1994

Heuristics for query optimization in distributed database systems.

Hung Kai (George). Mak
University of Windsor

HEURISTICS FOR QUERY OPTIMIZATION IN DISTRIBUTED DATABASE SYSTEMS

by

HUNG KAI (GEORGE) MAK

A Thesis
Submitted to the Faculty of Graduate Studies and Research
through the School of Computer Science in Partial
Fulfillment of the Requirements for the Degree of
Master of Science at the
University of Windsor

Windsor. Ontario, Canada
1993

ISBN 0-315-93290-2

Abstract

The technology of distributed databases (DDB) is based on two other technologies which have developed a sufficiently solid foundation during the seventies : ***computer networks technology*** and ***database technology.***

One of the main difficulties in distributed database systems is to select an execution strategy that minimizes resource consumption. Some optimization strategies such as AHY (Apers-Hevner-Yao) Algorithms only focus on reducing the amount of transmissions. They assume that the cost to transmit the packets from one site to another site is the same. However, this is not true in the real world, since the cost of transmission is dependent on the network load situation. Therefore, it is possible to develop some heuristics which consider the network load as well.

The objective of this thesis is to develop some heuristics which will take the network load into account and to compare the result with the AHY Algorithms.

To my Lord Jesus Christ
my father Yiu Sik,
my mother Shiu Ping &
my brother Hung Kam.

Acknowledgments

I would like to express my sincere thanks and appreciation to **Dr. Bandyopadhyay** and **Dr. Morrissey** for their support and guidance throughout the progress of this thesis. I would like to thank **Dr. Channen** who gave me a lot of comments and suggestions on my simulation program. I would also like to thank **Dr. Soltis** my external reader.

I am appreciative my family 's support of my study. I would also like to thank all the brothers and sisters who pray for me on this thesis. THANK YOU ALL VERY MUCH.

TABLE OF CONTENTS

List of Figures

List of Tables

CHAPTER 1 INTRODUCTION

In the past, organizing data in a large and expensive centralized computer was extensively used in data processing. In recent years, with the development of database technology, computer networks and workstations, distributed database systems have become predominant.

With today's VLSI technology, the price/performance ratio has changed to favor multiple low-performance processors rather than single high-performance processors. Furthermore communication costs have fallen dramatically in the past few years and Local Area Networks (LANs) are now cost effective. Faster, extensible and reliable systems are needed by users. Distributed databases can achieve this goal, since, as a rule of thumb, the data is stored close to the point where it is most frequently used and this will increase the efficiency of processing.

The technology of distributed databases (DDB) is based on two technologies which have developed sufficiently solid foundations during the seventies : ***computer networks technology*** and ***database technology.*** A distributed database system consists of a number of sites, each of which contains a local database system. Therefore, the local transactions can be accessed at any of the sites. The global transactions can access data in several sites by using a computer network.

One of the main difficulties in distributed database systems is to select an execution strategy that minimizes resource consumption, since each equivalent execution strategy may consume different computer resources. The procedure used is called a **distributed**

access plan[1]. An **optimizer** program generates the distributed access plan. It is convenient to divide the optimizer into two categories :

1. **Global optimization**
2. **Local optimization**

Global optimization consists of determining which data must be accessed at which sites and identifing which data files must consequently be transmitted between sites. Therefore, *communication costs* and *disk access costs* are the main optimization parameters. This is the main concern of this thesis.

Local optimization consists of deciding how to perform the local database accesses at each site. This is the traditional problem in nondistributed databases and will not be discussed any further.

1.1 THE MOTIVATION

Some Query Optimization Algorithms such as the AHY Algorithm, assume that the cost to send the same amount of attributes or relations from one site to another is independent of the address of the source and the destination node. Therefore, they are only concerned with minimizing the amount of transmission in order to reduce the cost. The routing used to transmit the relation from the source to the destination is ignored in such algorithms. In some cases, even though the number of packets communicated is reduced, the cost to transfer that amount of packets may be higher than the cost to transfer more packets through another routing in the network. Therefore, it is possible to develop some Query Optimization Algorithm in order to take both the *network load* and the *size of transmission* into account.

[1] The term *access path* refers to the data structures and the algorithms that are used to access the data.

1.2 THE THESIS STATEMENT

Query optimization heuristics that take into account network load are better than the AHY Algorithm.

1.3 THE OBJECTIVES AND SCOPE OF THE THESIS WORK

The objective of this thesis is to develop some Query Optimization Heuristics which take both the network load and the size of transmission into account in order to reduce the *Total Cost* or *Response Time*. The definition of Total Cost and Response Time is given in Section 2.6.1.

The scope of this thesis is to test the heuristics described here and compare them with the traditional Query Optimization Algorithm such as AHY Algorithm. The tests on those algorithms will be done using a simulation of a hypercube network with up to 64 sites.

1.4 THE ORGANIZATION OF THE THESIS

Chapter 1 of this thesis discusses the objectives of this thesis and the thesis statement. Chapter 2 discusses the basic concepts in this area. Chapter 3 describes the total cost heuristic and illustrates an example. Chapter 4 describes the response time heuristic and illustrates an example. Chapter 5 shows the results and the comparisons. Chapter 6 contains the conclusions, the findings, the recommendations and the future work of this thesis.

CHAPTER 2 BACKGROUND

2.1 WHAT IS A DISTRIBUTED DATABASE SYSTEM ?

A **distributed database (DDB)** is *a collection of multiple, logically interrelated databases distributed over a computer network* [20]. In other words, a distributed database is a collection of data which is distributed over different computers of a computer network. Each site of the network has an autonomous processing capability and can perform local applications. Each site also participates in the execution of at least one global application, which requires accessing data at several sites using a communication subsystem [5].

Figure 1 shows a traditional central database on a network. The database is centrally managed by one computer system and all the requests are routed to that site. None of the sites can execute a local application except the master site.

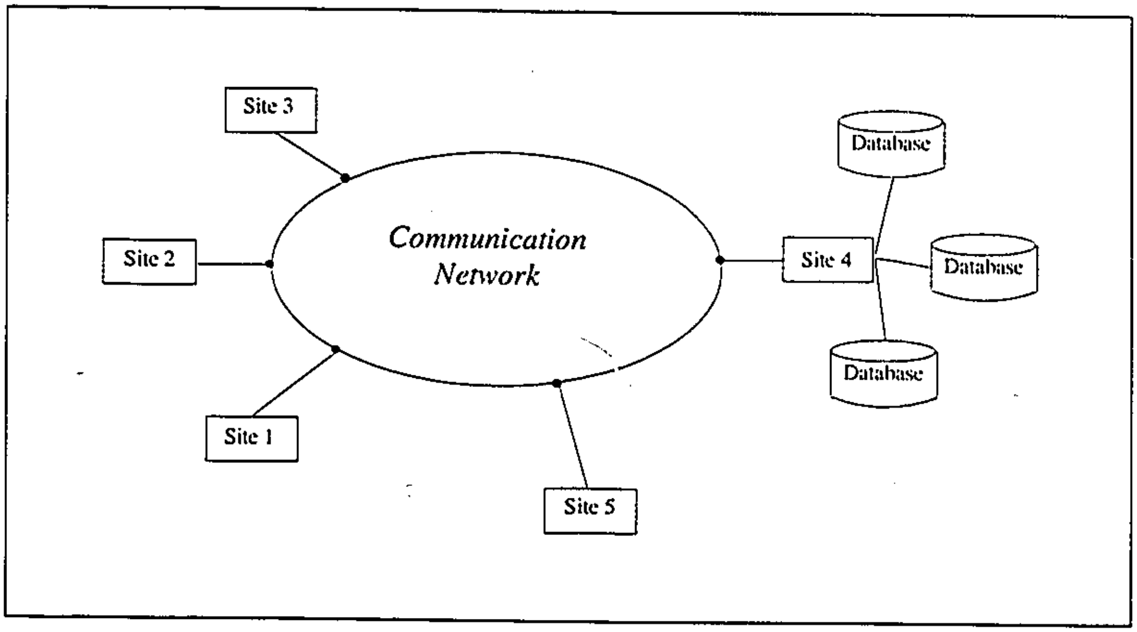


Figure 1 Central Database on a Network [21]

A **Distributed Database Management System (DDBMS)** is defined as *the software system that permits the management of a DDBS and makes the distribution transparent to the users* [20].

A **Distributed database system (DDBS)** refers to the combination of the DDB and the DDBMS.

Consider a bank system as an example. A local application is a debit or a credit application performed on an account stored at the same branch at which the application is requested. A typical global application is a transfer of funds from one branch to another.

A distributed database can be implemented in a large geographically dispersed network or in a small local network. These two configurations are shown in Figure 2 and Figure 3 . Notice that even though the physical structure of the connections has changed, the characteristic aspects of the architecture have remained the same. *That is, each computer involves its own database. Both local and global applications still apply on those systems.*

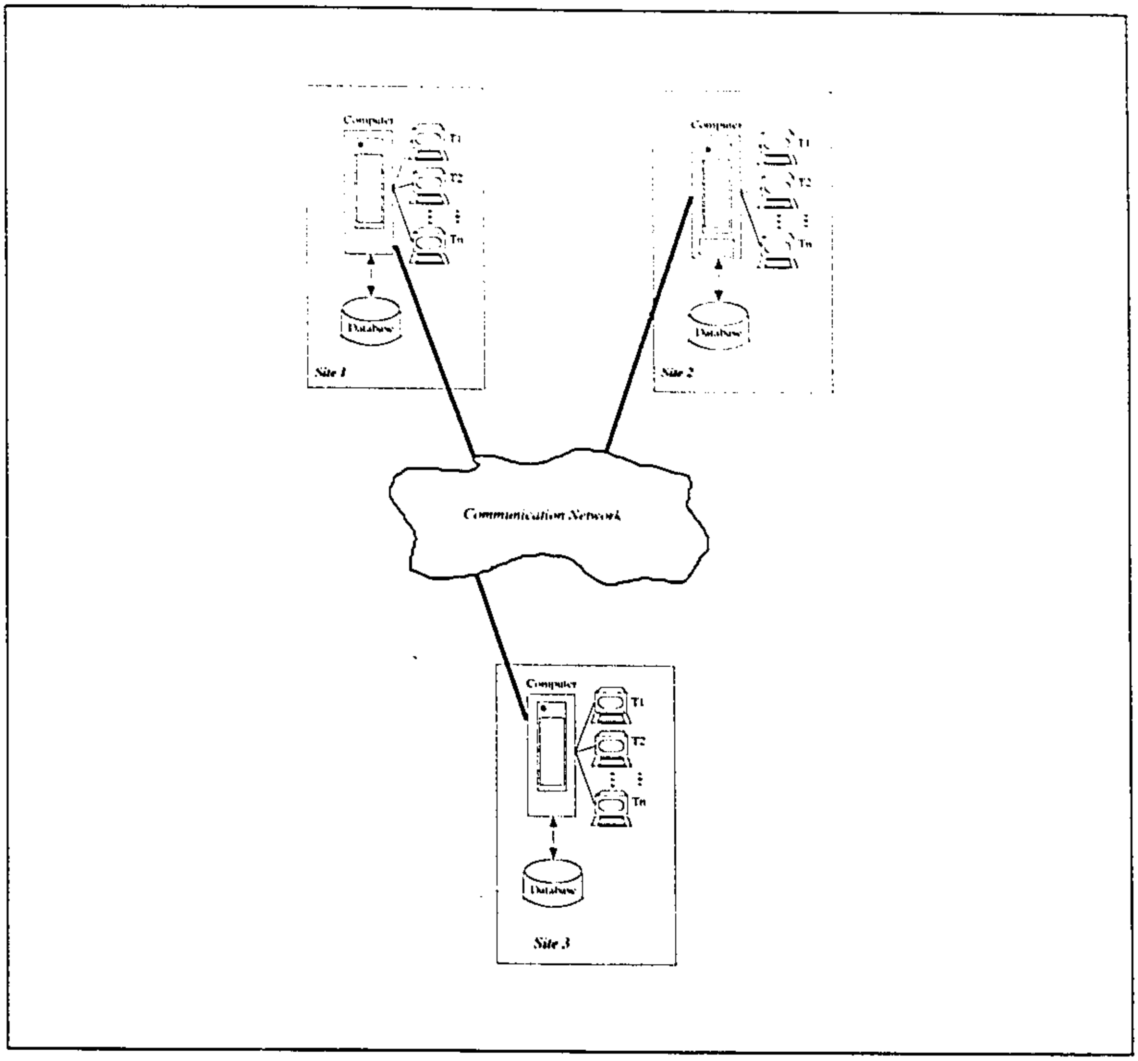


Figure 2 A Distributed Database System

2.2 THE OBJECTIVES OF DISTRIBUTED DATABASE SYSTEMS

Distributed database technology extends the concept of data independence to environments where *data is distributed* and *replicated over a number of nodes* connected by a network. Therefore, data independence is a fundamental form of transparency[2]. The aim is that database users would see a logically integrated, single image database even

[2] *Transparency* in distributed DBMS refers to separation of the higher-level semantics of a system from lower-level implementation issues. In other words, a transparent system "hides" the implementation details from users.

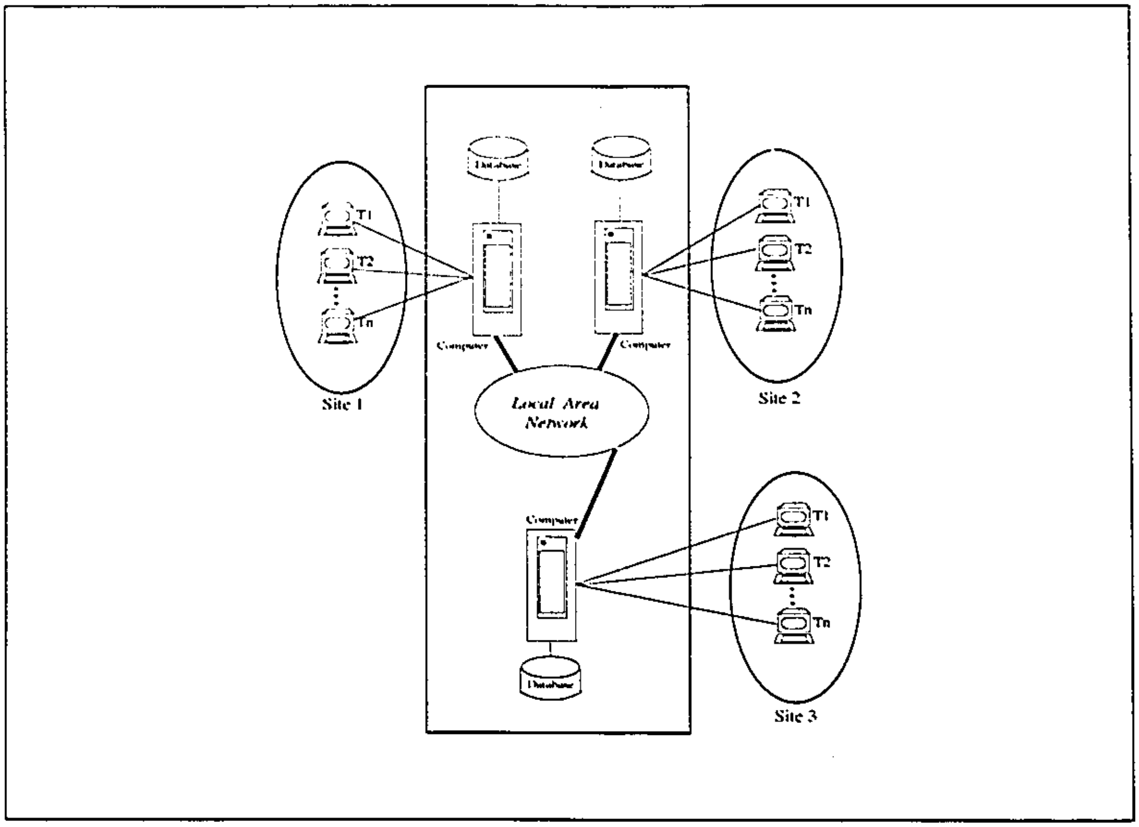


Figure 3 A Distributed Database System with LAN

though it may be physically distributed, enabling them to access the distributed database as if it was a centralized one.

Therefore, a **fundamental principle** is that a distributed system should look exactly like a nondistributed system [10].

2.2.1 Location Transparency

It is desirable that the user be unaware of the physical location of data items. This is called location transparency. It is achieved by creating a set of alternative names or aliases for each user. A user may thus refer to data items by simple names that are translated by the system to complete names. With aliases, the user can be unaware of the physical location of a data item.

2.2.2 Fragmentation Transparency

Rather than distributing relations, it is quite common to divide them into sub-relations, called **fragments**, which are then distributed. Similar to the location transparency, a user should not be required to know how a data item is fragmented and this is called fragmentation transparency.

There are several kinds of fragmentation and they are listed as follows :

1. Horizontal Fragmentation : If the relation r is fragmented, r is divided into a number of fragments $r_1, r_2, \ldots, r_n$. For horizontal fragmentation, a fragment may be defined as a *selection (σ)* on the global relation r [15].
2. Vertical Fragmentation : It splits the relation by decomposing the scheme of a relation using projection.
3. Mixed Fragmentation : In mixed fragmentation, the relation is divided into a number of horizontally or vertically fragmented relations.

