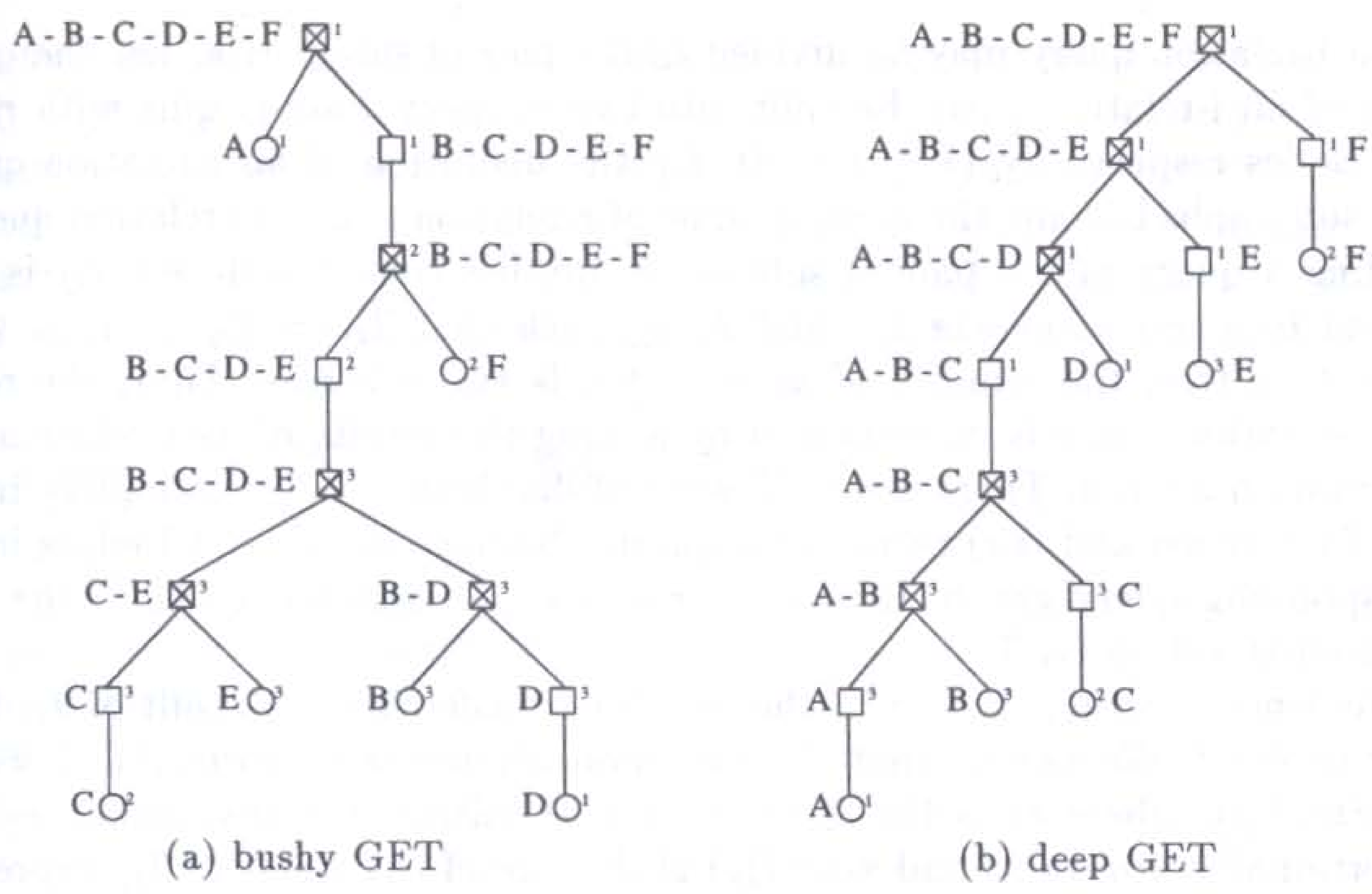


Fig. 3. Global Execution Tree

GET can be expressed as a sequence of operators. Let a, b be subplans of GET and $JN_x(a,b)$ be a join operation between the results of the subplans a and b at site x, $TR_{xt}(a)$ be an operation to transfer the result of the subplan a from site x to site t. The join node with site label x is transformed to $JN_x(-,-)$ operator and its children to operands. Also, the transmission node whose site label is t and has a child with site label x is to $TR_{xt}(-)$ operator and its child to operand. For example, the operator expression for the GET in Figure 3.a is $JN_1(A, TR_{2,1}(JN_2(TR_{3,2}(JN_3(JN_3(TR_{2,3}(C), E), JN_3(B, TR_{1,3}(D)))), F)))$.

2.3 Notions and Assumptions

Let an *i-relation query* ($1 \leq i \leq n$) be a subquery of a given query T such that the query graph of an i-relation query is a connected subgraph with i nodes constructed from that of a given query. Thus, the given query T becomes an n-relation query. An i-relation query is represented as a node set T_{ij} where i is the number of nodes in T_{ij} and j is a number for distinguishing feasible i-relation queries from one another. By the definition of an i-relation query, T_{ij} can not be null. The number of i-relation queries in T is represented as m_i. Thus, the total number of subqueries is $\sum m_i$. For a query with n relations, there are n 1-relation queries, T_{1j} ($1 \leq j \leq n$), each of which contains only one relation ($m_1 = n$). The n-relation query, T_{n1}, is the input query itself ($m_n = 1$). In addition, we use $RS(T_{ij})$ to denote the set of resident sites of relations captured by T_{ij}. The cardinality of a resident site set, $|RS(T_{ij})|$, depends on the initial distribution. For example, $|RS(T_{n1})| = s$.

An i-relation query may be divided into a pair of subqueries. Let the query graph of an i-relation query be split into two connected subgraphs with r and $i-r$ nodes respectively ($1 \leq r < i$). By the definition of an i-relation query, these subgraphs become the query graphs of r-relation and (i-r)-relation queries. Dividing a query into a pair of subqueries implies that a node set T_{ij} is partitioned into two node sets T_{rp} and $T_{i-r,q}$ such that $T_{ij} = T_{rp} \cup T_{i-r,q}$ when $i \geq 2$. In a tree, the removal of an edge yields two subtrees. Thus, the result of an i-relation query is materialized by joining the results of its r-relation and (i-r)-relation queries. There are $i-1$ ways of dividing an i-relation query into a pair of r-relation and (i-r)-relation subqueries because there are i-1 edges in the corresponding query graph. If r or $i-r$ is always restricted to be 1, the plan produced is a deep GET.

