OPTIMAL SEMIJOIN SCHEDULES FOR QUERY PROCESSING IN LOCAL DISTRIBUTED DATABASE SYSTEMS

Mohamed G. Gouda
Umeshwar Dayal

Department of Computer Sciences,
The University of Texas at Austin.

TR-162 November 1980.

ABSTRACT

Semijoin strategies are a technique for query processing in distributed database systems. In the past, methodologies for constructing minimum communication-cost strategies for solving tree queries have been developed. These assume point-to-point communication and ignore local processing costs and the limited communication capacity of the system. In this paper, query processing in bus or loop systems is considered. The definition of strategy is extended to allow for broadcast mode of communication. We then address the problem of finding the minimum response-time schedule for executing a given strategy in an m-bus system taking into account local processing and system capacity. It is shown that the problem is computationally intractable for general tree queries, even in a 1-bus system, and for special classes of tree queries in an m-bus system. However, there is a polynomial-time algorithm for simple queries in a 1-bus system.

1. INTRODUCTION

Semijoin strategies were introduced as a technique for query processing in general distributed database systems [Wo 79, HY 79, He 80, BC 81]. A semijoin strategy is a partially-ordered set of data moves between the different sites in the system to solve a given query. So far, most of the work in this area has been devoted to minimizing total communication cost assuming that local processing costs are negligible.
For this objective, the model of semijoin strategies is appropriate. However, in any specific system with limited communication capacity, parallel data moves in a strategy have to compete for that limited capacity. Consequently, the response time for solving the query is a function of the way these parallel moves are scheduled on the available communication links. Moreover, local processing costs can also affect response time.

This paper addresses the problem of scheduling a given strategy to minimize response time in a local distributed database system. The system architecture is shown in figure 1. It consists of a set of database processors and a front-end processor, connnected via a number of buses or loops*, e.g. ETHERNET [Me 76], MININET [MP 77].

The following assumptions are made:

(i) There is no data redundancy and each site stores one relation.

(ii) The system supports broadcast. Therefore, the same data can be sent to several sites without additional cost.

(iii) At any site, local processing can proceed in parallel with shipment of data to or from the site.

(iv) Communication on one bus is independent of communication on any other bus.

*From now on, we use the term "bus" to indicate either a bus or a loop.

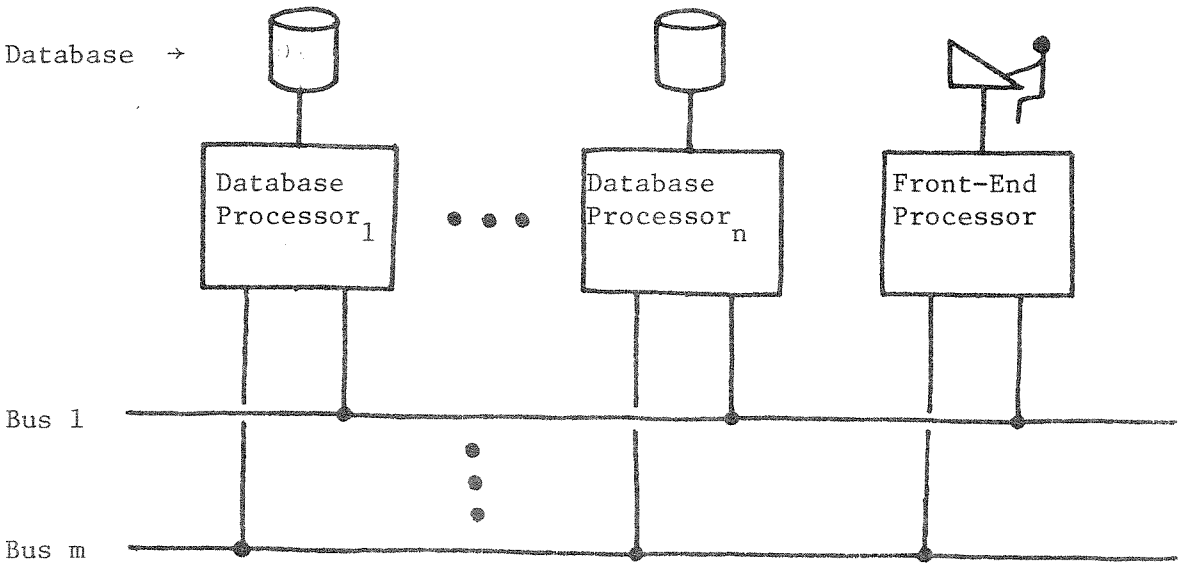


Figure 1. System Architecture

In Section 2 we modify the definition of a semijoin strategy to take into account broadcast capability, and we introduce the concept of a schedule for a semijoin strategy. We also define the m-bus Semijoin Scheduling Problem. In Section 3 this problem is shown to be computationally intractable for tree queries in a 1-bus system. In Section 4, we give a polynomial-time algorithm for a special class of tree queries, viz. simple queries. Finally, in Section 5, we consider systems with m buses and show that the problem is intractable even for the special classes of simple and chain queries.

2. SEMIJOIN STRATEGIES AND SCHEDULES

A query is an undirected graph $Q = (V,E)$, where V is a set of relation names taken from the set $\{R_i \mid 1 \leq i \leq n\}$; E is a set of labelled edges such that an edge $\{R_i, R_j\}$ is labelled A = B, where A is an attribute of R_i, and B an attribute of R_j. A and B are called joining attributes of R_i and R_j respectively. One relation, R_t, in the query is designated the target relation.

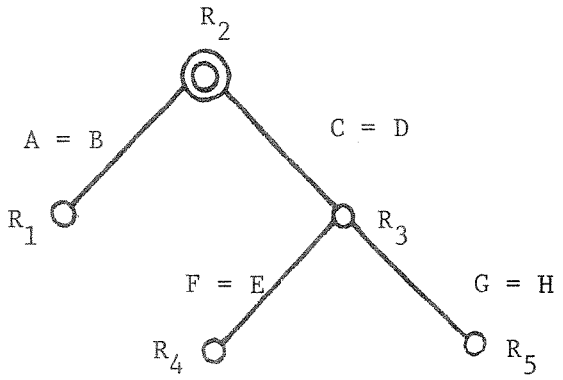
In this paper, we consider only queries that are trees. Examples of tree queries are shown in Fig.2. Consider the query of Fig.2(a). This can be interpreted as a conjunctive relational query whose target-list is in relation R_2 and whose qualification is given by

$$R_1.A = R_2.B \wedge R_2.C = R_3.D \wedge R_3.E = R_4.F \wedge R_3.G = R_5.H.$$

(See [BC 81] for details on tree queries and their interpretations.)

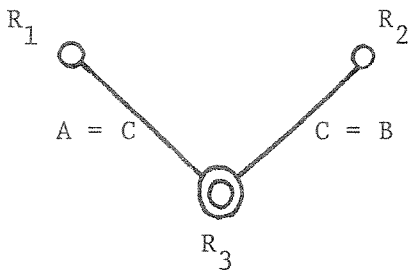
Two special classes of tree queries are of interest, viz. simple queries and chain queries. In a simple query the labels of all the edges incident on node R_i have the same joining attribute of R_i; for example, in the simple query of Fig.2(b), C is the joining attribute of R_3. In a chain query, each node R_i has at most 2 edges incident on it and the labels of these

q_a:



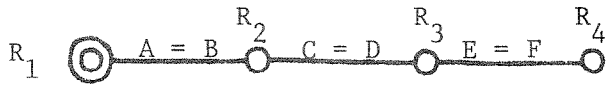
a. A tree query

q_b:



b. A simple query

q_c:



c. A chain query

Figure 2. Examples of tree queries.

edges have different joining attributes of R_i; for example, in the chain query of Fig.2(c), B and C are the two joining attributes of R_2.

Two relations R_i and R_j are joined in query q iff there is an edge $\{R_i,R_j\}$ in q or a path R_0, R_1, R_2,...,R_k, R_{k+1}, such that $R_0 = R_i$, $R_{k+1} = R_j$, and for every p, $1 \leq p \leq k+1$, the labels of the edges $\{R_{p-1},R_p\}$ and $\{R_p,R_p+1\}$ have the same joining attribute of R_p. Intuitively, R_i and R_j are joined in q iff there is a join between R_i and R_j when the qualification of q is closed under transitivity of =. For example, in Fig.2(b), any two relations are joined in q_b.