2.2.3 Replication Transparency

A distributed system may have several identical replicas (copies) of a relation stored in different sites. A user should not be required to know how many replicas are stored in the system. This is called **replication transparency**. **Full replication** means that a copy of a relation is stored in every site.

2.3 TRADE-OFFS IN DISTRIBUTED DATABASE SYSTEMS

2.3.1 Advantages of DDBS

There are several advantages of DDBS. First of all, each site is able to retain a degree of control or direct control over data stored locally, that is, site autonomy, resulting in increased data integrity and more efficient data processing.

The failure of a site does not necessarily imply that the system goes down. The remaining sites may be able to continue operating. When a failed site is repaired, some procedures must be available to re-install it back into the system. However, the distributed processing capability of different sites does not guarantee a higher overall reliability of the system, but it ensures a **graceful degradation** property.

It is possible to execute a query in parallel when it involves data at several sites. There are two types of parallelism : **inter-query parallelism** which results from the ability to execute multiple queries at the same time; and **intra-query parallelism** which is achieved by breaking up a single query into a number of subqueries each of which is executed at a different site.

2.3.2 Disadvantage of DDBS

The primary disadvantage of distributed database systems is the added complexity required to ensure that the system operates properly. This induces some drawbacks to the system.

To implement a distributed database system requires additional and more complex software. Thus a higher cost should be expected. With a DDBS, we have not only the problems of a centralized environment, but also a new set of problems. For example, the distribution of control can cause problems of synchronization. Security is also more complicated than in a centralized system because we have to maintain adequate security over the computer networks. It is more difficult to ensure the correctness of algorithms when the system operates in parallel. Some additional computation is required to exchange messages between sites. The cost of personnel required to support the system will be increased.

2.4 DISTRIBUTED DBMS ARCHITECTURES

The main components of the DDBMS at each site are ***a data communication component (DC), a data dictionary (DD), a database management component (DB) and a distributed database component (DDB).*** Usually, the term "DBMS" refers to the set of components which serve and manage a nondistributed database. That is DB, DC and DD components. The components of a DDBMS are shown in Figure 4.

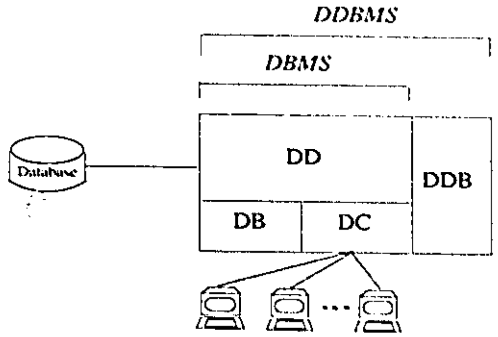


Figure 4 Components of A DDBMS

The DDBMS has several functions. Firstly, it manages the global data dictionary to store information about distributed data. Secondly, it defines distributed data definition. Thirdly, it provides distributed semantic data control. Fourthly, it offers distributed query processing, including distributed query optimization. Finally, it provides distributed transaction management such as distributed concurrency control, recovery and commit protocols.

2.4.1 Network Topology

The sites in the system can be connected in a number of different ways, such as star, hypercube, ring, mesh, tree and so on. The main differences between them are as follows :

— **Installation Cost :** The cost to install the physical link between the sites.

— **Communication Cost :** The cost in time and money to send a message between two sites.

— **Reliability :** The frequency with which a link or a site may fail.

— **Availability :** The portion of the data that can be accessed even with the failure of some links or sites.

Various query optimization strategies exist for each network topology. A detailed description of an algorithm used for a star network can be found in [7] and [14].

2.4.2 Architectural Models for Distributed DBMS

2.4.2.1 The Implementation Alternatives on DBMS

In [21], the possible ways in which multiple databases may be put together for sharing by multiple DBMSs is discussed. A classification (Figure 5) is used that organizes the systems with respect to (1) the autonomy of local systems, (2) their distribution and (3) their heterogeneity.

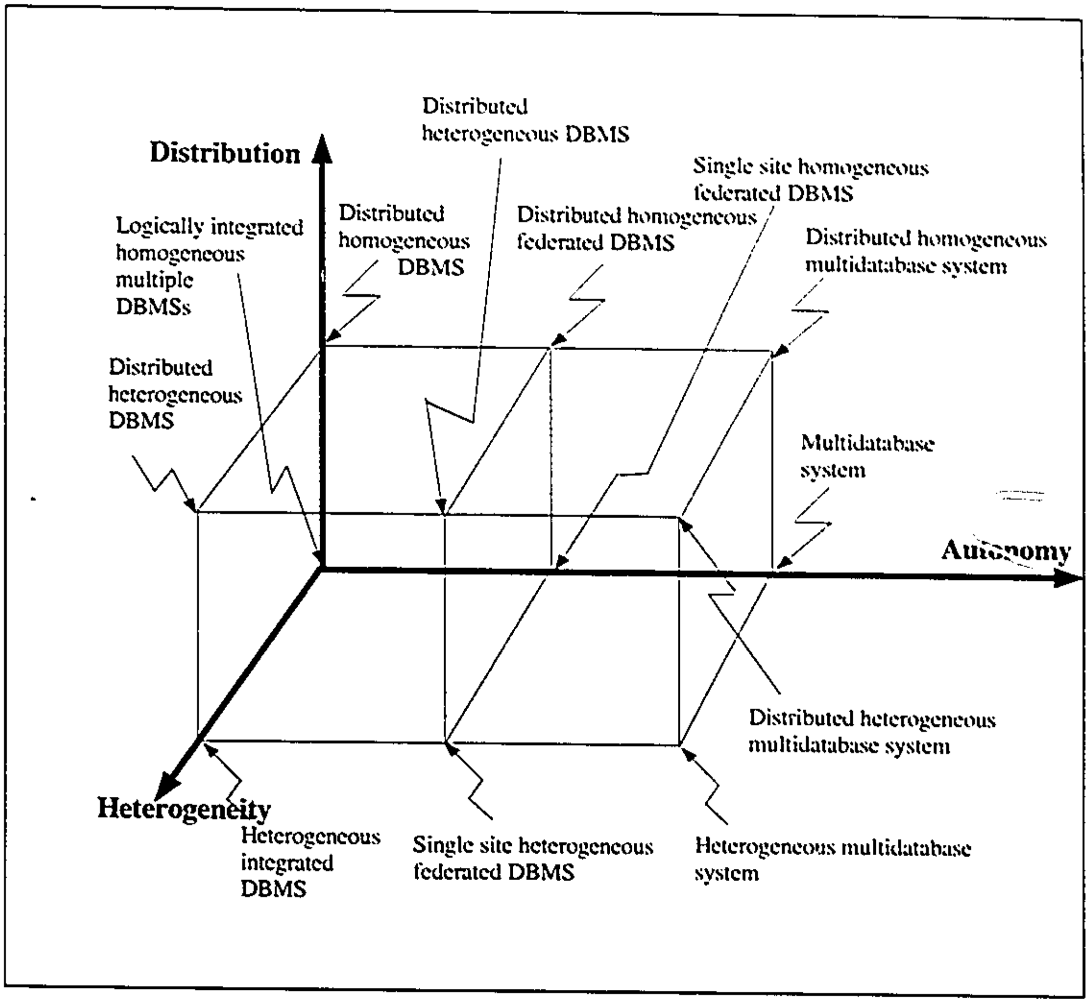

Distribution
Autonomy
Heterogeneity
Distributed heterogeneous DBMS
Single site homogeneous federated DBMS
Logically integrated homogeneous multiple DBMSs
Distributed homogeneous DBMS
Distributed homogeneous federated DBMS
Distributed homogeneous multidatabase system
Distributed heterogeneous DBMS
Multidatabase system
Distributed heterogeneous multidatabase system
Heterogeneous integrated DBMS
Single site heterogeneous federated DBMS
Heterogeneous multidatabase system

Figure 5 DBMS Implementation Alternatives [20]

Autonomy refers to the distribution of control and indicates the degree to which each DBMS can operate independently. **Distribution** refers to the fact that the data may be physically distributed at many sites or just one. **Heterogeneity** refers to the fact that the DBMS may be homogeneous or heterogeneous.

2.4.2.2 Distributed DBMS Architecture A distributed database can be implemented in many different ways depending on objectives and design choices. Similar to centralized databases, the three-level **ANSI/SPARC** architecture can be extended for distributed databases. Thus, data independence is easily achieved. The reference architecture (Figure 6) is frequently used (See [21] and [26]).

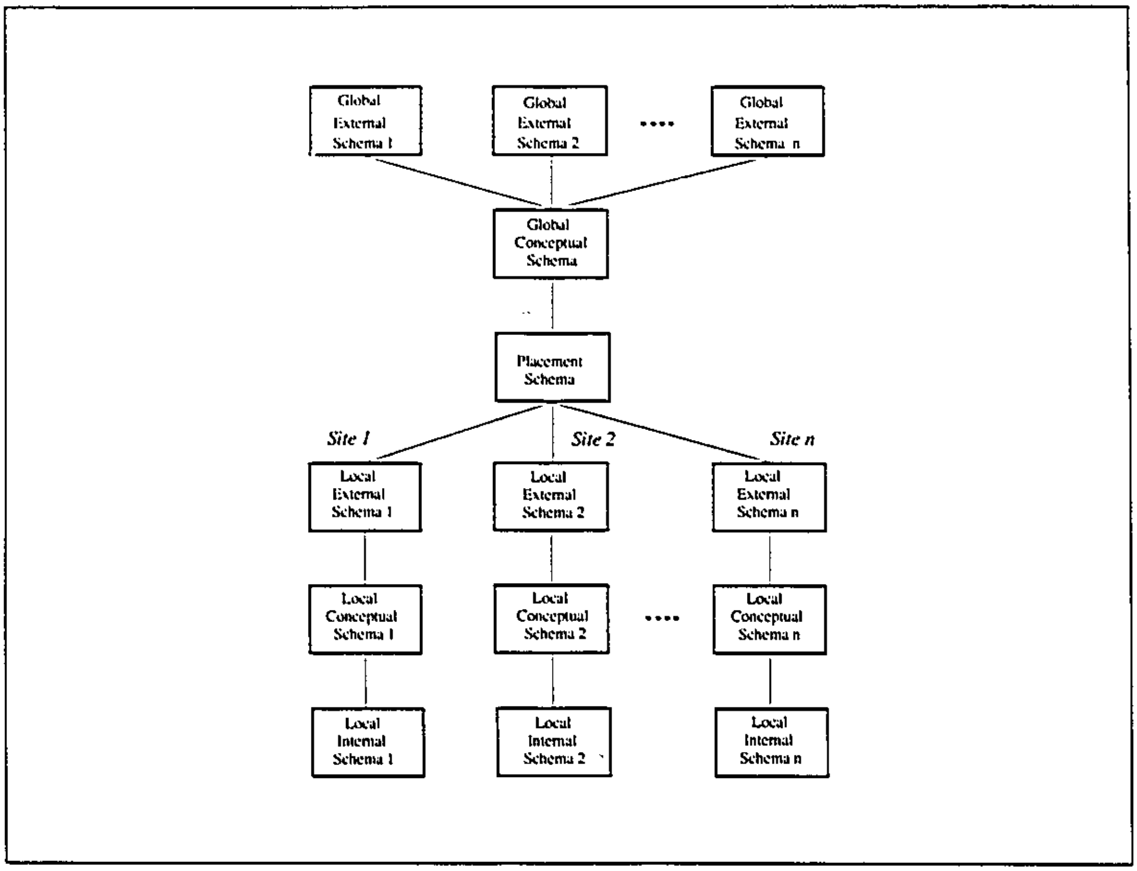


Figure 6 Reference Architecture for Distributed Databases [21] & [26]

2.5 OVERVIEW OF QUERY PROCESSING

Given a query, there are generally many different kinds of strategies for computing the answer. The system should transform the query, which is entered by the user, to an equivalent query which can be computed more efficiently. Generating this improved strategy for processing a query is called **query optimization.**

In other words, the objective of query optimization is to decide on a strategy for executing each query over the network in the most cost-effective way, that is, ***optimal ordering***. During this processing, a high-level query on a distributed database will be transformed into an efficient execution strategy expressed in low-level queries on local databases.

2.5.1 Objectives of Query Processing

A **query processor** is a procedure which is part of a DBMS. Its function is to construct the answer to the query which is given by the user.

Usually, the query will be decomposed into the relational algebra. The output language is some internal form of the relational algebra with some communication parameters. The query processing will typically consist of a global optimization, followed by a local optimization at each affected site (See Figure 7).

2.5.2 Functional Layers of Query Processing

The role of a distributed query processor is to translate a "high-level query"[3] on a distributed database into a sequence of efficient "low-lèvel database operations"[4] and execute them on the local databases. There are two important aspects of this translation :

1. The query must be translated correctly in order to obtain the correct result.
2. The strategy must be "optimal".

It is very difficult to achieve these two aspects together. It may be easier to decompose this problem into several components. In Figure 7 a generic layering schema for query processing is shown where each layer solves a *well-defined subproblem*.

[3] For example, a relational calculus query.

[4] For example, a relational algebra.

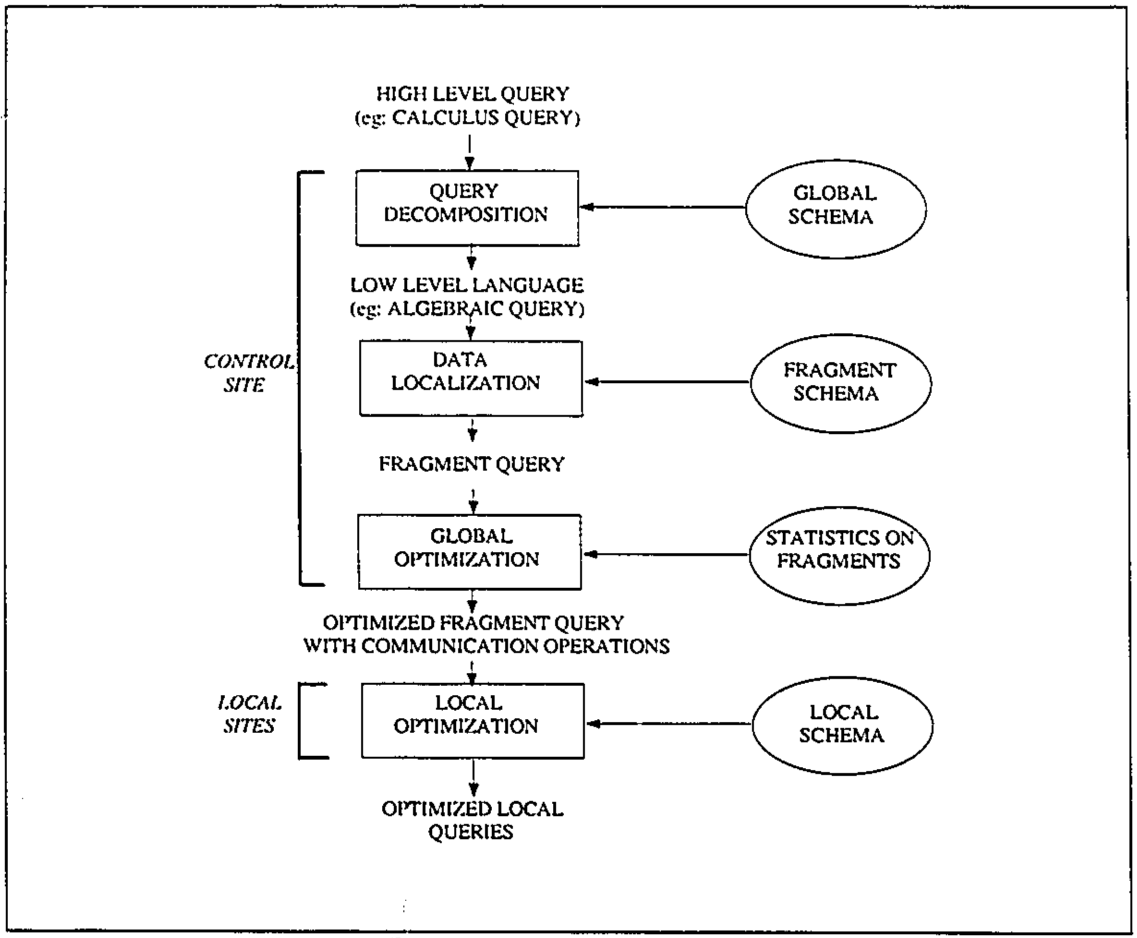


Figure 7 Generic Layering Scheme for Distributed Query Processing [21]

2.5.2.1 Query Decomposition The traditional way to decompose a query into a "better" algebraic specification is to start with an initial algebraic query and transform it in order to find a "good" one.

Query decomposition can be decomposed into four steps. Firstly, the calculus query is rewritten in a *normalized* form which involves the manipulation of the query quantifiers and of the query qualification by applying logical operator priority. Secondly, the normalized query is analyzed semantically so that incorrect queries are detected and rejected as soon as possible. Thirdly, the correct query which is expressed in relational calculus is simplified, for example, eliminating the redundant predicates in a query.

