Some Issues in Design of Distributed Deductive Databases

Mukesh K. Mohania
Dept. of Computer Science and Engg.
Indian Institute of Technology,
Bombay 400 076, INDIA
mukesh@cse.iitb.ernet.in

N. L. Sarda
Dept. of Computer Science and Engg.
Indian Institute of Technology,
Bombay 400 076, INDIA
nls@cse.iitb.ernet.in

Abstract

The design of a distributed deductive database system differs from the design of conventional non-distributed deductive database systems in that it requires design of distribution of both the database and rulebase. In this paper, we address the rule allocation problem. We consider minimisation of data communication cost during rule execution as a primary basis for rule allocation. The rule allocation problem can be stated in terms of a directed acyclic graph, where nodes represent rules or relations, and edges represent either dependencies between rules or usage of relations by rules. The arcs are given weights representing volume of data that need to flow between the connected nodes. We show that rule allocation problem is NP-complete. Next, we propose a heuristic for nonreplicated allocation based on successively combining adjacent nodes for placement at same site which are connected by highest weight edge, and study its performance vis-a-vis the enumerative algorithm for optimal allocation. Our results show that the heuristic produces acceptable allocations. We also extend our heuristic for partially replicated allocation.

Proceedings of the 20th VLDB Conference
Santiago, Chile, 1994

1 Introduction

Distributed deductive database systems have been an intensive field of research in the last few years. A Distributed Deductive Database System (DDedDBS) consists of many autonomous deductive database systems [CGT90] connected by a computer network to facilitate sharing of both databases and rulebases. Efforts are being focused on defining an architecture for such systems [CR91, HBH94, Li88, MS93c].

Distribution design is a phase in the development of distributed applications concerned with the allocation of data as well as rules and operations to the various processing sites according to the application's requirements. Distribution of data in a distributed database system has already received considerable attention [CP86], while partitioning and allocation of rulebase in a DDedDBS has not yet been addressed adequately. The problem of rule allocation is complex and challenging when large number of rules and sites are involved. Sellis [SRN90] quotes Hayes-Roth on expert systems reaching 100,000 rules by 1990 in applications such as engineering processes and manufacturing. Moss [Mos92] describes many individual applications (such as Airport management, Boeing's Connector Assembly Specification Expert, Integrated Circuits Layout, etc.) written in Prolog and using very large number of rules. The rulebase will be much larger when we wish to build an integrated set of applications.

In this paper, we illustrate the problem of partitioning and allocation of rulebase to sites. The rule allocation problem is defined by us as a mapping problem across a set of interconnected database systems so that for all possible execution of rules at various sites, the overall communication cost is minimised. A poorly designed rule allocation can lead to less efficient computation and higher communication cost, whereas a

well-designed rule allocation can enhance the rulebase availability, diminish the access time, and minimize the communication cost. Thus, it is crucial to provide DDedDBSs with a good means to achieve rule allocation.

Given (a) a set of sites (a site is a physical location of database system) with a possibly replicated allocation of relations to them, (b) the Global Rulebase (GRB) consisting of Datalog rules [CGT90], and (c) rule execution information pertaining to transmission cost, the objective is to allocate the rules of GRB across the set of sites in such a way that the overall communication cost is minimized. Our approach assumes that the problems of distribution of data and rulebase can be treated separately. Further, we assume that the former has already been done based on the dominant database queries (including updates) and that the latter needs to be done based on database usage by rules and interrelationships between the rules. This is obviously a simplifying assumption, since the distribution of one affects the other. In distributed databases, data allocation takes into account sites and frequencies of execution of read and update types of transactions at operational level. Here, execution of each transaction is independent of execution of other transactions. The objective of data allocation problem is to minimise the overall communication cost. This problem has been extensively studied in [Ape88, CNW83, RW79]. The allocation of rules to sites in distributed deductive databases is not only based on usage of data stored at multiple sites and frequencies of execution of rules, but also based on interdependencies between rules. That is, allocation of rules depends on allocation of their dependent rules. The objective of rule allocation problem is also to minimise the overall communication cost.

Since a rule may refer to another rule or data stored at another site, the individual DedDBS must cooperate during the execution of that rule. This cooperation consists of resolution of rules and/or database access and transmitting data from one site to another. Minimising communication cost for data is one important goal in allocation of rulebase.

We show that the rule allocation problem is NP-complete [GJ79]. The NP-compleness is shown by transforming *multiterminal cut* problem [DJP+92] into a restriction of the rule allocation problem. This problem has also been formulated as a 0-1 integer programming with quadratic function in [MS93b]. Since the time complexity of the optimal allocation of rules is exponential, this situation justifies the use a of heuristic algorithm which provides a sub-optimal solution for solving the rule allocation problem. We propose and evaluate a heuristic algorithm for nonreplicated allocation of rules, and compare the results of it with the exhaustive enumeration algorithm that produces allocations of rules with minimal communication cost. We also derive a heuristic for partial replication of rules which starts off with a given nonreplicated allocation.

The results presented in this paper can be contrasted with those of [DJP+92, SV91] which solve the so called partitioning problem for a graph. The partitioning problem for more than two partitions (say, k) has been used in a variety of applications [Bre77, Kri84, San89], most having to do with the minimisation of communication costs in parallel computing systems. In the literature, two types of graph partitioning problems are discussed, namely, *k-cut* problem [SV91] and *multiterminal cut* problem [DJP+92]. In [MS93a], we have shown that none of the solutions to any of the variations of graph partitioning problems are directly applicable to rule allocation problem and also shown that the *multiterminal cut* problem is the special case of rule allocation problem where each relation node is initially assigned to precisely one site. If rule allocation problem did not have replicated data allocation, then the approximation algorithm of *multiterminal cut* problem could have been used for rule allocation. The detailed comparison is discussed in [MS93a].

