Data & Knowledge Engineering 3 (1988/89) 261–284
North-Holland

A knowledge-based approach to multiple query processing

J.T. PARK*, T.J. TEOREY and S. LAFORTUNE

Computing Research Laboratory, Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI 48109-2122, U.S.A.

Abstract. The collective processing of multiple queries in a database system has recently received renewed attention due to its capability of improving the overall performance of a database system and its applicability to the design of knowledge-based expert systems and extensible database systems. A new multiple query processing strategy is presented which utilizes semantic knowledge on data integrity and information on predicate conditions of the access paths (plans) of queries. The processing of multiple queries is accomplished by the utilization of subset relationships between intermediate results of query executions, which are inferred employing both semantic and logical information. Given a set of fixed order access plans, the A^* algorithm is used to find the set of reformulated access plans which is optimal for a given collection of semantic knowledge.

Keywords. Database systems, Query optimization, Semantic knowledge.

1. Introduction

As knowledge-based systems are extended to more complex problems requiring large volumes of information and knowledge, the need for efficient processing of multiple queries and updates in a distributed environment becomes critical [7]. The handling of multiple queries is also found in extensions to existing database languages for the support of CAE applications such as VLSI design [9], and in deductive database systems [8].

An independent optimization of queries may overlook potential savings which can be achieved when queries are optimized collectively. We address the collective optimization of a set of queries such that, given a set of individual access plans of queries, a set of alternative access plans of queries which exhibits minimum cost is found using semantic knowledge on data objects.

The collective processing of batches of queries and update operations has been a popular technique in conventional file systems of the sixties and early seventies [10, 34]. The majority of the research in this area has focused on the processing of multiple queries in centralized DBMSs [6, 8, 10, 14, 16, 28, 32]. We consider a new technique based on the concept of subquery relationship for the efficient processing of multiple transactions which occur almost simultaneously in both centralized and distributed computing environments. Both the knowledge on the semantic data integrity constraints and the information of logical predicate conditions of the access plans of queries are utilized in order to find a set of reformulated distributed query execution plans that exhibit minimum cost. The task of processing multiple queries is achieved by a rule-based expert system, Multiple Transaction Processor (MTP), which employs a planning technique combined with a problem solving method. The plan step

* Current address: Jong-Tae Park, Samsung Semiconductor & Telecommunications Co., Ltd., San14, Nongseo-Ri, Kihung-Eup Yongin-Kun, Kyungki-Do, Korea.

infers the necessary constraints as in Dendral [18], and the problem solving step searches the state space to find an optimal solution using the A^* algorithm [24].

Query processing can be categorized as individual or multiple. Individual query processing implies that each query is processed independently with respect to other queries [13, 39]. Multiple query processing attempts to collectively optimize access plans of a set of queries occurring either simultaneously or not, by utilizing the commonality which exists among the set of queries in terms of accesses to relations, join/semi-join operations, and local physical data access [5, 6, 8, 10, 14, 16, 26, 28, 32, 34, 36].

Multiple query processing may be further classified as semantic or nonsemantic. Semantic query processing implies that semantic knowledge such as functional dependencies and semantic data integrity constraints is utilized to achieve more efficient query processing [4, 11, 12, 18, 15, 25, 35].

Finally, queries can be specified as either concurrent or nonconcurrent. In the concurrent case, a batch of transactions is assumed to occur almost simultaneously within a given time unit. The nonconcurrent case corresponds to the conventional data allocation problem in which frequencies of occurrences of transactions are given [2, 5, 6, 10, 14, 16, 28, 34, 36, 37]. The processing of nonconcurrent multiple queries attempts to improve the overall system performance by storing or creating fast access paths via index or pointers for the intermediate results of queries which do not necessarily occur concurrently.

The assumptions that we make are as follows. A precompiled individual optimal access plan for each query is available. For example, the system R^* query optimizer [21] generates the *global plan* for a query which is a procedural sequence of operations such as the accesses to relation, projection, join, inter-site transfers, and sorts whose estimated execution cost is minimal. The global plan is in a high-level form that lacks internal representations such as its parse tree structure or machine-executable code. Either joins or semi-joins or both are used as query processing tactic [1, 17, 19]. We assume that the speed of the computer network is relatively high such that the local processing cost cannot be negligible. The predicate conditions of a query are in a conjunctive form as assumed by other relevant research [39].

The paper is organized as follows. In Section 2, the subquery relationship is defined along with examples. In Section 3, the knowledge for efficient processing of multiple queries is described, and we present an algorithm for the reformulation of access plans of queries. In Section 4, the state space representation of the problem is formally presented, and the admissibility of the heuristic cost estimates for both general and simple cases is proved. The operation of MTP for a simple case is illustrated through an example in Section 5. Finally, a discussion on performance and some conclusions follow in Sections 6 and 7, respectively.

2. View identification and subquery definition

We assume that the reader is familar with the basic concepts of relational database theory [22]. Let DOM(A) be the *domain* of attribute A. Let t, u, v and w denote tuple variables. Consider relation r on scheme R, and let Λ denote an empty predicate formula. The standard relational operators will be denoted as follows: Projection, $\pi_X(r)$ with $X \subseteq R$; selection, $\sigma_P(r)$, with P a 1st-order predicate formula; join (equi-join), $J_P(r_1, r_2)$, with P a conjunction of equality clauses; and Cartesian product, $C(r_1, r_2) = J_\Lambda(r_1, r_2)$.

A relation in a database is referred to as a *base relation*. A *view* is the result of the execution of a relational operator such as selection (σ), projection (π), join (J), Cartesian product (C), union (U) and difference (D), on relations in a database. An *access plan* of a query is a sequence of relational operators applied to relations to get its result. When a view V is a subset of another view V', we say that there is a subset relationship between two views V and V'.

Processing multiple queries requires the identification of subset relationships between intermediate results (views) of queries, since some can be used for the processing of others. In a distributed environment, the recognition of these relationships among different access plans of queries can reduce the overall processing cost substantially by eliminating many expensive intersite joins. Such subset relationships can be inferred either from logical information of queries such as predicate conditions of queries, or from semantic information such as semantic data integrity constraints and functional dependencies existing among the attributes of each relation.

Example 2.1. Consider relations r_1 and r_2 with schemes AB and CD respectively, and two views $V_1 = J_{P_1}(r_1, r_2)$ and $V_2 = J_{P_2}(r_1, r_2)$, where $P_1 \equiv (A = C)$ and $P_2 \equiv (A = C) \wedge (B = D)$. Since P_2 is more restrictive, we know that V_2 is a subset of or equal to V_1. In this case, the relationship between the join predicates is represented by $\forall u_{/R_1} \forall v_{/R_2}(P_2(u, v) \Rightarrow P_1(u, v))$ in a closed well-formed formula. For notational convenience, it is denoted as $P_2 \Rightarrow P_1$.

Example 2.1 illustrates that a subset relationship between views can be inferred from logical information on predicates of queries. We now illustrate that a subset relationship can also be inferred from the semantic knowledge on the database such as semantic data integrity constraints and functional dependencies.

Example 2.2. Consider an automobile insurance company which maintains a distributed database containing two relations OWNER and ISSUER at sites S_1 and S_2, respectively, and whose schemes are as follows: ISSUER (<u>REPNAME</u>, BRANCH) and OWNER (<u>NAME</u>, ADDRESS, AGE, SEX, INCOME, INSURANCE), where underlined attributes denote the primary key of the corresponding relation. REPNAME is the name of the insurance representative at the given branch of the company, and the other attributes are self-explanatory.

We assume that the following semantic knowledge is obtained from the analysis of the user's requirements: "All representatives are living at cities at which they work, and, by managerial policy of the company, they are insured as owners at the branch where they work."

Consider two queries QT_1 and QT_2 which occur at site S_2; the first says "List the names of the owners insured by the company who work for the company as representatives"; and the second says "List the names of the owners insured by the company who live at cities where a branch of the company is located." From the knowledge, it is inferred that if the attribute NAME of OWNER is equal to REPNAME of ISSUER, then OWNER.ADDRESS is equal to ISSUER. BRANCH. Since the attributes NAME and REPNAME are keys of relations OWNER and ISSUER, respectively, it is found that $J_{P_1}(r_1, r_2)$ is a subset or equal to $J_{P_2}(r_1, r_2)$ where r_1 is OWNER; r_2 is ISSUER; $P_1 \equiv (\text{NAME} = \text{REPNAME})$ and $P_2 \equiv (\text{ADDRESS} = \text{BRANCH})$. The relationship between the join predicates P_1 and P_2 associated with QT_1 and QT_2 is represented by the assertion, $\forall u_{/R_1} \forall v_{/R_2} (P_1(u, v) \Rightarrow P_2(u, v))$.

The access plan of a query can be represented by a *query graph* **G** [28] which is a triple (N, E, f_D) where N is a set of nodes; $E \subseteq N \times N$ is a set of directed edges, and f_D: $N \rightarrow 2^{2^N}$ is called the decomposition mapping for **G**. Each node of **G** corresponds to a view, and it contains information on both the view and the processing method to obtain that view. $f_D(n_i)$ denotes the set of all possible sets of nodes in N from which the view corresponding to n_i can be constructed using suitable relational operations. The situation described in Example 2.2 is shown in Fig. 1a. It indicates that relation r_1 at site S_1 is transmitted to site S_2 to be joined with r_2 at site S_2, and the result of the join, either V_1 or V_2, is available at site S_2. If (V_1, S_2) is replaced by (V_1, S_1) in Fig. 1a, r_2 is transferred from site S_2 to site S_1 to be joined, and V_1 is available at site S_1. In this way, the distributed access plan of a query can be represented precisely.

Since $J_{P_1}(r_1, r_2) \subset J_{P_2}(r_1, r_2)$, view V_1 can be obtained by accessing view V_2 as shown in Fig. 1b. That is, two intersite joins (a) are replaced by one intersite join and one local selection (b). This can significantly reduce the total processing cost due to I/O, CPU and data transmission across the network. In this case, query QT_1 is said to be a *subquery* of query QT_2, meaning that the (intermediate) result of the former can be obtained from that of the latter. Alternatively, QT_2 is called a *superquery* of QT_1.

"View identification" addresses this recognition of subset relationships between access plans of two different queries to optimize collectively the processing of multiple queries. Generally, these relationships depend on both logical and semantic information, as shown in Examples 2.1 and 2.2. We will refer to such assertions as *integrity constraints*, denoted ICs. Throughout this paper, we assume that integrity constraints of the form $P_1 \Rightarrow P_2$ are inferred from either logical or semantic information. (More details on view identification can be found in [25].)

It is assumed that a distributed query processing strategy processes all the (unary) projection operations before any binary relational operations, as in [3]. All the attributes of relations which are required for join conditions and target list of analyzed queries are assumed to be projected before the intermediate result is transferred to the next site.

3. Multiple transaction processor: Reformulation algorithm

Multiple Transaction Processor (MTP) is a rule-based expert system for the collective processing of multiple queries. Its operation is divided into two steps: a "plan step" and a "search step". The plan step infers integrity constraints which can be employed in the following search step for the generation of superqueries. The search step uses the A^* heuristic search strategy [24]. In this paper, we shall only discuss the search step. The plan step is described in detail in [25]. It uses several heuristic rules to generate a set of ICs. This set is then pruned to retain only the "promising" ICs, where an IC is said to be promising if it can be used for the generation of a superquery.

We now describe the algorithm for the reformulation of access plans of queries during the search step. For simplicty, we omit site information in this section. The generalization to the case of a distributed database is straightforward (see next section). For the relational operations selection, projection and join, there are corresponding construction rules for reformulation.