Finally, the calculus query is restructured as an algebraic query.

2.5.2.2 Data Localization This layer determines which fragments are involved in the query and transforms the distributed query into a *fragment query*[5]. The main function of data localization is to *localize the data involved in the query*.

A distributed relation can be reconstructed by applying the fragmentation rules, and then deriving a program, called a *localization program*, of relational algebra operations which then acts on fragments. This process involves two steps. Firstly, the distributed query is mapped into a fragment query by substituting each distributed relation by its **materialization program** which is a reconstruction program. Secondly, the fragment query is simplified and restructured to produce another "good" query.

In short, a query expressed on database fragments will be stored at different sites. In addition, the result generated is far from optimal because information regarding fragments is not utilized.

2.5.2.3 Global Query Optimization The goal of global query optimization is to find an execution strategy for the query which is close to optimal. The major decisions that are made by the global query optimization are as follows :

1. The order of the operations.
2. The site to take place the operations.

The efficiency of query optimization depends on the database **statistics.** The statistics can help to decide which operations should be performed first. For example, algorithm serial, parallel and general which will be discussed in section 7.3.4 are highly dependent on these statistics. In some cases, the size of the intermediate relations must be estimated

[5] Relations are fragmented and stored in disjoint subsets, called fragments, each being stored at a different site.

based on the statistics, and this data will be updated periodically in order to maintain accuracy.

A typical simplification made by distributed DBMSs is to consider communication costs as the most significant factor to minimize. This is valid for Wide Area Networks (WAN) where the limited bandwidth makes communication costs much more expensive than the local processing costs.

There are many proposals for query optimization. For example, a semijoin operation is used to reduce the size of fragments in order to reduce the communication cost. However, more recent techniques, which consider local processing costs and communication costs, do not use semijoins because it might increase local processing costs. This is the most important topic of this survey and will be discussed in more detail later.

2.5.2.4 Local Query Optimization This layer is performed by all sites having fragments involved in the query. The advantage is that local query optimization can be carried out independently using the known methods for centralized systems. This survey is not concerned with this topic.

2.6 OPTIMIZATION OF DISTRIBUTED QUERIES

Finding the optimal solution of the query optimization problem is computationally intractable. Therefore, a popular way of reducing the cost of search is to use an **heuristic solution.** An important heuristic solution in distributed systems is using a *semijoin* instead of a *join* to minimize the communication cost. In [20], it is claimed that "More important, a different search strategy should be used depending on the kind of query (simple vs. complex) and the application requirements (ad hoc vs. repetitive)".

For centralized DBMSs, the primary parameters for measuring the cost of a particular strategy are the **I/O cost**[6] and the **CPU cost**[7]. In a distributed system, there are some other issues which should be take into account such as the **communication costs**.

2.6.1 Inputs to Query Optimization

The cost of a distributed execution strategy can be expressed with respect to either the total cost or the response time. The total cost is the sum of all cost components, while the response time is the elapsed time from the initiation to the completion of the query. The general formula for determining these costs will be discussed in the following section.

2.6.1.1 Cost Model The cost function is usually defined in terms of **time units**. They refer to computing resources such as disk space, disk I/O, buffer space, CPU cost, communication cost, and so on. There are two very important performance variables in query optimization, total cost and response time.

Total Cost : The Total Cost is a good measure of resource consumption. It is the sum of all the time incurred in processing the operations of the query at various sites and in intersite communication. A general formula for determining the total cost is as follows [21] :

$$Total\ Cost = C_{CPU} * \#insts + C_{I/O} * \#I/Os + C_{MSG} * \#msgs + C_{TR} * \#bytes$$

C_{CPU} : is the cost of a CPU instruction.

$C_{I/O}$: is the cost of a disk I/O.

C_{MSG}: is the fixed Cost of initiating & receiving a message.

[6] Disk access

[7] Operations on data in main memory.

C_{TR} : is the cost of transmitting a data unit[8] from one site to another. A typical assumption is that C_{TR} is a constant.

Response Time : This is the elapsed time for query execution. Since the operations can be executed in parallel at different sites, the response time of a query may be significantly less than its total cost. A general formula for determining the response time is as follows [21] :

$$Response\ Time = C_{CPU}*seq.\#insts + C_{I/O}*seq.\#I/Os + C_{MSG}*seq.\#msgs + C_{TR}*seq.\#bytes$$

Where seq.#x, in which x can be

1. instructions (insts)
2. I/O
3. messages (msgs)
4. bytes

And they choose the **maximum** number of x which must be done **sequentially** for the execution of the query.

Example : This example illustrates the difference between total cost and response time (Figure 8). We want to compute the answer to a query at site 3 with data transfer from site 1 and site 2.

Assumptions :

(1) We only consider the communication costs.

(2) C_{MSG} and C_{TR} are expressed in *time units.*

[8] The data unit is given here in terms of the number of bytes which is the sum of the sizes of all messages, but can be in different units (eg: packets).

Solutions :

Total Cost = $(C_{MSG} + C_{TR} * x) + (C_{MSG} + C_{TR} * y)$

$= 2 * C_{MSG} + C_{TR} * (x + y)$

Response Time = **Max**$\{ C_{MSG} + C_{TR} * x, C_{MSG} + C_{TR} * y \}$

since the transfers can be done in *parallel.*

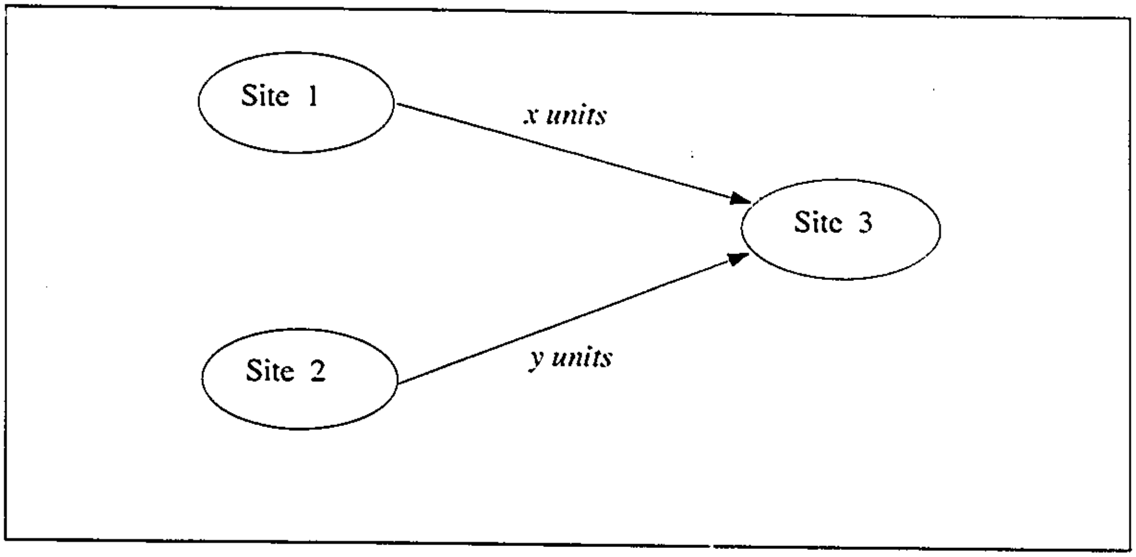


Figure 8 Example of Data Transfers for a Query

2.6.1.2 Database Statistics The main factor affecting the performance of an execution strategy is the **size** of the intermediate relations that are produced during execution. Therefore, it is necessary to ***estimate*** the size of the data transfers. This estimation [21] is based on statistical information about :

1. The base relations

2. Formulas[9] to predict the cardinalities of the results of the relational operations.

[9] Formulas to predict the resultant size of selection, cartesian product, join, semijoin, difference and so on (see [21] for details).

One important piece of statistical data is called the **join selectivity factor** for some pairs of relations, that is the proportion of tuples participating in the join. **SF$_J$(R, S)** denotes the join selectivity factor of relation R and S. It is a real value between 0 and 1 :

$$SF_J(R,S) = \frac{card(R \bowtie S)}{card(R) * card(S)}$$

The selectivity factor of other operations is discussed in detail in [21].

2.6.2 Two Approaches to Order Joins in Query Optimization

2.6.2.1 Join Ordering This approach tries to optimize the ordering of joins directly. It is important since joins between fragments may increase the communication cost. **Distributed INGRES** and **R*** algorithms both use join rather than semijoin. An example with two relations is shown in Figure 9. If the size of X is greater then the size of Y, then the relation in Site 1 will be transmitted to Site 2. Otherwise, the relation in Site 2 will be transmitted to Site 1.

Some other algorithms have been proposed using join as a reducer in distributed query processing, including [6], [9], [22]. In [1], [8] and [17] joins and fragmented relations in distributed database systems are discussed.

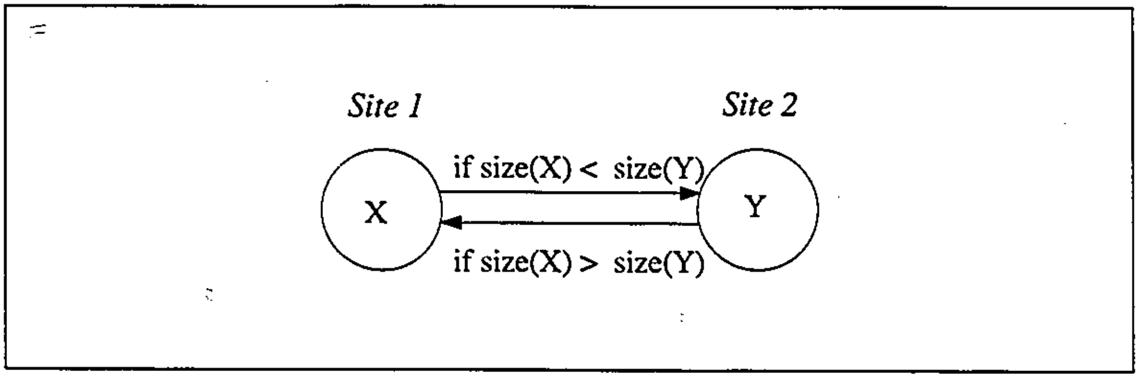


Figure 9 Transfer of Relation in a 2-way Join

2.6.2.2 Semijoin[10] The main disadvantage of the join approach is that *entire operand relations* must be transferred between sites. The semijoin acts as a size reducer in the same way as selection does. The use of semijoins is beneficial only if *the cost to produce the intermediate relation and send it to the other site is less than the cost of sending the whole relation and doing the actual join.* Then it can become a powerful *size reducer.* An example with two relations is shown in Figure 10 . In [3], [13] and [29] some semijoin algorithms are discussed. In [27] and [28] a One-Shot Fixed-Precision Semijoin is used to minimize the response time. The algorithms in [2] and [12] will be discussed in more detail in Section 7.3.4.

Given :

Site 2 requests a join with the relation X which is stored in Site 1.

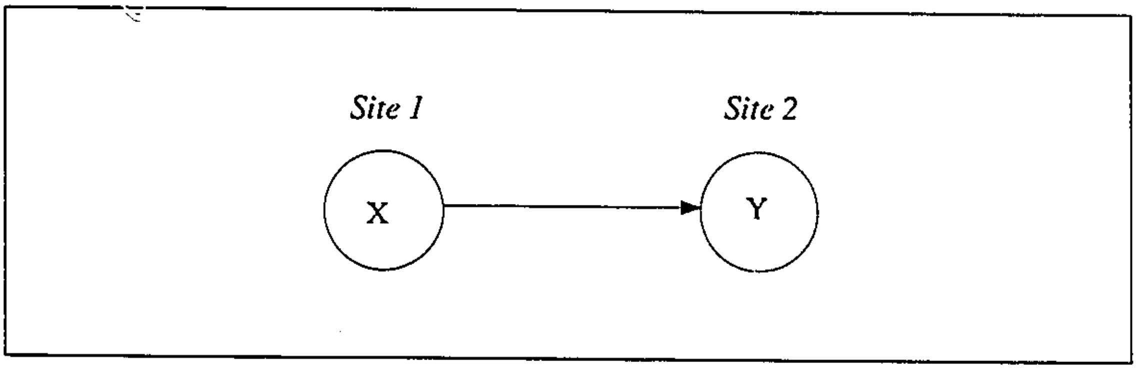


Figure 10 Transfer of Relation in a Binary Operation

Assumption : size(X) < size(Y)

Strategies 1 (Join-Based Algorithm) : $X \bowtie_A Y$

1. X → Site 2

2. Site 2 computes : $X \bowtie_A Y$

[10] Further details of the semijoin operation are given in Appendix F

Strategies 2 (Semijoin-Based Algorithm) : $(X \ltimes_A Y) \bowtie_A Y$

1. $\Pi_A(Y)$ → Site 1
2. Site 1 computes : $Temp.X = X \ltimes_A Y$
3. Temp.X → Site 2
4. Site 2 computes Temp.X $\bowtie_A$ Y

Conclusions :

Consider the communication cost of these two algorithms. The communication cost of the join-based algorithm only takes place in step 1, that is, in transferring X from site 1 to site 2. The communication cost of the semijoin-based algorithm takes place in steps 1 and 3. Therefore, the semijoin approach is better if

$$\text{size}(\Pi_A(Y)) + \text{size}(X \ltimes_A Y) < \text{size}(X)$$

The semijoin approach is better and it can act as a sufficient reducer if only a few of X participate in the join operation. The join approach is better if almost all tuples of X participate in the join, because the semijoin approach requires an additional *communication cost and local processing cost.* That is, the transfer of a projection on the join attribute.

Therefore, using semijoins might not be a good idea if the communication cost is not the dominant factor, as is the case with local area networks [21]. Since it may increase both **local processing cost** in that one extra project and join are required and the **communication cost** in that one extra transmission is required.

2.7 AHY (Apers-Hevner-Yao) ALGORITHMS

A family of algorithms which use *semijoins* to minimize the response time or total time was proposed in [2]. These methods, Algorithm Serial and Algorithm Parallel, are only applicable to **simple queries**. A simple query is one where after initial local processing, each relation in the query contains only *one common join attribute*. Therefore, a general query that contains multiple join attributes is decomposed into simple queries and then the algorithm can be applied to each of them. Figure 11 contains an example of a simple query and table 1 contains its statistics. There are three relations which are stored in three sites and the common join attribute is called "A".

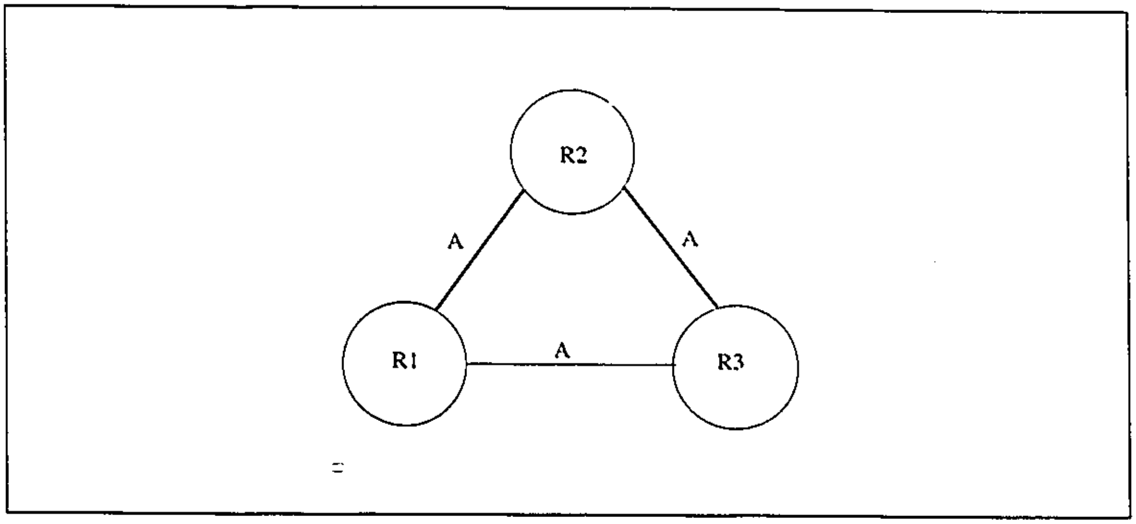


Figure 11 Example of a simple query

attribute	SF_{SJ}	size($\Pi_{attribute}$)
R_1.A	0.3	100
R_2.A	1.0	400
R_3.A	0.7	200

Table 1 Simple Query and Statistics

The steps in the AHY algorithm are as follows :

1. Apply local operations.
2. Decompose the query into simple queries for an optimal strategy using semijoin.
3. Integrate those decomposed queries in step 2 into an unique execution strategy.

In short, all relations are first reduced by semijoins and then sent to a unique result site to compute the result of the query.

Any query processing strategy can be *graphically* illustrated by a **schedule** [12]. A schedule is a tree in which the data transfers are represented by edges of length proportional to the transfer size [23]. AHY algorithms can be illustrated by using schedules. Figure 12 is an example of such a schedule. R1 and R3 send their fragments to R2 to perform the semijoin. After the semijoin operation is performed, the size of their fragments are reduced and finally those reduced fragments is transmitted to the result site.