We denote $trans_{xt}(T_{ij})$ to be the cost for transferring the result of T_{ij} from site x to site t. We assume that the communication cost is $trans_{xt}(T_{ij}) = c_1 + c_2 * size(T_{ij})$, where c_1 is the start-up cost of initiating transmission, c_2 is a proportionality constant, and $size(T_{ij})$ is the size of the result of T_{ij} expressed in bytes [16]. If $x = t$, $trans_{tt}(T_{ij}) = 0$. Also, we denote $join_x(T_{rp}, T_{i-r,q})$ to be the join cost between the results of T_{rp} and $T_{i-r,q}$ at site x.

It is assumed that there is no fast access path such as index for all non-leaf nodes and some leaf nodes in GET. This is justified by the observations that in distributed query processing,

- The join node is an intermediate result of a join. The intermediate result is not supported by fast access paths unless some are created dynamically.
- A fast access path is not valid outside the site where it was established. When a relation or intermediate result is sent from one site to another, its fast access path will not be sent. Thus, the transmission node is not supported by fast access paths.
- The operations like selection and projection are executed before joins. Thus, the leaf nodes which are qualified by these operations are also intermediate results.

Besides these, we assume that the usual statistical information for cost estimation is available, even though we have not explicitly expressed it in formalism. When multiple copies of a relation exist, we assume one copy has been preselected.

We make a *uniformity assumption* on a distributed environment, as in [1, 2, 3, 9, 10, 15]. That is, all sites have the same processing capabilities and the communication speeds between any two sites are the same. The assumption is then written formally as follows.

- if $x_1, x_2 \notin RS(T_{ij})$, $join_{x_1}(T_{rp}, T_{i-r,q}) = join_{x_2}(T_{rp}, T_{i-r,q})$.
- if $x_1 \in RS(T_{ij})$ and $x_2 \notin RS(T_{ij})$, $join_{x_1}(T_{rp}, T_{i-r,q}) \leq join_{x_2}(T_{rp}, T_{i-r,q})$.
- if $x_1 \neq t_1$ and $x_2 \neq t_2$, $trans_{x_1,t_1}(T_{ij}) = trans_{x_2,t_2}(T_{ij})$.

If an execution site x is not a site in the resident site set of T_{ij}, $RS(T_{ij})$, the results of subqueries T_{rp} and $T_{i-r,q}$ to be joined may be results materialized at the execution site x or transferred from other sites. In both cases, the intermediate result of join or transmission is stored in a temporary file and there is no index support for the file since we do not create index dynamically. However, if either or both of the subqueries to be joined is a 1-relation query, there may be index. Thus, the cost of join executed at the resident site of the relation captured in the 1-relation query may be lower than the join cost executed at the other site.

3 Two-Step Pruning Algorithm

3.1 Equivalence Criteria

The principle of optimality states that an optimal execution plan (optimal GET) for a query is composed of the optimal plan for its subqueries as subplans. Therefore, in order to produce an optimal plan for a given query through dynamic programming, the optimal plan for its subqueries must be produced in advance. That is, an optimal plan for an i-relation query is built from the optimal plans for its subqueries from 1-relation to (i-1)-relation queries.

If the results of r-relation and (i-r)-relation subqueries to be joined are not at the same site, one of them may be transferred to the site of the other or both of them transferred to a third site where a subsequent join between the i-relation query and another (k-i)-relation query will be executed. The third site can be any resident site of the relations captured in the (k-i)-relation query, since the (k-i)-relation query may materialize its result at this site. Furthermore, the result of the i-relation query materialized at the site where the results of its subqueries are located may be transferred to any resident site of the relations captured in the (k-i)-relation query because of the same reason. Thus, the site at which an i-relation query materializes its result, an *execution site of an i-relation query*, and the site to which an i-relation query delivers its result, a *delivery site of an i-relation query*, can be any resident site of the relations captured in a given query.

When building execution plans through dynamic programming, the optimizer systematically builds and compares equivalent partial plans through their cost estimates. Our optimizer builds the optimal plan of an i-relation query through two separate steps, one for join plan and the other for transmission plan. Let a *join plan of an i-relation query* be a plan which materializes the result of the query at a certain site, and a *transmission plan of an i-relation query* be a plan which transfers the result of the query to a certain site. Our optimizer uses the following *equivalence criteria* for pruning.

- Join plans are equivalent if they capture the same relations and have the same execution site.
- Transmission plans are equivalent if they capture the same relations and have the same delivery site.

For each i-relation query, the optimizer produces the optimal join and transmission plans by applying the pruning step twice based on the two criteria respectively. Thus, our optimizer has two chances to discard costly equivalent plans.

3.2 Principle of Optimality

To help us characterize the principle, we denote $a_x^u(T_{rp})$ to be the u-th transmission plan of T_{rp} at delivery site x among all equivalent transmission plans. The join and transmission plans of T_{ij} are produced by the following functions.

The function $buildJP_x(a_x^u(T_{rp}), a_x^v(T_{i-r,q}))$ combines the plans $a_x^u(T_{rp})$ and $a_x^v(T_{i-r,q})$ into the plan in which the results of T_{rp} and $T_{i-r,q}$ at site x are joined at this site, that is, $buildJP_x(a_x^u(T_{rp}), a_x^v(T_{i-r,q})) = JN_x(a_x^u(T_{rp}),\ a_x^v(T_{i-r,q}))$. By executing this plan, the result of T_{ij} is materialized at site x. The delivery site of subqueries T_{rp} and $T_{i-r,q}$, site x, becomes the execution site of T_{ij}. We denote $b_x^w(T_{ij})$ to be the plan produced by the function. It means the w-th join plan of T_{ij} at execution site x among all equivalent join plans.

The function $buildTP_{xt}(b_x^w(T_{ij}))$ extends the plan $b_x^w(T_{ij})$ into another plan in which the result of T_{ij} materialized at site x is transferred to site t, namely, $buildTP_{xt}(b_x^w(T_{ij})) = TR_{xt}(b_x^w(T_{ij}))$. The site t is the delivery site of T_{ij}. If sites x and t are equal, no transmission occurs. By executing this plan, we can get the result of T_{ij} at site t. The delivery site of the query T_{ij}, site t, may become the execution site of another query which employs T_{ij} as its subquery.