In [BC 81] it is shown that tree queries with one target relation can be completely solved by a sequence of semijoin operations. A semijoin of R_j by R_i on attributes A and B, denoted $R_i[A=B>R_j$, is defined to be the set of R_j-tuples

$$\{r_j \in R_j \mid (\exists r_i \in R_i)(r_i[A] = r_j[B])\}.$$

If R_i and R_j are stored at different sites i and j of a distributed database system, the semijoin $R_i[A=B>R_j$ is performed in two steps. First, the projection of R_i on A is moved from site i to site j on an available bus. This move step is denoted $R_i[A] \rightarrow R_j$. R_i is called the source, $R_i[A]$ the source attribute, and R_j the destination of this move step. Then, the semijoin is constructed at site j by restricting relation R_j. This local processing step is denoted also by $R_i[A=B>R_j$. Before performing the move and local processing steps to construct the semijoin, some preprocessing (e.g., selections and projections) may be done at sites i and j. These preprocessing steps are denoted $PREP_i$ and $PREP_j$.

In a bus system, data can be broadcast from one site to several different sites without incurring extra cost. Such a move step is denoted by $R_i[A] \rightarrow R_{j1},\ldots,R_{jr}$, where R_i is the source and $R_{j1},\ldots,R_{jr}$ are the destinations of the move step.

Informally, a semijoin strategy for a given tree query q is a sequence of move steps (corresponding to semijoins) that completely solves q. Some of the move steps can be performed in parallel, provided that enough buses are available. Formally, a semijoin strategy ST(q) for a tree query q with target relation R_t consists of a strategy graph, SG(q), and a set of functions associated with each node of SG(q). The strategy graph SG(q) is a directed acyclic graph, each node of which is labelled by a move step $R_i[A] \rightarrow R_{j1},\ldots,R_{jr}$, where R_i and each of the relations $R_{j1},\ldots,R_{jr}$ are joined in q with A being the joining attribute of R_i. SG(q) satisfies the following six conditions:

SG1. Every relation in q, except possibly R_t, is a source of some move step in SG(q).

SG2. Every terminal node (i.e., a node with no successors) has exactly one destination, which is the target relation, R_t, of q.

SG3. For every directed edge (v_1,v_2), the source of v_2 is one of the destinations of v_1.

SG4. For every non-terminal node, labelled $R_i[A] \rightarrow R_{j1},\ldots,R_{jr}$, and for every R_j, $j1 \leq j \leq jr$, which is not the target relation, there is a descendant node whose source is R_j.

SG5. There is a directed path between any pair of nodes with the same source attribute.

SG6. For every pair of nodes such that the source of one is a destination of the other, there is a directed path between the two nodes. □

The justification for SG1 and SG2 is straightforward: every relation in q has to participate in producing the result and the result is produced at the site of R_t[CH 80]. To justify conditions SG3-SG6, refer to the examples in Fig.3. Fig. 3(a) shows that if there were two successive nodes v_1 and v_2 such that the source of v_2 is not a destination of v_1, violating SG3, we could break the directed edge between them without

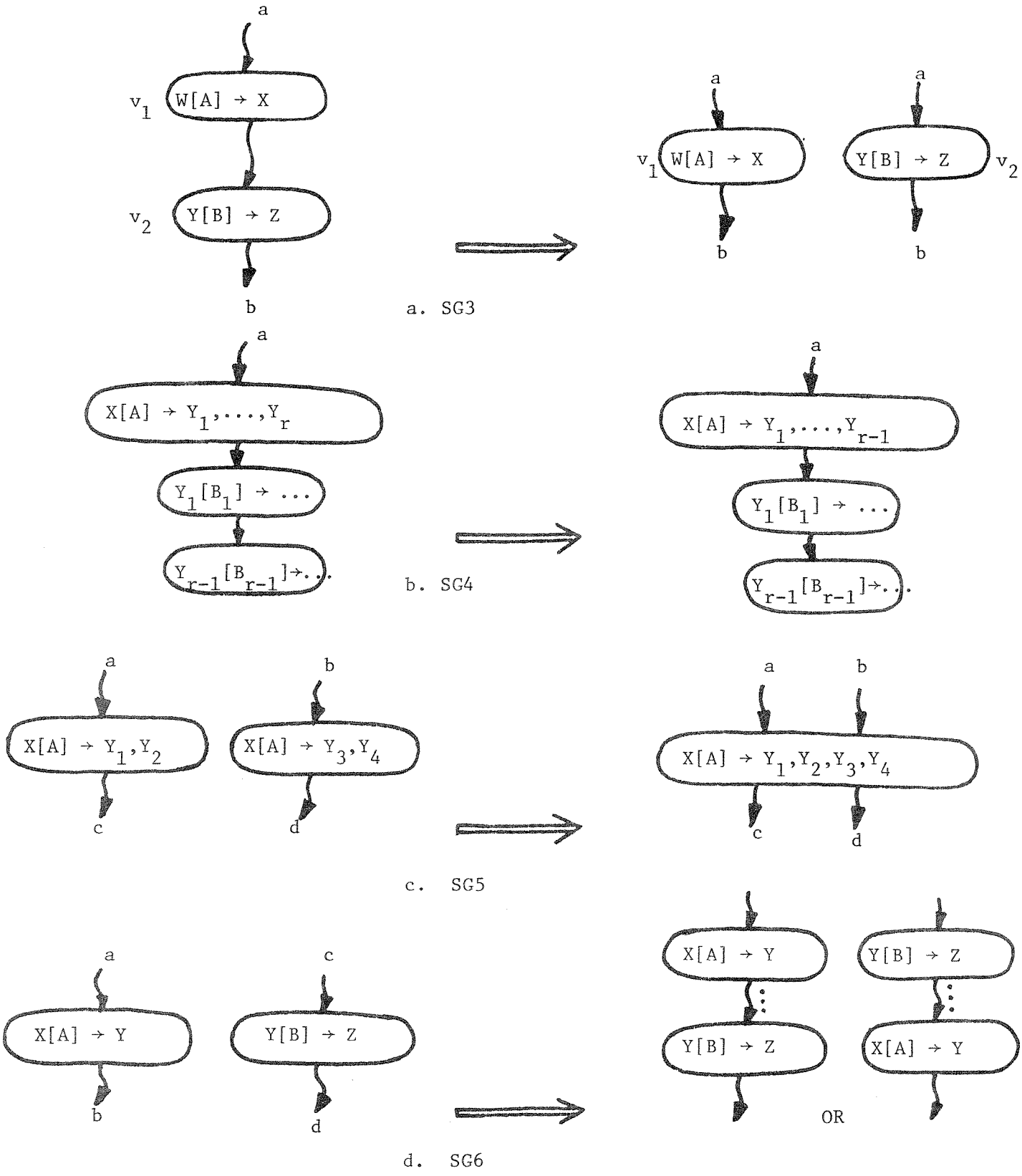
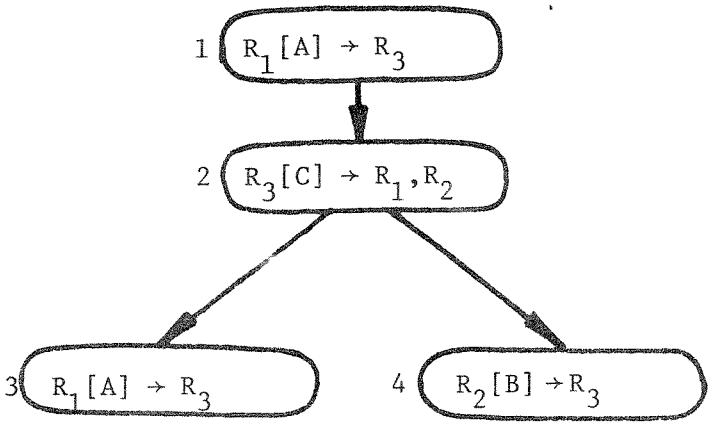


Figure 3. Justification of strategy graph conditions

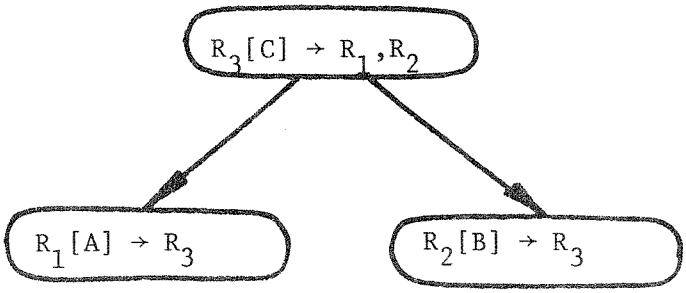
altering the result of executing the semijoin operations in the strategy. Figure 3(b) shows that if some destination Y_r of node v is not a source of any successor of v, then we need not move X[A] to Y_r. Figure 3(c) shows that if there were two parallel nodes v_1 and v_2 with the same source attribute X[A], then we could merge these into a single node. Figure 3(d) shows that if there were two parallel nodes v_1 and v_2 with the source of v_2 being a destination of v_1, then the strategy is ambiguous; it can be made unambiguous by forcing one node to precede the other.