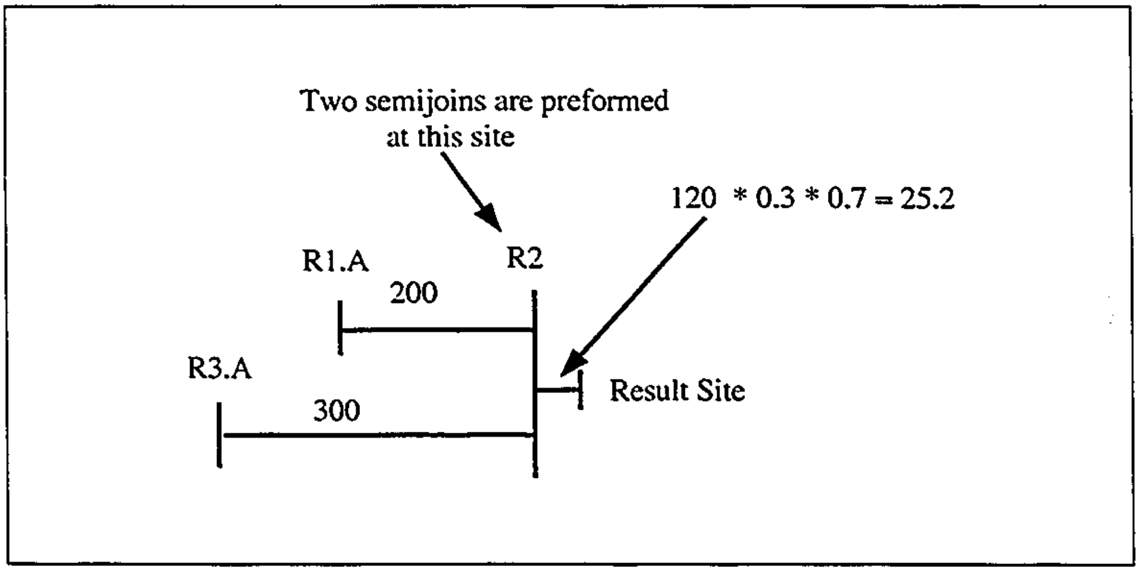


Figure 12 Example of a Schedule

The nodes and edges are the two components in this schedule. Their functions are as follows :

1. **Nodes :** The attributes or relations to transmit.
2. **Edges :** The size[11] of the data to transmit.

[11] The result of a semijoin = Size(R) * SF_{SJ}

(1) Algorithm Serial

The objective of the Algorithm Serial is to minimize the total amount of data transmitted in order to minimize the total cost. The algorithm tries to arrange the size of transmissions in ascending order. An example of the Algorithm Serial is shown in Figure 13 and the algorithm is given in Figure 14. Moreover, a more detailed algorithm is given in [21].

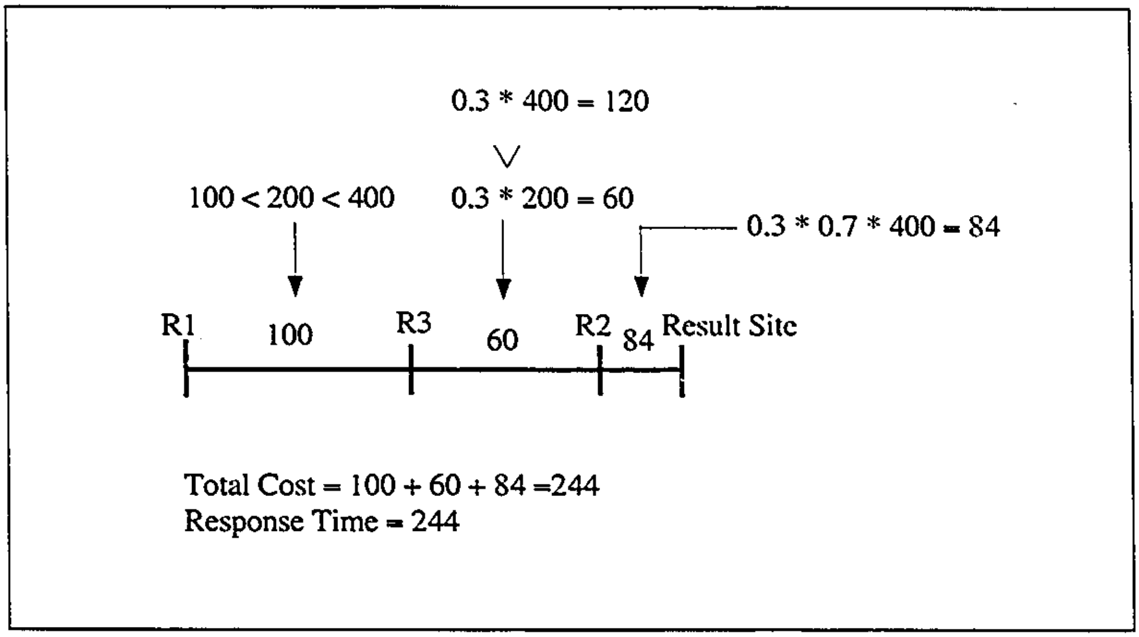


Figure 13 Example of Algorithm Serial

```
1. Order relations $R_i$ such that $s_1 \leq s_2 \leq \ldots \leq s_m$.
2. If no relations are at the result node, then select strategy :
   a. $R_1 \rightarrow R_2 \rightarrow \ldots \rightarrow R_n \rightarrow$ result node.
3. Or else if $R_r$ is a relation at the result node, then there are two strategies :
   (Select the one with minimum total time)
   a. $R_1 \rightarrow R_2 \rightarrow \ldots \rightarrow R_r \rightarrow \ldots \rightarrow R_n \rightarrow R_r$ or
   b. $R_1 \rightarrow R_2 \rightarrow \ldots \rightarrow R_{r-1} \rightarrow R_{r+1} \rightarrow \ldots \rightarrow R_n \rightarrow R_r$.
```

Figure 14 Algorithm Serial [12]

(2) Algorithm Parallel

The objective of the Algorithm Parallel is to minimize the total amount of data serially transmitted in order to minimize the response time. The algorithm chooses the initial feasible solution first, then the optimizer tries to improve on the solution by considering alternative schedules where some relations are sent to an intermediate site. An outline of this algorithm is shown in Figure 16 and a more detailed description is given in [21]. Finally, an example of the Algorithm Parallel is given in Figure 15.

We first order the relations according to their size and then choose the one with the minimum size as our step one. We choose the second smallest relation from the IFS and use the previous result to perform the semijoin in order to choose the one with a lower cost. This is shown in step two. The following step also uses the same technique to find the lowest cost. Finally, we will obtain the strategy for the Algorithm Parallel.

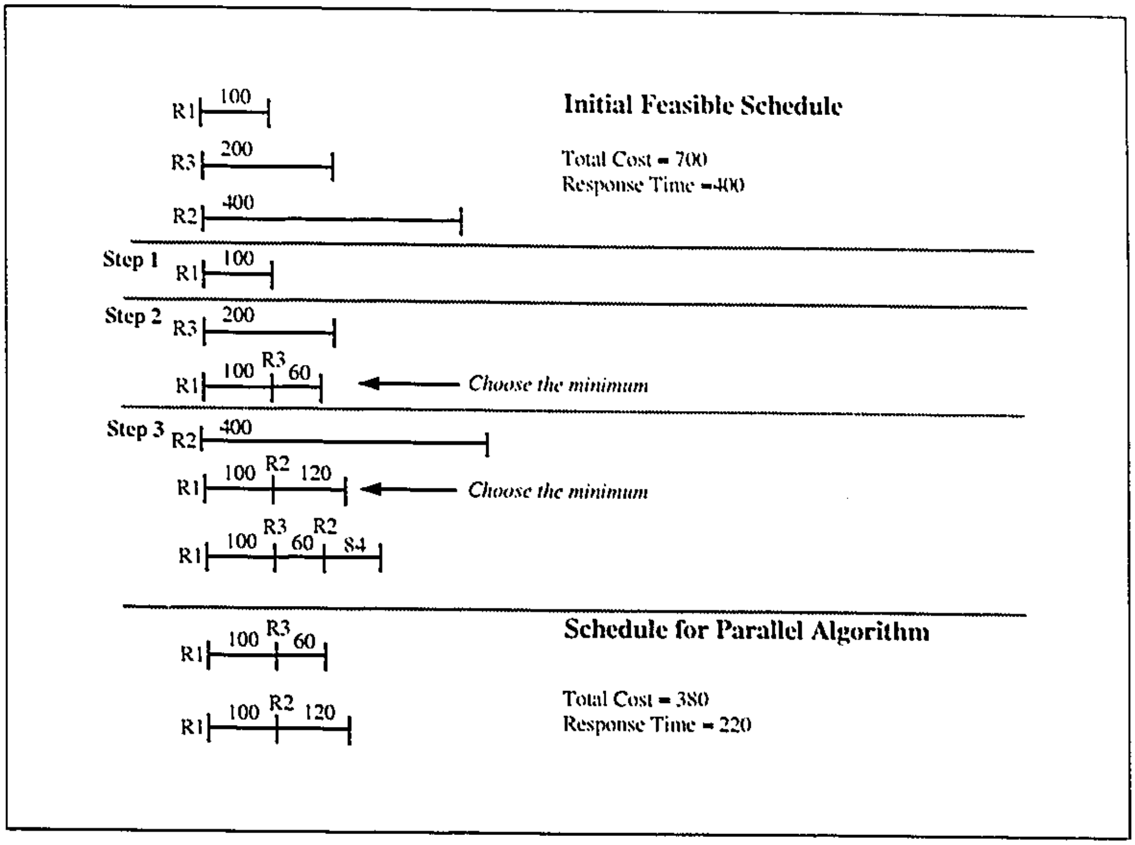


Figure 15 Example of Algorithm Parallel

1. Order relations R_i such that $s_1 \le s_2 \le ... \le s_m$.
2. Consider each relation R_i in ascending order of size.
3. For each relation $R_j(j<i)$, construct a schedule to R_j and all schedules of relations $R_k(k<j)$. Select the schedule with minimum response time.

Figure 16 Algorithm Parallel [12]

(3) General Algorithm

Several algorithms are proposed in [12] and [2] and reported in [23] which generalize the optimal algorithms for simple queries. The General algorithm is able to minimize either the **response time** or the **total cost** for the complex queries[12]. An outline of this

[12] Queries have more than one common join attributes.

algorithm is given in Figure 17. A completed example on both Response Time Version and Total Cost Version is from Figure 20 to Figure 24. The Relation Table used in this example is shown on Table 2. Since the scope of this thesis is not to explain this algorithm, it will not be discussed in more detail. However, a detailed explanation can be found in [2] and [12].

1. Do *all initial local processing*.
2. Generate candidate relation schedules. Isolate each of the σ joining attributes, and consider each to define a *simple query* with an undefined result node.
    a. To minimize *response time*, apply Algorithm PARALLEL to each simple query. Save all candidate schedules for integration in step 3.
    b. To minimize *total time*, apply Algorithm SERIAL to each simple query. This results in one schedule per simple query. From these schedules, the candidate schedules for each joining attribute are extracted. Consider joining attribute d_{ij}. Its candidate schedule is identical to the schedule produced by Algorithm SERIAL, applied to the simple query in which d_{ij} occurs, up to the transmission of d_{ij}. All transmissions after that are deleted from the schedule.
3. *Integrate the candidate schedules.* For each relation R_i, the candidate schedules are integrated to form a processing schedule for R_i. The integration is done by procedure RESPONSE for response time minimization and by procedure TOTAL or procedure COLLECTIVE for total time minimization.
4. *Remove schedule redundancies.* Eliminate relation schedules for relations which have been transmitted in the schedule of another relation.

Figure 17 General Algorithm [2]

1. *Candidate schedule ordering* : For each relation l order the candidate schedules on joining attribute d_{ij}, $j = 1, \dots, \sigma$ in ascending order of arrival time. Let ART_l denoted the arrival time of candidate schedule $CSCH_l$. (For the joining attributes not in R_i, disregard the corresponding candidate schedules.)
2. *Schedule integration* : For each candidate schedule $CSHI_l$ in ascending order, construct an integrated schedule for that consists of the parallel transmission of $CSCH_l$ and $CSCH_k$ with $k < l$. Select the integrated schedule with minimum response time.

Figure 18 Procedure RESPONSE [2]

1. *Adding candidate schedules* : For each relation R_i, each candidate schedule $CSCH_l$, do the following :

   If this schedule contains a transmission of a joining attribute of R_i, say d_{ij}, then add another candidate schedule which is the same as $CSCH_l$ except that the transmission d_{ij} is deleted.

2. *Select the best candidate schedule* : For each relation R_i and for each joining attribute d_{ij} $(j = 1, 2, \ldots, \sigma)$, select the candidate schedule which minimizes total time for transmitting R_i if only the joining attributes are considered which can be joined with d_{ij}.

3. *Candidate schedule ordering* : For each relation order the candidate schedules $BEST_{ij}$ on joining attribute d_{ij}, $j = 1, 2, \ldots, \sigma$, so that $ART_{i1} + C(s_1 * SLT_{i1}) \leq ART_{i\sigma} + C(s_i * SLT_{i\sigma})$. (For the joining attributes not in disregard $BEST_{ij}$.) **ART_{ij}** denotes the arrival time of the $BEST_{ij}$ schedule.

4. *Schedule integration* : For each $BEST_{ij}$ in ascending order of j, construct an integrated schedule to R_i that consists the parallel transmission of candidate schedule $BEST_{ij}$ and schedules $BEST_{ik}$ where $k < j$. Select the integrated schedules that results in the minimum total time value.

Figure 19 Procedure TOTAL [2]

RELATION	SIZE	d_{i1}		d_{i2}	
R_i	S_i	b_{i1}	p_{i1}	b_{i2}	p_{i2}
R1	1000	400	0.4	100	0.2
R2	2000	400	0.4	450	0.9
R3	3000	900	0.9	— — —	— — —

Table 2 The Relation Table

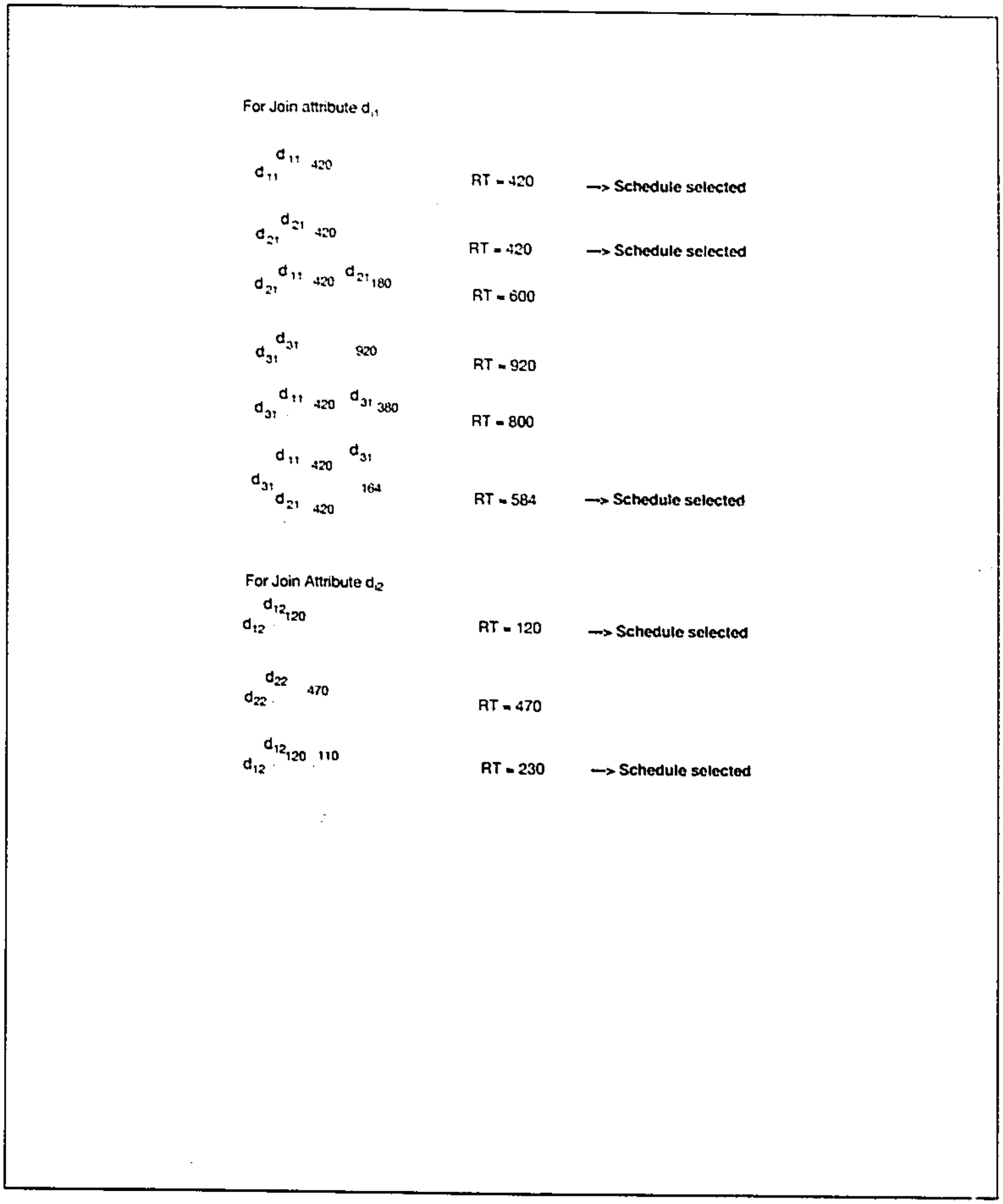


Figure 20 Example of General Algorithm (Response Time Version)