As the results of T_{rp} and $T_{i-r,q}$ to be joined must be at the same site, the cost of the join plan produced by $buildJP_x(a_x^u(T_{rp}), a_x^v(T_{i-r,q}))$ is the sum of the costs for getting the results of the subqueries T_{rp} and $T_{i-r,q}$ at site x and the cost for joining them at this site. Similarly, the cost of the transmission plan produced by $buildTP_{xt}(b_x^w(T_{ij}))$ is the sum of the cost for materializing the result of the query T_{ij} at site x and the cost for transferring the result from site x to site t. Let $Cost(p)$ be the cost of a plan p. Then, we have

$$Cost(buildJP_x(a_x^u(T_{rp}), a_x^v(T_{i-r,q}))) = Cost(a_x^u(T_{rp})) + Cost(a_x^v(T_{i-r,q})) + join_x(T_{rp}, T_{i-r,q}) \tag{4}$$

$$Cost(buildTP_{xt}(b_x^w(T_{ij}))) = Cost(b_x^w(T_{ij})) + trans_{xt}(T_{ij}) \tag{5}$$

To choose the optimal plans among the plans produced by $buildJP_x(-,-)$ and $buildTP_{xt}(-)$, we use the principle of optimality: If plans differ only in subplans, the plan with optimal subplans is also optimal. This principle can be expressed formally in the following theorem. Proofs for Theorem 1 and the following theorems are given in [8].

Theorem 1. *Let the subplans $a_x^1(T_{rp})$, $a_x^1(T_{i-r,q})$ and $b_x^1(T_{ij})$ be the lowest cost plans among all its equivalent plans $a_x^u(T_{rp})$, $a_x^v(T_{i-r,q})$ and $b_x^w(T_{ij})$ respectively. That is, $Cost(a_x^1(T_{rp})) \leq (\forall u)Cost(a_x^u(T_{rp}))$, $Cost(a_x^1(T_{i-r,q})) \leq (\forall v)Cost(a_x^v(T_{i-r,q}))$, and $Cost(b_x^1(T_{ij})) \leq (\forall w)Cost(b_x^w(T_{ij}))$. Then $Cost(buildJP_x(a_x^1(T_{rp}), a_x^1(T_{i-r,q}))) \leq (\forall u,v)Cost(buildJP_x(a_x^u(T_{rp}), a_x^v(T_{i-r,q})))$ and $Cost(buildTP_{xt}(b_x^1(T_{ij}))) \leq (\forall w)Cost(buildTP_{xt}(b_x^w(T_{ij})))$.*

We call $a_x^1(T_{rp})$ the optimal transmission plan of T_{rp} at site x and $b_x^1(T_{ij})$ the optimal join plan of T_{ij} at site x. According to the theorem, we can get the following equations.

$$Cost(buildJP_x(a_x^1(T_{rp}), a_x^1(T_{i-r,q}))) = \min_{\forall u,v}\{Cost(buildJP_x(a_x^u(T_{rp}), a_x^v(T_{i-r,q})))\} \tag{6}$$

$$Cost(buildTP_{xt}(b_x^1(T_{ij}))) = \min_{\forall w}\{Cost(buildTP_{xt}(b_x^w(T_{ij})))\} \tag{7}$$

Now, let's compute the cost of an optimal join plan of T_{ij} at site x, $Cost(b_x^1(T_{ij}))$. As there are $i-1$ pairs of subqueries T_{rp} and $T_{i-r,q}$ for a query T_{ij}, we must compare $i-1$ costs of all equivalent join plans. For each pair of subqueries, the results of subqueries to be joined must be at site x. Since there are many alternative plans to get the result of subquery at site x, we also compare all equivalent transmission plans of both subqueries at site x and then choose the lowest cost plans. Thus, we have

$$\begin{aligned} Cost(b_x^1(T_{ij})) &= \min_{\forall p,q,r}\{\min_{\forall u,v}\{Cost(buildJP_x(a_x^u(T_{rp}), a_x^v(T_{i-r,q})))\}\} \\ &= \min_{\forall p,q,r}\{Cost(buildJP_x(a_x^1(T_{rp}), a_x^1(T_{i-r,q})))\} \\ &= \min_{\forall p,q,r}\{Cost(a_x^1(T_{rp})) + Cost(a_x^1(T_{i-r,q})) + join_x(T_{rp}, T_{i-r,q})\} \end{aligned} \tag{8}$$

where $\forall p,q,r$ means all pairs of subqueries T_{rp}, $T_{i-r,q}$ such that $T_{ij} = T_{rp} \cup T_{i-r,q}$. This equation is developed from Equations (6) and (4).

Let's compute the cost of an optimal transmission plan of T_{ij} at site t, $Cost(a_t^1(T_{ij}))$. In order to get a result of T_{ij} at site t, the result may be directly materialized at site t or materialized at another site, for example, at site x and then transferred to site t. As the result of T_{ij} may be materialized at any site in the resident site set of the given query, $RS(T_{n1})$, we must compare the costs of all equivalent transmission plans. Before a transmission of a result of T_{ij} occurs, the result must be materialized. Since there are many equivalent join plans to materialize the result of T_{ij} at site x, we also compare them and then choose the lowest cost plan. Thus, we have

$$\begin{aligned} Cost(a_t^1(T_{ij})) &= \min_{\forall x \in RS(T_{n1})}\{\min_{\forall w}\{Cost(buildTP_{xt}(b_x^w(T_{ij})))\}\} \\ &= \min_{\forall x \in RS(T_{n1})}\{Cost(buildTP_{xt}(b_x^1(T_{ij})))\} \\ &= \min_{\forall x \in RS(T_{n1})}\{Cost(b_x^1(T_{ij})) + trans_{xt}(T_{ij})\} \end{aligned} \tag{9}$$

This equation is developed from Equations (7) and (5). In Equation (9), if $x = t$ the result of T_{ij} is directly materialized at site t because of $trans_{tt}(T_{ij}) = 0$. Otherwise, it is materialized at site x and then transferred to site t.

Equations (8) and (9) form an indirect recurrence. Since there is no join plan of 1-relation query, we will initialize the transmission plans of 1-relation query. The initial condition for Equation (8) is $Cost(a_t^1(T_{1j})) = trans_{xt}(R_j)$ where x is the resident site of relation R_j. If $x = t$, $Cost(a_t^1(T_{1j}))$ will be zero. If Equations (8) and (9) are computed for each i-relation query $(2 \le i \le n)$ varying x and t over all related sites, that is, $\forall x, t \in RS(T_{n1})$, an optimal plan of a given query at a query site can be produced.

Equations (8) and (9) say that the optimal join plan of T_{ij} at site x is constructed by the optimal transmission plans of its subqueries T_{rp} and $T_{i-r,q}$ at this site and the optimal transmission plan of T_{ij} at site t is constructed by the optimal join plan of T_{ij} at site x. That is, the optimal join and transmission plans are constructed recursively using each other.