In section 2, some preliminaries are given. Section 3 defines rule allocation problem more precisely. Section 4 contains the proof of NP-completeness of the rule allocation problem. A heuristic algorithm for allocation of rules to sites, which is based on the largest connection weight between two nodes, is presented in section 5. In section 6, we outline a heuristic procedure for partially replicated allocation of a rulebase. The experimental results for measuring effectiveness of our algorithm in finding near-optimal solutions are given in section 7. Finally, section 8 contains our conclusions and future plans.

2 Preliminaries

2.1 Definitions

A Datalog rule has the form

$P_0(X_0) : -P_1(X_1), \ldots, P_n(X_n)$

where P_i is a predicate name and X_i is a vector of terms, each involving a variable or a constant. Each $P_i(X_i)$ is called a literal. The left hand side of a rule is the head of rule. It is a derived predicate. The right hand side of the rule is the body. It contains either derived predicates or base predicates or both. A base predicate corresponds to a relation in the database. A query is a rule without a head. A rule is recursive if its head predicate appears in its body. A predicate P directly depends on predicate Q, if Q appears in the

body of P. The transitive dependence is called indirect dependence.

A cluster is a collection of rules. It represents a partition of the global rulebase. As rules in a cluster will have same site allocation, the clusters can also be viewed as units for allocation.

2.2 Issues in Replication

Let us briefly characterize the application scenarios of DDedDBMSs envisaged by us. A DDedDBMS may serve a single large organization, or multiple organizations may wish to share their data and rulebases for cooperative processing or profit. Assume a set of sites in these cases, each with a database. Either these database systems already exist as autonomous units or the data has been distributed across sites to meet their operational needs. A GRB is now given to be stored appropriately at the sites for efficient querying by the users. We are interested in applications where GRB may contain thousands of rules to be shared by users at tens of sites. The problem of replication of rules in DDedDBMS has been discussed at length in [MS93b]. It can be approached in various ways as summarized below:

(i) **No replication:** GRB can be partitioned into non-overlapping clusters, one for each site. If properly designed, most rules might compile and execute locally, without any communication overheads. For rules using RB from multiple sites, compilation can also exploit parallelism (which in effect corresponds to the access planning by Distributed DBMS for distributed execution of a query).

(ii) **Full replication:** Here, the GRB is stored fully at all sites. The rule allocation problem disappears in full replication, but can be justified only when GRB is fairly static and of moderate size. We note that partitioning of large RB even in single systems has been suggested for reducing search space and improving execution time [RB92] as well as for simplifying administration of RB [JF90]. Moreover, the GRB may be less time varying than the database, but it is not fully static, and we cannot ignore cost of updates in fully replicated RB. (We note further that full replication of even database catalogs is not always advocated in distributed database systems [CP86]). If GRB if fully replicated, rule compilation is completely local; however, the underlying distributed DBMS must perform access planning for distributed execution of the rule when it uses data from multiple sites.

(iii) **Controlled (i.e., part-) replication:** This approach falls in between the above two approaches. The replication may be at rule or cluster level.

3 The Rule Allocation Problem

We wish to allocate the rules of a rulebase to sites based on their usage of data stored at multiple sites and on rule execution information pertaining to transmission cost. A rule that uses data and/or rules from only one site is best stored at that site. However, if a rule uses databases or rulebases at multiple sites, it must be stored at that site which minimizes the communication cost. To minimize communication cost we need to consider some application specific data such as frequencies of executions, expected volume of results, etc. We propose the partitioning of a global rulebase into as many clusters as there are sites. We restrict the rules of a dependency cycle to be present at the same site. To meet this objective, we normalise the rules of GRB where a cycle is replaced by a single 'compiled recursive' rule. If the rules of a cycle are mapped to different sites, then participating sites have to communicate at every intermediate step of execution until the termination point of execution is reached. Hence, keeping the rules of a cycle in one site might reduce the execution time as well as communication cost.

To achieve this, we replace all the rules of a cycle by a single 'compiled recursive' rule. After allocation of the rules, the compiled recursive rules can be replaced back by the rules merged in their definitions.

Lemma 1 *Assignment of recursive rules to sites is not affected if recursive predicates are removed from the bodies of rules.*

Proof: If a recursive rule is executed, then it will be called by itself at the same site and there will not be any intersite communication cost due to recursive predicate. Therefore, recursive predicates do not affect the allocation of rules. ∎

Input and Output Data

The following matrices are related to rule execution information pertaining to computation of transmission cost. They will be employed as input data in the rule allocation problem. Let n be the number of rules, m be the number of sites and l be the total number of relations stored at various sites. Let a, i & i' be index variables for rules, j for relations, and b, k & k' for sites.

Let $P \in \{0,1\}^{n \times n}$ be the rule-rule dependency matrix where $p_{ii'} = 1$ if rule i is directly dependent on rule

i', and $p_{ii'} = 0$ otherwise. Let $Q \in \{0,1\}^{n \times l}$ be the rule-relation dependency matrix where $q_{ij} = 1$ if rule i is directly dependent on relation j, and $q_{ij} = 0$ otherwise. Let $A \in \{0,1\}^{l \times m}$ be the relation-site allocation matrix where $a_{jk} = 1$ if relation j is present at site k, and $a_{jk} = 0$ otherwise. Each rule r_i is associated with a relative execution frequency f_{ik}, which gives how frequently r_i is executed at site k. The frequency matrix F is defined as $F = \{f_{ik} \mid 1 \leq i \leq n \text{ and } 1 \leq k \leq m\}$. We assume that frequencies are measured over a suitable time frame. A relation size matrix T is defined as $T = \{t_j \mid 1 \leq j \leq l\}$, where t_j is the size of relation j. The result size cost matrix, Z is defined as $Z = \{z_i \mid 1 \leq i \leq n\}$, where z_i is the average answer size returned by execution of rule i.