ATTRIBUTE	RESPONSE TIME
d_{22}	230
d_{21}	420
d_{31}	584

Table 3 The Options for R_2

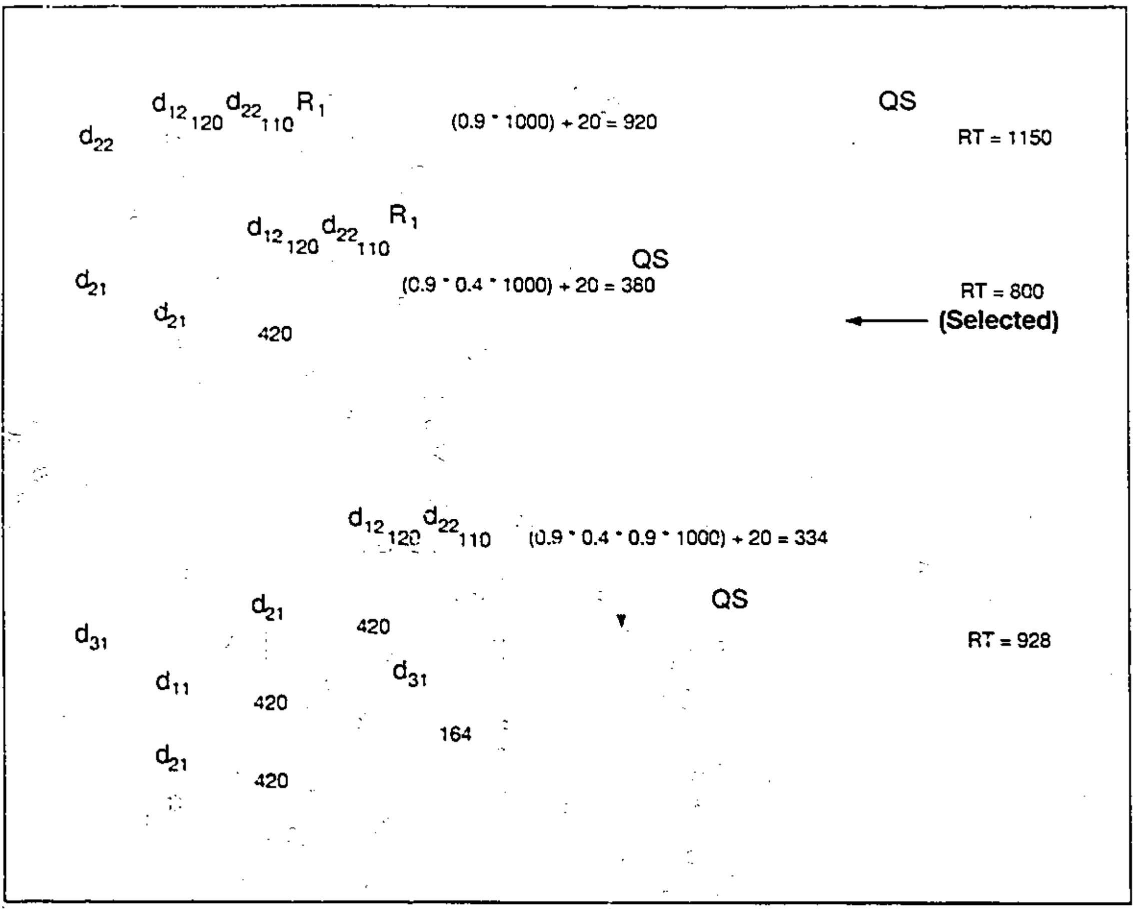


Figure 21 Example of Algorithm General (Response Time Version) Continuous...

ATTRIBUTE	RESPONSE TIME
d_{12}	120
d_{11}	420
d_{31}	584

Table 4 The Options for R_2

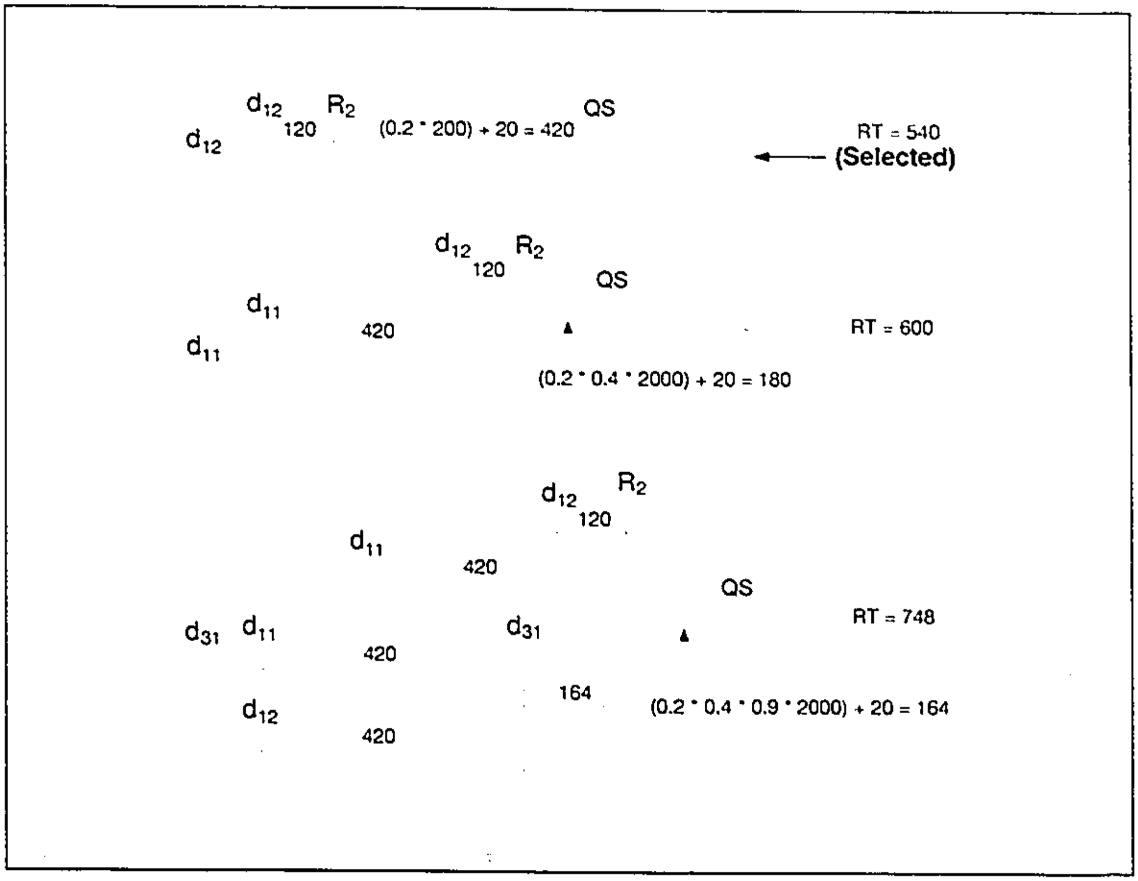


Figure 22 Example of Algorithm General (Response Time Version) Continuous...

ATTRIBUTE	RESPONSE TIME
d_{11}	420
d_{21}	420
— — —	— — —

Table 5 The Options for R_3

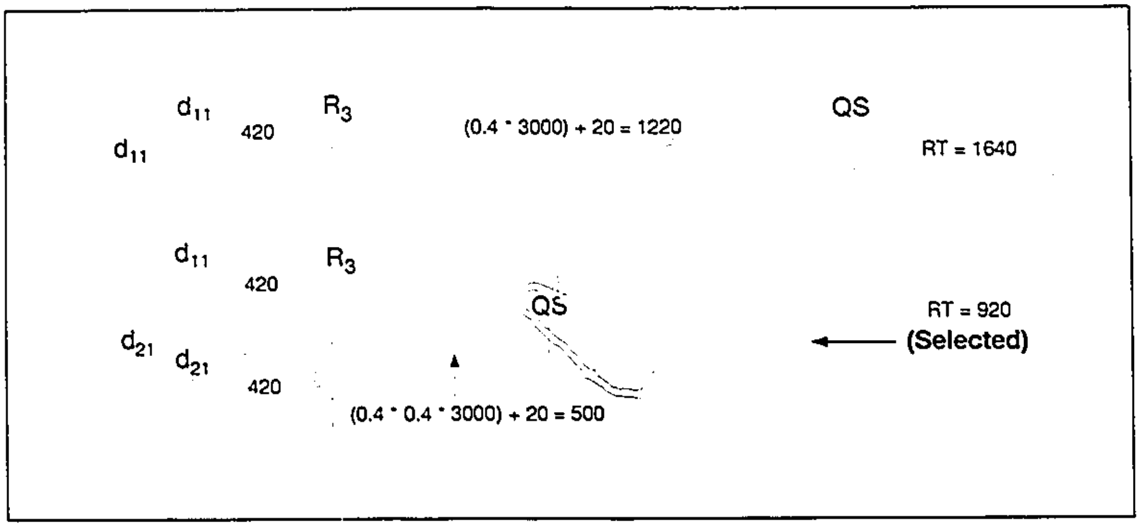


Figure 23 Example of Algorithm General (Response Time Version) Continuous...

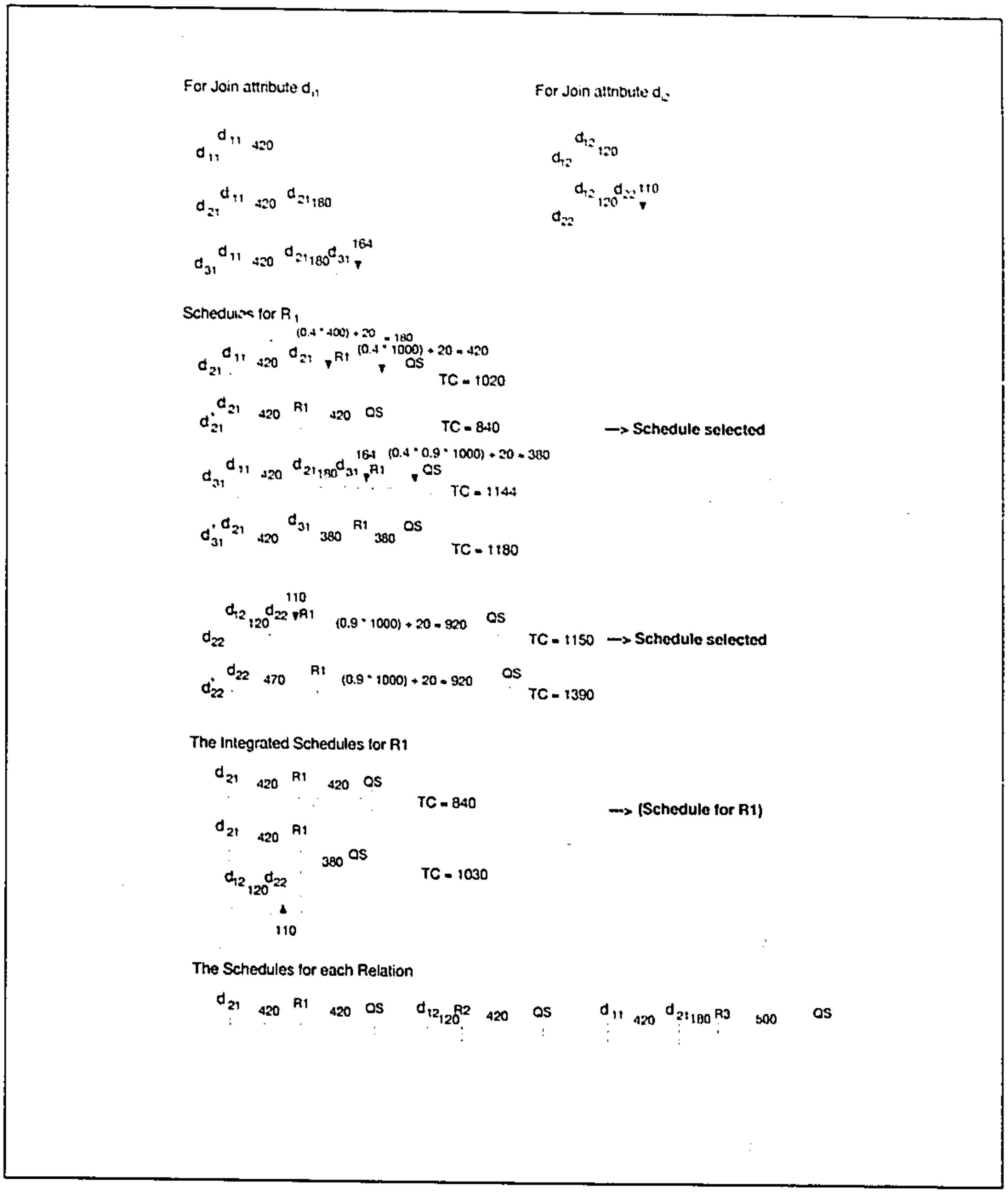


Figure 24 Example of Algorithm General (Total Cost Version)

CHAPTER 3 HEURISTIC FOR QUERY OPTIMIZATION (TOTAL COST VERSION)

The total cost heuristic is discussed in this chapter.

3.1 THE COST MODEL OF THE TOTAL COST HEURISTIC

The total cost heuristic measures the total amount of packets transmitted from one site to another. It assumes that the cost to transmit a packet from one site to another site is the same, but this is not always valid. For example, Site A wants to send 10 packets to Site B and Site C as shown in Figure 25.

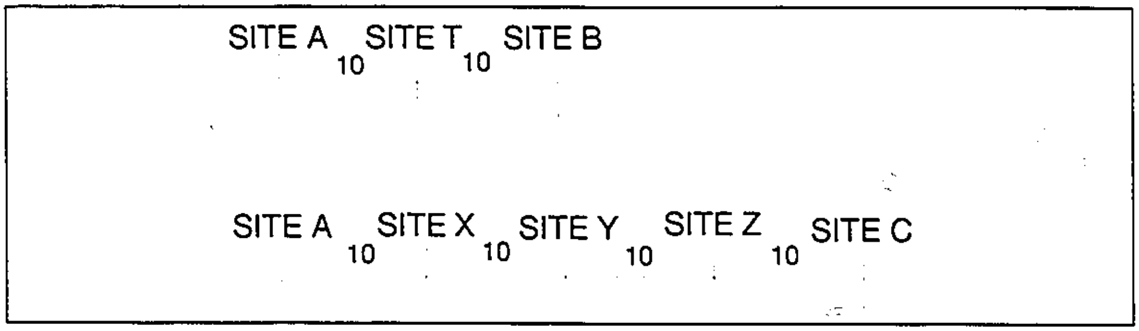


Figure 25 Example of Computing Total Cost

The total cost to send 10 packets from Site A to Site B is not the same as to send 10 packets from Site A to Site C. A better measure is to multiply the total number of links/hops between the source site and the destination site. Therefore, the total cost to send 10 packets from Site A to Site B is 20 and the total cost to send 10 packets from Site A to Site C is 40.

The total cost is defined as :

$$Total\ Cost = \sum_{\forall segments} \#\ of\ Packets * \#\ of\ Hops$$

The following is an example to compute the total cost from a hop table and a given schedule. Table 6 is a 8 x 8 hop table which is supplied by the simulator. Table 7 is an example of a schedule which is created by a query optimization algorithm.

	TO 1	To 2	To 3	To 4	To 5	To 6	To 7	To 8
From 1	0	1	3	2	3	1	2	3
From 2	2	0	3	2	1	2	3	2
From 3	1	2	0	3	3	1	2	3
From 4	2	2	1	0	1	2	3	1
From 5	1	2	2	3	0	2	1	1
From 6	2	1	1	2	3	0	3	1
From 7	3	1	1	2	2	2	0	1
From 8	2	2	3	3	1	3	2	0

Table 6 Example of Hop Table

	SIZE	HOPS	COST
Send : d_{11} —> R_5	200	3	200 * 3 = 600
Send : d_{51} —> R_7	250	1	250 * 1 = 250
Send : d_{71} —> R_3	150	1	150 * 1 = 150
	Total Cost = 600 + 250 + 150 = 1000		

Table 7 Example of a schedule

There are three segments in this schedule. The first segment sends 200 packets from the attribute d_{11} to R_5. The cost is 600. The cost of the second and third segment is computed at the same manner. Finally, the sum of all three gives a total cost of 1000.