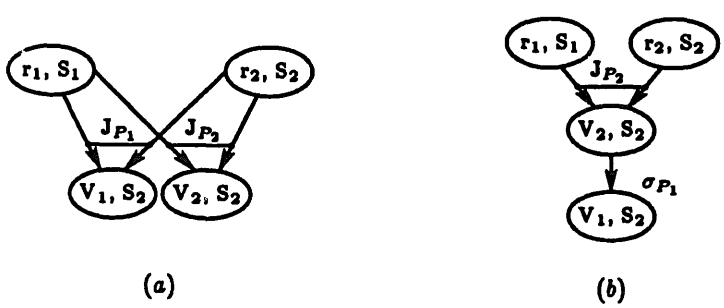

Fig. 1. Two joins are replaced by one join and one selection.

Since join is considered to be the most expensive operation in both centralized DBMSs and distributed DBMSs, it is assumed that the access plan of a query is represented by a sequence of join operations where local unary relational operations are considered aggregately. Let uop_i be a unary relational operator. A sequence of unary operations $uop_{i+k}(uop_{i+k-1}(\ldots(uop_i(V_i)\ldots)))$ applied to a view V_i is represented by $\mathbf{uop}_i(V_i)$ for notational convenience. The *access plan* of a query is then represented by the sequence $\langle J_{P_2}(V_1, V_2),\ J_{P_4}(V_3, V_4), \ldots, J_{P_{2n}}(V_{2n-1}, V_{2n})\rangle$ where, for $l = 1, 2, \ldots, V_{2l} = \mathbf{uop}_{2l}(J_{P_j}(V_{j-1}, V_j))$ and $V_{2l-1} = \mathbf{uop}_{2l-1}(J_{P_i}(V_{i-1}, V_i))$ for some even $i, j, 2 \leq i, j < 2l$. Here, each join operation, $J_{P_i}(V_{i-1}, V_i)$ for $i = 2, 4, \ldots, 2n$ in the access sequence, is defined as an *access step* of the sequence $\langle J_{P_2}(V_1, V_2), J_{P_4}(V_3, V_4), \ldots, J_{P_{2n}}(V_{2n-1}, V_{2n})\rangle$. In Theorem 1 of Section 4.2, each individual query access plan will be assumed to be optimal. But until then, this restriction is not enforced.

1. *Construction rule for the selection operation.*

Let us assume that there are two selection operations $\sigma_{Q_1}(V_k)$ and $\sigma_{Q_2}(V_k)$ where V_k is a view, as shown in Fig. 2a. Suppose that there are two ICs, $Q_1 \Rightarrow Q_3$ and $Q_2 \Rightarrow Q_3$. The construction rule allows for the query graph on the left-hand side to be transformed into that on the right-hand side; two selection operations σ_{Q_1} and σ_{Q_2} are replaced by one selection operation σ_{Q_3}. It is noted that the views V_1' and V_2' can be derived from the view V_3' in the reformulated query graph as follows; $V_1' = \sigma_{Q_1}(V_3')$, $V_2' = \sigma_{Q_2}(V_3')$.

2. *Construction rule for the projection operation.*

The projection operations $\pi_{W_1}(V_3')$ and $\pi_{W_2}(V_3')$ are replaced by the projection $\pi_{W_1 \cup W_2}(V_3')$ as shown in Fig. 2b. The views V_1'' and V_2'' can be derived from the view V_3'' in the reformulated query graph since $V_1'' = \pi_{W_1}(V_3'')$, and $V_2'' = \pi_{W_2}(V_3'')$.

3. *Construction rule for the join operation.*

Let $V_1 = J_{P_1}(V', V'')$ and $V_2 = J_{P_2}(V', V'')$. If there are two ICs, $P_1 \Rightarrow P_3$ and $P_2 \Rightarrow P_3$, one join operation J_{P_3} substitutes for two join operations J_{P_1} and J_{P_2}. The views V_1 and V_2 can also be derived from the view V_3; $V_1 = \sigma_{P_1}(V_3)$ and $V_2 = \sigma_{P_2}(V_3)$. Two join operations are replaced by one join operation, as shown in Fig. 3.

Now, let us look at a more general case in which the length of the query access plan is greater than one. For notational convenience, let us define V_1, V_2, V_4 and V_5 as follows: $V_1 = J_{P_1}(\pi_{X_1}(V_i),\ \pi_{X_2}(V_j))$, $V_2 = J_{P_2}(\pi_{X_1}(V_i),\ \pi_{X_2}(V_j))$, $V_4 = J_{P_4}(\pi_{Y_1}(V_1),\ \pi_{W_1}(\sigma_{Q_1}(V_k)))$, and $V_5 = J_{P5}(\pi_{W_2}(\sigma_{Q_2}(V_k)))$. Suppose that there are two queries whose access plans are described below.

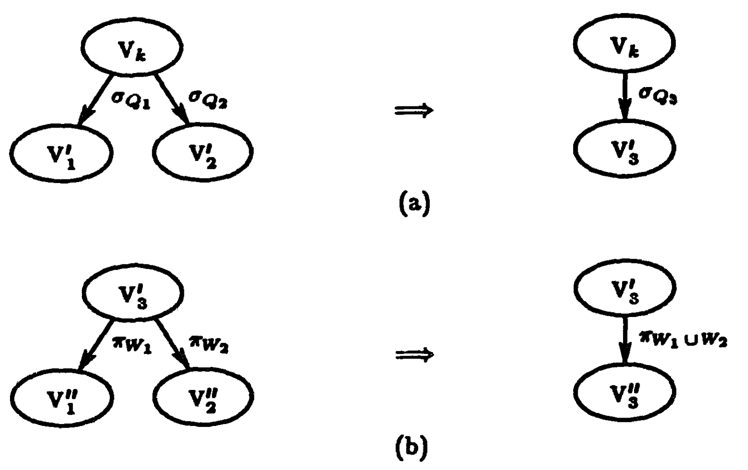

Fig. 2. Construction rules for (a) selection and (b) projection operations.

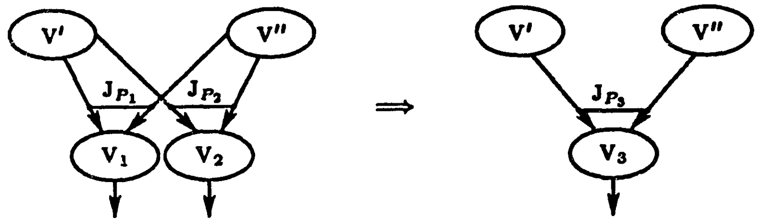

Fig. 3. Construction rule for the simple join operation.

- access-plan ($QT_l^{r_l}$): $\langle J_{P_1}(\pi_{X_1}(V_i), \pi_{X_2}(V_j)), J_{P_4}(\pi_{Y_1}(V_1), \pi_{W_1}(\sigma_{Q_1}(V_k))), \pi_{Z_1}(V_4)\cdots\rangle$
- access-plan ($QT_m^{r_m}$): $\langle J_{P_2}(\pi_{X_1}(V_i), \pi_{X_2}(V_j)), J_{P_5}(\pi_{Y_2}(V_2), \pi_{W_2}(\sigma_{Q_2}(V_k))), \pi_{Z_2}(V_5)\cdots\rangle$

(The superscripts r_l and r_m denote the set of relations referenced by queries QT_l and QT_m, respectively.)

The following ICs are assumed to be inferred from the knowledge base; $P_1 \Rightarrow P_3$, $P_2 \Rightarrow P_3$, $Q_1 \Rightarrow Q_3$, $Q_2 \Rightarrow Q_3$, $P_4 \Rightarrow P_6$, and $P_5 \Rightarrow P_6$. The rule for the simple join operation in Fig. 3, using ICs $P_1 \Rightarrow P_3$ and $P_2 \Rightarrow P_3$, allows to obtain the reformulated query graph shown in Fig. 4a. Here, it is said that a superquery $QG_1^{\{V_i,V_j\}}$ of $QT_l^{r_l}$ and $QT_m^{r_m}$ is generated whose access plan is $J_{P_3}(\pi_{X_1}(V_i), \pi_{X_2}(V_j))$.

The *current access step*, denoted CAS, of a query (superquery) is defined to be the most recently reformulated access step of the access plan of the query. The *current view* of a query is defined as the result of the execution of the current access step of the query. The CAS of both $QT_l^{r_l}$ and $QT_m^{r_m}$ is $J_{P_3}(\pi_{X_1}(V_i), \pi_{X_2}(V_j))$ in Fig. 4a. Since the number of eliminated intersite join operations tends to be proportional to the length of the access sequence of the superquery, it may be a good strategy for the reduction of the total processing cost to stretch out the superquery $QG_1^{\{V_i,V_j\}}$ to $QG_1^{\{V_i,V_j,V_k\}}$ such that the intermediate results V_4 and V_5 of the executions of the queries $QT_l^{r_l}$ and $QT_m^{r_m}$ can be obtained from $QG_1^{\{V_i,V_j,V_k\}}$. Exploiting the knowledge $Q_1 \Rightarrow Q_3$, $Q_2 \Rightarrow Q_3$, $P_4 \Rightarrow P_6$, and $P_5 \Rightarrow P_6$ in the search step, the query graph of Fig. 4a is transformed into that of 4b. It is noted that V_4 and V_5 can be obtained from V_6 which is the result of the execution of $QG_1^{\{V_i,V_j,V_k\}}$. In this case, $QG_1^{\{V_i,V_j\}}$ is said to be *extended* into $QG_1^{\{V_i,V_j,V_k\}}$. Here, two join operations are again replaced by one join operation. This motivates the following heuristic in the search step:

Generate a superquery whose access plan is as long as possible.

In order to describe the above construction rule in more detail, we need to introduce some definitions. Let $V = \sigma_P(r_i)$. *attr*(P) is defined as the set of all attributes appearing in the selection predicate P. For example, if $V = \sigma_{(A=B)\wedge(C=\text{"value"})}(r_i)$, *attr*$((A = B) \wedge (C = \text{"}value\text{"}))$ = ABC. Similarly, *lattr*(P) and *rattr*(P) are defined for the join predicate P. Let $V = J_P(r_i, r_j)$ with the schemes of r_i and r_j being R_i and R_j respectively. *lattr*(P) is defined as the set of all attributes appearing in both P and R_i; *rattr*(P) as the set of all attributes in both P and R_j. For example, *lattr*(P) = A and *rattr*(P) = B for $J_{R_i.A=R_j.B}(r_i, r_j)$.