The functions associated with each node are cost functions and selectivity functions. The cost functions define the preprocessing costs, move costs, and local processing costs for constructing the semijoins. These costs are dependent on the current sizes of the relations, which may change after each semijoin operation. Hence, we need the selectivity functions of the joining attributes to compute the new size of a relation after a semijoin [He 79]. We assume that all of these functions monotonically increase with the sizes of the relations involved.

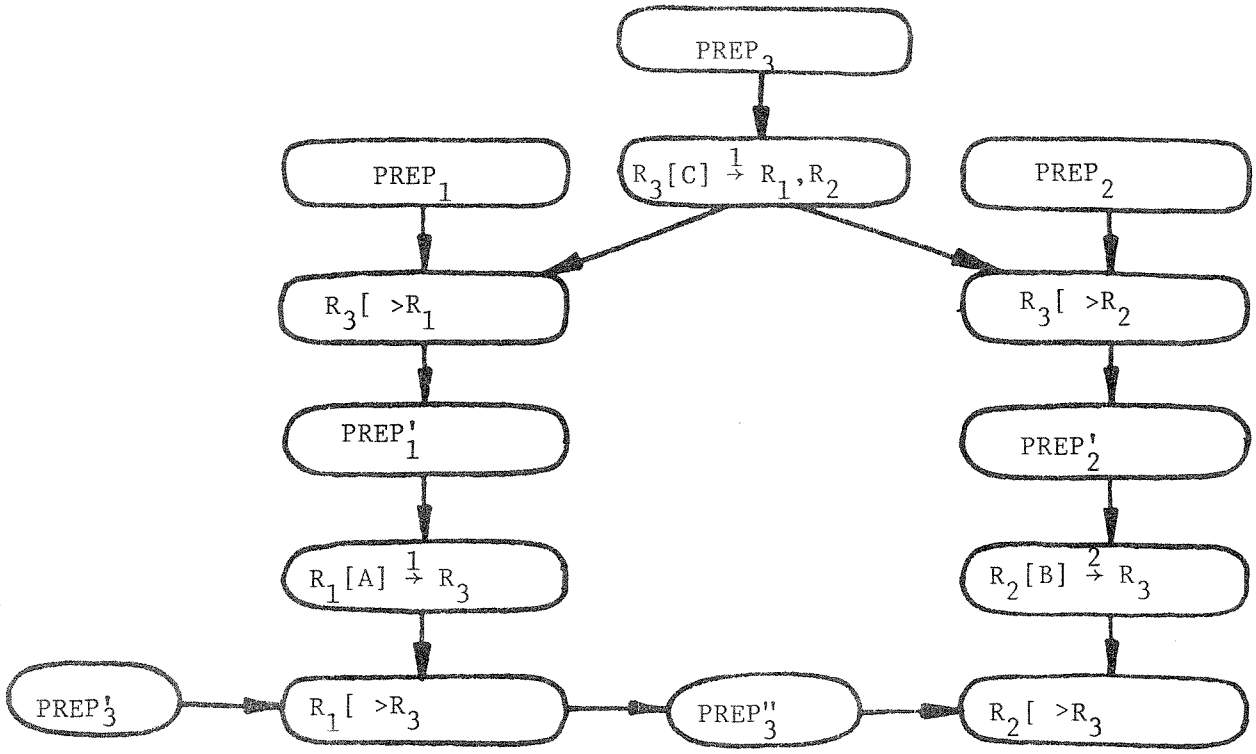
An example of a strategy graph for the query of Fig.2(b) is shown in Fig 4(a). For convenience, nodes are identified by unique numbers. Notice that both node 1 and node 3 represent the same move step. This implies a redundancy in the strategy [CH 80]. Node 1 must be excised to obtain the non-redundant strategy graph in Fig.4(b). Recall that strategies ignore local processing and the number of buses available in the system. For instance, the strategy in Fig.4(b) tacitly assumes the availability of two buses. In practice, however, the number of buses available is determined a priori. Thus, there is a need to construct a schedule, describing the exact sequence of move steps on each bus and local processing and preprocessing steps on each processor. For example, Fig.4(c) shows one possible 2-bus schedule for the strategy graph in Fig.4(b).



a. Redundant strategy



b. Non-redundant strategy



c. A schedule

$PREP'_3$ and $PREP''_3$ are the preprocessing steps at site 3 preceding the local processing steps $R_1[> R_3$ and $R_2[> R_3$, respectively.

Figure 4. An example of a strategy graph and a 2-bus schedule for it.

Formally, an m-bus schedule SC(ST(q),m) for a given strategy ST(q) is a directed acyclic graph constructed from ST(q) as follows:

SC1. Change the label of each node v in SG(q), the strategy graph of ST(q), from $R_i[A] \rightarrow R_{j1},\ldots,R_{jr}$ to $R_i[A] \xrightarrow{B} R_{j1},\ldots,R_{jr}$, where $1 \leq B \leq m$. This assigns a bus to each move step. v is called a move node, and B is called the bus of v.

SC2. For each move node v, labelled $R_i[A] \rightarrow R_{j1},\ldots,R_{jr}$, add (see Fig.5):

(i) preprocessing nodes u, $u_1,\ldots, u_r$ to represent preprocessing at the sites of R_i, $R_{j1},\ldots, R_{jr}$;

(ii) immediate successor nodes, $v_1,\ldots,v_r$, called local processing nodes, to represent the local processing steps at the sites of j1,...,jr following the move step.

SC3. Add directed edges to satisfy the following conditions:

(i) for every pair of move nodes with the same bus B, there is a directed path from one to the other;

(ii) let v_1, v_2 be preprocessing or local processing nodes for the same site, such that v_1 corresponds to a move node u_1 and v_2 corresponds to a move node u_2; then

(a) if u_1 precedes u_2, then there is a directed path from v_1 to v_2;

(b) if there is no directed path between u_1 and u_2, then there is a directed path from v_1 to v_2 or from v_2 to v_1.

For any query, there may be more than one strategy that solves it. In [CH 80] the problem of constructing the minimum communication-cost strategy for solving a given tree query is addressed, and a dynamic programming algorithm is presented. But, for any strategy, there may be more than one schedule on a given number of buses. Our objective is to find a minimum-cost schedule SC(ST(q),m) given a strategy ST(q) and a pre-defined number m of buses. We call this the m-bus Semijoin Scheduling Problem. The cost measure that we adopt is response time. Each node v in a schedule has an execution time, c(v), derived from the functions associated with the corresponding node of the strategy graph.