3.2 THE DESCRIPTION OF THE TOTAL COST HEURISTIC

The algorithm of this heurictic is shown on Appendix D. The first input parameter of this heuristic is a relation table which contains the physical address of each relation, the size of each relation, the attribute size and selectivity factor of each relation. The second input parameter is a hop table.

A Cost and Benefit Table is created as shown in Figure 51. The cost to send an attribute d_{ij} at $Site_i$ to another attribute d_{xy} at $Site_x$ is defined as the size of attribute d_{ij} plus the overhead cost and then multiply the number of hops between $Site_i$ and $Site_x$. It is denoted by $C[b_{ij} \rightarrow b_{xy}]$. The benefit to send an attribute d_{ij} from $Site_i$ to d_{xy} at $Site_x$ is defined as the reduced relation size, that is $(1-p_{ij}) * R_x$ plus the overhead cost and then multiply the number of hops between $Site_x$ and the query site. It is denoted by $B[b_{ij} \rightarrow b_{xy}]$.

First, the transmissions where $C[b_{ij} \rightarrow b_{xy}] < (B[b_{ij} \rightarrow b_{xy}] +$ Threshold Value$)$ are considered. If only one such transmission is found, then this transmission is chosen. If any free transmissions[13] are found, then that transmission is removed from the Cost/Benefit table. If more than one transmission has the same smallest cost, then we chose the transmission with the best selectivity. If there is still a tie then we choose that transmission where the receiving site has the best selectivity. Finally, if there is still a tie then we choose the transmission with the best benefit. The process is repeated until no more transmissions are possible.

3.3 AN EXAMPLE OF THE TOTAL COST HEURISTIC

Three assumptions are made to simplify the example and to show all the cases in this heuristic.

1. All the values in the hop table are set to one.
2. The value of threshold is equal to zero.
3. The overhead cost is zero.

[13] If d_{ij} is semi-joined with d_{xy}, and d_{xy} is semi-joined with d_{ab}, then the semi-join of d_{ij} with d_{ab} is for free.

The following hop table is used in this example.

	To R_1	To R_2	To R_3	To QS
From R_1	0	1	1	1
From R_2	1	0	1	1
From R_3	1	1	0	1

Table 8 The Hop Table

ITERATION ONE

RELATION	SIZE	d_{i1}		d_{i2}	
R_i	S_i	b_{i1}	p_{i1}	b_{i2}	p_{i2}
R1	1000	400	0.4	100	0.2
R2	2000	400	0.4	450	0.9
R3	3000	900	0.9	———	———

Table 9 The Relation Table

	To :d_{11}	To :d_{21}	To :d_{31}	To :d_{12}	To :d_{22}
From : d_{11}	———	400/1200	400/1800	———	———
From : d_{21}	400/600	———	400/1800	———	———
From : d_{31}	900/100	900/200	———	———	———
From : d_{12}	———	———	———	———	100/1600
From : d_{22}	———	———	———	450/100	———

Table 10 The Cost/Benefit Table

Table 10 shows the first Cost/Benefit Table for the first iteration. For example, the cost of sending d_{11} to d_{21} is equal to :

$$Cost = (b_{11} + Overhead\ Cost) * Hop\ Table[1,2]$$

Since the overhead cost is equal to zero and the value of Hop_Table[1, 2] is equal to one, the cost of sending d_{11} to d_{21} is equal to (400 + 0) * 1 = 400. The benefit of sending d_{11} to d_{21} is equal to :

$$Benefit = (((1 - b_{11}) * R_2) + Overhead\ Cost) * Hop\ Table[2, QS]$$

The overhead cost is equal to zero and the value of Hop_Table[2, QS] is equal to one. Therefore, the benefit of sending d_{11} to d_{21} is equal to $(((1 - 0.4) * 2000) + 0) * 1 = 1200$. The rest of the cost and benefit values are computed at the same manner. Once this table is completed, it is clear that the transmissions d_{31} to d_{11}, d_{31} to d_{21} and d_{22} to d_{12} are not considered since the cost is higher than the benefit.

The transmission d_{12} to d_{22} is selected since it has the lowest cost. Since d_{12} to d_{22} is selected from the first iteration, the value of R_2 and b_{22} in the Relation Table and the Cost/Benefit Table must be updated before starting the second iteration. The updated R_2 and b_{22} are equal to :

$$R_2 = p_{12} * R_2$$

$$= 0.2 * 2000$$

$$= 400$$

$$b_{22} = P_{12} * b_{22}$$

$$= 0.2 * 450$$

$$= 90$$

All the cost values which contain the sending attribute d_{12} must be updated. There is only one transmission in Table 12 which is sending d_{22} to d_{12}. The new cost to send d_{22} is equal to 90. All the benefit values which contain R_2 must be updated. Therefore, the benefit of sending d_{11} to d_{21} and d_{31} to d_{21} must be updated. The benefit value for sending d_{11} to d_{21} and d_{31} to d_{21} are equal to :

$$B[d_{11} \rightarrow d_{21}] = ((1 - p_{11}) * R_2 + \text{Overhead Cost}) * \text{Hop\_Table}[2, QS]$$

$$= ((1 - 0.4) * 400 + 0) * 1$$

$= 240$

$B[d_{31} \text{ -> } d_{21}] = ((1 - p_{31}) * R_2 + \text{Overhead Cost}) * \text{Hop\_Table}[2. \text{QS}]$

$= ((1 - 0.9)* 400 + 0) * 1$

$= 40$

ITERATION TWO

RELATION	SIZE	d_{i1}		d_{i2}	
R_i	S_i	b_{i1}	p_{i1}	b_{i2}	p_{i2}
R1	1000	400	0.4	100	0.2
R2	***400***	400	0.4	***90***	0.9
R3	3000	900	0.9	— — —	— — —

Table 11 The Relation Table

	To :d_{11}	To :d_{21}	To :d_{31}	To :d_{12}	To :d_{22}
From : d_{11}	— — —	400/***240***	400/1800	— — —	— — —
From : d_{21}	400/600	— — —	400/1800	— — —	— — —
From : d_{31}	900/100	900/***40***	— — —	— — —	— — —
From : d_{12}	— — —	— — —	— — —	— — —	* * * *
From : d_{22}	— — —	— — —	— — —	***90***/100	— — —

Table 12 The Cost/Benefit Table

Those updated values are shown on Table 11 and Table 12. The transmissions d_{31} to d_{11}, d_{11} to ${}_{21}$ and d_{31} to d_{21} cannot be considered because the cost is greater than the benefit. The transmission d_{22} to d_{12} is selected because it has the lowest cost.

ITERATION THREE

RELATION	SIZE	d_{i1}		d_{i2}	
R_i	S_i	b_{i1}	p_{i1}	b_{i2}	p_{i2}
R1	***900***	400	0.4	***90***	0.2
R2	400	400	0.4	90	0.9
R3	3000	900	0.9	———	———

Table 13 The Relation Table

	To :d_{11}	To :d_{21}	To :d_{31}	To :d_{12}	To :d_{22}
From : d_{11}	———	400/240	400/1800	———	———
From : d_{21}	400/***540***	———	400/1800	———	———
From : d_{31}	900/***90***	900/40	———	———	———
From : d_{12}	———	———	———	———	* * *
From : d_{22}	———	———	———	* * *	———

Table 14 The Cost/Benefit Table

	PHASE 1	PHASE 2	PHASE 3	PHASE 4
d_{11} —> d_{21}	400/240	———	———	———
d_{11} —> d_{31}	400/1800	$d_{11} = 0.4$	$d_{31} = 0.9$	———
d_{21} —> d_{11}	400/540	$d_{21} = 0.4$	$d_{11} = 0.4$	Ben. = 540
d_{21} —> d_{31}	400/1800	$d_{21} = 0.4$	$d_{31} = 0.9$	———
d_{31} —> d_{11}	900/90	———	———	———
d_{31} —> d_{21}	900/40	———	———	———

Table 15 The Comparison Table

Table 13 and 14 show the updated values. Table 15 shows how to select the next transmission. All possible transmissions have the same cost, 400. All the sending attributes have the same selectivity factor which is 0.4. But the receiving attribute d_{11} has the best selectivity factor, 0.4. Therefore, we send d_{21} to d_{11}.

ITERATION FOUR : choose to send d_{11} to d_{21}

RELATION	SIZE	d_{i1}		d_{i2}	
R_i	S_i	b_{i1}	p_{i1}	b_{i2}	p_{i2}
R1	***360***	***160***	0.4	90	0.2
R2	400	400	0.4	90	0.9
R3	3000	900	0.9	———	———

Table 16 The Relation Table

	To :d_{11}	To :d_{21}	To :d_{31}	To :d_{12}	To :d_{22}
From : d_{11}	———	***160***/240	***160***/1800	———	———
From : d_{21}	* * *	———	400/1800	———	———
From : d_{31}	900/***36***	900/40	———	———	———
From : d_{12}	———	———	———	———	* * *
From : d_{22}	———	———	———	* * *	———

Table 17 The Cost/Benefit Table

ITERATION FIVE : choose to send d_{11} to d_{31}

RELATION	SIZE	d_{i1}		d_{i2}	
R_i	S_i	b_{i1}	p_{i1}	b_{i2}	p_{i2}
R1	360	160	0.4	90	0.2
R2	***160***	***160***	0.4	90	0.9
R3	3000	900	0.9	———	———

Table 18 The Relation Table

	To :d_{11}	To :d_{21}	To :d_{31}	To :d_{12}	To :d_{22}
From : d_{11}	———	* * *	160/1800	———	———
From : d_{21}	* * *	———	***160***/1800	———	———
From : d_{31}	900/36	900/***16***	———	———	———
From : d_{12}	———	———	———	———	* * *
From : d_{22}	———	———	———	* * *	———

Table 19 The Cost/Benefit Table

Now we see that there is a free transmission, d_{21} to d_{31}. There are no more possible transmissions, as shown.

ITERATION SIX

RELATION	SIZE	d_{i1}		d_{i2}	
R_i	S_i	b_{i1}	p_{i1}	b_{i2}	p_{i2}
R1	360	100	0.4	90	0.2
R2	160	160	0.4	90	0.9
R3	***480***	***144***	0.9	— — —	— — —

Table 20 The Relation Table

	To :d_{11}	To :d_{21}	To :d_{31}	To :d_{12}	To :d_{22}
From : d_{11}	— — —	* * *	* * *	— — —	— — —
From : d_{21}	* * *	— — —	* * *	— — —	— — —
From : d_{31}	***144***/36	***144***/16	— — —	— — —	— — —
From : d_{12}	— — —	— — —	— — —	— — —	* * *
From : d_{22}	— — —	— — —	— — —	* * *	— — —

Table 21 The Cost/Benefit Table

The transmissions generated in this example are :

1. d_{12} —> d_{22} [SIZE = 100]
2. d_{22} —> d_{12} [SIZE = 90]
3. d_{21} —> d_{11} [SIZE = 400]
4. d_{11} —> d_{21} [SIZE = 160]
5. d_{11} —> d_{31} [SIZE = 160]
6. R_1 —> QS [SIZE = 360]
7. R_2 —> QS [SIZE = 160]
8. R_3 —> QS [SIZE = 480]

The total cost is 1910 units.

CHAPTER 4 HEURISTIC FOR QUERY OPTIMIZATION (RESPONSE TIME VERSION)

The cost model, the description and an example of the Response Time Heuristic are discussed in this chapter. Section 4.1 discusses how to measure the cost of the heuristic. Section 4.2 discusses the algorithm of this heuristic and Section 4.3 illustrates an example of this heuristic.

4.1 THE COST MODEL OF THE RESPONSE TIME HEURISTIC

The cost model used to measure the Response Time Heuristic is the **Response Time.** This is the elapsed time for the query execution. Since the operations can be executed in parallel at different sites, the response time of query may be significantly less than its total cost. A discussion and an example on response time is given in Section 2.6.1.

When a query is processed by the simulator (discussed in Appendix A), the simulator will record the query's starting time. When the query is finished, i.e., all the transmissions corresponding to a query are completed, the simulator will record this time of termination. Therefore, the response time on processing this query is the time of termination minus the starting time. The response time is measured in the default unit time used in the Simscript II.5.

4.2 THE DESCRIPTION OF THE RESPONSE TIME HEURISTIC

The Response Time Heuristic is discussed in this section. The algorithm is shown on Appendix E. The symbols used in this heuristic are defined in Figure 57.

This heuristic is shown in Figures 58 to Figure 61. The first input parameter of this heuristic is a Relation Table which contains the physical address of each relation, the size of each relation, the attribute size and selectivity factor for each of the relation. The second input parameter is a Delay Table which is given by the simulator. This table stores the delay to send a packet from one site to every other site.

The Cost Table is created in Figure 58. The cost to send an attribute d_{ij} at $Site_i$ to $Site_k$ is defined as the size of attribute d_{ij} plus the overhead cost multiplied by the delay cost between $Site_i$ and $Site_k$. It is denoted by $C[b_{ij} -> k]$. The general equation of $C[b_{ij} -> k]$ is shown as follow :

$$C[b_{ij}- > k] = (b_{ij} + Overhead\,Cost) * D_{ik}$$

The symbol b_{ij} indicates the size of attribute d_{ij} and D_{ik} indicates the delay cost between $Site_i$ and $Site_k$. The costs can be evaluated if

1. i is not equal to k.

2. The size of receiving attribute b_{kj} is not equal to zero.

3. The size of sending attribute b_{ij} is not equal to zero.

Otherwise, the cost is "UNDEFINED".

The variable NUM_SELECTION in Figure 59 is used to count the number of relations that do not generate the optimum schedule. For each of the relations given in the Relation Table the procedure is given below. A relation (eg: R_x) that does not have the optimum schedule yet is located. If the cost in the Cost Table is not equal to "UNDEFINED", then the cost has to be sorted in ascending order. The results are stored in a temporary storage called SELECTION_LIST.

The cost to send R_x from its own site to the Query Site is computed and stored under this schedule in Schedule_R_x(1). This is the IFS which is the first schedule for R_x. The rest of the schedules for R_x is generated as follow :

1. Find the minimum cost from SELECTION_LIST (eg: C[b_{ab} —> c])
2. If this is the first iteration, then use P_{ab} to carry out a Semi-Join with R_x and store the result in Schedule_R_x(2). Otherwise, use p_{ab} to do the Semi-Join with the previous schedule which is stored in the rest of Schedule_R_x.

This minimum cost will be deleted from the SELECTION_LIST. The previous two steps will be repeated until SELECTION_LIST is empty for R_x. The schedule with the minimum response time from Schedule_R_x will be selected and stored to another storage MIN_Schedule_R_x. Figure 59 and Figure 60 is the algorithm to find the MIN_Schedule_R_x for each of all relations which don't have the optimum schedule.

Figure 61 shows the last few steps of this heuristic. The schedule with the minimum response time will be selected as the ***Optimum Schedule*** for R_y. Before finding the next optimum schedule for other relations, the Relation Table and the Cost Table must be updated. The attributes b_{yj}, for all j, and relation R_y are modified by doing the Semi-Join as the Optimum Schedule for R_y. All the costs where sending attribute is from R_y and is not "UNDEFINED" must be re-computed, using the modified Relation Table. All cost for R_y will not be considered again. The process from Figure 59 to Figure 61 will be repeated until all the relations obtain their Optimum Schedule. A step-by-step example of this heuristic is shown on Section 4.3.

4.3 AN EXAMPLE OF THE RESPONSE TIME HEURISTIC

This section illustrates a complete example of the Response Time Heuristic.

THE FIRST ITERATION

RELATION	SIZE	d_{i0}		d_{i1}	
R_i	S_i	b_{i0}	p_{i0}	b_{i1}	p_{i1}
R_0	1000	400	0.4	100	0.2
R_1	2000	400	0.4	450	0.9
R_2	3000	900	0.9	———	———

Table 22 The Relation Table

	To : R_0	To : R_1	To : R_2	To : QS
From : R_0	0	1.59	1.36	2.97
From : R_1	1.30	0	2.66	2.75
From : R_2	1.30	2.88	0	3.15

Table 23 The Delay Table

	TO : R_0	TO : R_1	TO : R_2
SEND : d_{00}	———	667.80	571.20
SEND : d_{01}	———	190.80	———
SEND : d_{10}	546.00	———	1117.20
SEND : d_{11}	611.00	———	———
SEND : d_{20}	1196.00	2649.60	———
SEND : d_{21}	———	———	———

Table 24 The Cost Table

Table 22 is the Relation Table used in this example. Table 23 is the Delay Table used in this example. For instance, the cost to send a packet from the site of R_0 to the site of R_1 is 1.59 Unit Time. This Delay Table is obtained from the Simulator. Table 24 is the Cost Table which is created in the first iteration. The cost to send b_{ij} from the site with R_i to the site with R_k is equal to :

$$C[b_{ij} -> k] = (b_{ij} + Overhead\,Cost) * D_{ik}$$

Let the overhead cost is equal to 20. The Cost to send the following attributes are "UNDEFINED" because those attributes are stored in their own site (ie. i = k).