Recall that the head of a rule may have many arguments, and in a query, one or more of these may be bound (i.e., instantiated), affecting size of result produced by its execution. We consider average result size based on results produced by execution of same rule with different instantiations and different frequencies. (Note: we could have treated each instantiation as a different rule.)

The matrices T and Z will be used for computing communication costs, which, we assume, is directly proportional to the amount of data transferred from one site to another (and not on how the sites are connected by the topology).

It must be re-iterated here that the allocation results obtained would depend on how well we have characterized the usage pattern of data and rules in the application. This problem is faced in most problems concerned with allocation such as data/file allocation, load balancing, etc. We expect that the input parameters chosen by the designer are representative of expected usage.

We first consider nonreplicated allocation of rules in this paper. That is, each rule should be assigned to only one site. The problem of replicated allocation is briefly discussed in section 6.

Note: Rules with same head predicates, or rules forming a recursive cycle are merged or replaced as discussed above. The input data should be prepared so as to take these into account.

Example

Let the matrices given below be the input values for a given rulebase. We will use this as a running example to illustrate rule allocation problem. In this example, there are 3 rules and 2 sites.

$$P = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \quad A = \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 1 & 0 \end{bmatrix} \quad F = \begin{bmatrix} .5 & .5 \\ .5 & .5 \\ .5 & .5 \end{bmatrix}$$

$$Q = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \quad T = \begin{bmatrix} 100 \\ 80 \\ 10 \\ 80 \\ 10 \end{bmatrix} \quad Z = \begin{bmatrix} 50 \\ 20 \\ 50 \end{bmatrix}$$

∎

4 Complexity of the Rule Allocation Problem

In this section, we show that the problem of rule allocation in distributed deductive database systems is NP-complete. In fact, our proof demonstrates that the rule allocation problem is NP-complete even if we do not allow replication of relations across the sites.

4.1 Rulebase as a Dependency Graph

A rulebase can be represented as a dependency graph $G_1 = (V, E)$ with relations as leaf nodes and rules as non-leaf nodes, and edges representing usage of rules and relations. Nodes are assigned with either relation sizes or result sizes. The dependency graph can be considered as a Directed Acyclic Graph (DAG), since it does not contain any cycle (recall that recursive predicates are removed since they do not affect the allocation). We define the *depth* of each node and the *height* of dependency graph in the following definitions.

Definition 1 The *depth* for leaf nodes (i.e. relation nodes) is 0. The *depth* d of a node is defined as max(*depth* of descendents)+1.

Definition 2 The *height* h of the dependency graph is defined as maximum *depth* of any node in the graph.

The graph in figure 1 is the dependency graph of our running example. Here, the relation nodes at leaf level are shown by rectangles with the result of their allocation and weights in curly and small parentheses respectively. The rule nodes are represented by small circles along with their weights in parentheses.

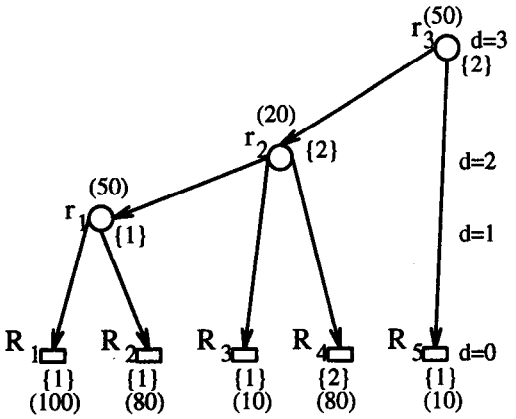


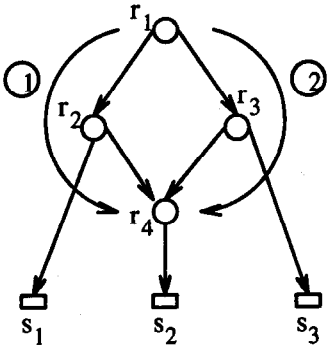
Figure 1: A dependency graph G_1

4.2 Conversion to Dependency Graph with Edge Weights

We reduce a dependency graph into another dependency graph having weights associated with edges rather than nodes. This graph is constructed in three steps.

(i) Convert a dependency graph G_1 having node weights $w(v)$ into another graph G_2 having edges weight $w(e)$. The edge weight between nodes i' and i in graph G_2 represents the total volume of data flowing from node i' to node i for all executions of i and ancestors of i. The procedure of calculating edge weight from i' to i, where node i' could be either a rule or relation node, is as follows:

- find out the set I of all ancestor nodes of node i in graph G_1; include i also in I,
- for each ancestor node $a \in I$, find out the number of paths to reach to node i' from a in the dependency graph. Here the number of paths indicates that rule i' will execute as many time as the number of paths if rule a is executed. This can be understood from the following dependency graph.



Here, when rule r_1 executes, rule r_4 will execute two times, first as a dependent of r_2 and second of r_3. There are two paths to reach to node r_4 from r_1. These are shown by the arrows in the above dependency graph.

- $w(i', i) = \sum_{a \in I} \sum_{k=1}^{m} \mathcal{P}_{ai'} \cdot f_{ak} \cdot \begin{cases} z_{i'} & \text{if node } i' \text{ is a rule node} \\ t_{i'} & \text{if node } i' \text{ is a relation node} \end{cases}$

where $\mathcal{P}_{ai'}$ gives the total number of paths from a to i'. We calculate the weight of all edges of graph G_1 of figure 1, which are shown in graph G_2 of figure 2.