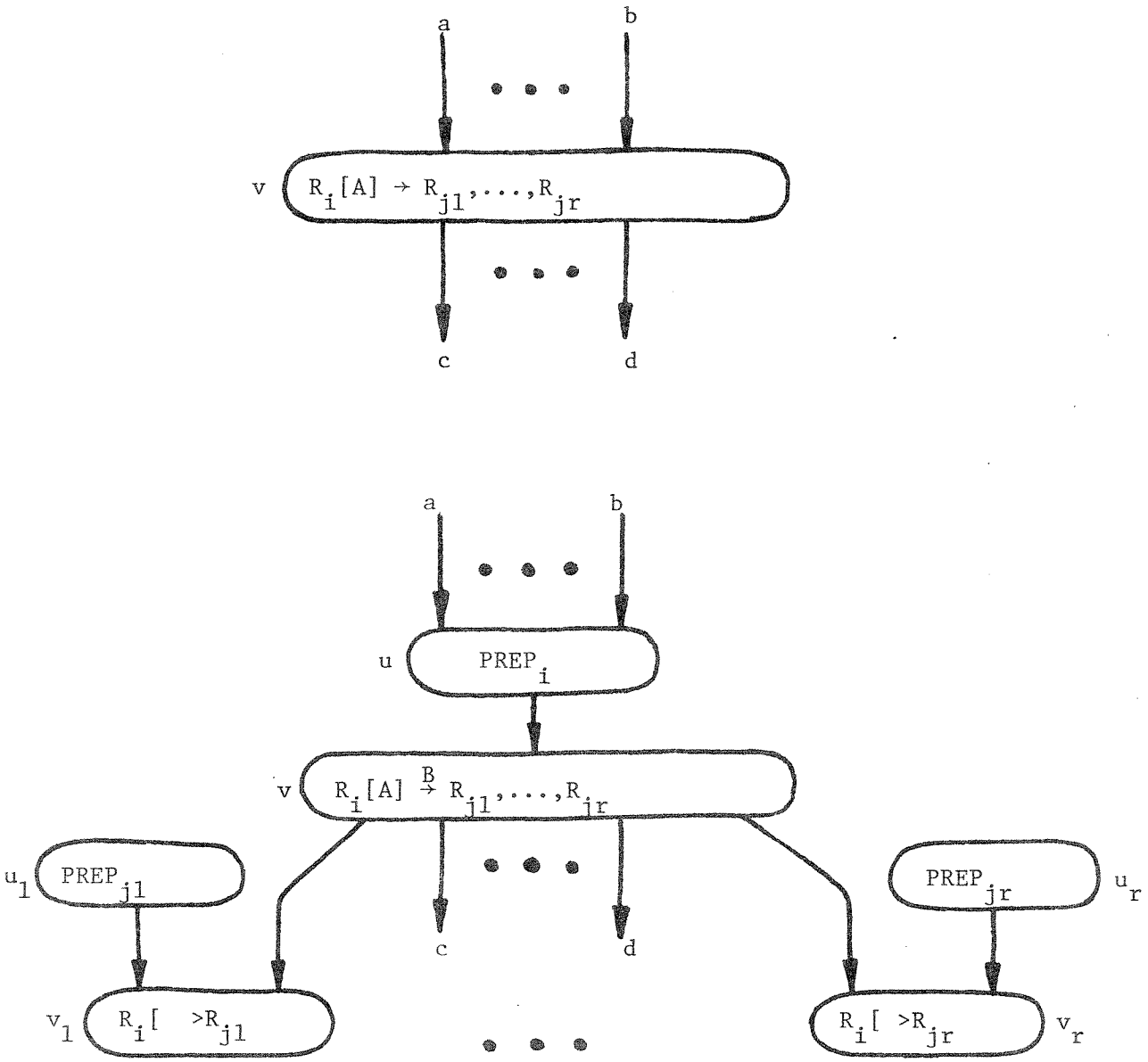


Figure 5. Transforming a strategy graph node to schedule nodes.

The response time of a schedule S is defined as:

$$c(S) = \max_{P} \{ \sum_{v \text{ in } P} c(v) \mid P \text{ is a path in } S \}.$$

3. OPTIMAL 1-BUS SCHEDULES FOR TREE QUERIES

In this section, we prove the surprising result that even for a 1-bus system, the problem of constructing the optimal schedule for a given strategy is computationally complex, viz, NP-hard.

Theorem 1: Given a strategy ST(q) for solving the tree query q, the problem of constructing the minimum response-time 1-bus schedule SC(ST(q),1) is NP-hard.

Proof: We show a polynomial-time reduction from the Job-Shop Scheduling problem which is known to be NP-hard [GS 78, GJ 79]. An instance of the Job-Shop Scheduling problem is the following:

Given - two processors;

- a set $J = \{1,\ldots,n\}$ of jobs;
- each $j \in J$ consisting of an ordered sequence of tasks $t_k[j]$, $1 \leq k \leq 3$;
- for each such task t an execution time $c(t) \in Z_0^+$ and a processor p(t), where $p(t_k[j]) \neq p(t_{k+1}[j])$ for all $j \in J$ and $1 \leq k \leq 3$.

Find the minimum response-time schedule for J.

Construct an instance of the 1-bus Semijoin Scheduling problem as follows: Let the two processors be a bus B and a processor P. The sequence of processors required for the sequence of tasks in any $j \in J$ is one of the following [B], [P], [BP], [PB], [BPB], [PBP]. Each $j \in J$ is mapped into a sequence of move, preprocessing, and local processing steps to solve a tree query q of the form shown in Fig.6 where the target relation R is stored at processor P; each of the other relations is stored at a different processor; and the processors are connected by bus B.

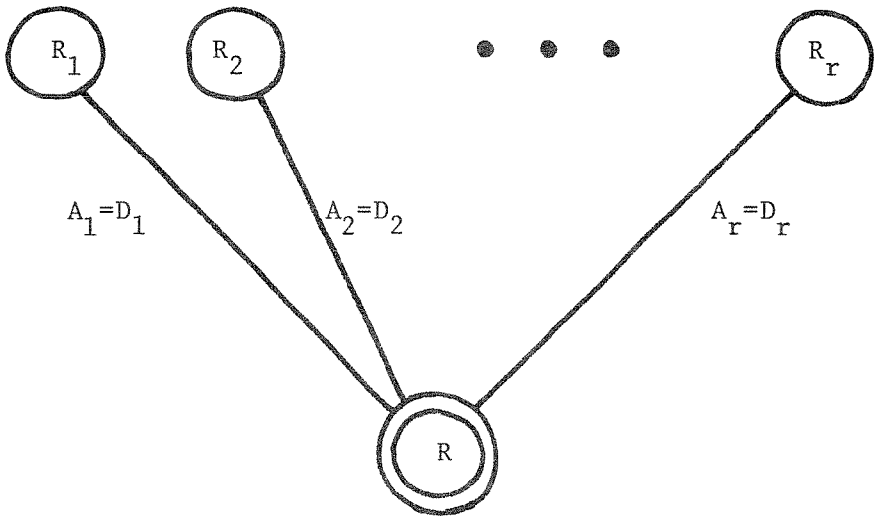


Figure 6. A tree query for the proof of Theorem 1.

Case 1 (j is of type [B]): j is mapped to the sequence*:

1. move: $R_j[A_j] \xrightarrow{B} R$
2. local processing: $R_j[A_j=D_j>R$

Choose the size of $R_j[A_j]$ such that the cost of step 1 is the given $c(t_1[j])$, and choose the size of $R[D_j]$ such that the cost of step 2 is less than $\frac{1}{n}$.

Case 2 (j is of type [P]): j is mapped to the same sequence as in Case 1. Choose the size of $R_j[A_j]$ such that the cost of step 1 is less than $\frac{1}{n}$; and choose the size of $R[D_j]$ such that the cost of step 2 is the given $c(t_1[j])$.

Case 3 (j is of type [BP]): j is mapped to the same sequence as in Case 1. Choose the size of $R_j[A_j]$ such that the cost of step 1 is the given $c(t_1[j])$; and choose the size of $R[A_j]$ such that the cost of step 2 is the given $c(t_2[j])$.

Case 4 (j is of type [PB]): j is mapped to the sequence:

1. move: $R_j[A_j] \xrightarrow{B} R$
2. local processing: $R_j[A_j=D_j>R$
3. move: $R[D_{n+j}] \xrightarrow{B} R_{n+j}$
4. local processing: $R[D_{n+j}=A_{n+j}>R_{n+j}$
5. move: $R'_{n+j}[A_{n+j}] \xrightarrow{B} R$
6. local processing: $R'_{n+j}[A_{n+j}=D_{n+j}>R$

where R'_{n+j} is R_{n+j} after the reduction in step 4. Choose the sizes of $R_j[A_j]$, $R_{n+j}[A_{n+j}]$, $R'_{n+j}[A_{n+j}]$ such that the sum of the costs of steps 1,4,5,6 is less than $\frac{1}{n}$; and choose the sizes of $R[D_j]$ and $R[D_{n+j}]$ such

---

*In this construction, all the preprocessing steps which are not mentioned explicitly are assumed zero, for example, in Case 1, the preprocessing to get $R_j[A_j]$ and $R[B_j]$ is assumed to be zero.

that the costs of steps 2 and 3 are the given costs $c(t_1[j])$ and $c(t_2[j])$ respectively.

Case 5 (j is of type [BPB]): j is mapped to the same sequence as in Case 4. Choose the sizes of $R_{n+j}[A_{n+j}]$, $R'_{n+j}[A_{n+j}]$ such that the sum of the costs of steps 4,5,6 is less than $\frac{1}{n}$; choose the sizes of $R_j[A_j]$, $R[D_j]$ and $R[D_{n+j}]$ such that the costs of steps 1,2, and 3 are the given $c(t_1[j])$, $c(t_2[j])$, and $c(t_3[j])$, respectively.