- Send d_{00} to R_0
- Send d_{01} to R_0
- Send d_{10} tc R_1
- Send d_{11} to R_1
- Send d_{20} to R_2
- Send d_{21} to R_2

The costs to send d_{01} to R_2 and d_{11} to R_2 are "UNDEFINED" because the *receiving attribute* d_{21} is equal to zero. The cost to send d_{21} to R_0 and d_{21} to R_1 are "UNDEFINED" because the *sending attribute* d_{21} is equal to zero. The cost to send d_{10} to R_0 is equal to $(b_{10} + 20) * D_{10} = (400 + 20) * 1.30 = 546$. The rest of the cost is computed in the same manner.

Attribute	Cost	Selectivity
d_{10}	546.00	0.4
d_{11}	611.00	0.9
d_{20}	1196.00	0.9

Table 25 The Selection List for R_0

The Cost of IFS to send R_0 to QS = 1020 * 2.97 = 3029.40

	SELECTIVITY	COST
Send : d_{10} —> R_0	0.4	546.00
Send : R_0 —> QS	— — — —	1247.40
Max. Cost to Send Attributes		546.00
Max. Cost to send R_0 to Query Site		1793.40

Table 26 Schedule 1 for R_0

Table 25 shows the SELECTION_LIST for R_0 in the first iteration. The cost for sending the attributes to R_0 is rearranged in ascending order. There are three attributes in the SELECTION_LIST. The value of COUNT become three in Figure 59. The cost of IFS for R_0 which sends the whole R_0 directly to the Query Site is (1000 + 20) * 2.97 = 3029.40. This schedule is stored in Schedule_R_0(1) as one of the options for sending R_0. The second option for sending R_0 is to find the minimum cost from the SELECTION_LIST. This is to send d_{10} to R_0. This attribute is used to do the Semi-Join with R_0 and store the result in Schedule_R_0(2). This schedule is shown in Table 26. The cost to send d_{10} to R_0 and R_0 to Query Site is equal to 546.00 and 1247.40 correspondingly. The "Maximum Cost to send attribute" is equal to 546.00 since there is only one attribute. This will be explained more clearly in the following paragraph. The value of "Maximum Cost to send R_x to Query Site" is the sum of the cost of sending the reduced R_x to the Query Site and the "Maximum cost to send attributes". In Table 26, this is equal to (1247.40 + 546.00) = 1793.40.

	SELECTIVITY	COST
Send : d_{11} —> R_0	0.90	611.00
Send : d_{10} —> R_0	0.40	546.00
Send : R_0 —> QS	— — — —	1128.60
Max. Cost to Send Attributes		1157.00
Max. Cost to send R_0 to Query Site		2285.60

Table 27 Schedule 2 for R_0

The next smallest cost from the SELECTION_LIST (Table 25) is to send d_{11} to R_0. Therefore, this transmission will be used to do the Semi-Join with the previous schedule for R_0 which forms Schedule_R_0(3). Table 27 shows this new schedule for R_0. Attributes d_{11} and d_{10} are both sent to R_0 to do the Semi-Join at the same instant. Even though there are two attributes here, they are both in the source site. Therefore, the cost to send those two attributes is accumulated. In Table 27, this value is equal to (611.00 + 546.00) = 1157.00. The maximum of the accumulated costs will be the value of "Maximum Cost to send attributes". If there is a single attribute sent to R_0 with a cost that is bigger than any of the accumulated costs, then that cost will be the "Maximum Cost to send attributes". As shown in Table 27, this value is equal to (1128.60 + 1157.00) = 2285.60.

	SELECTIVITY	COST
Send : d_{20} —> R_0	0.9	1196.00
Send : d_{11} —> R_0	0.9	611.00
Send : d_{10} —> R_0	0.4	546.00
Send : R_0 —> QS	— — — —	1021.68
Max. Cost to Send Attributes		1196.00
Max. Cost to send R_0 to Query Site		2217.68

Table 28 Schedule 3 for R_0

Attribute	Cost	Selectivity
d_{01}	0.2	190.80
d_{00}	0.4	667.80
d_{20}	0.9	2649.60

Table 29 The Selection List for R_1

The Cost of IFS to send R_1 to QS = 2020 * 2.75 = 5555

	SELECTIVITY	COST
Send : d_{01} —> R_1	0.20	190.80
Send : R_1 —> QS	— — — —	1155.00
Max. Cost to Send Attributes		190.80
Max. Cost to send R_1 to Query Site		1345.80

Table 30 Schedule 1 for R_1

	SELECTIVITY	COST
Send : d_{00} —> R_1	0.40	667.80
Send : d_{01} —> R_1	0.20	190.80
Send : R_1 —> QS	— — — —	495.00
Max. Cost to Send Attributes		858.60
Max. Cost to send R_1 to Query Site		1353.60

Table 31 Schedule 2 for R_1

	SELECTIVITY	COST
Send : d_{20} —> R_1	0.90	2649.60
Send : d_{00} —> R_1	0.40	667.80
Send : d_{01} —> R_1	0.20	190.80
Send : R_1 —> QS	— — — —	451.00
Max. Cost to Send Attributes		2649.60
Max. Cost to send R_1 to Query Site		3100.60

Table 32 Schedule 3 for R_1

Attribute	Cost	Selectivity
d_{00}	0.40	571.20
d_{10}	0.40	1117.20

Table 33 The Selection List for R_2

The Cost of IFS to send R_2 to QS = 3020 * 3.15 = 9513

	SELECTIVITY	COST
Send : d_{00} —> R_2	0.40	571.20
Send : R_2 —> QS	— — — —	3843.00
Max. Cost to Send Attributes		571.20
Max. Cost to send R_2 to Query Site		4414.20

Table 34 Schedule 1 for R_2

	SELECTIVITY	COST
Send : d_{10} —> R_2	0.40	1117.20
Send : d_{00} —> R_2	0.40	571.20
Send : R_2 —> QS	— — — —	1575.00
Max. Cost to Send Attributes		1117.20
Max. Cost to send R_2 to Query Site		2692.20

Table 35 Schedule 2 for R_2

	SELECTIVITY	COST
Send : d_{01} —> R_1	0.20	190.80
Send : R_1 —> QS	— — — —	1155.00
Max. Cost to Send Attributes		190.80
Max. Cost to send R_1 to Query Site		1345.80

Table 36 The Optimum Schedule at First Iteration (Schedule 1 of R_1)

The last schedule for R_0 which is Schedule_R_0(4) is shown in Table 28 and is formed in the same manner. Once this is finished, the schedule with the minimum cost in the storage Schedule_R_0 will be found and stored in MIN_Schedule_R_0. Table 26 the Schedule 2 for R_0 will be selected for R_0 in this example, since its cost is 1793.40 which is the smallest cost found so far. Table 25 to Table 32 and Table 33 to Table 36 are doing the same process for R_1 and R_2 correspondingly. Therefore, MIN_Schedule_R_1 is on Table 30 and MIN_Schedule_R_2 is on Table 35.

The schedule with the minimum cost which is MIN_Schedule_R_1 will be selected as the ***Optimum Schedule*** for R_1 as shown in Table 36. This schedule sends d_{01} to R_1 and then sends R_1 to the Query Site.

THE SECOND ITERATION

RELATION	SIZE	d_{i0}		d_{i1}	
R_i	S_i	b_{i0}	p_{i0}	b_{i1}	p_{i1}
R_0	1000	400	0.4	100	0.2
R_1	400	400	0.4	90	0.9
R_2	3000	900	0.9	— — —	— — —

Table 37 The Relation Table

	To : R_0	To : R_1	To : R_2	To : QS
From : R_0	0	1.59	1.36	2.97
From : R_1	1.30	0	2.66	2.75
From : R_2	1.30	2.88	0	3.15

Table 38 The Delay Table

	TO : R_0	TO : R_1	TO : R_2
SEND : d_{00}	———	*****	571.20
SEND : d_{01}	———	*****	———
SEND : d_{10}	546.00	———	1117.20
SEND : d_{11}	333.80	———	———
SEND : d_{20}	1196.00	*****	———
SEND : d_{21}	———	———	———

Table 39 The Cost Table

Before creating the schedules for the second iteration, there are some values in the Relation Table and the Cost Table that must be modified. Since d_{01} is sent to R_1, then the size of R_1 is reduced, it becomes (p_{01} * R_1) = (0.2 * 2000) = 400. The attribute d_{11} is reduced by doing the Semi-Join with d_{01}. Therefore, d_{11} becomes (0.2 * 450) = 90. The cost of sending d_{11} to R_0 (Cost Table in Table 39) has to be modified. Since the cost to modify d_{11} is equal to 190.80 which is shown on Table 36, the new cost to send d_{11} to R_0 is equal to (190.80 + (90 + 20) * 1.3) = 333.80. This cost is updated and showed on Table 39.

All the transmissions which have the destination site equal to R_1 will not be considered any more, since the Optimum Schedule for R_1 is already found. Therefore, the cost for sending d_{00} to R_1, d_{01} to R_1 and d_{20} to R_1 will be erased from the Cost Table as shown on Table 39.

Attribute	Cost	Selectivity
d_{11}'	0.90	333.80
d_{10}	0.40	546.00
d_{20}	0.90	1196.00

Table 40 The Selection List for R_0

The Cost of IFS to send R_0 to QS = 1020 * 2.97 = 3029.40

	SELECTIVITY	COST
Send : d_{11}' —> R_0	0.90	333.80
Send : R_0 —> QS	— — — —	2732.40
Max. Cost to Send Attributes		333.80
Max. Cost to send R_0 to Query Site		3066.20

Table 41 Schedule 1 for R_0

	SELECTIVITY	COST
Send : d_{10} —> R_0	0.40	546.00
Send : d_{11}' —> R_0	0.90	333.80
Send : R_0 —> QS	— — — —	1128.60
Max. Cost to Send Attributes		879.80
Max. Cost to send R_0 to Query Site		2008.40

Table 42 Schedule 2 for R_0

	SELECTIVITY	COST
Send : d_{20} —> R_0	0.90	1196.00
Send : d_{10} —> R_0	0.40	546.00
Send : d_{11}' —> R_0	0.90	333.80
Send : R_0 —> QS	— — — —	1021.68
Max. Cost to Send Attributes		1196.00
Max. Cost to send R_0 to Query Site		2217.68

Table 43 Schedule 3 for R_0

Attribute	Cost	Selectivity
d_{00}	571.20	0.40
d_{10}	1117.20	0.40

Table 44 The Selection List for R_2

The Cost of IFS to send R_2 to QS = 3020 * 3.15 = 9513

	SELECTIVITY	COST
Send : d_{00} —> R_2	0.40	571.20
Send : R_2 —> QS	— — — —	3843.00
Max. Cost to Send Attributes		4414.20
Max. Cost to send R_2 to Query Site		4414.20

Table 45 Schedule 1 for R_2

	SELECTIVITY	COST
Send : d_{10} —> R_2	0.40	1117.20
Send : d_{00} —> R_2	0.40	571.20
Send : R_2 —> QS	— — — —	1575.00
Max. Cost to Send Attributes		1117.20
Max. Cost to send R_2 to Query Site		2692.20

Table 46 Schedule 2 for R_2

	SELECTIVITY	COST
Send : d_{10} —> R_0	0.40	546.00
Send : d_{11}' —> R_0	0.90	333.80
Send : R_0 —> QS	— — — —	1128.60
Max. Cost to Send Attributes		879.80
Max. Cost to send R_0 to Query Site		2008.40

Table 47 The Optimum Schedule at Second Iteration (Schedule 2 of R_0)

THE THIRD ITERATION

RELATION	SIZE	d_{i0}		d_{i1}	
R_i	S_i	b_{i0}	p_{i0}	b_{i1}	p_{i1}
R_0	360	160	0.4	90	0.2
R_1	400	400	0.4	90	0.9
R_2	3000	900	0.9	— — —	— — —

Table 48 The Relation Table

	To : R_0	To : R_1	To : R_2	To : QS
From : R_0	0	1.59	1.36	2.97
From : R_1	1.30	0	2.66	2.75
From : R_2	1.30	2.88	0	3.15

Table 49 The Delay Table

	TO : R_0	TO : R_1	TO : R_2
SEND : d_{00}	— — —	*****	571.20
SEND : d_{01}	— — —	*****	— — —
SEND : d_{10}	*****	— — —	1117.20
SEND : d_{11}	*****	— — —	— — —
SEND : d_{20}	*****	*****	— — —
SEND : d_{21}	— — —	— — —	— — —

Table 50 The Cost Table

Attribute	Cost	Selectivity
d_{00}	571.20	0.40
d_{10}	1117.20	0.40

Table 51 The Selection List for R_2

The Cost of IFS to send R_2 to QS = 3020 * 3.15 = 9513

	SELECTIVITY	COST
Send : d_{00} —> R_2	0.40	571.20
Send : R_2 —> QS	— — — —	3843.00
Max. Cost to Send Attributes		571.00
Max. Cost to send R_2 to Query Site		4414.20

Table 52 Schedule 1 for R_2

	SELECTIVITY	COST
Send : d_{10} —> R_2	0.40	1117.20
Send : d_{00} —> R_2	0.40	571.20
Send : R_2 —> QS	— — — —	1575.00
Max. Cost to Send Attributes		1117.20
Max. Cost to send R_2 to Query Site		2692.20

Table 53 Schedule 2 for R_2

	SELECTIVITY	COST
Send : d_{10} —> R_2	0.40	1117.20
Send : d_{00} —> R_2	0.40	571.20
Send : R_2 —> QS	— — — —	1575.00
Max. Cost to Send Attributes		1117.20
Max. Cost to send R_2 to Query Site		2692.20

Table 54 The Optimum Schedule at Third Iteration (Schedule 2 of R_2)

The Final Optimal Schedules

	SELECTIVITY	COST
Send : d_{01} —> R_1	0.20	190.80
Send : R_1 —> QS	— — — —	1155.00
Max. Cost to Send Attributes		190.80
Max. Cost to send R_1 to Query Site		1345.80

Table 55 The Optimum Schedule at First Iteration (Schedule 1 of R_1)

	SELECTIVITY	COST
Send : d_{10} —> R_0	0.40	546.00
Send : d_{11}' —> R_0	0.90	333.80
Send : R_0 —> QS	— — — —	1128.60
Max. Cost to Send Attributes		879.80
Max. Cost to send R_0 to Query Site		2008.40

Table 56 The Optimum Schedule at Second Iteration (Schedule 2 of R_0)

	SELECTIVITY	COST
Send : d_{10} —> R_2	0.40	1117.20
Send : d_{00} —> R_2	0.40	571.20
Send : R_2 —> QS	— — — —	1575.00
Max. Cost to Send Attributes		1117.20
Max. Cost to send R_2 to Query Site		2692.20

Table 57 The Optimum Schedule at Third Iteration (Schedule 2 of R_2)

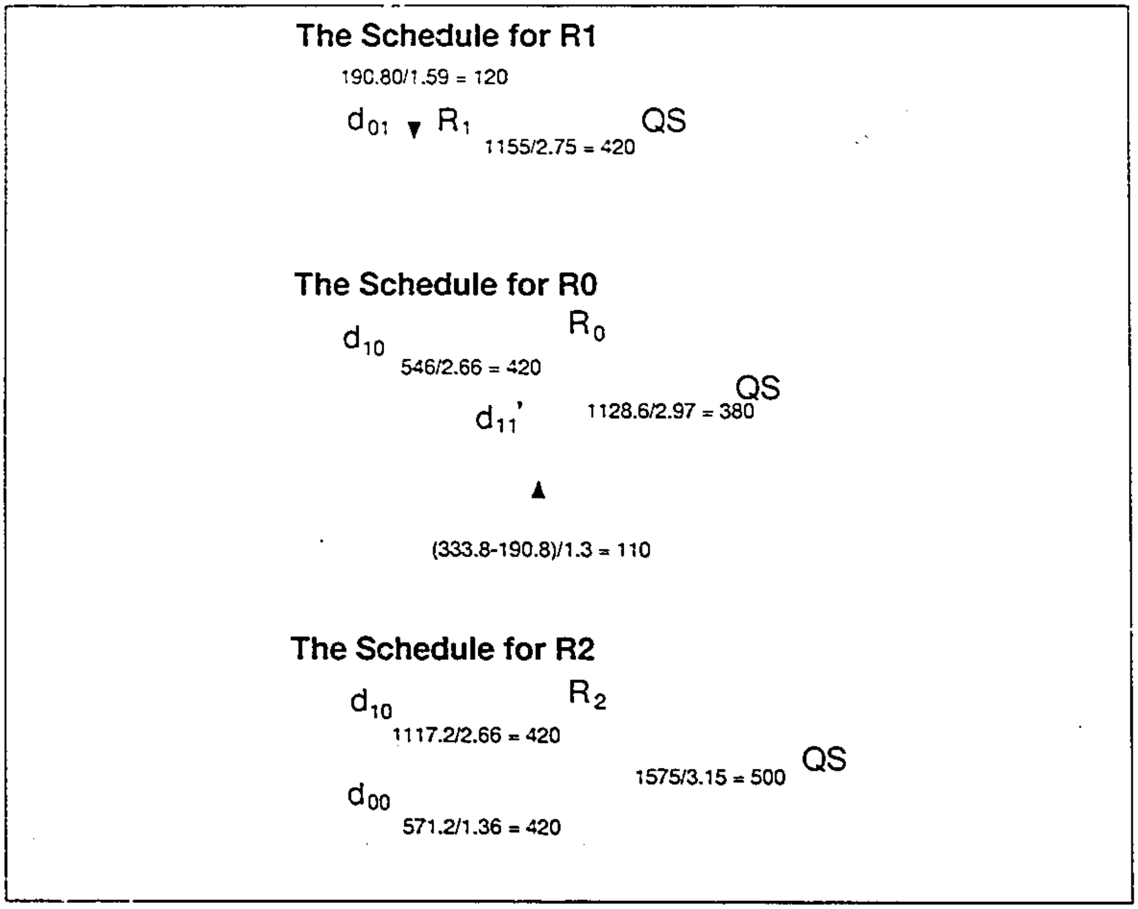


Figure 26 The Graphically Optimal Schedule

The Optimum Schedule for R_0 and R_2 are generated in the same manner which is shown from Table 37 to Table 53. The Optimum Schedules for all three relations are shown from Table 55 to Table 57. The graphical optimal schedules are shown on Figure 26.