3.3 Plan Aggregation and Elimination

We will aggregate some plans to avoid redundant computations, and eliminate some plans from the execution space in order to prevent wasteful searches. The optimal plans of an i-relation query are *redundant* if they always have the same cost irrespective of the contents of the relations captured in the query. For example, relation A at site 1 and relation B at site 2 may be joined at site 3 or 4 where a subsequent join will be executed. The costs of the optimal join plans at site 3 and 4 are computed as follows.

$$\begin{aligned}
Cost(b_3^1(\{A,B\})) &= Cost(buildJP_3(a_3^1(\{A\}), a_3^1(\{B\}))) \\
&= Cost(a_3^1(\{A\})) + Cost(a_3^1(\{B\})) + join_3(\{A\},\{B\}) \\
&= trans_{13}(\{A\}) + trans_{23}(\{B\}) + join_3(\{A\},\{B\}) \\
Cost(b_4^1(\{A,B\})) &= trans_{14}(\{A\}) + trans_{24}(\{B\}) + join_4(\{A\},\{B\})
\end{aligned}$$

Because of the uniformity assumption, $Cost(b_3^1(\{A,B\}))$ is equal to $Cost(b_4^1(\{A, B\}))$.

Similarly, the join result between A and B may be directly materialized at site 3 or materialized at a site other than site 3 and then be transferred to site 3. This is also true for site 4. Thus, the costs of the optimal transmission plans at site 3 and 4 are

$$\begin{aligned}
&Cost(a_3^1(\{A,B\})) \\
&= \min\{\ Cost(buildTP_{13}(b_1^1(\{A,B\}))), Cost(buildTP_{23}(b_2^1(\{A,B\}))), \\
&\qquad Cost(buildTP_{33}(b_3^1(\{A,B\}))), Cost(buildTP_{43}(b_4^1(\{A,B\})))\} \\
&= \min\{\ Cost(b_1^1(\{A,B\})) + trans_{13}(\{A,B\}), Cost(b_2^1(\{A,B\})) + trans_{23}(A,B), \\
&\qquad Cost(b_3^1(\{A,B\})), Cost(b_4^1(\{A,B\})) + trans_{43}(\{A,B\})\} \\
&Cost(a_4^1(\{A,B\})) \\
&= \min\{\ Cost(b_1^1(\{A,B\})) + trans_{14}(\{A,B\}), Cost(b_2^1(\{A,B\})) + trans_{24}(\{A,B\}), \\
&\qquad Cost(b_3^1(\{A,B\})) + trans_{34}(\{A,B\}), Cost(b_4^1(\{A,B\}))\}
\end{aligned}$$

Since $Cost(b_3^1(\{A,B\})) = Cost(b_4^1(\{A,B\}))$ and $trans_{x_1,t_1}(\{A,B\}) = trans_{x_2,t_2}(\{A,B\})$ for $x_1 \neq t_1$ and $x_2 \neq t_2$ according to the uniformity assumption, $Cost(a_3^1\ (\{A,B\}))$ is equal to $Cost(a_4^1(\{A,B\}))$. These can be generalized for any i-relation query in the following theorem.

Theorem 2. *Let* $x, y, s, t \in RS(T_{n1}) - RS(T_{ij})$. *Then* $Cost(b_x^1(T_{ij})) = Cost(b_y^1(T_{ij}))$ *and* $Cost(a_s^1(T_{ij})) = Cost(a_t^1(T_{ij}))$.

The above theorem states that all optimal join and transmission plans of T_{ij} at a site not in $RS(T_{ij})$ are redundant. The implication of this theorem is that it is unnecessary to compute the cost of an optimal plan of an i-relation query at all resident sites. Therefore, without losing optimality, it is possible to aggregate the redundant join plans into a single plan whose execution site is marked as *undetermined* and also aggregate the redundant transmission plans into a single plan whose delivery site is also marked as undetermined. The undetermined execution and delivery site of an i-relation query T_{ij} can be any site not in $RS(T_{ij})$. An undetermined site is not yet known when the i-relation query is being optimized. It will be determined later when the query employing the i-relation query as a subquery has been optimized.

Since we introduce the concept of an indefinite site, it becomes impossible to compute the transmission cost between the undetermined sites. From the steps of proving Theorem 2, we fortunately know that the only case of any transmission plan from an undetermined site to another undetermined site to be optimal is when both sites are the same. Thus, we can let this transmission cost be zero. By plan aggregation, we can compute Equations (8) and (9) varying execution and delivery site on the restricted set of sites, that is, $\forall x, t \in RS(T_{ij}) \cup$ {undetermined site} instead of $\forall x, t \in RS(T_{n1})$.

A plan of an i-relation query is *inferior* if there is a lower cost equivalent plan irrespective of the contents of the relations involved in the query [15]. Without losing optimality, we can eliminate the inferior plans of an i-relation query when considering the equivalent plans of the query. Let's reconsider the above chain query. The cost of the optimal transmission plan of a subquery A, B at delivery site 3 is computed as follows.

$$\begin{aligned}
&Cost(a_3^1(\{A,B\}))\\
&= \min\{\ Cost(buildTP_{13}(b_1^1(\{A,B\}))), Cost(buildTP_{23}(b_2^1(\{A,B\}))),\\
&\qquad Cost(buildTP_{33}(b_3^1(\{A,B\}))), Cost(buildTP_{43}(b_4^1(\{A,B\})))\}\\
&= \min\{\ Cost(b_1^1(\{A,B\})) + trans_{13}(\{A,B\}), Cost(b_2^1(\{A,B\})) + trans_{23}(\{A,B\}),\\
&\qquad Cost(b_3^1(\{A,B\})), Cost(b_4^1(\{A,B\})) + trans_{43}(\{A,B\})\}\\
&= \min\{\ trans_{21}(\{B\}) + join_1(\{A\},\{B\}) + trans_{13}(\{A,B\}),\\
&\qquad trans_{12}(\{A\}) + join_2(\{A\},\{B\}) + trans_{23}(\{A,B\}),\\
&\qquad trans_{13}(\{A\}) + trans_{23}(\{B\}) + join_3(\{A\},\{B\}),\\
&\qquad trans_{14}(\{A\}) + trans_{24}(\{B\}) + join_4(\{A\},\{B\}) + trans_{43}(\{A,B\})\}
\end{aligned}$$

According to the uniformity assumption, the join cost at site 3 is equal to that at site 4 because of $RS(\{A,B\}) = \{1,2\}$. Furthermore, the transmission plan from site 4, that is, $buildTP_{43}(b_4^1(\{A,B\}))$ has an additional transmission cost compared to that from site 3. Thus, the transmission plans from site 4 is inferior to that from site 3. We will show that there may exist inferior transmission plans of an i-relation query in the following theorem.