We now describe the construction rule of the join operation in Fig. 4. First, the conditions for the extensibility of superquery $QG_1^{\{V_i,V_j\}}$ are checked. This consists of the identification of any ICs with respect to join predicates P_4 and P_5. If this condition is satisfied, we then check the feasibility of extending the superquery by verifying the conditions related to the local selection and projection operations. Since these conditions are satisfied by $P_4 \Rightarrow P_6$ and $P_5 \Rightarrow P_6$, and $Q_1 \Rightarrow Q_3$ and $Q_2 \Rightarrow Q_3$, we try to construct the intermediate result from which both intermediate results of $QT_l^{r_l}$ and $QT_m^{r_m}$ can be obtained. This construction of the intermediate result of the superquery requires the projection operation π_{R_1} to be carried out, in order to provide the necessary attributes and the corresponding data for the selection

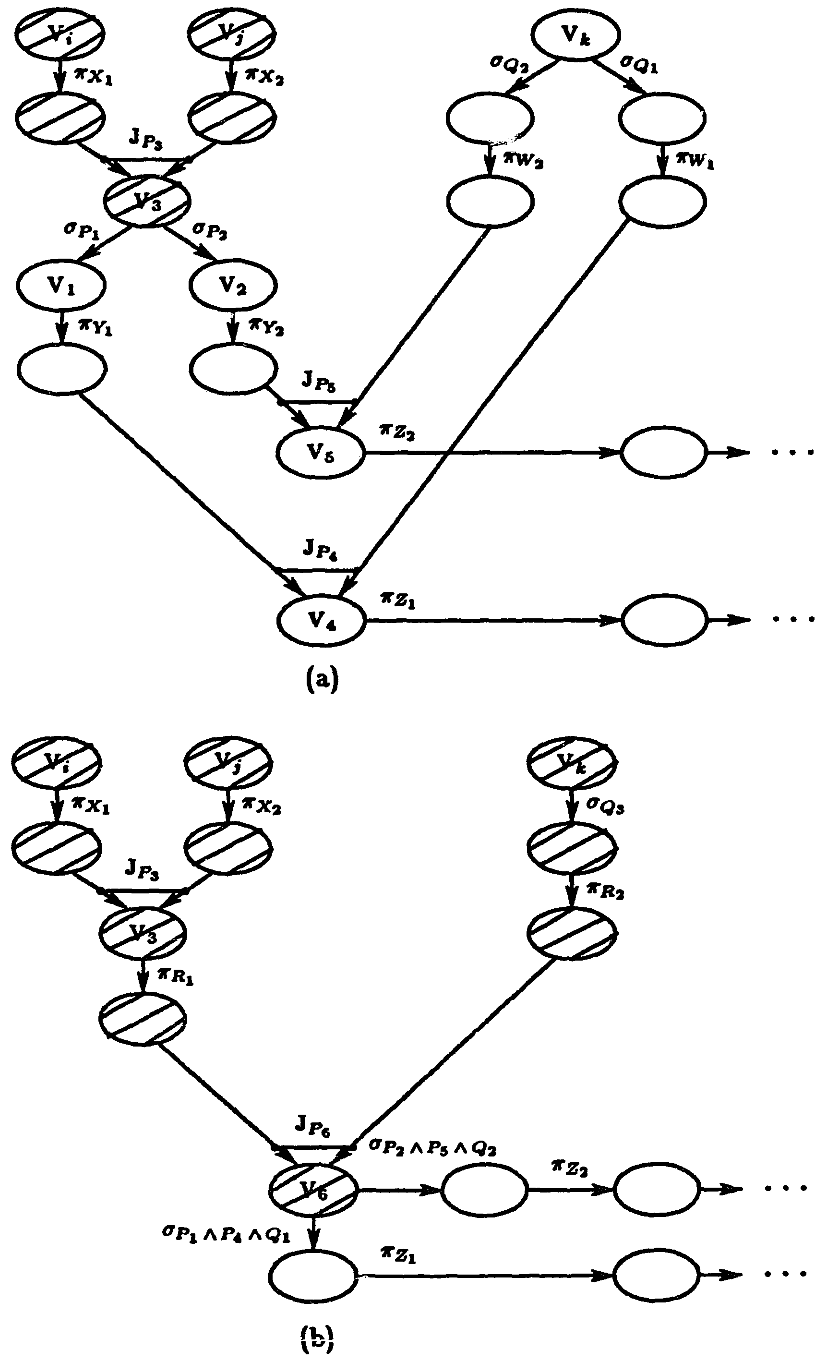

Fig. 4. Construction rule including unary and binary operations.

operations σ_{P_1} and σ_{P_2} to be executed at the next access step. It must also provide the necessary attributes for the join operation J_{P_6}. Here, $R_1 = (\cup_{i=1}^{2} Y_i) \cup (\cup_{i=1}^{2} attr(P_i)) \cup lattr(P_6)$.

Second, the construction rules for the local selection and projection operations, σ_{Q_1} and σ_{Q_2}, are applied to the view V_k. The join operation J_{P_6} also requires the attribute $lattr(P_6)$ to be projected. The selection operation σ_{Q_1} and σ_{Q_2} are also deferred to the next stage, which requires the attributes $attr(Q_1)$ and $attr(Q_2)$. Since all these projection operations can be performed simultaneously in a centralized DBMS, all the attributes involved in the projection operations can be combined. This composite attribute is represented by R_2 with $R_2 = W_1 \cup W_2 \cup rattr(P_6) \cup attr(Q_1) \cup attr(Q_2)$.

Finally, the local selection operations σ_{P_1} and σ_{P_2}, and σ_{Q_1} and σ_{Q_2}, are postponed to the next access step where these are carried out together with the selection operations σ_{P_4} and σ_{P_5}, respectively, to get V_4 and V_5 from V_6.

When the above transformation occurs, it is said that a state transition occurs from the current state represented by Fig. 4a to the state corresponding to Fig. 4b. This state transition is defined as a one-step transition since only one intersite join operation is accounted for the state transition. All the local selection and projection operations involved in an intersite join operation are considered to be executed at the access step associated with that join operation.[1] The CAS is also changed to $J_{P_6}(\pi_{R_1}(V_3), \pi_{R_2}(\sigma_{Q_3}(V_k)))$ for both queries. In the above one-step transition, one join, one selection, and two projection operations are saved. If $Q_3 = Q_2$, $\sigma_{P_2 \wedge P_5 \wedge Q_2}$ is equivalent to $\sigma_{P_2 \wedge P_5}$. Furthermore, if $P_5 \equiv P_6$, then $\sigma_{P_2 \wedge P_5} \equiv \sigma_{P_2}$. All the views which are involved in the generation of a superquery up to the current access step in Fig. 4 are marked by ///.

The above construction rules for the general case are summarized below.

Step 1. Check the feasibility of extending a superquery. There must be promising join and selection conditions for the predicate formulas in the access plans of the subqueries.
Step 2. If it is feasible, then compose R_1 and R_2.
Step 3. Apply the construction rules for the local selection and projection operations related to V_k.
Step 4. Extend the access plan of the superquery by replacing two join operations of a subquery by one join operation, and postpone all the join, selection and projection operations as shown in Fig. 4a and b.
Step 5. Update CAS and mark the views which are involved in the reformulated access plans.

During the plan step, we can also identify the equivalence relationship $P_4 \Leftrightarrow P_5$. In this case, we can further reduce the search space as follows. Let us assume that the attributes appearing in π_{R1} and π_{R_2} are R_4 when P_6 is replaced with P_4, and R_5 when P_6 is replaced with P_5. We further assume that $SIZE(R_4) \geq SIZE(R_5)$. The heuristic is as follows.

If $P_4 \Leftrightarrow P_5$, then select P_5 as the predicate for the superquery, i.e. let P_5 play the role of P_6 in Fig. 4. Since $SIZE(R_4) \geq SIZE(R_5)$, $VOL(V_6)$ derived using J_{P_4} is always greater than or equal to that derived using J_{P_5}. This guarantees that we always select the smaller volume of intermediate results in the process of superquery generation associated with P_4 and P_5. Since we do not generate the superquery associated with IC $P_5 \Rightarrow P_4$, the subtree using $P_5 \Rightarrow P_4$ can be pruned off in the search space.

4. Multiple transaction processor: Search step

4.1 Formal representation of problem space

In this section, we present formal definitions for state, initial and goal states, heuristic cost evaluation functions, and we describe the rules in the search step. These consist of generation and test rules which together embody the A^* search strategy.

[1] This aggregation of local unary operations and an intersite join operation is considered for the efficient representation of a state transition.

Let QT_COST_k be the processing cost for the execution of the query QT_k along its access plan in a distributed environment. Let **G** be a query graph which is constructed by integrating all the individual access plans of queries. The objective is to reformulate the access plans in **G** using the available knowledge such that the total processing cost over n queries, $\Sigma_{k=1}^{n} QT\_COST_k$, is minimized.

A *state* is informally defined as a set of access plans which are represented by a query graph with attached proper information required to estimate the total processing cost in a distributed environment. A state transition occurs whenever a new query graph is constructed by adding or modifying relational operations, thereby reformulating the access plans of the queries, using both logical (syntactic) and semantic knowledge under the constraint that all the views in the current state can be obtained from the new state. This state transition occurs by the activation of the generation rules described below. The generation rules allow for two types of state transitions to occur; one is advancing an access step of a query, and the other merges two current access steps of two different queries and generates a superquery according to the construction rule of the previous section. Both types of state transitions are subject to the following constraints: (i) any view at the current state should be derivable from the newly generated one; (ii) the state transition is only possible using the knowledge associated with the current access step (one-step transition).

A state ω is formally defined as a 5-tuple (**G**, **AP**, **CAS**, $g(\omega)$, $\hat{h}(\omega)$). **G** is a query graph which is constructed by integrating all the individual access plans (see e.g. Figs 10–13). **AP** is the set of (reformulated) access plans of queries inferred from **G**:

$$\boldsymbol{AP} = \{\textit{access-plan}(QT_k) \mid k = 1, 2, \ldots, n\} . \tag{1}$$

When a query is using the (intermediate) results of another query or superquery, QG_k, its access plan should be reformulated as follows: $\langle \textit{access-plan}\ (QG_k), V_i, V_{i+1}, \ldots, V_m \rangle$. **CAS** is the set of all the current access steps of queries:

$$\boldsymbol{CAS} = \{CAS \text{ of } QT_k \mid k = 1, 2, \ldots, n\} . \tag{2}$$

The value $g(\omega)$ is the sum of all the processing costs of queries from their initial access steps to their respective current access steps along their (reformulated) access sequence. The value $\hat{h}(\omega)$ is the estimate of the total remaining cost for all the queries from the current state to reach the goal state, using the strategy of multiple transaction processing.

4.2 Generation rules

We describe the cost functions g and $\hat{h}$, and the generation and test rules in more detail. There are three generation rules: *I*-rule, *M*-rule, and *F*-rule. *I*-rule moves forward the access step of the query along the access sequence to reflect the *individual* distributed query processing strategy. *M*-rule tries to generate a profitable superquery by *merging* access plans of queries, and *F*-rule is useful for the cost evaluation at the end of the superquery generation steps. The firing of the generation rules is controlled by the specificity ordering of the conflict resolution strategy. *M*-rule has more priority than *I*-rule and *F*-rule, and *I*-rule has more priority than *F*-rule. These priorities reflect one of the inference-guiding heuristics in the search step.

Let $\langle V_1, V_2, \ldots, V_n \rangle$ be the access sequence of query QT_k. *I*-rule is described as follows:

I-rule: Move forward one-step further the current access step of the query along the access sequence $\langle V_1, V_2, \ldots, V_n \rangle$. Initially, the *CAS* of QT_k is set to V_1. The activation of *I*-rule enables the *CAS* of QT_k to become V_2; the next firing to become V_3 and so on.

When there are two current access steps for which promising integrity constraints exist, M-rule reformulates the access plans of the queries to generate a superquery relationship by invoking the construction rule described in Section 3.

M-rule: If there exist promising ICs which can be applied to views in the current access steps of two different queries, then reformulate the access plans using the construction rule, and make the newly generated access step to be the current access steps of the corresponding queries.

The answer to a query can be represented by $\boldsymbol{uop}_n(V_n)$ for some "final" view V_n of the query. Since it cannot move forward any further, we introduce the final view V_f such that V_n can be advanced to it without incurring any processing cost. This is actually a stopping rule.

F-rule: If the result of the current access step of the query QT_k is view V_n where the answer to the query is equal to $\boldsymbol{uop}_n(V_n)$, then move forward the current access step to V_f.

I-rule, M-rule and F-rule are complete in the sense that all the possible ways of reformulating the access plans of queries can be enumerated along the access sequences within a given knowledge. They are also nonredundant since each rule is fired only once using given ICs. Finally, the search step is informed by using the priority existing between I-rule, M-rule and F-rule as well as being guided by the heuristic cost evaluation function $\hat{h}$ which will be described subsequently. Note that the identification of the same views can be facilitated by using ICs of the form $P_i \Rightarrow P_i$.