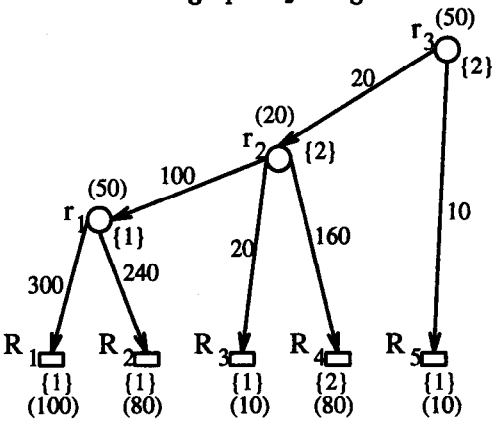


Figure 2: The graph G_2

(ii) A rule stored at one site can be invoked from another site. To include the weight due to this, we do the following:

(a) add a set of specified nodes $S_1, \cdots, S_k$ (equal to the number of sites), where node S_b corresponds to site b

(b) connect all non-specified nodes (rule nodes) of graph G_2 with each newly added specified node

(c) calculate the weight of these newly edges as follows:
The weight of edge connecting rule node i to newly added node S_k is:
$w(i, k) = f_{ik} \cdot z_i \ \{1 \leq i \leq n, \text{ and } 1 \leq k \leq m\}$
where $w(i,\ k)$ shows communication cost of sending results of rule i at site k, if it is executed at remote site.

The idea of adding a set of nodes equal to the number of sites is to include the cost of communication on account of execution of rules at remote sites. Thus, this new graph, say G_3, contains $n + l + m$ nodes. We calculate the weight of all new edges which are shown in graph G_3 of figure 3.

(iii) Reduce $n+l+m$ nodes of graph G_3 into another graph G having $n + m$ nodes. We merge rela-

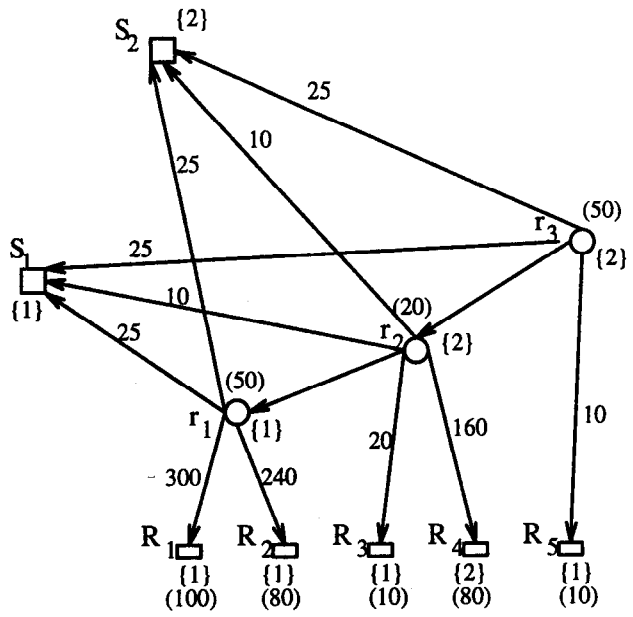


Figure 3: The graph G_3

tion nodes with specified site nodes where they are stored by a single node and connect edges to latter node. Those edges which have same endpoints will be replaced by a single edge having weight equal to the sum of weights of replaced edges. If a relation node is stored at more than one site, then this node will be considered at all sites for replacement.

An edge-weight matrix W can be obtained from graph G, which gives weights between nodes of graph G. The graph G in figure 4 is the final graph obtained at step (iii). In this graph, s_1 and s_2 are specified nodes, and r_1, r_2 and r_3 are nonspecified nodes.

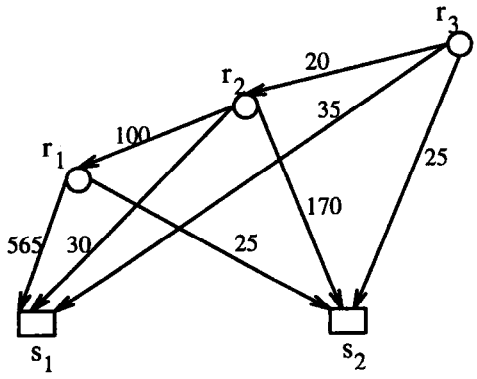


Figure 4: The graph G

The edge-weight matrix W for graph G is

$$W = \begin{array}{c} \\ s_1 \\ s_2 \\ r_1 \\ r_2 \\ r_3 \end{array} \begin{array}{c} \begin{array}{ccccc} s_1 & s_2 & r_1 & r_2 & r_3 \end{array} \\ \begin{bmatrix} 0 & 0 & 565 & 30 & 35 \\ 0 & 0 & 25 & 170 & 25 \\ 565 & 25 & 0 & 100 & 0 \\ 30 & 170 & 100 & 0 & 20 \\ 35 & 25 & 0 & 20 & 0 \end{bmatrix} \end{array}$$

Definition 3
The *multiterminal cut* problem[DJP+92] can be defined as follows:

Instance: Graph $G = (V, E)$, a set $S = \{s_1, \cdots, s_k\}$ of k specified vertices, also called terminals, *$\{s_1, \cdots, s_k\} \in V$, and a positive weight for each edge $e \in E$.*

Question : Find a minimum weight set of edges $E' \subseteq E$ such that removal of E' from E disconnects each terminal from all the others.

Lemma 2 *Multiterminal cut problem for any fixed $k \geq 3$ is NP-complete, even if all weights are equal to 1.*

Proof: When k=2 *multiterminal cut* problem reduces to the well known *Min-cut* problem which can be solved in polynomial time by standard network flow techniques [PS82]. The proof of NP-completeness of *multiterminal cut* problem for k >2 is by a transformation from the *Simple Maxcut* problem [GJ79] and is given in [DJP+92]. ∎

4.3 Proof of NP-completeness

We are now ready to prove the NP-completeness of rule allocation problem. The following theorem proves the NP-completeness.