Theorem 3. *Let* $t \in RS(T_{n1})$, $x \in RS(T_{ij}) \cup \{t\}$, $y \in RS(T_{n1}) - \{RS(T_{ij}) \cup \{t\}\}$. *Then* $Cost(buildTP_{xt}(b_x^1(T_{ij}))) \leq Cost(buildTP_{yt}(b_y^1(T_{ij})))$.

Theorem 3 essentially says that, for any delivery site t, a transmission plan of an i-relation query T_{ij} from a site not in $RS(T_{ij}) \cup \{t\}$ is inferior to that from a site in $RS(T_{ij}) \cup \{t\}$. We can then choose the optimal transmission plan without comparing all equivalent plans. Thus, we have

$$\begin{aligned}
&min_{\forall x \in RS(T_{n1})}\{Cost(buildTP_{xt}(b_x^1(T_{ij})))\} =\\
&min_{\forall x \in RS(T_{ij}) \cup \{t\}}\{Cost(buildTP_{xt}(b_x^1(T_{ij})))\} \qquad (10)
\end{aligned}$$

The upshot of Theorem 3 is that we need not search through all the possible transmission plans to find one guaranteed to be optimal. Elimination of inferior plans reduces the execution space That must be searched to find an optimal plans, and thereby makes the process of optimization more efficient.

3.4 Two-Step Pruning

In this subsection, we describe the dynamic programming algorithm to generate an optimal plan of a given query (n-relation query, T_{n1}) at a query site and compute its cost. The optimal cost is obtained by recursively computing the following costs. Let $CostJ_x(T_{ij})$ be the cost of an optimal join plan of T_{ij} at site x and $CostT_t(T_{ij})$ be the cost of an optimal transmission plan of T_{ij} at site t, that is, $CostJ_x(T_{ij}) = Cost(b_x^1(T_{ij}))$ and $CostT_t(T_{ij}) = Cost(a_t^1(T_{ij}))$. By the definition of $CostJ$, $CostT$ and Theorem 2, 3, Equations (8) and (9) are rewritten as follows.

$$CostJ_x(T_{ij}) = min_{\forall p,q,r}\{CostT_{x_1}(T_{rp}) + CostT_{x_2}(T_{i-r,q}) + join_x(T_{rp}, T_{i-r,q})\} \tag{11}$$

$$CostT_t(T_{ij}) = min_{\forall x \in RS(T_{ij}) \cup \{t\}}\{CostJ_x(T_{ij}) + trans_{xt}(T_{ij})\} \tag{12}$$

In Equation (11), if $x \in RS(T_{rp})$, $x_1 = x$. Otherwise, x_1 is an undetermined site. Similarly, if $x \in RS(T_{i-r,q})$, $x_2 = x$. Otherwise, x_2 is an undetermined site. The initial condition for Equation (11) is $CostT_t(T_{1j}) = trans_{xt}(R_j)$ for all $t \in RS(T_{1j}) \cup$ {undetermined site} = {x, undetermined site} where x is the resident site of relation R_j.

If $CostJ_x(T_{ij})$ and $CostT_t(T_{ij})$ are computed for each i-relation query T_{ij} varying x and t over $RS(T_{ij}) \cup$ {undetermined site} and this computations are repeatedly applied for $i = 2, 3, \cdots, n$, $CostT_{query\_site}(T_{n1})$ for n-relation query T_{n1} can be finally obtained. When computing $CostJ_x(T_{ij})$, Equation (11) prunes equivalent join plans except the optimal one. Similarly when computing $CostT_t(T_{ij})$, equivalent transmission plans except the optimal one are pruned by Equation (12). The obtained $CostJ_x(T_{ij})$ and $CostT_t(T_{ij})$ are saved in order to reuse when the query T_{ij} is employed as a subquery of another larger query.

Generating an optimal plan of an i-relation query is the process of constructing feasible optimal subtrees (subplans) of an optimal GET. For each i-relation query, computing the costs $CostJ_x(T_{ij})$ and $CostT_t(T_{ij})$ means computing the costs of optimal subtrees whose roots are the corresponding join and transmission nodes in GET. That is, $CostJ_x(T_{ij})$ is the cost of an optimal subplan whose root is a join node with site label x, and $CostT_t(T_{ij})$ is the cost of an optimal subplan whose root is a transmission node with site label t.

The dynamic programming equations are coded in the algorithm TSP in Figure 4. The algorithm first initializes $CostT_t(T_{1j})$ for each 1-relation query in lines 1-5. In lines 6-23, it then uses Equations (11) and (12) to compute $CostJ_x(T_{ij})$ and $CostT_t(T_{ij})$ for each i-relation query varying x and t over the resident site set of the query and an undetermined site. This process is repeatedly applied in bottom-up fashion, that is, for $i = 2, 3, \cdots, n$.

An optimal GET of a given n-relation query is constructed from that of an i-relation query. For each i-relation query, TSP algorithm maintains not only the optimal join and transmission costs but also additional information such as optimal join pair and execution site in its optimal plan, $OptPlan_t(T_{ij})$. Thus, we can construct an optimal GET of a given query by the postorder traversal from the root node, $OptPlan_{query\_site}(T_{n1})$. The algorithm *Build* in Figure 5 constructs

```
Algorithm TSP
Input: All i-relation query $T_{ij}$'s ($1 \leq j \leq m_i$) and its resident site set $RS(T_{ij})$'s.
Output: An optimal GET with its cost.
1  for j:=1 to $n$ do
2      let $x$ be the resident site of $R_j$;
3      for all $t \in \{x$, undetermined site$\}$ do
4          $CostT_t(T_{1j}) := trans_{xt}(T_{1j})$;
5          save $x$ into $OptPlan_t(T_{1j})$;
6  for i:=2 to $n$ do
7      for j:=1 to $m_i$ do
           /* optimal join plan of $T_{ij}$ at site $x$ */
8          for all $x \in RS(T_{ij}) \cup$ {undetermined site} do   /* execution site */
9              $CostJ_x(T_{ij}) := \infty$;
10             for all pair $T_{rp}, T_{i-r,q}$ s.t. $T_{ij} = T_{rp} \cup T_{i-r,q}$ do
11                 if $x \in RS(T_{rp})$ then $x_1 := x$
                       else $x_1 :=$ undetermined site;
12                 if $x \in RS(T_{i-r,q})$ then $x_2 := x$
                       else $x_2 :=$ undetermined site;
13                 $jc := CostT_{x_1}(T_{rp}) + CostT_{x_2}(T_{i-r,q}) + join_x(T_{rp}, T_{i-r,q})$;
14                 if $CostJ_x(T_{ij}) > jc$ then   /* pruning */
15                     $CostJ_x(T_{ij}) := jc$;
16                     save $p, q, r$ and $CostJ$ into $OptPlan_x(T_{ij})$;
           /* optimal transmission plan of $T_{ij}$ at site $t$ */
17         for all $t \in RS(T_{ij}) \cup$ {undetermined site} do   /* delivery site */
18             $CostT_t(T_{ij}) := \infty$;
19             for all $x \in RS(T_{ij}) \cup \{t\}$ do
20                 $tc := CostJ_x(T_{ij}) + trans_{xt}(T_{ij})$;
21                 if $CostT_t(T_{ij}) > tc$ then   /* pruning */
22                     $CostT_t(T_{ij}) := tc$;
23                     save $x$ and $CostT$ into $OptPlan_t(T_{ij})$;
24 Plan := $OptPlan_{query\_site}(T_{n1})$ with $CostT_{query\_site}(T_{n1})$;
```

Fig. 4. Two-step pruning algorithm

an optimal GET from partial plans. The initial call is Build($n, 1, query\_site$, TRANS).