For an access step $J_P(\boldsymbol{uop}_i(V_i), \boldsymbol{uop}_j(V_j))$, we want to allow for various access strategies which take into account the different intersite communication costs and different local processing costs. In order to do that, we describe the cost function associated with a query graph where each node of the query graph contains site information. Let $V_k = J_{P_k}(\boldsymbol{uop}_i(V_i), \boldsymbol{uop}_j(V_j))$. The access step $J_{P_k}(\boldsymbol{uop}_i(V_i), \boldsymbol{uop}_j(V_j))$ is represented by the query graph in Fig. 5.

Here, since the oval contains site information, we represent the node of the query graph by the notation VS_k where VS_k is defined to be an ordered pair (V_k, S_k), denoting view V_k located at site S_k.

We describe the processing cost evaluation functions. $\psi^l_{s_i}(uop, V)$ is defined as the local processing cost to carry out the relational operator uop to view V at site S_i; $\psi^c_{s_i s_j}(V)$ as the communication cost to transmit the volume of view V, $VOL(V)$, from site S_i to site S_j, and $\psi^l_{s_i}(J_{P_k}, V, V')$ as the local processing cost to carry out the join operation J_{P_k} for the views V and V' at site S_i.[2] For a unary operation $\boldsymbol{uop}(V_i)$, the corresponding query graph is shown in Fig. 6, where $V_j = \boldsymbol{uop}(V_i)$.

If $S_i = S_j$ in Fig. 6, $\boldsymbol{uop}$ is performed at site S_i. Otherwise, $\boldsymbol{uop}$ is performed at site S_i, but the result V_j of its execution is transmitted to site S_j, and stored there for further processing.

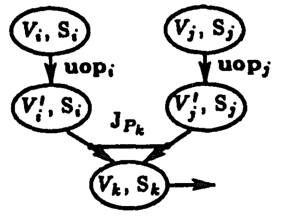

Fig. 5. The access step of $J_{P_k}(\boldsymbol{uop}_i(V_i), \boldsymbol{uop}_j(V_j))$.

Fig. 6. The access step of $\boldsymbol{uop}(V_i)$.

[2] If necessary, other cost functions can be defined; for example, semi-join operations can be considered.

The corresponding cost function, $cst(VS, VS_j)$ with $VS = \{VS_i\}$ is defined as follows.

$$cst(VS, VS_j) = \begin{cases} \psi^l_{s_i}(uop, V_i) & \text{for } S_i = S_j \\ \psi^l_{s_i}(uop, V_i) + \psi^c_{s_i s_j}(V_j) & \text{for } S_i \neq S_j . \end{cases} \tag{3}$$

Going back to the query graph in Fig. 5, we assume that all the unary operations are carried out before any intersite data transmission occurs. However, the join operation J_P can be performed at different sites according to the following protocol. If $S_k = S_i$, it is assumed that view V'_j at site S_j is transmitted to site S_i, and J_P is carried out on V'_i and V'_j at site S_i where the result V_k is stored. If $S_k \neq S_i$ and $S_k \neq S_j$, both V'_i and V'_j are transmitted to site S_k, and the join is performed at site S_k. The result V_k is also stored at site S_k. According to the above protocol, every distributed query access plan using the joins as processing tactic can be represented precisely by the above modified query graph. We describe the cost formula for each case below. Let $VS = \{VS_i, VS_j\}$ where $VS_i = (V_i, S_i)$ and $VS_j = (V_j, S_j)$. Let $VS_k = (V_k, S_k)$. The corresponding cost formula, $cst(VS, VS_k)$, is shown below.

$$cst(VS, VS_k) = \begin{cases} local\_cst + \psi^c_{S_j S_i}(V'_j) + \psi^l_{S_i}(J_{P_k}, V'_i, V'_j) & \text{for } S_k = S_i \\ local\_cst + \psi^c_{S_i S_j}(V'_i) + \psi^l_{S_j}(J_{P_k}, V'_i, V'_j) & \text{for } S_k = S_j \\ local\_cst + \psi^c_{S_i S_k}(V'_i) + \psi^c_{S_j S_k}(V'_j) & \\ \quad + \psi^l_{S_k}(J_{P_k}, V'_i, V'_j) & \text{otherwise ,} \end{cases} \tag{4}$$

where $local\_cst = \psi^l_{S_i}(uop_i, V_i) + \psi^l_{S_j}(uop_j, V_j)$ and where $V'_i = uop_i(V_i)$, $V'_j = uop_j(V_j)$. Note that we do not assume any specific cost model for the evaluation of the access step.

Let VS_k be a view which is an intermediate result from the execution of a query along its access plan. Let $\mathbf{VS}$ be the set of views such that $VS \in \mathbf{VS}$ implies that VS is involved in the part of the access plan yielding VS_k. $VS\_COST(VS_k)$ determines the processing cost to get VS_k along the access plan of the query, and it is defined as follows:

$$VS\_COST(VS_k) = cst(\mathbf{VS}, VS_k) + \sum_{VS \in \mathbf{VS}} VS\_COST(VS) . \tag{5}$$

Assume that state ω consists of m superqueries, QG_l for $l = 1, 2, \ldots, m$, and n original queries[3] $QT^{r_k}_k$ for $k = 1, 2, \ldots, n$ where the current view of the query $QT^{r_k}_k$ is denoted as $VS_k = (V_k, S_k)$. Let VS'_l be the result of executing QG_l, and NQG_l be the number of queries $QT^{r_k}_k$ which use the intermediate result VS'_l to get their current view VS_k. Then,

$$g(\omega) = \sum_{k=1}^{n} VS\_COST(VS_k) - \sum_{l=1}^{m} (NQG_l - 1) VS\_COST(VS'_l) . \tag{6}$$

The second term in Equation 6 reflects the fact that the processing cost related to using the intermediate result should be accounted for only once due to the strategy of multiple query processing. VS_COST is evaluated by a backward recursion.

We now wish to develop the heuristic cost estimate $\hat{h}(\omega)$ and prove its admissibility. We need to define new functions. For a given access plan $\langle J_{P_2}(V_1, V_2), J_{P_4}(V_3, V_4), \ldots, J_{P_{2n}}(V_{2n-1}, V_{2n}) \rangle$ of a query $QT^{r_k}_k$, let us define a function $\delta^l_k(VS_i)$ such that

[3] We mean by the original queries those supplied as input to *MTP*.

$\delta_k^I(VS_i)$ returns the remaining cost starting from the view VS_i along the individual access plan of $QT_k^{r_k}$ to get the result of $QT_k^{r_k}$. Here,

$$\delta_k^I(VS_i) = VS\_COST(VIEW(QT_k^{r_k})) - VS\_COST(VS_i)\,, \tag{7}$$

where the superscript I of δ_k^I indicates that the processing cost is evaluated along the access plan which is determined by an individual distributed query processing strategy. We define the function Γ as follows:

$$\Gamma(\boldsymbol{VS}_1, \boldsymbol{VS}_2) = \{(V_k, S_k) \mid \exists J_{P_k}((V_k = J_{P_k}(V_1, V_2)) \wedge (V_1, S_1) \in \boldsymbol{VS}_1 \wedge (V_2, S_2) \in \boldsymbol{VS}_2)\}\,. \tag{8}$$

Let $x = (V, S)$ indicate a view V stored at site S. Let $\delta^I(x)$ denote the remaining processing cost of an individual query access plan starting from the view V. Let $\hat{\delta}(VS_k)$ denote the estimate of the remaining processing cost associated with a current view VS_k. Then,

$$\hat{\delta}(VS_k) = \sum_{y \in \boldsymbol{VS}'} \left\{ \min_{x \in \Gamma(VS_k,\, y) \wedge S'' \in \boldsymbol{S}} \{cst(\{VS_k, y\}, (V, S''))\} + \max_{x \in \Gamma(VS_k,\, y) \wedge S'' \in \boldsymbol{S}} \{\delta^I(x) - \psi^c_{SS''}(V)\} \right\}, \tag{9}$$

where $\boldsymbol{S}$ is the set of all sites in the given computer network; $\boldsymbol{VS}' = \{(V_l, S_l) \mid \exists J_{P_l}(J_{p_l}(V_k, V_l) = V_i) \wedge (V_i, S_i) \in \Gamma(V_k, V_l) \wedge S_i, S_l \in \boldsymbol{S})\}$.

Let $\boldsymbol{CVS}$ be the set of all the current views in state ω. Then,

$$\hat{h}(\omega) = \sum_{VS \in \boldsymbol{CVS}} \hat{\delta}(VS)\,. \tag{10}$$

We prove the admissibility of $\hat{h}$ below.

Theorem 1 *Suppose that the individual access plan of each query is optimal. Then $h(\omega) \geq \hat{h}(\omega)$ where $h(\omega)$ is the minimal cost to reach the goal state ω_g from the current state ω.*

Proof. Without loss of generality, let

$$\hat{h}(\omega) = \sum_{y \in \boldsymbol{VS}'} \left\{ \min_{x \in \Gamma(VS_k,\, y) \wedge S'' \in \boldsymbol{S}} \{cst(\{VS_k, y\}, (V, S''))\} + \max_{x \in \Gamma(VS_k,\, y) \wedge S'' \in \boldsymbol{S}} \{\delta^I(x) - \psi^c_{SS''}(V)\} \right\}. \tag{11}$$

We make the assumption that there exists only one current VS_k and only one y. The situation is shown in Fig. 7. In Fig. 7, $\Gamma(VS_k, y) = \{(V_i, S_i) \mid i = 1, 2, \ldots, n\}$, $\boldsymbol{S} = \{S_1, S_2, \ldots, S_s\}$ and $\boldsymbol{VS}' = \{y\}$.

Suppose that we generate a superquery QG starting from (V_k, S_k) such that all the results of queries $QT_k^{r_k}$ for $k = 1, 2, \ldots, n$ can be obtained by accessing the result of QG. Furthermore, let us assume that the access plan of QG is optimal among all the feasible access plans of superqueries. As shown in Fig. 7, let us assume that (V', S') is the view in the access plan of QG from which the views (V_i, S_i) for $i = 1, 2, \ldots, n$ can be obtained. Let $\delta'((V', S'))$ be the remaining processing cost of QG starting from (V', S'). Since each individual access plan of queries is assumed to be optimal, we know that $\psi^c_{SS'}(V_i) + \delta'((V_i, S')) > \delta_i^I((V_i, S_i))$ for all $i = 1, 2, \ldots, n$. Otherwise, it leads to a contradiction since

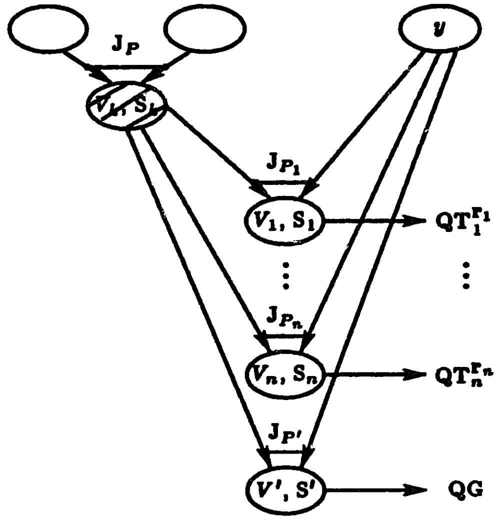

Fig. 7. A state with only one current view.

$\delta_i^I((V_i, S_i))$ does not become minimal in such a case. Since $VOL(V') \geq VOL(V_i)$, $\delta^I((V', S')) \geq \delta^I((V_i, S')) > \delta_i^I((V_i, S_i)) - \psi_{SS'}^c(V_i)$ for any site $S' \in S$ and $i = 1, 2, \ldots, n$. Thus, $\delta^I((V', S')) > \max_{x \in \Gamma(VS_k, y) \wedge S'' \in S} \{\delta^I(x) - \psi_{SS''}^c(V)\}$.