Theorem 1 *For any fixed $k \geq 3$, Rule Allocation Problem (RAP) is NP-complete.*

Proof: It is easy to see that $RAP \in NP$, since a nondeterministic algorithm need only guess a set of edges E' and check in polynomial time that sum of weights of the edges E' from E that have one endpoint in one cluster and one endpoint in another cluster is minimized.

The NP-completeness of RAP is proven by restriction. Each relation is restricted to be assigned to precisely one site. Now the restricted RAP is defined as follows:

Instance : Graph $G = (V, E)$, a set $S = \{s_1, \cdots, s_k\}$ of k specified site vertices $\{s_1, \cdots, s_k\} \in V$, and a positive weight $w(e)$ for each edge $e \in E$.

Question : Find a partition of V into k disjoint clusters $V_1, \cdots, V_k$ such that $s_1 \in V_1, \cdots, s_k \in V_k$ and sum of the weights of the cutting edges across the clusters is minimized.

It is easy to see that restricted RAP is in one-to-one correspondence with the *multiterminal cut* problem where site vertices correspond to terminals and rule vertices correspond to non-terminals. We obtain a restricted RAP identical to the *multiterminal cut* problem which has been shown to be NP-complete. Since a restriction on the general RAP is NP-complete, the RAP itself is NP-complete. ∎

5 Algorithm for Nonreplicated Allocation of Rules

We have shown in section 4 that the optimal rule allocation problem is NP-complete. Since the time complexity of the optimal allocation of rules is exponential, this situation justifies the use of heuristic algorithm. In this section, we propose a new heuristic method for rule allocation based on largest connection weight between two nodes in a graph. Our heuristic algorithm can be used for nonreplicated allocation of thousands of rules to sites.

5.1 A Heuristic Method for Allocation

We propose a heuristic for solving the nonreplicated rule allocation problem represented by an edge weight graph. This heuristic is a based on assigning two nodes of a graph to the same site if they are connected by highest weight edge. If one of them is a specified (i.e., site) node, then the other node will be assigned to the same site of the specified node. Nodes are clubbed (merged) in the descending order of their connection weights. This continues until number of nodes becomes equal to the number of specified vertices (i.e. sites). The heuristic algorithm operates on the edge-weight matrix W, defined in section 4, which contains the weights of the connections between nodes of a graph. Since the edge-weight matrix W is symmetric with respect to its diagonal, we consider the upper diagonal matrix for computations. This reduces the number of comparisons by half. The procedure of clubbing nodes and their assignment to sites is shown in figure 5. In this procedure, *home* is an output array and *home*(i) gives the site of rule i.

The **create_allocation** procedure gives us the allocation of rules to sites. It does not give the overall communication cost for executing rules over all sites at specified frequencies, because relations may be replicated at multiple sites which add more communication

```
global data:
    input W: matrix;
    Output home: array;
procedure create_allocation( )
    k̄ = m + n;
    ∀_{i∈n} merge(i) = i;
    /* merge(i) is a set of rule nodes which can be
      merged with i, initially assigned with i only. */
    repeat
     w(i,i') = Max{w(j, j')|(j > m or j' > m)
                          and 1 ≤ j, j' ≤ k̄ };
      /* find the two nodes i and i' (at least one
       should be a non-specified node) with the
       largest connection weight between them
       (ties are broken arbitrarily).*/
     for t = 1 to k̄
      {
       w(i, t) = { w(i, t) + w(i', t)   if i' ≠ i
                 { 0                    otherwise
          /* combine rows i' and i in matrix W */
        If both i and i' are nonspecified nodes,
        then merge(i) = merge(i) + merge(i');
          /* merge i and i' and replace by i */
        If i is specified node and i' is nonspecified,
        then ∀_{a∈merge(i')} home(a) = i;
          /* gives home site i of rule a */
      }
     endfor
     V = V − {i'};
     delete w(i', t), w(t, i');
       /* adjust W by dropping row and column */
     k̄ = k̄ − 1;
    until k̄ = m
end_create_allocation.
```

Figure 5:

cost. If the relations are not replicated, then the overall communication cost will be the sum of the weights of the edges across the partitions. If relations are replicated, we have to compute communication cost separately.

This procedure can have n iterations and $((n + m)(n + m - 1)/2)$ comparisons are required to find the maximum value in W. Thus, the complexity of this procedure is $O(n(n+m)^2)$.

5.2 Cost Estimation

For a given allocation of rules (derived from **create_allocation** procedure), we calculate the overall communication cost by the following equations. We calculate the cost for a rule in a bottom-up fashion with respect to the dependency graph.

Let rule i be allocated at site k. The cost of exe-

cuting rule i at site k once is

$$single(i,k) = \sum_{j=1}^{l} q_{ij} \cdot (1 - a_{jk}) \cdot t_j + \sum_{i'=1,\, i \neq i'}^{n} \left(\begin{cases} z_{i'} + single(i',k') & \text{if } p_{ii'} = 1,\ k \neq k' \\ single(i',k') & \text{if } p_{ii'} = 1,\ k = k' \\ 0 & \text{if } p_{ii'} \neq 1 \end{cases} \right) \quad (1)$$

Here, the first term gives communication cost of accessing relations and the second term gives cost of obtaining results of descendent rules which are stored at some other site.