Case 6 (j is of type [PBP]): j is mapped to the sequence:

1. preprocessing to get $R[D_j]$
2. move: $R[D_j] \xrightarrow{B} R_j$
3. local processing: $R[D_j=A_j>R_j$
4. move: $R'_j[A_j] \xrightarrow{B} R$
5. local processing: $R'_j[A_j=D_j>R$
6. preprocessing to get $R[D_{n+j}]$
7. move: $R[D_{n+j}] \xrightarrow{B} R$
8. local processing: $R[D_{n+j}=A_{n+j}>R_{n+j}$
9. move: $R'_{n+j}[A_{n+j}] \xrightarrow{B} R$
10. local processing: $R'_{n+j}[A_{n+j}=D_{n+j}>R$

Choose the sizes of $R_j[A_j]$, $R'_j[A_j]$, $R[D_{n+j}]$, $R_{n+j}[A_{n+j}]$, and $R'_{n+j}[A_{n+j}]$ such that the sum of the costs of steps 3,4,5,7,8,9, and 10 is less than $\frac{1}{n}$. Choose the cost of the preprocessing in step 1 to be $c(t_1[j])$, and the size of $R[D_j]$ such that the cost of step 2 is $c(t_2[j])$, and the cost of the preprocessing in step 6 to be $c(t_3[j])$.

Notice that in this reduction, we constructed an "augmented" strategy consisting of move, preprocessing and local processing steps. The implied strategy ST(q) can be obtained by deleting all the preprocessing and local processing steps. To make the construction valid, we must ensure that

the size of $R[D_i]$ is unaffected by the semijoin $R_j[A_j=D_j>R$ for all i,j, $i \neq j$. This can be achieved by assuming that relation R is the Cartesian product of its projections.

Suppose we can find a minimal response-time 1-bus schedule S = SC(ST(q),1) for the strategy constructed above. Then the solution to the Job Shop Scheduling problem is obtained as follows. Construct one-processor schedules for bus B and processor P. From each of these schedules delete all steps of cost less than $\frac{1}{n}$. The resulting schedules constitute the minimal response-time Job Shop Schedule, because the total cost of the deleted steps is less than 1, and the costs of the tasks in the given instance of the Job Shop Scheduling problem are all non-negative integers. This proves that 1-bus Semijoin Scheduling is NP-hard. □

Having shown that 1-bus Semijoin Scheduling is computationally intractable, we are faced with two alternatives. One is to look for approximate solutions to the problem for tree queries in general. (This is a subject for future research.) The other is to consider special classes of tree queries and special classes of strategies for which the Scheduling problem is tractable, i.e. of polynomial-time complexity. In the next section, we consider simple queries and linear strategies (to be defined precisely) for solving them.

4. THE SPECIAL CASE OF SIMPLE QUERIES

Recall, from Section 2, that a simple query is a tree query all of whose relations are joined on a single common attribute. A simple query can be solved by intersecting the joining columns of all of its relations, and then joining this intersection with the target relation. We consider a special class of semijoin strategies, called linear strategies, for solving simple queries.

A linear strategy ST(q) for solving a simple query q with target relation R_t is defined as follows.

LS1. Its strategy graph is a chain of the form shown in Figure 7, where $R_1, R_2, \ldots, R_n$ is a permutation of the relations in q.

LS2. The functions associated with each node, labelled $R_i[A_i] \rightarrow R_{i+1}, \ldots, R_n$, in the strategy graph, are defined as follows.

Communication cost of the move step = $c_1 |R_i[A_i]|$*.

There is no preprocessing at the sites of $R_i, R_{i+1}, \ldots, R_n$.

Local processing cost at the site of R_j is

$$c_2 |R_i[A_i]| |R_j[A_j]|, \quad i+1 \leq j \leq n.$$

Selectivity of the joining attribute A_i is $\frac{|R_i[A_i]|}{|dom(A_i)|}$.

Hence, after the semijoin $R_i[A_i = A_j > R_j$, the cardinality of $R_j[A_j]$ becomes $\frac{|R_i[A_i]|}{|dom(A_i)|} \cdot |R_j[A_j]|$, $i+1 \leq j \leq n$.

Linear strategies appear to have some merit. For example, referring to non-linear strategy graph in Figure 8(a), we note that by broadcasting W[A] to Y as well, the cardinality of Y[B] can be reduced before it is shipped to $Z_1, \ldots, Z_n$, thus reducing communication cost.

Given a linear strategy ST(q) for solving a simple query q, the problem of finding the minimum response-time 1-bus schedule for ST(q) is trivial: there is exactly one 1-bus schedule SC(ST(q),1). (Figure 9 shows an example of this.) Hence, in this section, we consider the following problem: given a simple query q, find, of the 1-bus schedules for all linear strategies that solve q, the schedule S with minimum response-time, i.e., $\min_S \{c(S) \mid \exists$ linear strategy ST(q) such that S = SC(ST(q),1)$\}$ We call this the minimum response-time linear schedule for q.

---

* $|X|$ denotes the cardinality of X.

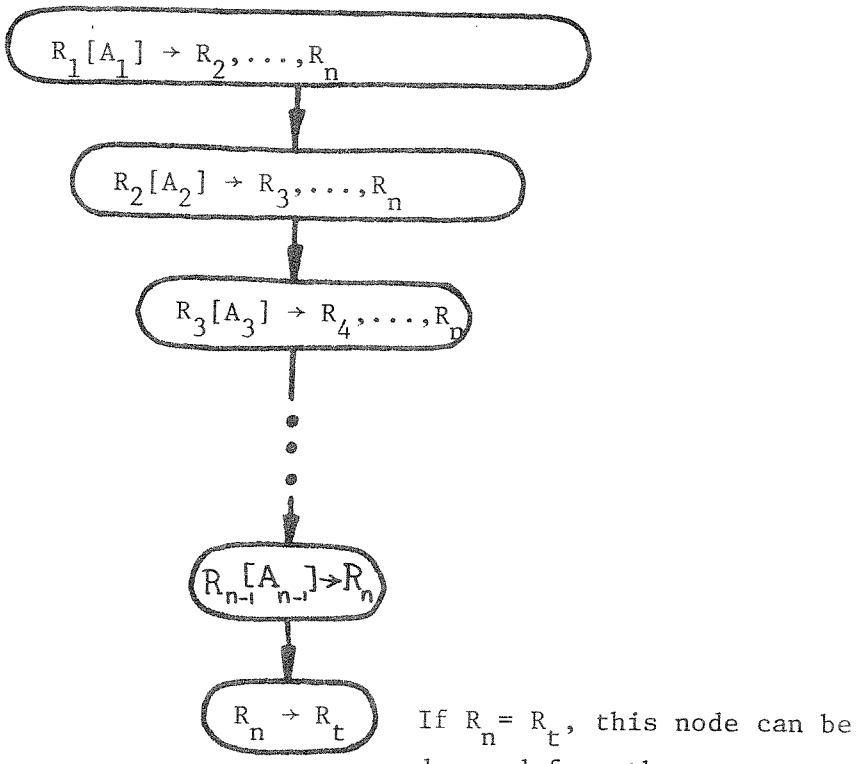


Figure 7. A linear strategy graph for a simple query with relations $R_1,\ldots,R_r$, of which R_t is the target relation