Now, let us consider all the possible ways of constructing V_i for $i = 1, 2, \ldots, n$ from the views VS_k and y. For each V_i, we have s ways of constructing it where s is the number of sites. Let the view (V'', S'') be the view such that $(V'', S'') \in \{(V_i, S_i) | i = 1, 2, \ldots, n\}$ and $cst(\{VS_k, y\}, (V'', S''))$ is minimal. Since $VOL(V') \geq VOL(V'')$, we know that $cst(\{VS_k, y\}, (V', S')) \geq cst(\{VS_k, y\}, (V'', S')) \geq cst(\{VS_k, y\}(V'', S''))$. In other words, $cst(\{VS_k, y\}, (V', S')) \geq \min_{x \in \Gamma(VS_k, y) \wedge S'' \in S} \{cst(\{VS_k, y\}, (V, S''))\}$, where $x = (V, S)$. Thus, the lower bound of $h(VS_k)$ where $h(VS_k)$ is the remaining processing cost associated with the current view VS_k when an optimal superquery is generated, is

$$\hat{h}(VS_k) = \left\{ \min_{x \in \Gamma(VS_k, y) \wedge S'' \in S} \{cst(\{VS_k, y\}, (V, S''))\} \right. \\ \left. + \max_{x \in \Gamma(VS_k, y) \wedge S'' \in S} \{\delta^I(x) - \psi_{SS''}^c(V)\} \right\} \quad \square$$

It is easily shown that $\hat{\delta}(VS_k)$ is simplified to $\delta_k^I(VS_k)$ when VS_k is the current view of $QT_k^{r_k}$ which is marked by the activation of generation rule I-rule. This is summarized below:

$$\hat{\delta}(VS_k) = \begin{cases} \delta_k^I(VS_k) & \text{if } \mathrm{CARD}(\Gamma(VS_k, y)) = 1 \\ \text{Eq. (9)} & \text{otherwise}, \end{cases}$$

where CARD denotes the cardinality of a set.

Finally, we wish to describe the test rule. For each search step, we evaluate $\hat{f}(\omega) = g(\omega) + \hat{h}(\omega)$. Among all the states which are generated at previous search steps, but are not expanded, or which are generated at the current search step, we select the state which has the minimal cost estimate $\hat{f}(\omega)$, and we test that state whether it is a goal state or not. If that state is found to be a goal state, then the search process is stopped. Otherwise, we expand that state, and the search process resumes.

4.3 A special case

In this section we present heuristic cost evaluation functions for a simple case in which the access plans of the given queries comply with the following assumption, called *the assumption of fixed access order*: (i) for an access step $J_P(V_1, V_2)$, the join is performed at the site where V_2 is located by moving V_1 there; (ii) the order of the access step in the access sequence of a newly generated superquery precisely follows those of the original queries which use the (intermediate) result of that new superquery.[4] Due to this assumption, the site information is implicitly ignored throughout this section.

Here, we do not assume that each individual access plan is optimal. Under the assumption of fixed access order, it is generally not feasible to infer subquery relationships in the middle of the access sequences of two different queries.[5] Hence, we can assume that all the superquery relationships which would be profitable can be inferred from the query graph shown in Fig. 8. The function g is defined as in Equation 6.

For $\hat{h}$, we need to define the two functions $\hat{\rho}$ and Γ. Let $\mathbf{V}_1$ and $\mathbf{V}_2$ be sets of views. Γ is defined as follows:

$$\Gamma(\mathbf{V}_1, \mathbf{V}_2) = \{V \mid \exists J_{P_k}((V = J_{P_k}(V_1, V_2)) \wedge (V_1 \in \mathbf{V}_1) \wedge (V_2 \in \mathbf{V}_2))\} \tag{12}$$

For a set $\mathbf{V}$ of views, the function $\hat{\rho}(\mathbf{V})$ optimistically estimates the sum of the remaining costs of queries which use the intermediate results in $\mathbf{V}$.

Let x, y and z denote views (see Fig. 8). Then,

$$\hat{\rho}(\mathbf{V}) = \sum_{y \in \mathbf{V}'} \left\{ \max_{(x \in \mathbf{V}) \wedge (z \in \Gamma(x, y))} \{cst((x, y), z)\} + \hat{\rho}(\Gamma(\mathbf{V}, y)) \right\}, \tag{13}$$

where $\mathbf{V}' = \{V' \mid \exists J_{P_k}((J_{P_k}(V, V') = V'') \wedge (V \in \mathbf{V}) \wedge (V'' \in \Gamma(V, V')))\}$. $\hat{\rho}(\mathbf{V})$ is evaluated by

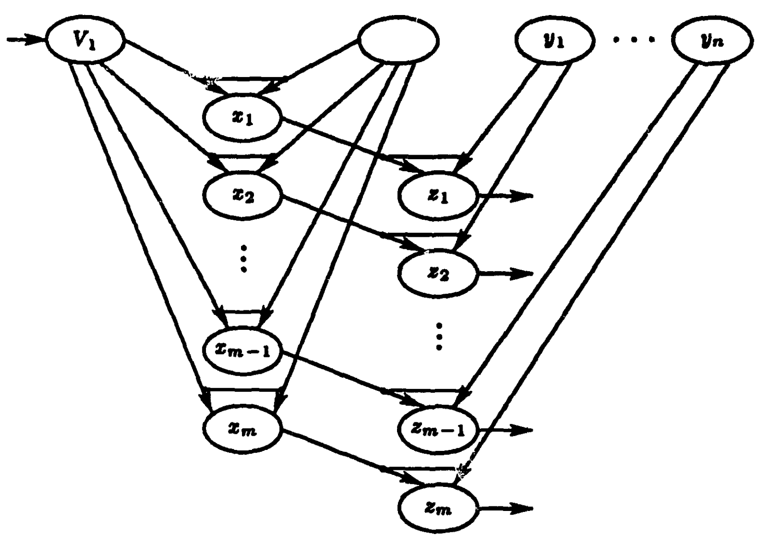

Fig. 8. A f[illegible] le query graph.

[4] This assumption is more likely to be valid in a centralized environment, but it may also be applicable to local area networks.

[5] Refer to [25].

a forward recursion. The boundary conditions are as follows:

Case 1. *When* $V_3 = J_P(V_1, V_2)$, $\hat{\rho}(V_1) = cst((V_1, V_2), V_3)$.
Case 2. *When* $V_2 = \mathbf{uop}(V_1)$, $\hat{\rho}(V_1) = cst(V_1, V_2)$.
Case 3. *When* $\hat{\rho}(V) = cst(V, V_f)$, $\hat{\rho}(V) = 0$.

Now, let $\boldsymbol{CV}$ be the set of all the current views in state ω. The heuristic cost evaluation function $\hat{h}$ is defined as follows:

$$\hat{h}(\omega) = \sum_{V \in CV} \hat{\rho}(V) . \tag{14}$$

The initial state ω_0 is $(\boldsymbol{G}, \boldsymbol{AP}, \boldsymbol{CAS}, 0, \hat{h}(\omega_0))$ where $\boldsymbol{AP}$ is the set of access plans which are not reformulated, and $\boldsymbol{CAS}$ is the empty set. The goal state ω_g is $(\boldsymbol{G}, \boldsymbol{AP}, \boldsymbol{CAS}, g(\omega_g), 0)$ where $\boldsymbol{G}$ and $\boldsymbol{AP}$ are the reformulated query graph and the set of reformulated access plans, respectively; $\boldsymbol{CAS}$ is $\{V_f\}$.

We prove the admissibility of $\hat{h}$ below.

Theorem 2. $h(\omega) \geq \hat{h}(\omega)$ *where* $h(\omega)$ *is the minimal cost from the current state* ω *to the goal state* ω_g.

Proof. Let V_i for $i = 1, 2, 3, 4$ denote views. Let $J_{P_1}(V_1, V_2)$ be an access step chosen for the evaluation of some $\hat{\rho}(V)$ defined in Equation 13 where V is a current view of a query at state ω. Let $\rho(V)$ be an optimal remaining processing cost, at state ω, for the current view V using the multiple query processing strategy. (Equation (13) is clearly true if the individual access plan are used.) For the access step $J_{P_1}(V_1, V_2)$ of the query, let the corresponding access step of the superquery be $J_{P_3}(V_3, V_4)$ which is involved in the construction of an optimal access plan. Let $V_1' = J_{P_1}(V_1, V_2)$ and $V_3' = J_{P_3}(V_3, V_4)$. Since $J_{P_3}(V_3, V_4)$ is an access step of the superquery, the processing cost of performing this access step is greater than or equal to any corresponding access step of the query which uses the result of execution of $J_{P_3}(V_3, V_4)$. Hence, $cst((V_3, V_4), V_3') \geq cst((V_1, V_2), V_1')$. Since $cst((V_3, V_4), V_3') \geq cst((V_1, V_2), V_1')$ for any access step $J_{P_1}(V_1, V_2)$ which comes behind the current view V, $\rho(V) \geq \hat{\rho}(V)$. Therefore, $h(\omega) \geq \Sigma_{V \in CV}\, \hat{\rho}(V)$. □

5. An example

This example illustrates the operation of Multiple Transaction Processor. For simplicity, attention is restricted to the special case described in Section 4.3. The example shows how multiple queries occurring concurrently in a distributed database system can be processed optimally by utilizing jointly both database semantics and logical information of predicate conditions of queries.

The distributed database $\boldsymbol{r}$ contains three relations r_1, r_2 and r_3 at site 1, site 2, and site 3, respectively, as shown in Fig. 9. Let $V_a = J_{P_4 \wedge P_5}(\pi_{Y_1}(r_1), r_2)$ and $V_b = J_{P_3}(\pi_{Z_1}(r_1), r_2)$. Two queries QT_2 and QT_3 occur at site 1 with the following access plans:

access_plan(QT_2): $\langle J_{P_4 \wedge P_5}(\pi_{Y_1}(r_1), r_2), J_{P_7 \wedge P_8}(\pi_{Y_2}(V_a), r_3)\rangle$, and
access_plan(QT_3): $\langle J_{P_3}(\pi_{Z_1}(r_1), r_2), J_{P_q}(\pi_{Z_2}(V_b), r_3)\rangle$,

where Y_1, $Z_1 \subseteq R_1$, $Y_2 \subseteq R_1 \cup R_2$, and $Z_2 \subseteq R_1 \cup R_2$. The access plan of QT_2 implies that

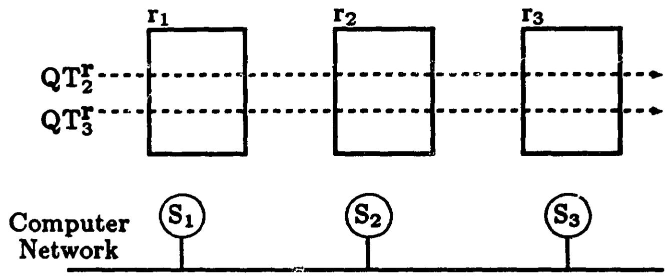

Fig. 9. Example of a distributed database with two queries.

$\pi_{Y_1}(r_1)$ is performed at site S_1; $\pi_{Y_2}(J_{P_4 \wedge P_5}(V', r_2))$ with $V' = \pi_{Y_1}(r_1)$ being transferred from S_1 is performed at site S_2; $J_{P_7 \wedge P_8}(V'', r_3)$ is performed at S_3 with $V'' = \pi_{Y_2}(J_{P_4 \wedge P_5}(\pi_{Y_1}(r_1), r_2)$. The access plan of QT_3 can be interpreted similarly.

During the planning step, a set of integrity constraints is inferred from logical and semantic information using inference-guiding heuristics.[6] The plan step infers only *relevant* ICs for the efficient identification of subquery relationships and systematic generation of superqueries in the subsequent search step. The following is the list of all the relevant ICs associated with relations r_1 and r_2, and r_2 and r_3.