The rule i can also be executed from other sites. So, the execution site of i should send results to other sites. Hence, the net communication cost for executing rule i over all sites at specified frequencies is

$$Val(i,k) = \left(\sum_{k'=1}^{m} f_{ik'} \right) \cdot single(i,k) + \left(\sum_{k'=1,\, k' \neq k}^{m} f_{ik'} \cdot z_i \right) \quad (2)$$

Here, the first term represents the cost of accessing relations and rules, and the second term represents the cost of sending results of executing rule i when it is invoked from other sites. The algorithm **Rule_Allocation** is described in figure 6. The complexity of **create_allocation** procedure is $O(n(n+m)^2)$. The complexity of calculating the overall communication cost is $O(n)$. Hence, the overall time complexity of the heuristic algorithm will remain as $O(n(n+m)^2)$.

```
global data:
   input P, Q, A, F, T, Z, W: matrices;
   Output home: array; total_cost: real;
procedure Rule_Allocation();
   create_allocation( );
   total_cost=0;
   for d = 1 to h do
      for each node i having distance d do
        {
         k = home(i);
         compute single(i,k), Val(i,k) as per
         equations 1 and 2 respectively;
         total_cost = total_cost + Val(i,k);
        }
      endfor
   endfor
   print home, total_cost;
end_Rule_Allocation.
```

Figure 6: Algorithm for Rule Allocation

Example

In figure 4, we first combine node r_1 and s_1, because they have the largest connection weight. These nodes will be replaced by node s_1. This implies that r_1 will be allocated at site 1. Next node r_2 and s_2 will be combined by node s_2 as edge(r_2, s_2) has maximum weight of 170 and then, node r_3 will combine with node s_2. Hence, rule r_1 is allocated to site 1, and rules r_2 and r_3 are allocated to site 2. For this allocation, we calculate the overall communication cost. The cost of executing rule r_1 at site 1 once is

$$single(1,\ 1) = 0$$

and net communication cost for executing rule r_1 over all sites at specified frequencies is

$$Val(1,\ 1) = 1 * 0 + .5 * 50 = 25.$$

Likewise, $single(2,\ 2) = 10 + 50 = 60$, $single(3,\ 2) = 10 + 60 = 70$, $Val(2,\ 2) = 1 * 60 + .5 * 20 = 70$, and $Val(3,\ 2) = 1 * 70 + .5 * 50 = 95$. Therefore, the overall communication cost will be 190. In this example, it is the same as the sum of the weights of the cutting edges among specified vertices as shown above. Note that relations are not replicated in this example.

6 Partially Replicated Allocation of Rules

The introduction of replication may be at rule and/or cluster level. The degree of replication is a design parameter. That is, the degree of replication becomes a variable of the problem. In this section, we limit our discussion to replication at rule level and present a greedy heuristic method for introducing 'additional replication' of rules. We start with a given nonreplicated allocation and determine all beneficial sites for replication of a rule. A beneficial site k' is a site other than home site (a site where the rule is allocated by the nonreplicated allocation algorithm) where the cost of allocating one copy of the rule and executing it at this site for specified frequency $f_{ik'}$ is less than the executing it at home site for the same frequency.

As we have discussed in section 2.2, the introduction of replication of rules in a distributed deductive database may lead to important advantages both from the viewpoint of performance and reliability of the system. However, there is price to be paid since replication leads to an increase in the cost of rulebase management due to the need for maintaining the consistency of redundant copies of the rules.

Keeping replicated rules at multiple sites results in performance gains due to the fact that any copy of a replicated rule can be used for an execution. A higher level of knowledge availability can be achieved by storing multiple copies of rules. The improved reliability

is obviously due to the availability of several redundant copies of the same information. We assume that the DDedDBMS access planner which determines the execution site of a rule selects the best site among the sites where it is stored based on the minimization of access and transmission cost.

Rulebase distribution should reflect the cost and availability of storage at the different sites. There may be some sites which may not support much storage. However, we neglect this constraint in our algorithm.

A heuristic algorithm for progressively introducing redundancy by replication of rules is discussed here. It starts with a nonreplicated allocation. These allocation will be referred as home sites of the rules. We place a rule i at all sites k' where the cost of executing rule i at site k' with frequency $f_{ik'}$ and the cost of updation of this rule is less than the cost of accessing it for execution when stored elsewhere. The benefit of placing rule i at site k' is evaluated as follow:

$$benefit(i,k') = single(i,k) \cdot f_{ik'} + z_i \cdot f_{ik'} - single(i,k') \cdot f_{ik'} - C \cdot (copies(i)+1) \cdot f_i^u \quad (3)$$

where the constant C is a ratio of update cost to retrieval cost, $copies(i)$ gives total number of copies of rule i, and f_i^u is the update frequency of rule i. Recall that $single(i,k)$ is the cost of executing rule i at site k once as discussed in section 5. If $benefit(i,k')$ is positive, then we replicate rule i at site k'. The algorithm **Partial_Rule_Allocation** is described in figure 7. In this procedure, *copies* is an array such that $copies(i)$ gives total number of copies of rule i, *non_home* is a set such that $non\_home(i)$ gives a set of beneficial sites of rule i, and overall_cost gives the overall comm. cost for partially replicated rules.

The complexity of this algorithm is $O(mn)$, where m is the number of sites and n is the number of rules. This follows easily from the fact that we have choice of m sites for each rule.

Example

In our running example, rule r_3 will be replicated to site 1 because

$$\begin{aligned} benefit(3,1) &= 70*.5 + 50*.5 - 80*.5 - 2*2*1 \\ &= 60 - 44 = 16 \end{aligned}$$

is positive. Thus, site 1 will be the non_home site of rule r_3. Rules r_1 and r_2 will not be replicated to sites 2 and 1 respectively. Now the overall_cost will be overall_cost=total_cost-$benefit(3,2) = 190-16 = 174$.

7 Experiments

We have carried out a number of experiments to measure the performance of the proposed heuristic algorithm for nonreplicated allocation of rules. The algorithm has been designed for carrying out the allocation efficiently in situations involving hundreds or thousands of rules and tens of sites. The running time of Exhaustive Enumeration (EE) algorithm in such situations is expected to be unacceptable even though it produces an optimal allocation. This is due to the fact that the number of permutations to be examined are exponential in nature. The second objective in conducting our experiments was to measure the quality of the results given by our heuristic algorithm as compared to the optimal algorithm. In this section, we report our experimental results for these algorithms. They were coded in C and executed on the ICL/ICIM DRS 6000 machine (based on the RISC chip SPARC).