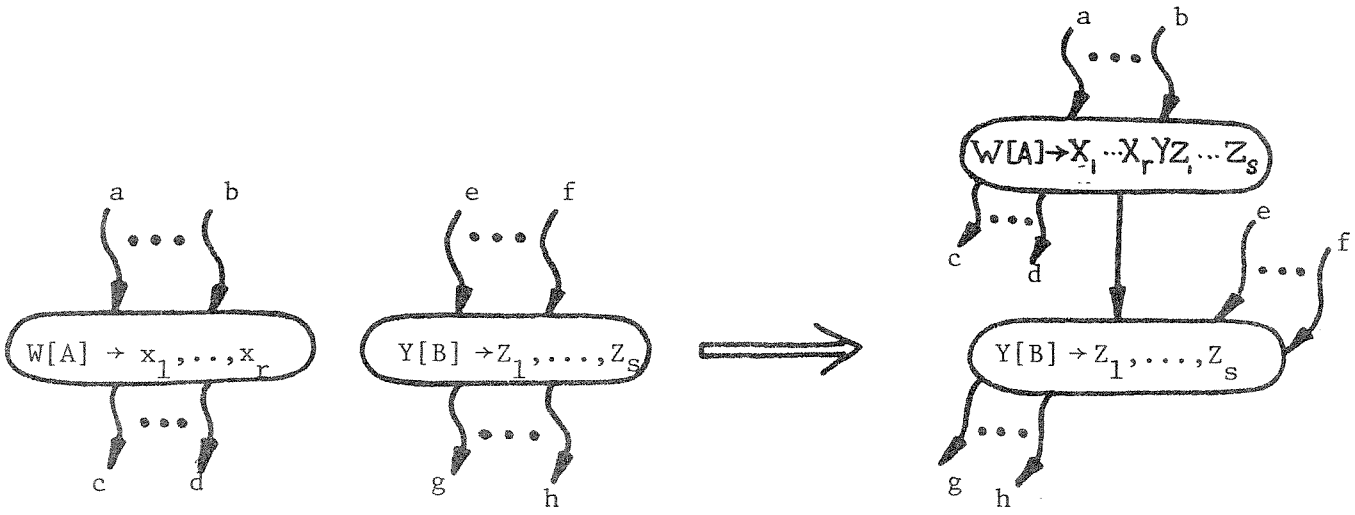


Figure 8. Strategy graph linearization for simple queries

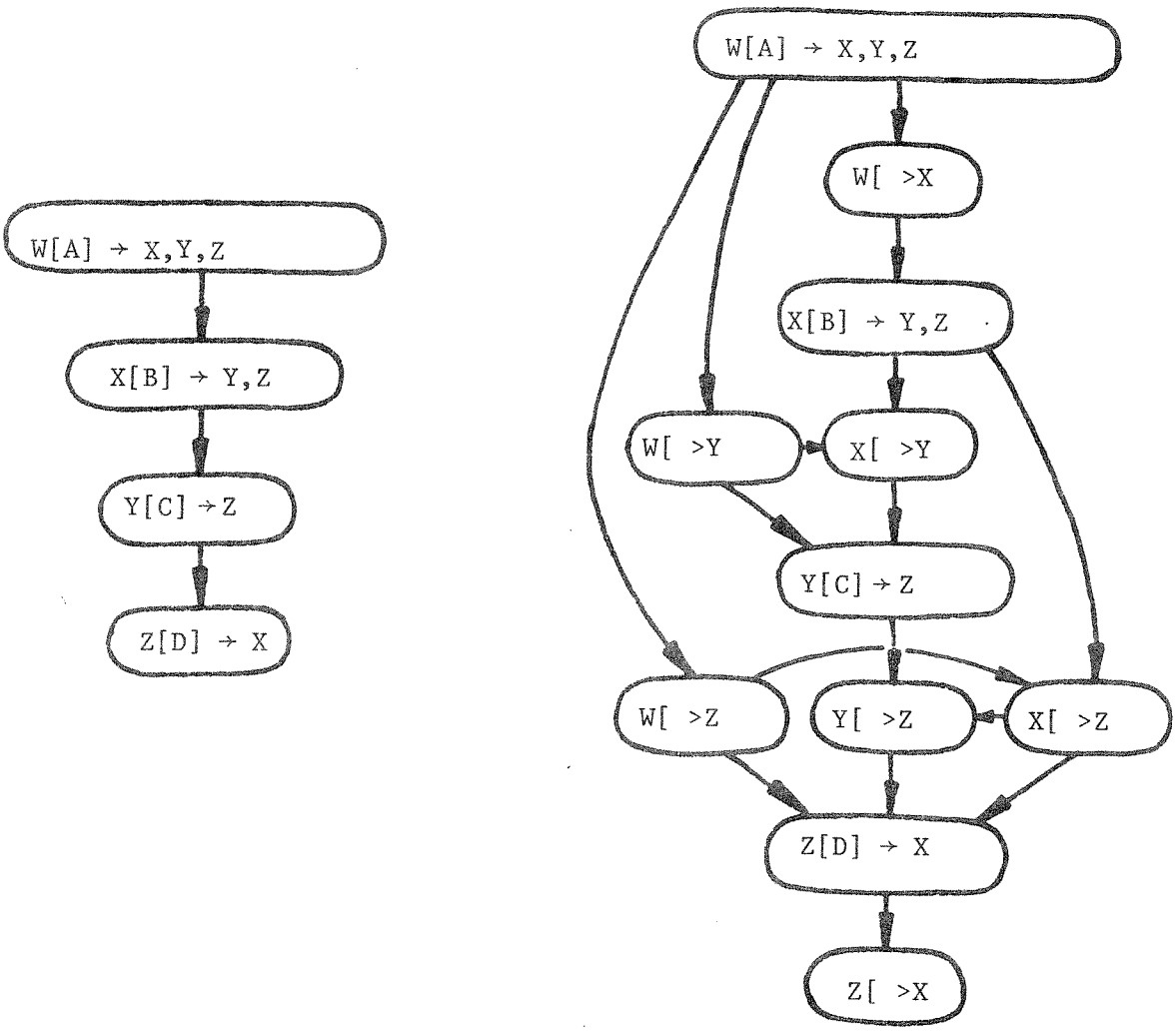


Figure 9. Schedule for a linear strategy that solves a simple query with relations W,X,Y,Z, where X is the target relation.

Theorem 2. Let q be a simple query with target relation R_t. Arrange the relations in ascending order w.r.t. the cardinalities of their joining attributes. Let this order be $R_1, R_2, \ldots, R_n$; i.e., $|R_1[A_1]| \leq |R_2[A_2]| \leq \ldots \leq |R_n[An]|$. Then, the 1-bus schedule, S_0, for the linear strategy whose graph is shown in Figure 7 is a minimum response-time linear schedule for q.

Proof: Let S be a minimum response-time linear schedule for q. Suppose S is not S_0, the 1-bus schedule for the linear strategy in Figure 7. Then there must be two relations X,Y in q such that $|X[B]| \leq |Y[C]|$, where B and C are the joining attributes of X,Y, respectively, and Y[C] is moved over the bus immediately before X[B] is. We show in the Appendix that the Schedule S' obtained by exchanging these move steps has a response time less than or equal to that of S. This argument can be repeated a finite number of times to show that S_0 is a minimum response-time linear schedule for q. □

As an example, for the schedule of Fig.9(b) to be a minimum response-time linear schedule, we must have $|W[A]| \leq |X[B]| \leq |Y[C]| \leq |Z[D]|$.

5. OPTIMAL M-BUS SCHEDULES

In Section 3 we showed that the 1-bus Semijoin Scheduling Problem for tree queries is NP-hard. We now show that when there are several buses in the system, the problem is NP-hard even for the special cases of simple queries and chain queries.

Theorem 3. Given a strategy ST(q) for solving a query q and a fixed number m of buses, the problem of constructing the minimum response-time m-bus schedule SC(ST(q),m) is NP-hard when

(i) q is a simple query
(ii) q is a chain query
(iii) q is an arbitrary tree query

Proof: First, observe that if we prove the problem NP-hard for (i) or (ii), this will imply that it is NP-hard for (iii).

To prove (i) and (ii) we show a polynomial-time reduction from the Multiprocessor Scheduling problem, which is known to be NP-hard [GJ 79].

An instance of the multiprocessor scheduling problem is the following. Given a set $T=\{1,2,\ldots,n\}$ of tasks, a number m of processors, and an execution time $c(t) \in Z^+$ for each $t \in T$, find the minimum m-processor schedule for T.

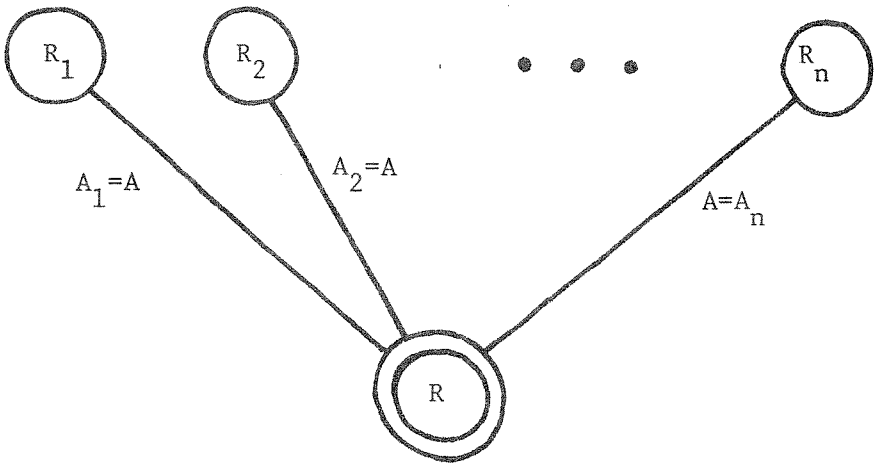
(i) q is a simple query:

Construct an instance of the m-bus Semijoin Scheduling problem as follows. Let the m processors be buses $B_1,\ldots,B_m$. Let q be the simple query of Figure 10(a), and ST(q) the strategy of Figure 10(b). For each i, $1 \leq i \leq n$, choose the size of $R_i[A_i]$ such that the cost of the move step $R_i[A_i] \rightarrow R$ is the given c(i). Choose the size of R[A] such that the cost of the local processing $R_i[A_i=A>R$ is less than $\frac{1}{n}$, for each i, $1 \leq i \leq n$. Note that since the cost of local processing monotonically decreases as the sizes of the relations being semijoined decreases, we are guaranteed that the cost of each local processing step $R_i[A_i=A>R$ will always be less than $\frac{1}{n}$, even after R has been reduced by previous semijoins.