ICs related to r_1 and r_2		
$P_1 \wedge P_2$	$\Rightarrow$	$P_1 \wedge P_2$
$P_4 \wedge P_5$	$\Rightarrow$	$P_1 \wedge P_2$
$P_1 \wedge P_2$	$\Rightarrow$	P_1
$P_4 \wedge P_5$	$\Rightarrow$	P_1
P_3	$\Rightarrow$	P_1
$P_1 \wedge P_2$	$\Rightarrow$	P_2
$P_4 \wedge P_5$	$\Rightarrow$	P_2
P_3	$\Rightarrow$	P_2
$P_4 \wedge P_5$	$\Rightarrow$	$P_4 \wedge P_5$
P_3	$\Rightarrow$	P_3
P_1	$\Rightarrow$	P_1
P_2	$\Rightarrow$	P_2

ICs related to r_2 and r_3		
P_7	$\Rightarrow$	P_7
$P_7 \wedge P_8$	$\Rightarrow$	P_7
P_8	$\Rightarrow$	P_8
$P_7 \wedge P_8$	$\Rightarrow$	P_8
$P_7 \wedge P_8$	$\Rightarrow$	P_6
$P_7 \wedge P_8$	$\Rightarrow$	$P_7 \wedge P_8$
P_8	$\Rightarrow$	P_6
P_9	$\Rightarrow$	P_9
P_7	$\Rightarrow$	P_{10}
$P_7 \wedge P_8$	$\Rightarrow$	P_{10}
P_9	$\Rightarrow$	P_{10}
P_{10}	$\Rightarrow$	P_{10}

The initial state for this example is shown in Fig. 10 where the access plans of QT_2 and QT_3 are integrated.

The processing costs for QT_2 and QT_3 are described below when individual distributed query processing strategies are used.

$$\begin{aligned} QT\_COST_2 &= \psi^l(\pi_{Y_1}, r_1) + \psi^c(\pi_{Y_1}(r_1)) + \psi^l(J_{P_4 \wedge P_5}, \pi_{Y_1}(r_1), r_2) \\ &\quad + \psi^l(\pi_{Y_2}, V_a) + \psi^c(\pi_{Y_2}(V_a)) + \psi^l(J_{P_7 \wedge P_8}, \pi_{Y_2}(V_a), r_3) \\ QT\_COST_3 &= \psi^l(\pi_{Z_1}, r_1) + \psi^c(\pi_{Z_1}(r_1)) + \psi^l(J_{P_3}, \pi_{Z_1}(r_1), r_2) \\ &\quad + \psi^l(\pi_{Z_2}, V_b) + \psi^c(\pi_{Z_2}(V_b)) + \psi^l(J_{P_9}, \pi_{Z_2}(V_b), r_3)\,. \end{aligned} \tag{15}$$

[6] The planning technique is described in detail in [25].

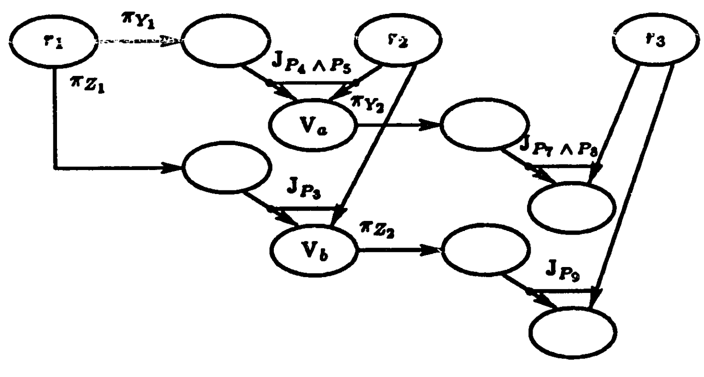

Fig. 10. A graphical representation of the initial state.

All site information is ignored here since we assume that all the sites have the same processing capability and that the communication cost does not depend on sites. In Equation 15, $cst_2((\pi_{Y_1}(r_1), r_2), V_a) = \psi^l(\pi_{Y_1}, r_1) + \psi^c(\pi_{Y_1}(r_1)) + \psi^l(J_{P_4 \wedge P_5}, \pi_{Y_1}(r_1), r_2)$ which is associated with the intersite join operation $J_{P_4 \wedge P_5}$, and $cst_2((\pi_{Y_2}(V_a), r_3), J_{P_7 \wedge P_8}(\pi_{Y_2}(V_a), r_3)) = \psi^l(\pi_{Y_2}, J_{P_4 \wedge P_5}(\pi_{Y_1}(r_1), r_2)) + \psi^c(\pi_{Y_2}(J_{P_4 \wedge P_5}(\pi_{Y_1}(r_1), r_2))) + \psi^l(J_{P_7 \wedge P_8}, \pi_{Y_2}(J_{P_4 \wedge P_5}(\pi_{Y_1}(r_1), r_2)), r_3)$, which is related to the intersite join operation $J_{P_7 \wedge P_8}$.

We first discuss the search process. For example, by employing the two promising ICs $P_3 \Rightarrow P_1$ and $P_4 \wedge P_5 \Rightarrow P_1$, state 1, which is shown in Fig. 11, is generated from the initial state. In Fig. 11, the current access step of the query QG_1 is $J_{P_1}(\pi_{Y_1 \cup Z_1}(r_1), r_2)$, and the current view of the query is V_1. We make the assumption that the attributes involved in the join operation J_{P_1} are contained in $Y_1 \cup Z_1$.

The construction rule for the join operation permits two intersite join operations $J_{P_4 \wedge P_5}$ and J_{P_3} to be replaced by one intersite join J_{P_1}. Here, the superquery QG_1 of both QT_2 and QT_3 is generated with respect to $\langle r_1, r_2 \rangle$. The view V_1 which is the result of the superquery execution allows for the production of intermediate results V_a and V_b of QT_2 and QT_3 since $V_a = \pi_{Y_1 \cup R_2}(\sigma_{P_4 \wedge P_5}(V_1))$ and $V_b = \pi_{Z_1 \cup R_2}(\sigma_{P_3}(V_1))$.

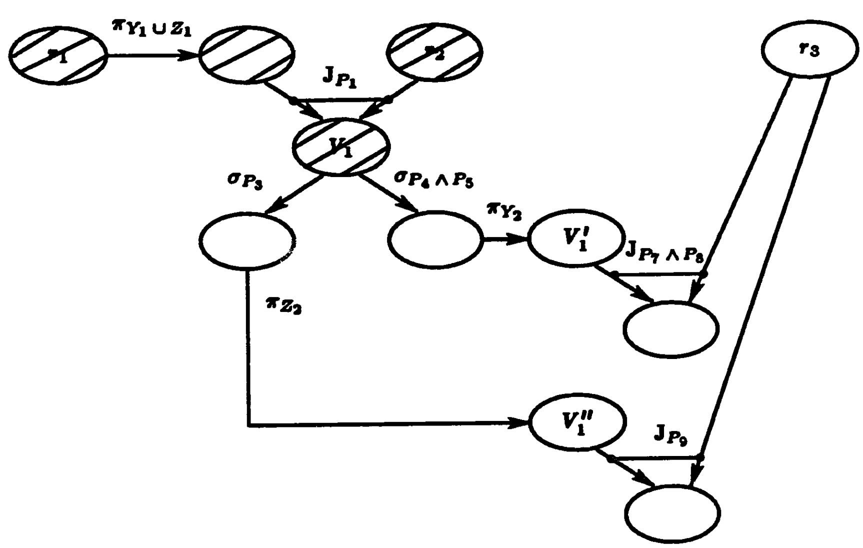

Fig. 11. A graphical representation of state 1.

State 1 is generated by the activation of M-rule, and state 4 which is shown in Fig. 12 is generated by the activation of I-rule. All the views which are utilized in the search step are marked by / / / in both Figs 11 and 12. Note that state 1 is generated following the access sequence of execution plans of both QT_2 and QT_3.

$g(\omega)$ is the processing cost incurred at the current state ω. Let us define $\hat{h}_I(\omega)$ as the heuristic cost evaluation function which estimates the sum of the incremental remaining processing cost for each query, from the current access step to the final access step, when the individual distributed query processing strategies are used. $h(\omega)$ is the minimal incremental cost from the current state ω which can occur by reformulating access plans in the context of multiple query processing. As before, let $\hat{h}(\omega)$ be an optimistic estimate of $h(\omega)$.

Let $V_1' = \pi_{Y_2}(\sigma_{P_4 \wedge P_5}(V_1))$, and $V_1'' = \pi_{Z_2}(\sigma_{P_3}(V_1))$ as shown in Fig. 11. Also, let ω be state 1.

$$\begin{aligned} g(\omega) &= QG\_COST_1(r_1, r_2) \\ &= cst((\pi_{Y_1 \cup Z_1}(r_1), r_2), V_1) \\ &= \psi^l(\pi_{Y_1 \cup Z_1}, r_1) + \psi^c(\pi_{Y_1 \cup Z_1}(r_1)) + \psi^l(J_{P_1}, \pi_{Y_1 \cup Z_1}(r_1), r_2) \end{aligned}$$

$$\begin{aligned} \hat{h}_I(\omega) &= cst_2((V_1', r_3), J_{P_7 \wedge P_8}(V_1', r_3)) \\ &\quad + cst_3((V_1'', r_3), J_{P_9}(V_1'', r_3)) \,. \end{aligned}$$

$cst((\pi_{Y_1 \cup Z_1}(r_1), r_2), V_1)$ denotes the processing cost of the current access step of the superquery QG_1. The access step $J_{P_1}(\pi_{Y_1 \cup Z_1}(r_1), r_2)$ is the current access step for all the queries QT_2, QT_3 and QG_1 at the state shown in Fig. 11. cst_2 denotes the incremental cost incurred using the intermediate result V_1 to get the answer of the query QT_2. Similarly, cst_3 denotes that for the query QT_3, using the same intermediate result V_1. Here,

$$\begin{aligned} cst_2((V_1', r_3), J_{P_7 \wedge P_8}(V_1', r_3)) &= \psi^l(\sigma_{P_4 \wedge P_5}, V_1) + \psi^l(\pi_{Y_2}, \sigma_{P_4 \wedge P_5}(V_1)) \\ &\quad + \psi^c(V_1') + \psi^l(J_{P_7 \wedge P_8}, V_1', r_3) \\ cst_3((V_1'', r_3), J_{P_9}((V_1''), r_3)) &= \psi^l(\sigma_{P_3}, V_1) + \psi^l(\pi_{Z_2}, \sigma_{P_3}(V_1)) \\ &\quad + \psi^c(V_1'') + \psi^l(J_{P_9}, V_1'', r_3) \,. \end{aligned}$$

Now, we attempt to extend the superquery relationship since it may reduce the total processing cost further. Since we do not know the existence of ICs at state 1, we do an optimistic conjecture that either $P_9 \Rightarrow P_7 \wedge P_8$ or $P_7 \wedge P_8 \Rightarrow P_9$ or both might be true. This

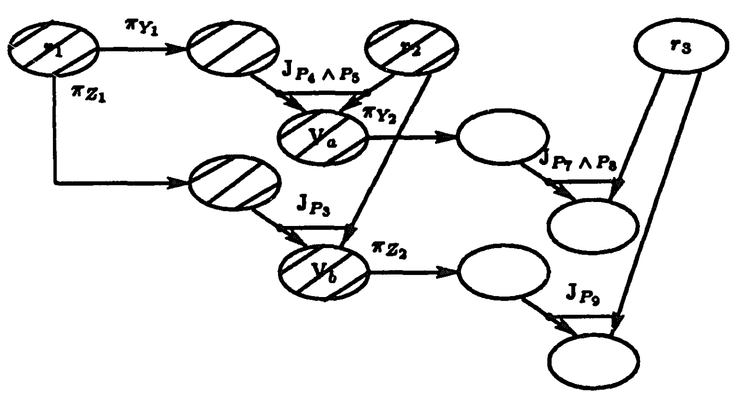

Fig. 12. A graphical representation of state 4.

optimistic conjecture allows us to estimate the remaining cost estimate $\hat{h}(\omega)$ at state 1. Using the construction rule for the join operation, the feasible state generated from state 1 which is based on the optimistic conjecture is depicted in Fig. 13.