When constructing an optimal GET,the algorithm Build decides the definite sites of the undetermined sites in lines 2-6. Since the decisions are made in a top-down fashion, an undetermined delivery site of an i-relation query T_{ij} is decided before its undetermined execution site. If the delivery site of an i-relation query T_{ij} is in $RS(T_{ij})$, it is not an undetermined site. In this case, the undetermined execution site may be any site not in $RS(T_{ij})$. But if the delivery site of the query is not in $RS(T_{ij})$, it was the undetermined site when the query was being optimized and its definite site has been decided in previous construction stages. In that case, the undetermined execution site is equal to its delivery site, because the only case of any transmission plan from an undetermined site to another undetermined site to be optimal is when both sites are the same.

```
Algorithm Build($i, j, t, flag$)
Input: Optimal plan of i-relation query $T_{ij}$, $OptPlant(T_{ij})$. If $flag$ is TRANS,
$t$ is the delivery site of $T_{ij}$. Otherwise, $t$ is the execution site of $T_{ij}$.
Output: Execution sequence of operations in an optimal GET.
1   if $flag$ = TRANS then
2       if $OptPlant(T_{ij}).x$ = undetermined site then
            /* execution site is undetermined */
3           if $t \in RS(T_{ij})$ then  /* delivery site is not undetermined */
4               let $OptPlan_t(T_{ij}).x$ be any site not in $RS(T_{ij})$
5           else  /* delivery site was undetermined */
6               $OptPlan_t(T_{ij}).x := t$;
7       if $i \neq 1$ then
8           Build($i, j, OptPlan_t(T_{ij}).x$, JOIN);
9       if $OptPlan_t(T_{ij}).x \neq t$ then
10          print "$TR_{xt}(T_{ij})$ : transfer $T_{ij}$ from site $x$ to $t$"
11  else  /* $flag$ = JOIN */
        /* let the execution site of $T_{ij}$ be the delivery sites of its subqueries */
12      Build($OptPlan_t(T_{ij}).r, OptPlan_t(T_{ij}).p, t$, TRANS);
13      Build($i - OptPlan_t(T_{ij}).r, OptPlan_t(T_{ij}).q, t$, TRANS);
14      print "$JN_t(T_{rp}, T_{i-r,q})$ : join between $T_{rp}$ and $T_{i-r,q}$ at site $t$";
```

Fig. 5. Optimal GET construction algorithm

3.5 Complexity of Algorithm

For each site in the resident site set of an i-relation query and an undetermined site, TSP algorithm considers all equivalent join and transmission plans except the inferior plans and chooses the lowest cost join and transmission plans as the optimal ones. Thus, the time complexity of the algorithm is determined by the number of considered plans, which depends on the initial distribution of relations and the number of subqueries.

The number of i-relation queries is determined by the shape of the query graph and the number of relations in the graph. Let's consider the special cases of a tree query, namely, a *chain query* and a *star query* whose corresponding query graphs are chain and star respectively. For a chain query with n relations, an i-relation query is composed of i consecutive relations. Thus, the number of i-relation queries, m_i, is $n - i + 1$. Also, for a star query with n relations, an i-relation query is constructed by choosing $i - 1$ relations from $n - 1$ relations around the root relation. Thus, we have $m_i = \binom{n-1}{i-1}$. It has been shown that, for a tree query, the total number of subqueries, $\sum m_i$, is the largest with a star query and the smallest with a chain query [12]. Thus, if the initial distributions of relations involved in chain and star queries are equal, the algorithm has the worst case complexity with a star query and the best case with a chain query.

For a tree query with n relations distributed over s sites, let's analyze the complexities of TSP algorithm. The time complexity of the algorithm is determined by the number of plans considered in lines 14 and 21. For a chain query, $m_i = n - i + 1$ and there are $i - 1$ pairs of subqueries for each i-relation query. The cardinality of the resident site set of an i-relation query varies according to the initial distribution and is no more than s. Thus, the number of join plans considered in line 14 is

$$\sum_{i=2}^{n}\sum_{j=1}^{n-i+1}\sum_{x=1}^{|RS(T_{ij})|+1}\sum_{p=1}^{i-1} 1 \leq \sum_{i=2}^{n}\sum_{j=1}^{n-i+1}\sum_{x=1}^{s+1}\sum_{p=1}^{i-1} 1 = (s+1)\frac{(n-1)n(n+1)}{6}$$

and the number of transmission plans considered in line 21 is

$$\sum_{i=2}^{n}\sum_{j=1}^{n-i+1}\sum_{t=1}^{|RS(T_{ij})|+1}\sum_{x=1}^{|RS(T_{ij})|+1} 1 \leq \sum_{i=2}^{n}\sum_{j=1}^{n-i+1}\sum_{t=1}^{s+1}\sum_{x=1}^{s+1} 1 = (s+1)^2\frac{(n-1)n}{2}$$

Clearly, considering join and transmission plans take time $O(sn^3)$ and $O(s^2n^2)$ respectively. Because s is less than or equal to n, the total best case time complexity is less than $O(sn^3)$.

For a star query, the process of complexity computation is the same as that of the chain query except $m_i = \binom{n-1}{i-1}$. Thus, the numbers of join and transmission plans considered in lines 14 and 21 are

$$\sum_{i=2}^{n}\sum_{j=1}^{\binom{n-1}{i-1}}\sum_{x=1}^{|RS(T_{ij})|+1}\sum_{p=1}^{i-1} 1 \leq \sum_{i=2}^{n}\sum_{j=1}^{\binom{n-1}{i-1}}\sum_{x=1}^{s+1}\sum_{p=1}^{i-1} 1 = (s+1)(n-1)2^{n-2}$$

and

$$\sum_{i=2}^{n}\sum_{j=1}^{\binom{n-1}{i-1}}\sum_{t=1}^{|RS(T_{ij})|+1}\sum_{x=1}^{|RS(T_{ij})|+1} 1 \leq \sum_{i=2}^{n}\sum_{j=1}^{\binom{n-1}{i-1}}\sum_{t=1}^{s+1}\sum_{x=1}^{s+1} 1 = (s+1)^2(2^{n-1}-1)$$

Therefore, the total worst case time complexity is $O(sn2^{n-1})$ according to the same reason.