Suppose we can find a minimum response-time m-bus schedule for ST(q). Deleting the steps of cost less than $\frac{1}{n}$ yields the minimum m-processor schedule for the given instance of the Multiprocessor Scheduling problem. This proves that m-bus Semijoin Scheduling is NP-hard for simple queries.

(ii) q is a chain query:

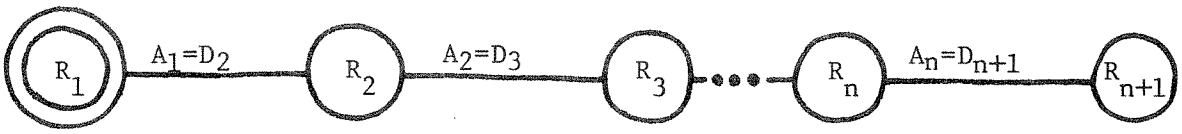
Construct an instance of the m-bus Semijoin Scheduling problem as follows. Let the m processors be buses $B_1,\ldots,B_m$. Let q be the chain query of Figure 11(a), and ST(q) the strategy of Figure 11(b). For each



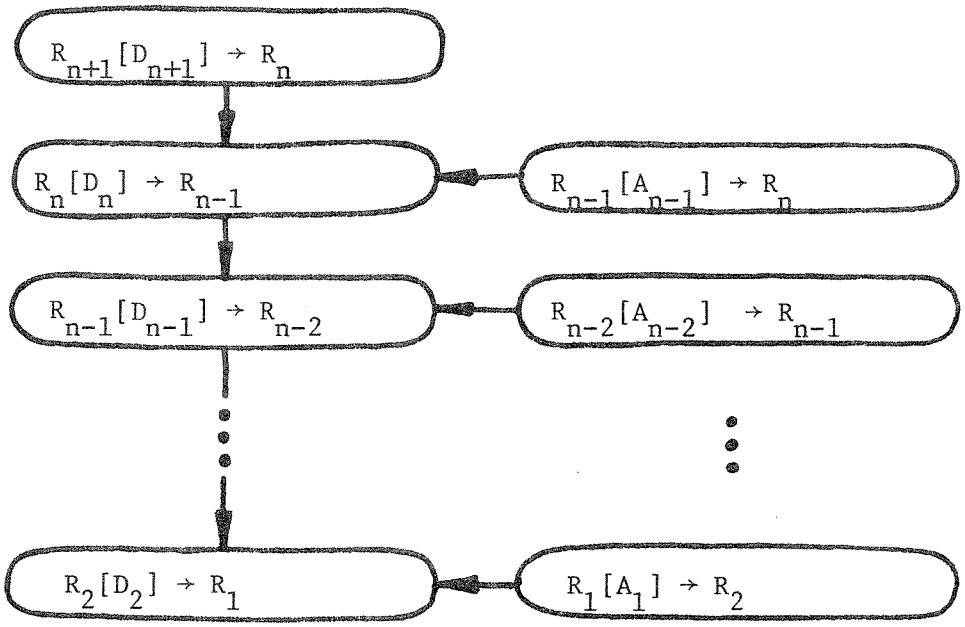
a. Simple query q

b. Strategy ST(q)

Figure 10. A simple query and strategy for the proof of Theorem 3(i).



a. Chain query q



b. Strategy ST(q)

Figure 11. A chain query and strategy for the proof of Theorem 3(ii).

i, $1 \leq i \leq n-1$, choose the initial size of $R_i[A_i]$ such that the cost of the move step $R_i[A_i] \rightarrow R_{i+1}$ is the given $c(i)$. Choose the size of $R_{n+1}[A_{n+1}]$ such that the cost of the move step $R_{n+1}[A_{n+1}] \rightarrow R_n$ is the given $c(n)$. Choose the sizes of $R_n[D_n]$ and $R_n[A_n]$ such that the cost of each of the local processing steps $R_{n-1}[A_{n-1}=D_n>R_n$ and $R_{n+1}[D_{n+1}=A_n>R_n$ is less than $\frac{1}{n+1}$. Choose the selectivity of $R_i[A_i]$, $1 \leq i \leq n-1$, such that the size of $R_i[A_i=D_{i+1}>R_{i+1}$ is small enough to make the cost of each move step $R_{i+1}[D_{i+1}] \rightarrow R_i$ and each local processing step $R_{i+1}[D_{i+1}=A_i>R_i$, $1 \leq i \leq n-1$, less than $\frac{1}{n+1}$.

Now suppose that we can find a minimum response-time m-bus schedule for ST(q). Deleting the steps of cost less than $\frac{1}{n+1}$ yields the minimum m-processor schedule for the given instance of the Multiprocessor Scheduling problem. This proves that m-bus Semijoin Scheduling is NP-hard for chain queries. □

6. CONCLUSIONS

We have suggested a two-step technique to process queries in distributed database systems. First, a semijoin strategy to solve the query is constructed based on some criterion,e.g. minimizing the total communication cost. Then, a schedule for that strategy is constructed to minimize its response time. In this paper, we have investigated the second problem in some detail for bus or loop systems. We have shown that the problem of constructing minimum response-time schedules for a given strategy is computationally intractable, even in a 1-bus system. For the case of m buses the problem is NP-hard even for special classes of tree queries, viz. simple and chain queries. This suggests that exact optimization is very hard. A pragmatic approach, then, would be to develop heuristics that yield good approximate

solutions to this problem. This is a subject for future research. Another issue that merits investigation is the use of "mixed" strategies, i.e., strategies involving both semijoins and joins; these are appropriate when queries are permitted to have more than one target relation.

ACKNOWLEDGEMENT:

We are thankful to Carol Engelhardt for her careful typing at such short notice.

REFERENCES

[BC 81]
Bernstein, P.A., and D-M. Chiu, "Using Semi-Joins to Solve Relational Queries," (to appear) JACM 28:1, Jan. 1981.

[CH 80]
Chiu, D-M., and Y-C. Ho, "A Methodology for Interpreting Tree Queries into Optimal Semi-Join Expressions," Proc. ACM-SIGMOD Intl. Conf. on Management of Data, May 1980, pp.169-178.

[GJ 79]
Garey, M.R., and D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, W.H. Freeman and Company, San Francisco, 1979.

[GS 78]
Gonzalez, T., and S. Sahni, "Flowshop and Jobshop Schedules: Complexity and Approximation," Operations Research 26, 1978, pp. 36-52.

[He 80]
Hevner, A.R., "The Optimization of Query Processing on Distributed Database Systems," Ph.D. Diss., Tech. Rep. DB-80-02, Dept. of Computer Sci., Purdue Univ., Lafayette, Indiana, 1980.

[HY 79]
Hevner, A.R., and S.B. Yao, Query Processing in Distributed Database Systems," IEEE Trans. on Software Engg., 5:3, May 1979, pp. 177-187.

[Me 76]
Metcalfe, R.M., "ETHERNET: Distributed Packet Switching for Local Computer Networks," CACM 19:7, July 1976, pp. 395-403.

[MP 77]
Manning, E.G., and R.W. Peebles, "A Homogeneous Network for Data-sharing Communications," Computer Networks 1, 1977, pp. 211-224.

[Wo 77]
Wong, E., "Retrieving Dispersed Data in SDD-1: A System for Distributed Databases," Proc. 1977 Berkeley Workshop on Distributed Data Management and Computer Networks, May 1977, pp. 217-235.

APPENDIX

Proof of Theorem 2

Let S be a minimum response-time linear schedule for q. Suppose S is not S_0. Then, there must be two relations X, Y in q such that $|X[B]| \leq |Y[C]|$, where B and C are the joining attributes of X, Y, respectively, and Y[C] is moved over the bus immediately before X[B] is. We show here that the schedule S' obtained by exchanging these move steps has a response time less than or equal to that of S.