In the process of extending the superquery with respect to $\langle r_1, r_2\rangle$ in Fig. 11 to that with respect to $\langle r_1, r_2, r_3\rangle$ in Fig. 13, the local selection operations σ_{P_3} and $\sigma_{P_4\wedge P_5}$ are postponed to the next stage in the access sequence, and a new projection operation π_Z is added. $Z = Z_2 \cup Y_2 \cup X$ where X denotes the attributes involved in the selection operations σ_{P_3} and $\sigma_{P_4\wedge P_5}$. Here, we can define two cost estimates for the remaining cost; one is $\hat{h}_1(\omega)$ based on the optimistic conjecture $P_7 \wedge P_8 \Rightarrow J_{P_9}$, and the other $\hat{h}_2(\omega)$ based on $J_{P_9} \Rightarrow P_7 \wedge P_8$. These estimates are evaluated as follows:

$$\hat{h}_1(\omega) = cst((\pi_Z(V_1)r_3), V_2) + \delta_l(\omega)$$
$$\hat{h}_2(\omega) = cst((\pi_Z(V_1)r_3), V_2') + \delta_l'(\omega)\,,$$

where $V_2' = J_{P_7\wedge P_8}(\pi_Z(V_1), r_3)$; δ_l and δ_l' account for the local processing cost at the final access step to get the result of each query from those of superqueries. Here,

$$cst((\pi_Z(V_1)r_3), V_2) = \psi^l(\pi_Z, V_1) + \psi^c(\pi_Z(V_1)) + \psi^l(J_{P_9}, \pi_Z(V_1), r_3)$$
$$cst((\pi_Z(V_1)r_3), V_2') = \psi^l(\pi_Z, V_1) + \psi^c(\pi_Z(V_1)) + \psi^l(J_{P_7\wedge P_8}, \pi_Z(V_1), r_3)$$
$$\delta_l(\omega) = \psi^l(\sigma_{P_3}, V_2) + \psi^l(\pi_{Z_2\cup R_3}, \sigma_{P_3}(V_2)) +$$
$$\psi^l(\sigma_{P_4\wedge P_5\wedge P_7\wedge P_8}, V_2) + \psi^l(\pi_{Y_2\cup R_3}, \sigma_{P_4\wedge P_5\wedge P_7\wedge P_8}(V_2))$$
$$\delta_l'(\omega) = \psi^l(\phi_{P_3\wedge P_9}, V_2') + \psi^l(\pi_{Z_2\cup R_3}, \sigma_{P_3\wedge P_9}(V_2')) +$$
$$\psi^l(\sigma_{P_4\wedge P_5}, V_2') + \psi^l(\pi_{Y_2\cup R_3}, \sigma_{P_4\wedge P_5}(V_2'))\,.$$

As described before, we take $\hat{h}(\omega) = \min\{\hat{h}_l(\omega), \hat{h}_1(\omega), \hat{h}_2(\omega)\}$.

Now, we illustrate the search process with the actual input data given in Table 1. We assume that the local processing costs due to projection and selection operations are

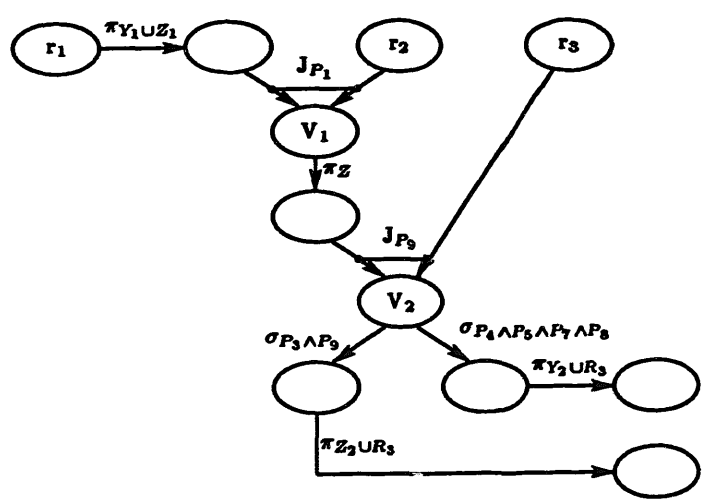

Fig. 13. A query graph based on an optimistic conjecture at state 1.

Table 1.
The input data.

	$\psi^c(\cdot)$		$\psi^l(\cdot, \cdot, \cdot)$
$\pi_{Y_1}(r_1)$	5	$(J_{P_4 \wedge P_5}, \pi_{Y_1}(r_1), r_2)$	5
$\pi_{Z_1}(r_1)$	5	$(J_{P_3}, \pi_{Z_1}(r_1), r_2)$	5
$\pi_{Y_1 \cup Z_1}(r_1)$	5	$(J_{P_1}, \pi_{Y_1 \cup Z_1}(r_1), r_2)$	6
–	–	$(J_{P_2}, \pi_{Y_1 \cup Z_1}(r_1), r_2)$	9
–	–	$(C, \pi_{Y_1 \cup Z_1}(r_1), r_2)$	12
$\pi_{Y_2}(V_a)$	2	$(J_{P_7 \wedge P_8}, \pi_{Y_2}(V_a), r_3)$	3
$\pi_{Z_2}(V_b)$	2	$(J_{P_9}, \pi_{Z_2}(V_b), r_3)$	3
$\pi_Z(V_1)$	3	$(J_{P_{10}}, \pi_Z(V_1), r_3)$	4

negligible in comparison with intersite data transmission and join operations.

The search tree generated by MTP is shown in Fig. 14. The costs are listed in Table 2 for states 0 to 6.

In Fig. 14, the value of $\hat{f}$ for each state is circled on the upper-right corner of the box representing the state, and the uncircled number on the upper-left corner is the state number; in this example, the state numbers also indicate the order in which states are expanded, except for states 4, 5, 6, and 7 which are never expanded. The darkened line shows the search path leading to the optimal solution. State 3 is generated by employing ICs $P_7 \wedge P_8 \Rightarrow P_{10}$ and $P_9 \Rightarrow P_{10}$ to state 1; state 2 by utilizing the individual distributed query processing strategies at state 1. State 5 corresponds to the application of individual distributed processing to the initial state without employing the knowledge available. State 6 is generated by using $P_4 \wedge P_5 \Rightarrow P_2$ and $P_3 \Rightarrow P_2$; state 4 by using $P_4 \wedge P_5 \Rightarrow \Lambda$ and $P_3 \Rightarrow \Lambda$. These states are never expanded since the heuristic evaluation function always underestimates the total remaining cost. In this way, many subtrees can be pruned off in more complex examples. State 5 is not expanded into states using any ICs associated with the access steps $J_{P_7 \wedge P_8}(\pi_{Y2}(V_a), r_3)$ and $J_{P_9}(\pi_{Z_2}(V_b), r_3)$, even though there are several ICs available like $P_7 \wedge P_8 \Rightarrow P_{10}$ and $P_9 \Rightarrow P_{10}$. This is because, once a join operation is carried out, it is generally not possible to infer any superquery relationships on the following access step.

Among the nodes expanded from the initial state, state 1 has the minimal processing cost of 17. It is expanded, generating states 2 and 3. Among states 2, 3, 4, 5, and 6, state 3 has the minimal cost. Since state 3 cannot be expanded further, we conclude that state 3 is the goal state with the total processing cost of 19. States 4, 5 and 6 are never expanded. If state 5 were expanded, we would have state 7 as shown by the dotted line in Fig. 14. This state corresponds to the state when individual distributed query processing strategies are employed. It is easily found that the total processing cost of state 7 is equal to that of state 5; $T\_COST = QT\_COST_2 + QT\_COST_3 = \hat{f}(\omega_5) = 30$. The savings in the total processing cost due to multiple query processing strategy amounts to 11 $(30 - 19 = 11)$ (neglecting the overhead due to MTP).

Table 2
The cost estimates

	ω_0	ω_1	ω_2	ω_3	ω_4	ω_5	ω_6
g	0	12	22	19	18	20	15
$\hat{h}$	15	5	0	0	5	10	5

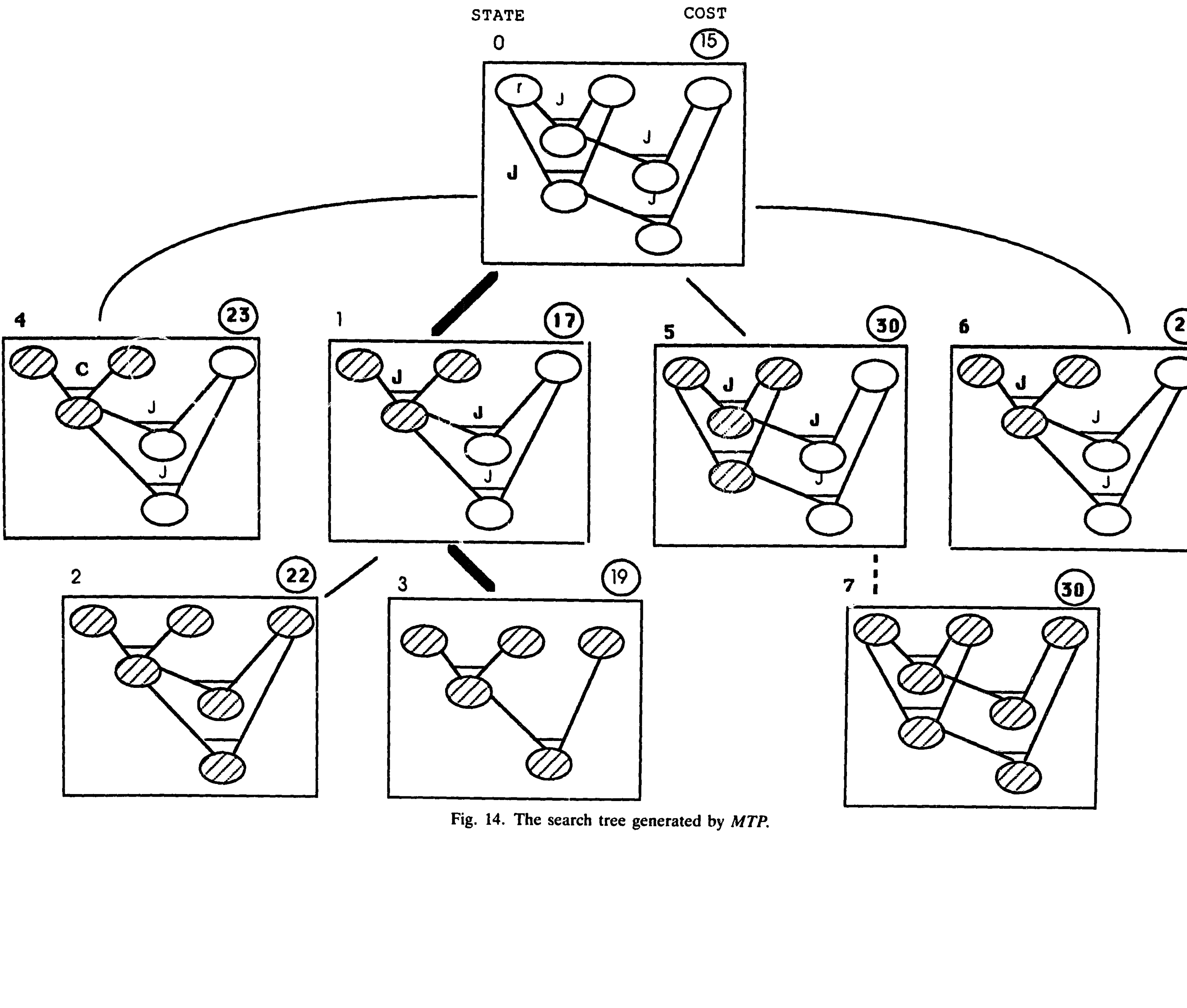

Fig. 14. The search tree generated by *MTP*.