The above analysis only shows upper bounds of complexities because we use the maximum upper bound of the cardinality of a resident site set, that is, $|RS(T_{ij})| = s$ instead of a precious bound. Actually, $|RS(T_{ij})|$ is bounded by the following inequalities.

$$\begin{cases} 2 \leq i \leq s, \ |RS(T_{ij})| \leq i \\ s < i \leq n, \ |RS(T_{ij})| \leq s \end{cases}$$

The complexities are lowered by using these actual upper bounds of the cardinality. For example, the number of join plans considered in the case of a star query is rewritten as follows.

$$\sum_{i=2}^{n}\sum_{j=1}^{\binom{n-1}{i-1}}\sum_{x=1}^{|RS(T_{ij})|+1}\sum_{p=1}^{i-1}1 \leq \sum_{i=2}^{s}\sum_{j=1}^{\binom{n-1}{i-1}}\sum_{x=1}^{i+1}\sum_{p=1}^{i-1}1 + \sum_{i=s+1}^{n}\sum_{j=1}^{\binom{n-1}{i-1}}\sum_{x=1}^{s+1}\sum_{p=1}^{i-1}1$$
$$= \sum_{i=1}^{s-1}\binom{n-1}{i}(i+2)i + (s+1)\sum_{i=1}^{n-s}\binom{n-1}{i+s-1}(i+s-1)$$

Since this equation can not be expressed as a closed form of n and s [4], we can not help using the maximum upper bound of the cardinality. This implies that the above analysis does not reflect the effects of plan aggregation and elimination unfortunately. However, the actual optimization work is considerably smaller compared to the analysis.

Though the number of considered plans is reduced by plan aggregation and elimination, the worst case complexity is still exponential in terms of the number of relations in the query. We do not believe that the exponential complexity at the worst case hinders the practicality of our algorithm. This is because queries with more than 10 joins are rare. In a survey of 30 major DB2 customers [14], for example, the most complex join query involves just 8 relations. Furthermore, a star query is just a hypothetical query that is rarely made by a user.

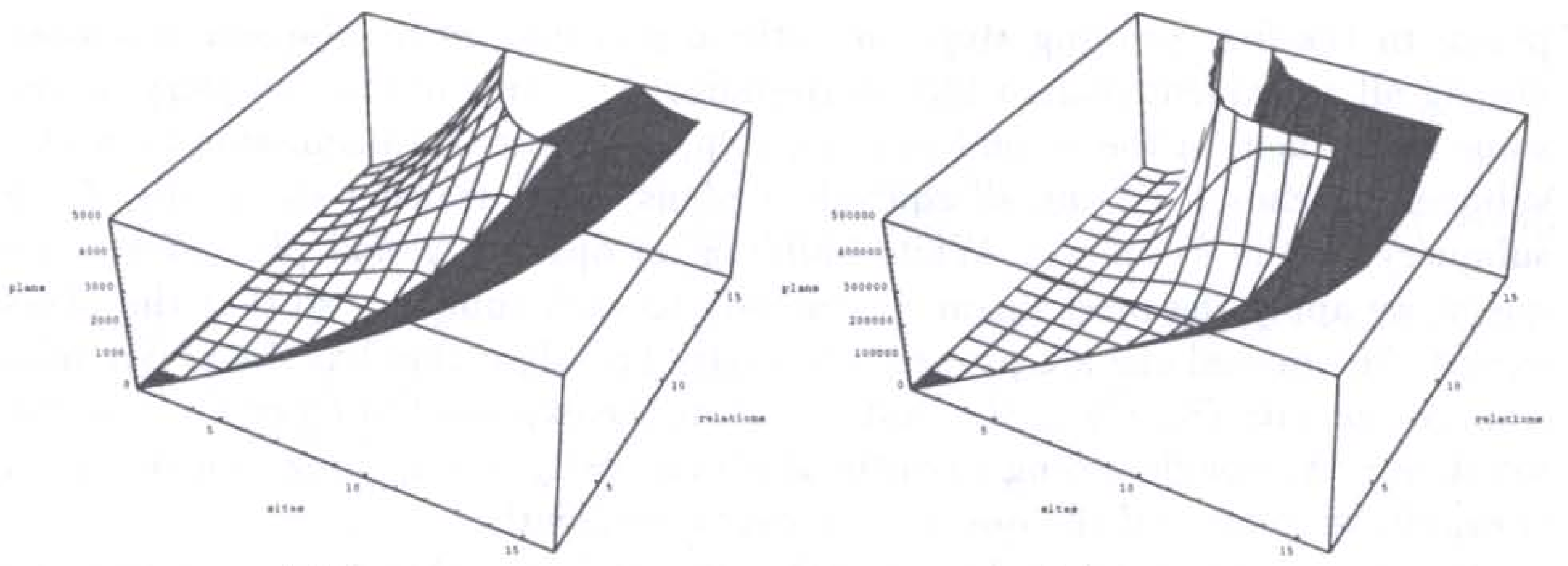
3.6 Comparison

Though OSP, semi-TSP and our TSP algorithm all employ dynamic programming as a search strategy, their characteristics and search efficiencies are different. The time complexity of OSP algorithm is determined by the number of plans considered in Equation (1). For chain and star queries,these numbers are computed as follows.

$$\sum_{i=3}^{n}\sum_{j=1}^{n-i+1}\sum_{t=1}^{|RS(T_{ij})|+1}\sum_{p=1}^{2}\sum_{x=1}^{|RS(T_{ij})|+1}1 \leq \sum_{i=3}^{s}\sum_{j=1}^{n-i+1}\sum_{t=1}^{i+1}\sum_{p=1}^{2}\sum_{x=1}^{i+1}1 + \sum_{i=s+1}^{n}\sum_{j=1}^{n-i+1}\sum_{t=1}^{s+1}\sum_{p=1}^{2}\sum_{x=1}^{s+1}1$$

$$\sum_{i=3}^{n}\sum_{j=1}^{\binom{n-1}{i-1}}\sum_{t=1}^{|RS(T_{ij})|+1}\sum_{p=1}^{i-1}\sum_{x=1}^{|RS(T_{ij})|+1}1 \leq \sum_{i=3}^{s}\sum_{j=1}^{\binom{n-1}{i-1}}\sum_{t=1}^{i+1}\sum_{p=1}^{i-1}\sum_{x=1}^{i+1}1 + \sum_{i=s+1}^{n}\sum_{j=1}^{\binom{n-1}{i-1}}\sum_{t=1}^{s+1}\sum_{p=1}^{i-1}\sum_{x=1}^{s+1}1$$