The performance of the various algorithms for a set of randomly chosen RB hierarchies is given in tables 1 and 2. Table 1 compares communication costs for the two algorithms, viz, optimal and heuristic. Each row represents one experiment. In each example, we decided on the number of rules and sites, but interrelationships between the rules as well as between rules

```
global data:
   input P, Q, A, F, T, Z: matrices;
         home:array; total_cost: real;
   Output copies: array; non_home: set;
         overall_cost: real;
procedure Partial_Rule_Allocation();
   overall_cost=total_cost;
   for i = 1 to n do
    {
     copies(i) = 1
     k = home(i)
     non_home(i) = { }
         /* initially, this set is empty; */
     for k' = 1 to m and k' ≠ k do
      {
       compute single(i, k) as per equation 1;
       compute benefit(i, k') as per equation 3;
       if benefit(i, k') > 0, then
        {
         non_home(i) = non_home(i) + {k'};
         copies(i) = copies(i) + 1;
         overall_cost=overall_cost-benefit(i, k');
        }
       endif
      }
     endfor
    }
   endfor
   print copies, non_home, overall_cost;
end_Partial_Rule_Allocation.
```

Figure 7: Algorithm for Partially Replicated Allocation of Rules

Table 1: Comparison between communication costs

Rules and Sites	Comm. cost of optimal Solution	Comm. cost of heuristic Solution
12 rules, 2 sites	66.52	66.52
34 rules, 3 sites	2032.80	2132.80
20 rules, 4 sites	802.00	929.00
40 rules, 5 sites	22240.42	28813.50
60 rules, 6 sites	52547.48	59778.50
100 rules, 7 sites	144021.30	165431.88
104 rules, 8 sites	151146.80	170418.30

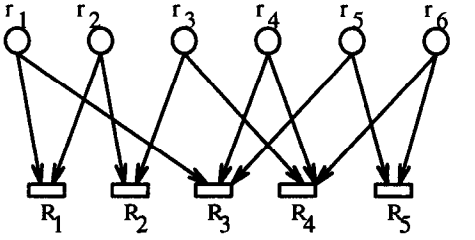
Table 2: Comparison of execution times

Rules and Sites	Running time of EE algo. (milliseconds)	Running time of heuristic algo. (milliseconds)
12 rules, 2 sites	28	12
34 rules, 3 sites	0.9×10^3	42
20 rules, 4 sites	8.3×10^2	35
40 rules, 5 sites	2.7×10^4	53
60 rules, 6 sites	1.8×10^6	74
100 rules, 7 sites	9.5×10^6	92
104 rules, 8 sites	3.8×10^7	96

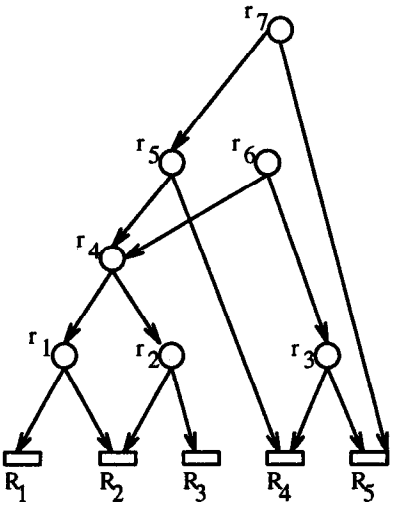
and relations were chosen randomly. We found that going beyond the 104 rules and 8 sites case resulted in very high execution time for the exhaustive enumeration algorithm.

Effect of Height

A rulebase may be large horizontally or vertically with reference to its hierarchical representation. That is, the RB may be 'broad' or 'deep' (as shown in figure 8) or something in between. Intuitively, one feels that real-world rulebases will have many more nodes at lower levels (i.e., at lower distances d), and the heights of RB hierarchies will be quite limited. We were interested in observing the performance of our heuristic algorithms for rulebases of different heights. It is simple to show that for h=1, the heuristic algorithm gives the optimal result. The proposed heuristic algorithm may deviate from optimum as height increases, since it considers only the largest connection weight edge. This is confirmed by our results in table 3, which compares communication costs of allocations obtained by our heuristic algorithm with optimal solution for a rulebase for 20 rules, 3 sites and 42 edges, arranged in hierarchies of heights 1 to 6. For lower heights, the heuristic algorithm gives allocations whose communication costs are same as the optimal algorithm.



(a) 'Broad' RB



(b) 'Deep' RB

Figure 8: RB characterization on height

Table 3: Comparison of Costs for different h

Height (h)	Comm. cost of optimal solution	Comm. cost of proposed heuristic solution
1	315.70	315.70
2	522.50	522.50
3	560.70	560.70
4	710.50	792.50
5	828.50	862.90
6	886.60	932.70

8 Conclusions

This paper has addressed the problem of allocating rules in a distributed deductive database system, where rulebase (a collection of rules) and database are shared across autonomous sites. The problem is relevant and challenging in applications involving large rulebases. We have identified communication cost as the primary consideration in allocation of rules. The optimal allocation is shown to be an NP-complete problem and solving it for a large rulebase is impractical.

We have proposed a heuristic algorithm for allocating the rules of a rulebase over a set of sites without replication. We represent the rulebase as a dependency graph with relations as leaf nodes and rules as non-leaf nodes, and edges representing usage of rules and database relations. Nodes are associated with either relation sizes or result sizes. Our rule allocation method converts a dependency graph into another graph of edge weights and then allocates the nodes (rules) on the basis of largest connection weight between them. The complexity of the heuristic algorithm is $O(n(n+m)^2)$, where n is the number of rules and m is the number of sites. We also proposed a method for replicated allocation based on the heuristic of all beneficial sites.