Without loss of generality, assume that W is the relation whose joining attribute W[A] is moved on the bus immediately before X[B] and Y[C] are in S and S'; and that Z is the relation whose joining attribute Z[D], is moved on the bus immediately after X[B] and Y[C] are in S and S'. Referring to Fig. 12, T', the response-time of S', is given by:

$$T' = K' + LP'_x + MOVE'_x + LP'_y + MOVE'_y + \Delta LP'_z + L'$$

where:

K is the cost of all steps in S' upto and including the move of W[A];

LP'_x is the local processing at the site of X from the time that X receives W[A] until the move of X[B]; $LP'_x = \Delta LP'_x + c_2.w.x$, where*

$\Delta LP'_x$ is the remaining local processing at the site of X resulting from the move steps preceding the move of W[A]; and

c_2wx is the local processing cost of W[A=B>X;

$MOVE'_x$ is the cost of the move step X[B] → Y,Z,...

$$MOVE'_x = \frac{c_1.w.x}{D} \quad ;$$

*w denotes $|W[A]|$, x denotes $|X[B]|$, etc. D denotes the cardinality of the domain of the joining attributes.

LP'_y is the local processing at the site of Y from the time that Y receives X[B] until the move of Y[C]

$LP'_y = \Delta LP'_y + c_2\frac{wxy}{D}$, where

$\Delta LP'_y$ is the remaining local processing at the site of Y resulting from the move steps preceding the move of X[B];

and $c_2\frac{wxy}{D}$ is the local processing cost of X[B=C>Y;

$MOVE'_y$ is the cost of the move step Y[C] → Z,...

$MOVE'_y = c_1\frac{wxy}{D^2}$;

$\Delta LP'_z$ is the remaining local processing at the site of Z resulting from the move steps preceding the move of Y[C]

$\Delta LP'_z = \max(L_z - K_1, 0)$

where L_z is the total local processing at the site of Z resulting from all move steps preceding the moves of X[B] and Y[C];

$K_1 = K' + LP'_x + MOVE'_x + LP'_y + MOVE'_y$;

L' is the cost of all the steps in S' from the local processing of Y[C=D>Z to the end.

Symmetrically, T, the response time of S, is given by:

$$T = K + LP_y + MOVE_y + LP_x + MOVE_x + \Delta LP_z + L$$

But,

K = K', because the move steps preceding the move of W[A] are the same in S and S';

$L \geq L'$ because the local processing of X[B=D>Z in S' is the same as the local processing of Y[C=D>Z in S, and all subsequent steps cost at least as much in S as they do in S' (since the former use y and the latter use x, and $y \geq x$);

$LP_y = \Delta LP_y + c_2wy$

$= \Delta LP'_x + \Delta LP'_y + \Delta + c_2wy$ where ≥ 0

This is because $y \geq x$ (see Figure 12).

$MOVE_y = c_1\frac{wy}{D}$

$LP_x = c_2\frac{wyx}{D}$ since X would certainly have finished all local processing resulting from preceding move steps before Y has finished all its local processing resulting from the same move steps.

$$MOVE_x = c_1\frac{wyx}{D^2}$$

$$\Delta LP_z = \max(L_z - K_2, 0)$$

where L_z is as before, and

$$K_2 = K + LP_y + MOVE_y + LP_x + MOVE_x$$

Thus,

$$K_1 = K + \Delta LP'_x + c_2 wx + c_1\frac{wx}{D} + \Delta LP'_y + c_2\frac{wxy}{D} + c_1\frac{wxy}{D^2}$$

and $$K_2 = K + \Delta LP'_x + \Delta LP'_y + \Delta + c_2 wy + c_1\frac{wy}{D} + c_2\frac{wxy}{D} + c_1\frac{wxy}{D^2}$$

$K_1 \leq K_2$ since $x \leq y$ and $\Delta \geq 0$

There are now three cases to consider:

Case 1 ($L_z - K_1 \leq 0$ and hence $L_z - K_2 \leq 0$):

This implies $\Delta LP'_z = \Delta LP_z = 0$.

Hence, $T' = K_1 + 0 + L' \leq K_2 + 0 + L = T$

Case 2 ($L_z - K_1 > 0$ and $L_z - K_2 \leq 0$):

This implies $\Delta LP'_z = L_z - K_1$, $\Delta LP_z = 0$

Hence, $T' = K_1 + L_z - K_1 + L' = L_z + L' \leq K_2 + L = T$

Case 3 ($L_z - K_1 > 0$ and $L_z - K_2 > 0$):

This implies $\Delta LP'_z = L_z - K_1$, $\Delta LP_z = L_z - K_2$

Hence, $T' = L_z + L' \leq L_z + L = T$.

Thus, we have shown that the response time T' of S' is less than or equal to the response time T of S. This argument can be repeated a finite number of times to show that S_0 is a minimal response-time linear schedule for q. □

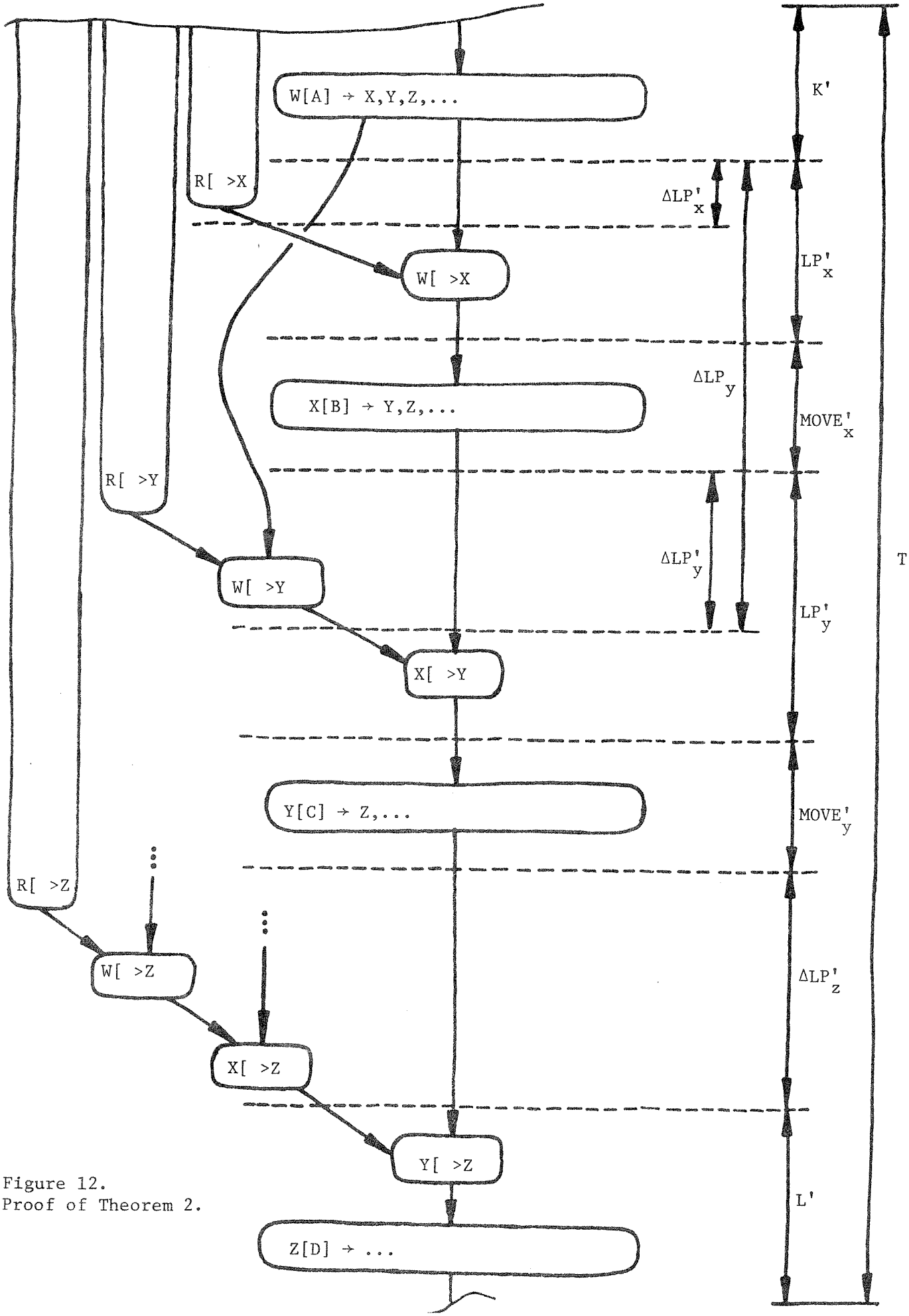


Figure 12.
Proof of Theorem 2.