In Figure 6, we draw diagrams concerning the number of plans considered in OSP and TSP algorithms varying n and s such that $n \geq 2$ and $n \geq s \geq 2$. In the figure, the bright surface represents the number of plans considered in OSP algorithm and the dark surface, the number of plans considered in TSP algorithm. By comparing the two surfaces, we know that the number of plans considered in TSP algorithm is smaller than that in OSP algorithm when $n \geq 3$ and $s \geq 3$. As OSP algorithm does not search the bushy execution plans, this algorithm produces a suboptimal plan. Therefore, in most cases, our TSP algorithm produces an optimal plan while considering less partial plans than the suboptimal algorithm OSP.



(a) best case (b) worst case

Fig. 6. The number of plans considered in OSP and TSP algorithm

The time complexity of semi-TSP algorithm is determined by the number of trajectories considered in Equation (3) and this number depends on the number of states generated. As in the cases of algorithms OSP and TSP, the number of states generated is determined by the initial distribution and the shape of the query graph. Contrary to these two algorithms, the number is the smallest with a star query because the bushy states like $(-;-;AB,CD;-)$ are not generated with a star query. Thus, semi-TSP algorithm has the best case time complexity with a star query. However, if all n relations are stored at different sites, the number of states generated with a star query is exponential in n (though that number is reduced by the equivalence class technique in [9]). Thus, in case of this initial distribution, the best case time complexity is also exponential in n. Since our TSP algorithm has a polynomial time complexity at the best case with this initial distribution, and its extra work in generating all subqueries is much less than that in constructing a state space of semi-TSP algorithm, our algorithm has computational advantages over semi-TSP algorithm.

4 Conclusion

We have presented an algorithm which determines the optimal global plan for a tree query in a distributed database. The objective function of optimization is the total processing time including not only the communication costs but also the local processing costs. The optimizer we have developed considers the execution plans which may be deep or bushy trees. The search work done by our optimizer is reduced by applying dynamic programming technique. We devise the two-step pruning mechanism to choose an optimal plan among all feasible plans. In the first pruning step, an optimal join plan of a subquery is chosen among all equivalent plans which materialize the result of the subquery at the same site. Then, in the second pruning step, an optimal transmission plan of a subquery is chosen among all equivalent plans which transfer the result of the subquery to the same site. While building an optimal global plan of a given query, we apply the mechanism successively to each subquery so that the plans except the optimal one are pruned efficiently. The algorithm has the polynomial time complexity $O(sn^3)$ at the best case, but the exponential $O(sn2^{n-1})$ at the worst case. Although finding an optimal plan is costly, the optimization overhead is rapidly amortized if the query is executed frequently.

We show that, without losing optimality, it is possible to aggregate some redundant plans into a single plan. Furthermore, we can always guarantee the optimality though we eliminate the inferior plans from the execution space of our optimizer. This means that we can produce the optimal plan without exhaustive search. Clearly, the plan aggregation and elimination significantly reduce the optimization work.

Future work includes: extending the algorithm to minimize the response time of processing a query and to be used in the case of recursive queries, elaborating the cost model to predict the cost of query processing exactly.

Acknowledgement

This work was partially supported by the Korea Science and Engineering Foundation (KOSEF) under Grant No. 931-0900-022-2.

References

1. Apers, P., Hevner, A., and Yao, S.B., "Optimization algorithm for distributed queries", IEEE Trans. Software Engineering, SE-9, No.1, Jan. 1983, pp.57-68.
2. Bernstein, P.A., Goodman, N., Wong, E., Reeve, C.L., and Rothnie, Jr., J.B., "Query Processing in a System for Distributed Databases (SDD-1)", ACM Trans. Database Systems, Vol.6, No.4, Dec. 1981, pp.602-625.
3. Chiu, D-M, Bernstein, P. and Ho, Y-C, "Optimizing chain queries in a distributed database system", SIAM Journal of Computing, Vol.13, No.1, Feb., 1984, pp.116-134.
4. Graham, R.L., Knuth, D.E., and Patashnik, O., *Concrete Mathematics*, Addison-Wesley Publishing Company, Inc., 1989, pp.165-166.
5. Hevner, A.R., and Yao, S.B., "Querying Distributed Databases on Local Area Networks", Proceedings of The IEEE, Vol.75, No.5, May 1987, pp.563-572.
6. Horowitz, E., Sahni, S., *Fundamentals of Computer Algorithms*, Computer Science Press, Inc., 1978, pp.198-202.
7. Ioannidis, Y.E., Kang, Y.C., "Left-deep vs. bushy trees: An analysis of strategy spaces and its implications for query optimization", Proc. ACM SIGMOD, May 1991, pp.168-177.
8. Kim, H., Lee, S., Kim, H.-J., "Two-step pruning algorithm for distributed query optimization", Technical report DBTR-95-1, 1995.

9. Lafortune, S., and Wong, E., "A state transition model for distributed query processing", ACM Transactions on Database Systems, Vol.11, No.3, Sep. 1986, pp.294-322.

10. Lohman, G.M., Mohan, C., Haas, L.M., Lindsay, B.G., Selinger, P.G., Wilms, P.F., "Query processing in R*", IBM Research Report RJ4272, San Jose, Calif., April 1984.

11. Mackert, L.M., Lohman, G.M., "R* optimizer validation and performance evaluation for distributed queries", Proc. VLDB, August 1986, pp.149-159.

12. Ono, K. and Lohman, G.M., "Measuring the complexity of join enumeration in query optimization", Proc. VLDB, August 1990, pp.314-325.

13. Ozsu, M.T., and Valduriez, P., *Principle of Distributed Database Systems*, Prentice-Hall International, Inc., 1991, pp.230-252.

14. Tsang, A., Olschanowsky, M., "A study of Database 2 customer queries", Technical Report 03.413, IBM Santa Teresa Laboratory, April 1991.

15. Yu, C.T., "Optimization of distributed tree query", Journal of Computer and System Science, Academic Press, Inc., Vol.29, 1984, pp.409-445.

16. Yu, C.T. and Chang C.C., "Distributed query processing", ACM Computing Surveys, Vol.16, No.4, Dec., 1984, pp.399-432.