We have analyzed the performance and effectiveness of the heuristic algorithm for a set of rulebases, characterized in terms of number of nodes, sites and edges. We found that in most of the cases, allocations produced by it had total communication costs well within 20% of the optimal.

For performance analysis, we have attempted a characterization of large rulebases. Our characterizations are based on the premises that real-world rulebases will be more 'broad' than 'deep'. We have defined these characterizations with respect to the hierarchical structure of a rulebase. We then studied performance of the algorithm for rulebase hierarchies of varying heights.

For the variety of situations considered, our heuristic algorithm gives results that are reasonably close to optimal. The divergence of the heuristic from the optimal solution will be larger as height of RB increases. It is difficult to give a bound on divergence from the optimal, as well as convince ourselves by a formal argument that we will not be uselessly off from the optimum solution. Even for the experimental purpose, we have been unable to consider RB of more than (only) 104 rules due to prohibitive time taken by the optimal algorithm. It appears reasonable to expect satisfactory results for large RBs (expected to be more 'broad' than 'deep'). Another observation we wish to make in this connection is about critical dependence of results (optimal or not) on quantification of work load. There is bound to be some approximations in estimates of frequencies, expected result sizes, etc. In this 'fluid' situation, it seems better to be satisfied with a 'reasonably good' solution.

The future work includes the characterization of the real-world rulebases in greater depth. We are investigating more algorithms to reduce the overall communication cost even further for nonreplicated allocation.

Acknowledgment

We are grateful to S. Seshadri and A. A. Diwan for their helpful comments.

References

[Ape88] P.M.G. Apers. Data allocation in distributed database systems. *ACM Transactions on Database Systems*, 13(3):263–304, 1988.

[Bre77] M.A. Breuer. Min-cut placement. *Journal of Design Automation and Fault-tolerant Computation*, 1:343–362, 1977.

[CGT90] S. Ceri, G. Gottlob, and L. Tanca. *Logic Programming and Databases.* Springer-Verlag Berlin Heidelberg New York, 1990.

[CNW83] S. Ceri, S. B. Navathe, and G. Wiederhold. Distribution design of logical database schemas. *IEEE Transaction on Software Engineering*, 9(4):487–503, 1983.

[CP86] S. Ceri and G. Pelagatti. *Distributed Databases: Principles and Systems.* McGraw-Hill Book Co., 1986.

[CR91] D.A. Carlson and S. Ram. An architecture for distributed knowledge base systems. *DATA BASE*, 22(1):11–21, 1991.

[DJP+92] E. Dahlhaus, D.S. Johnson, C.H. Papadimitriou, P.D. Seymour, and Yannakakis M. The complexity of multiway cuts. In *ACM STOC*, pages 241–251, 1992.

[GJ79] M.R. Garey and D.S Johnson. *Computers and Intractability: A Guide to the theory of NP-Completeness.* W.H. Freeman and Company NewYork, 1979.

[HBH94] Cao Hua, D.A. Bell, and M.E.C. Hull. Architecture of a LAN-based parallel deductive database system - PDDS. In

Proc. 5th Int. Workshop on Next Generation Database Systems, Hongkong, Febuary 1994.

[JF90] R.J.K. Jacob and J.N. Froscher. A software engineering methodology for rule-based systems. *IEEE Transactions on Knowledge and Data Engineering*, 2(2):173–189, June 1990.

[Kri84] B. Krishnamurthy. An improved min-cut algorithm for partitioning vlsi networks. *IEEE Transactions on Computers*, 33(5):438–446, 1984.

[Li88] Y.P. Li. DKM: A distributed knowledge representation framework. In *Proc. 2nd Int. Conf. on Expert Database*, pages 313–331, 1988.

[Mos92] Chris Moss. Commercial applications of large Prolog knowledge bases. In *LNCS, Number 567*, pages 32–40, 1992.

[MS93a] Mukesh K. Mohania and N.L. Sarda. Allocation of rules in distributed deductive databases: NP-completeness and a heuristic approach. Technical Report TR-131-93, Dept. of Computer Science and Engg., IIT Bombay, November 1993.

[MS93b] Mukesh K. Mohania and N.L. Sarda. Rule allocation in distributed deductive database systems. Technical Report TR-122-93, Dept. of Computer Science and Engg., IIT Bombay, October 1993.

[MS93c] Mukesh K. Mohania and N.L. Sarda. An architecture for a distributed deductive database system. In *Proc. IEEE TENCON'93, Volume 1*, pages 196–200, Beijing, China, 1993.

[PS82] C.H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Networks and Matroids*. Prentice-Hall Inc., Englewood Cliffs, NJ, 1982.

[RB92] Tzvi Raz and N.A. Botten. The knowledge base partitioning problem: Mathematical formulation and heuristic clustering. *Journal of Data & Knowledge Engineering*, 8(4):329–337, 1992.

[RW79] C.V. Ramamoorthy and B.W. Wah. The placement of relations in a distributed relational database. In *Proc. 1st Int. Conf. on Distributed Computing Systems*, 1979.

[San89] L.A. Sanchis. Multiple-way network partitioning. *IEEE Transactions on Computers*, 38(1):62–81, Jan. 1989.

[SRN90] Timos K. Sellis, N. Roussopoulos, and Raymond T. Ng. Efficient compilation of large rule bases using logical access paths. Technical Report UMIACS-TR-90-8, University of Maryland, College Park, January 1990.

[SV91] H. Saran and V. V. Vazirani. Finding k-cuts within twice the optimal. In *Proc. of the 32nd Annual Symp. on Foundations of Computer Science, IEEE Computer Society*, pages 743–751, 1991.