6. Discussion

Suppose that we have two queries QT_2 and QT_3 where QT_2 is a superquery of QT_3 as shown in Fig. 13. The total processing costs T_COST_I, using individual query processing strategies, and T_COST_M, using *MTP*, are

$$T\_COST_I = \text{cost of executing } QT_2 + \text{cost of executing } QT_3$$

$$T\_COST_M = \text{cost of executing } QT_2 + \psi^l(\mathbf{uop}, VIEW(QT_2)) + \Delta\,,$$

where $\psi^l(\mathbf{uop}, VIEW(QT_2))$ is the local processing cost to obtain $VIEW(QT_3)$ from the result of QT_2, and where Δ is the overhead of executing *MTP*. The main factor in Δ is *CPU* time associated with running *MTP*. The *I/O* cost is assumed to be small since the size of the knowledge base containing the semantic knowledge is unlikely to be large in our problem. On the other hand, the cost of executing QT_3 usually involves intersite data communication costs in addition to *CPU* and *I/O* costs. Thus one can conjecture that using a multiple query processing strategy such as *MTP* will be beneficial in environments where communication costs are significant and where the queries and ICs are such that some commonality can be found between the given access plans. This conjecture is strengthened by the promising empirical results recently reported in [33], where multiple query optimization produced a decrease of 20–50% in both *CPU* and *I/O* time in a series of experiments on a *centralized* database, and by the seminal work of King [15] on semantic query optimization.

The central issue for the cost of running *MTP* lies in the performance of the heuristic search. First, we claim that the number of feasible states in the search space is $O(s \cdot k^k \cdot m^l)$, where s is the number of sites, k the number of queries, l the length of common access plans, and m is the average number of superqueries which subsume a subset $\mathbf{QT}$ of the set of k queries but do not subsume any superset of $\mathbf{QT}$. This result is proved in [25]. In brief, assume that there is at least one superquery which subsumes any subset of the k queries (worst case analysis). The question is then how many collective query execution plans can be generated. This problem is equivalent to the problem of partitioning a set of k elements where each collective query execution plan corresponds to a partition. It is known that the total number of partitions of a set with cardinality k is B_k where B_k is the kth Bell number,

$$B_k = \sum_{i=0}^{k-1} \binom{k-1}{i} B_i\,,$$

where $B_0 = 1$. B_k is almost of order k^k for large k; for $k = 15$, it is almost 10^9. Therefore, for k queries, the number of partitions on the set $\mathbf{QT}$ of queries $\cong O(k^k)$. For each partition, there are m^l possible state transitions. Thus, the total number of state transitions $\cong O(k^k \cdot m^l)$. If in addition joins can be performed at any site, i.e. if we relax the first condition in the assumption of fixed access order, then the total number of state transitions is $O(s \cdot k^k \cdot m^l)$.

A straightforward enumeration of all the possible collective execution plans, without using any inference-guiding heuristics, is thus computationally intractable either for the case where the number of queries is not small, or for a real-time environment in which the overhead due to enforcing the multiple query processing strategy cannot be overlooked.

On the other hand, it is difficult to formally evaluate the performance of the heuristic search of *MTP*, especially due to its dependence on the knowledge base and thus on the predicate conditions of individual queries. As argued in [27], a worst case analysis assumes that A^* exhibits its poorest performance; the average performance of A^* is often significantly better. To quantify this statement, one needs to develop a probabilistic model of *MTP* and to apply the results in [27], Chap. 6. This is beyond the scope of this paper. However, we point out that since there are only a few types of rules which can activate state transitions, the search should be efficient if proper semantic knowledge is available.

7. Conclusion

We have introduced a new multiple query processing strategy that makes use of functional dependencies and semantic integrity constraints to determine subset relationships between intermediate results of different queries. The concept of the conventional query graph is extended to represent distributed query processing strategies by including site information. Given some semantic knowledge, the least cost solution is found by a rule-based expert system, Multiple Transaction Processor (*MTP*), in which the planning technique is combined with a search method. We define the cost function (g) and the estimated cost function ($\hat{h}$) for the case of distributed query processing. We also provide a proof that the estimated function ($\hat{h}$) is always underestimating to ensure that an optimal solution will be produced by the A^* algorithm.

Acknowledgement

The authors gratefully acknowledge the critical analysis and suggestions by the referees.

References

[1] P.A. Bernstein, N. Goodman, E. Wong, C.L. Reeve and J.B. Rothnie, Query processing in a system for distributed databases (SDD-1), *ACM Trans. on Database Systems*, vol. 6, no. 4 (Dec. 1981) 602–625.

[2] R. Blankinship, D. Dorris and A.R. Hevner, The file allocation problem – A survey and annotated bibliography, *MS/S 85-013*, Database Systems Research Center, University of Maryland (June 1985).

[3] S. Ceri, and G. Pelagatti, *Distributed Databases, Principles and Systems* (McGraw-Hill, 1984).

[4] U.S. Chakravarthy, D.H. Fishman and J. Minker, Semantic query optimization in expert systems and database systems, in *Proc. 1st Intl. Workshop on Expert Data Base Systems*, L. Kerschberg (Ed.) (Benjamin/Cummings Publ. Co. Inc.) 659–674 (1986).

[5] U.S. Chakravarthy and J. Minker, Processing multiple queries in database system, *IEEE Database Eng.*, vol. 5 (Sept. 1982) 38–43.

[6] S. Finkelstein, Common expression analysis in database applications, in *Proc. 1982 ACM-SIGMOD Int. Conf. Management of Data*, 235–245.

[7] M.S. Fox and J. McDermott, The role of databases in knowledge-based systems, *CMU-RI-TR-86-3*, Department of Computer Science, Carnegie-Mellon University (Feb. 1986).

[8] J. Grant and J. Minker, Optimization in deductive and conventional relational database systems, in *Advances in Data Base Theory*, Vol. 1, H. Gallaire, J. Minker and J.-M. Nicolas (Eds) (Plenum, New York, 1981) 195–234.

[9] A. Guttman, *New Features for Relational Database Systems to Support CAD Applications*, Ph. D Thesis, University of California, Berkeley (June 1984).

[10] P.A.V. Hall, Optimization of single expressions in a relational database system, *IBM J. Res. Develop.*, vol. 20 (May 1976) 144–257.

[11] M.M. Hammer and S.B. Zdonik, Jr., Knowledge-based query processing, in *Proc. Sixth Intl. Conf. on VLDB*, Montreal (Oct. 1980) 137–147.

[12] M. Jarke, J. Clifford and Y. Vassiliou, An optimizing PROLOG front-end to a relational query system, in *Proc. 1984 ACM-SIGMOD Intl. Conf. Management of Data*, Boston, MA (June 1984).

[13] M. Jarke and J. Koch, Query optimization in database systems, *Computing Surveys*, vol. 16, no. 2 (June 1984) 111–152.

[14] M. Jarke, Common subexpression isolation in multiple query optimization, in *Query Processing in Database Systems*, W. Kim, D. Reiner, and D. Batory (Eds.) (Springer-Verlag, 1985) 191–205.

[15] J.J. King, *Query Optimization by Semantic Reasoning* (UMI Research Press, Ann Arbor, Michigan, 1981).

[16] W. Kim, Global optimization of relational queries: A first step, *Query Processing in Database Systems*, W. Kim, D. Reiner and D. Batory (Eds.) (Springer-Verlag, 1985).

[17] S. Lafortune and E. Wong, A state transition model for distributed query processing, *ACM Trans. on Database Systems*, vol. 11, no. 3 (Sept. 1986) 294–322.

[18] R.K. Lindsay, B.G. Buchanan, B.A. Feigenbaum and J. Lederberg, *Applications of Artificial Intelligence for Organic Chemistry: The Dendral Project* (McGraw-Hill, 1980).

[19] B.G. Lindsay, L.M. Hass, C. Mohan, P.F. Wilms et al., Computation and communication in R^*: a distributed database manager, *ACM Trans. on Computer Systems*, vol. 2, no. 1 (Feb. 1984) 24–38.

[20] L. Lilien and B. Bhargava, Database integrity block construct, *IEEE Trans. on Software Eng.*, vol. SE-11, no. 9 (Sept. 1985) 865–885.

[21] G. Lohman, C. Mohan, L. Hass, D. Daniels, B. Lindsay, P. Selinger and P. Wilms, Query processing in R^*, in *Query Processing in Database Systems*, W. Kim, D. Reiner and D. Batory (Eds.) (Springer-Verlag, 1985).

[22] D. Maier, *The Theory of Relational Databases* (Computer Science Press, Rockville, Maryland, 1983).

[23] J. McDermott, R1: A rule-based configurer of computer systems, *Artificial Intelligence*, vol. 19 (1982) 39–88.

[24] N.J. Nilsson, *Principles of Artificial Intelligence* (Tioga Publishing Co., CA, 1980).

[25] J.T. Park, *A Knowledge-Based Approach to Multiple Transaction Processing and Distributed Database Design*, Ph.D. Thesis, University of Michigan, Ann Arbor (1987).

[26] J.T. Park and T.J. Teorey, A knowledge-based approach to multiple query processing in distributed database systems, *Proc. 1987 ACM-IEEE Fall Joint Computer Conference*, Dallas, TX, (Oct. 1987) 461–468.

[27] J. Pearl, *Heuristics: Intelligent Search Strategies for Computer Problem Solving* (Addison-Wesley, 1984).

[28] N. Roussopoulos, View indexing in relational databases, *ACM Trans. on Database Systems*, vol. 7, no. 2 (June 1982) 258–290.

[29] N. Roussopoulos, The logical access path schema of a database, *IEEE Trans. on Software Eng.*, vol. SE-8, no. 6 (Nov. 1982) 563–573.

[30] E.D. Sacerdoti, *A structure for plans and behavior* (American Elsevier, New York, 1977).

[31] M. Schkolnick, A clustering algorithm for hierarchical structures, *ACM Trans. on Database Systems*, vol. 2, no. 1, pp. 27–44 (Mar. 1977).

[32] T.K. Sellis, Global query optimization, in *Proc. 1986 ACM-SIGMOD Intl. Conf. Management of Data*, Washington, D.C. (May 1986) 191–205.

[33] T.K. Sellis, Multiple-Query Optimization, *ACM Trans. on Database Systems*, Vol. 13, No. 1 (March 1988) 23–52.

[34] B. Shneiderman and V. Goodman, Batched searching of sequential and tree structured files, *ACM Trans. on Database Systems*, vol. 1, no. 3 (Sept. 1976) 268–275.

[35] S.T. Shenoy and Z.M. Ozsoyoglu, A system for semantic query optimization, in *Proc. 1987 ACM-SIGMOD Int. Conf. Management of Data*. 181–195.

[36] S.Y.W. Su, K.P. Mikkilineni, R.A. Liuzzi and Y.C. Chow, A distributed query processing strategy using decomposition, pipelining and intermediate result sharing techniques, in *Proc. IEEE 1986 Intl. Conf. on Data Engineering*, Los Angeles, CA, (Feb. 1986) 94–102.

[37] T.J. Teorey and J.P. Fry, *Design of Database Structures* (Prentice-Hall, Englewood Cliffs, N.J., 1982).

[38] P. Valduriez, Join indices, *ACM Trans. on Database Systems*, vol. 12, no. 2, (June 1987) 218–246.

[39] C.T. Yu and C.C. Chang, Distributed query processing, *Computing Surveys*, vol. 16, no. 4 (Dec. 1984) 399–433.