CHAPTER 5 COMPARISONS AND RESULTS

This chapter shows and compares the results of all simulations.

5.1 RESULTS OF THE SAME QUERY PROCESS AT DIFFERENT INSTANT (Total Cost Heuristic)

This section shows the total cost of five queries at low load, medium load and high load situations using the Total Cost Heuristic. Row 1 shows the total cost of the 5 queries at times T_1 to T_5 (Low Load). Row 2 shows the total cost of the queries at medium load. The remaining 5 rows show the cost at 5 different high low times.

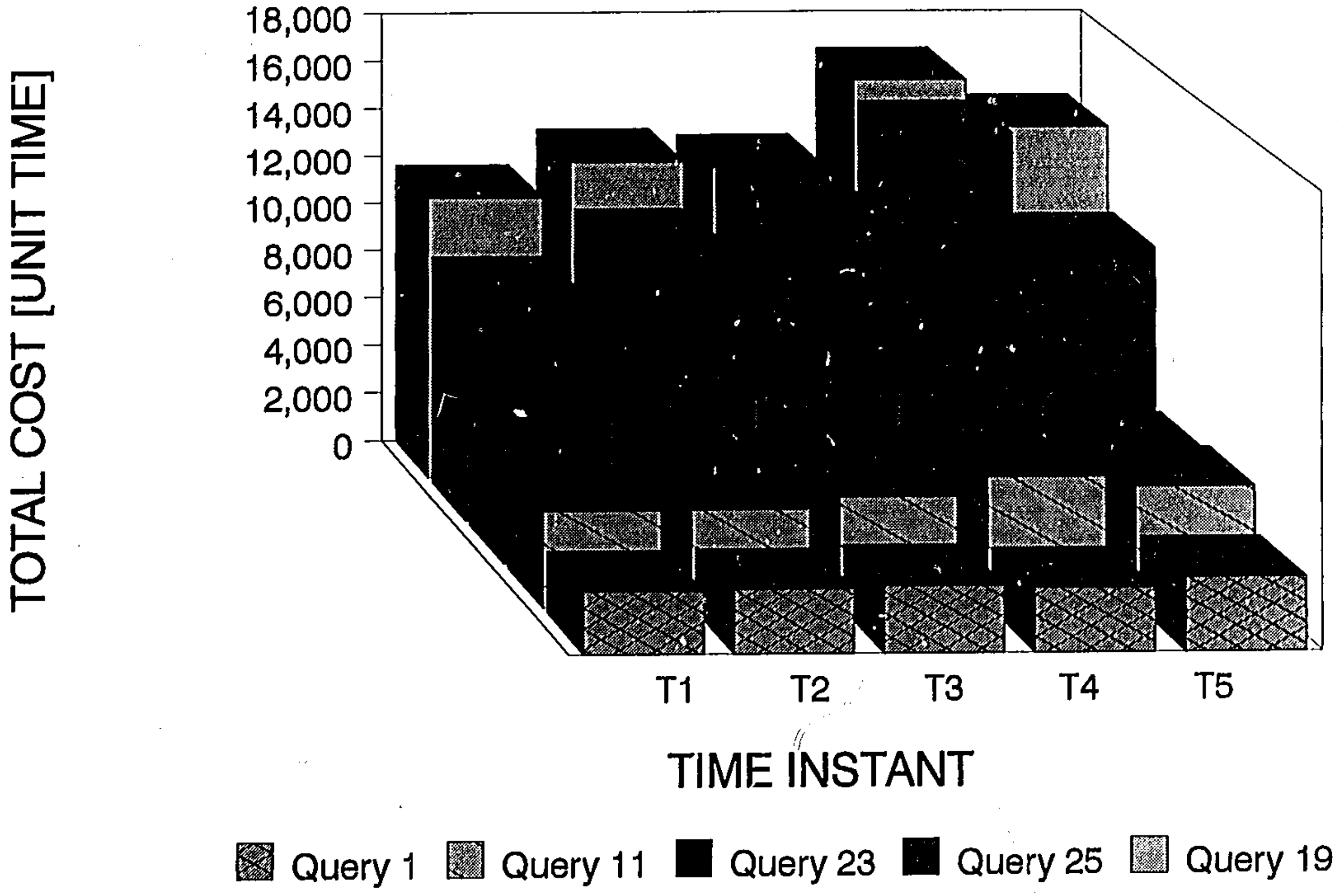
The Hop Table at T_1 is used by the Total Cost Heuristic to generate the schedules for a given query and compute its total cost. The same schedules are used to compute the total cost at different times as T_2, T_3, T_4 and T_5. The Hop Tables which are generated at those times will be used to compute the total cost. The graph of the total cost at a high load situation is on next page.

QUERY NUMBER	1	11	19	23	25
LOW LOAD (T_1 to T_5)	2050	3764	11100	3644	9308
MED. LOAD (T_1 to T_5)	2050	3764	11100	3644	9308
HIGH LOAD AT T_1	2454	3912	11596	3435	9308
HIGH LOAD AT T_2	2518	3924	13096	4085	11264
HIGH LOAD AT T_3	2638	4404	12828	4881	12916
HIGH LOAD AT T_4	2518	5164	16468	5179	15764
HIGH LOAD AT T_5	2946	4764	14492	4441	11020

Table 58 The Result of Total Cost Heuristic at Different Instant

T.C. HEURISTIC AT DIFFERENT INSTANT

(HIGH LOAD SITUATION)



5.2 RESULTS FOR TOTAL COST VERSION

The tables used in Section 5.2.1 to Section 5.2.2 have the following format. The first column is the Query Number. Each query can be found in Appendix D. The 2nd, 3rd and 4th columns give the total cost for the query at a low, medium and high load respectively.

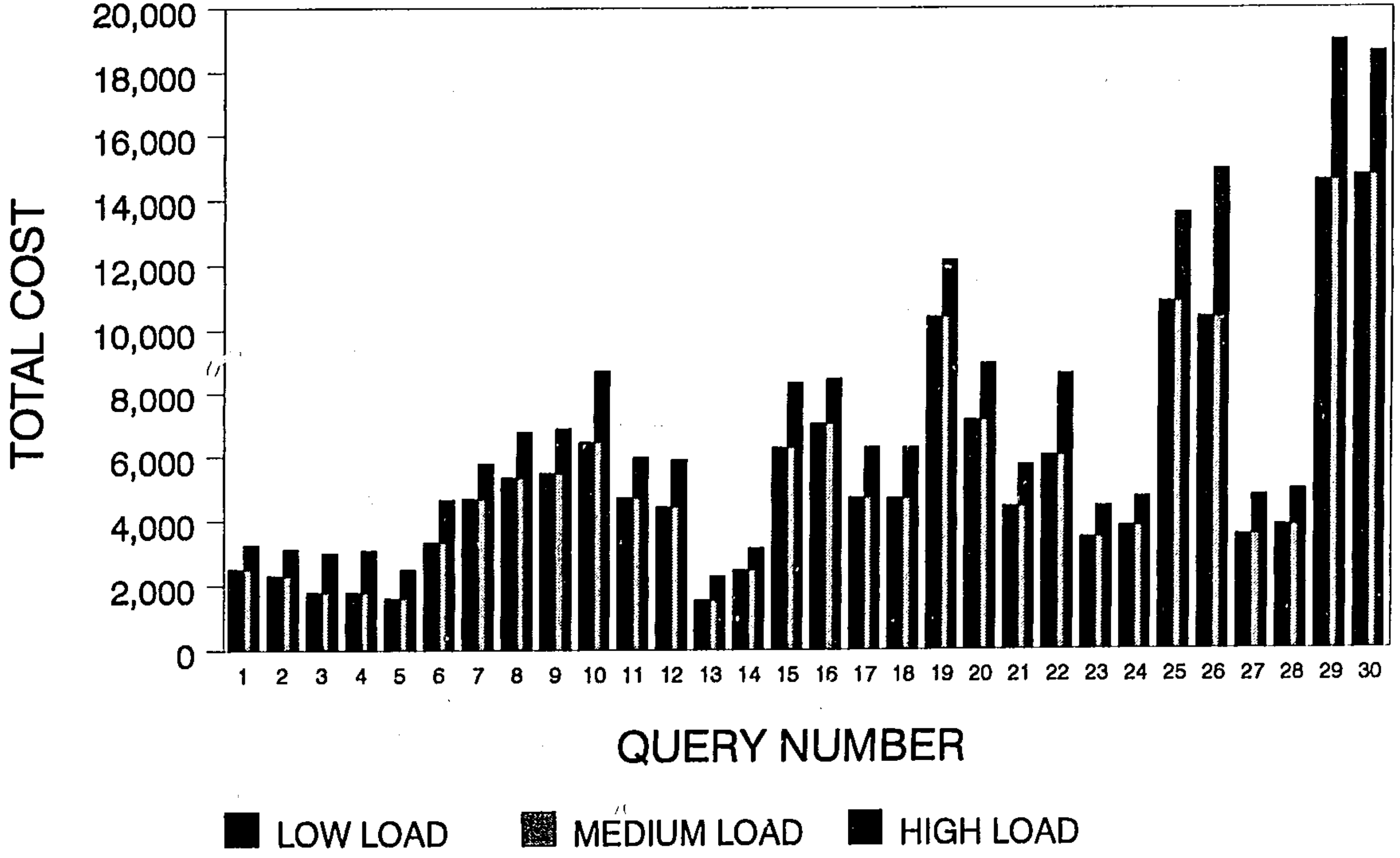
5.2.1 The Results for AHY (Total Cost Version)

Table 59 gives the total cost for the AHY heuristic (Total cost Version). The graph of Table 59 is shown on the next page.

Query Num.	LOW LOAD (50%-55%)	MED. LOAD 75%-80%)	HIGH LOAD (25%-30%)
1	2506	2506	3251.20
2	2300	2300	3130.00
3	1789	1789	3130.40
4	1789	1789	3091.20
5	1607	1607	2492.40
6	3341	3341	4667.60
7	4705	4705	5798.00
8	5368	5368	6792.40
9	5506	5506	6889.40
10	6460	6460	8720.00
11	4740	4740	5988.00
12	4452	4452	5910.40
13	1527	1527	2266.60
14	2475	2475	3156.90
15	6275	6275	8327.60
16	7037	7037	8455.00
17	4722	4722	6279.50
18	4696	4696	6270.00
19	10396	10396	12138.40
20	7142	7142	8909.20
21	4423	4423	5721.20
22	6013	6013	8594.20
23	3475	3475	4425.20
24	3803	3803	4711.40
25	10870	10870	13612.00
26	10390	10390	14954.00
27	3534	3534	4770.20
28	3822	3822	4960.00
29	14606	14606	18992.80
30	14760	14760	18635.20

Table 59 The Results for AHY (Total Cost Version)

THE COMPARSION OF ALGORITHM SERIAL AT LOW LOAD, MEDIUM LOAD & HIGH LOAD



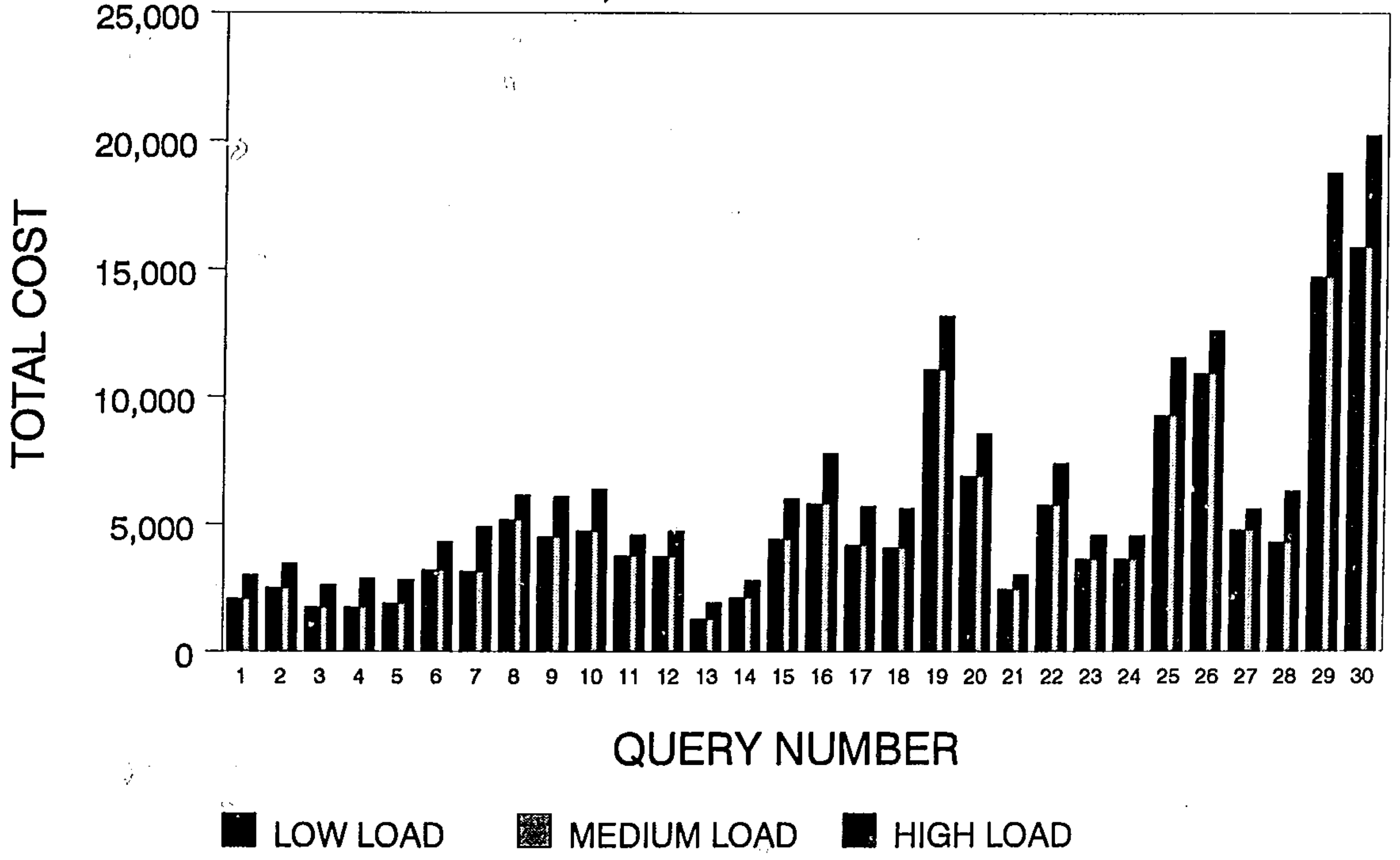
5.2.2 The Results for Total Cost Heuristic

Table 60 is the total cost for the Total Cost Heuristic. The graph of Table 60 is shown on the next page.

Query Num.	LOW LOAD (25% - 30%)	MED. LOAD (50% - 55%)	HIGH LOAD (50% - 55%)
1	2050	2050	2985.60
2	2460	2460	3425.20
3	1690	1690	2602.30
4	1690	1690	2850.00
5	1867	1867	2820.50
6	3197	3197	4316.20
7	3143	3143	4913.15
8	5195	5195	6174.30
9	4512	4512	6102.40
10	4750	4750	6416.50
11	3764	3764	4582.80
12	3722	3722	4745.50
13	1254	1254	1884.20
14	2112	2112	2798.00
15	4424	4424	6053.10
16	5830	5830	7813.70
17	4176	4176	5730.60
18	4066	4066	5640.30
19	11100	11100	13175.90
20	6920	6920	8589.60
21	2420	2420	3002.80
22	5776	5776	7415.22
23	3644	3644	4566.40
24	3602	3602	4555.90
25	9308	9308	11557.00
26	10927	10927	12598.90
27	4764	4764	5593.10
28	4275	4275	6295.56
29	14714	14714	18786.20
30	15859	15859	20250.20

Table 60 The Results for Total Cost Heuristic

THE COMPARSION OF HEURISTIC (TOTAL COST) AT LOW LOAD, MEDIUM LOAD & HIGH LOAD



5.3 RESULTS FOR RESPONSE TIME VERSION

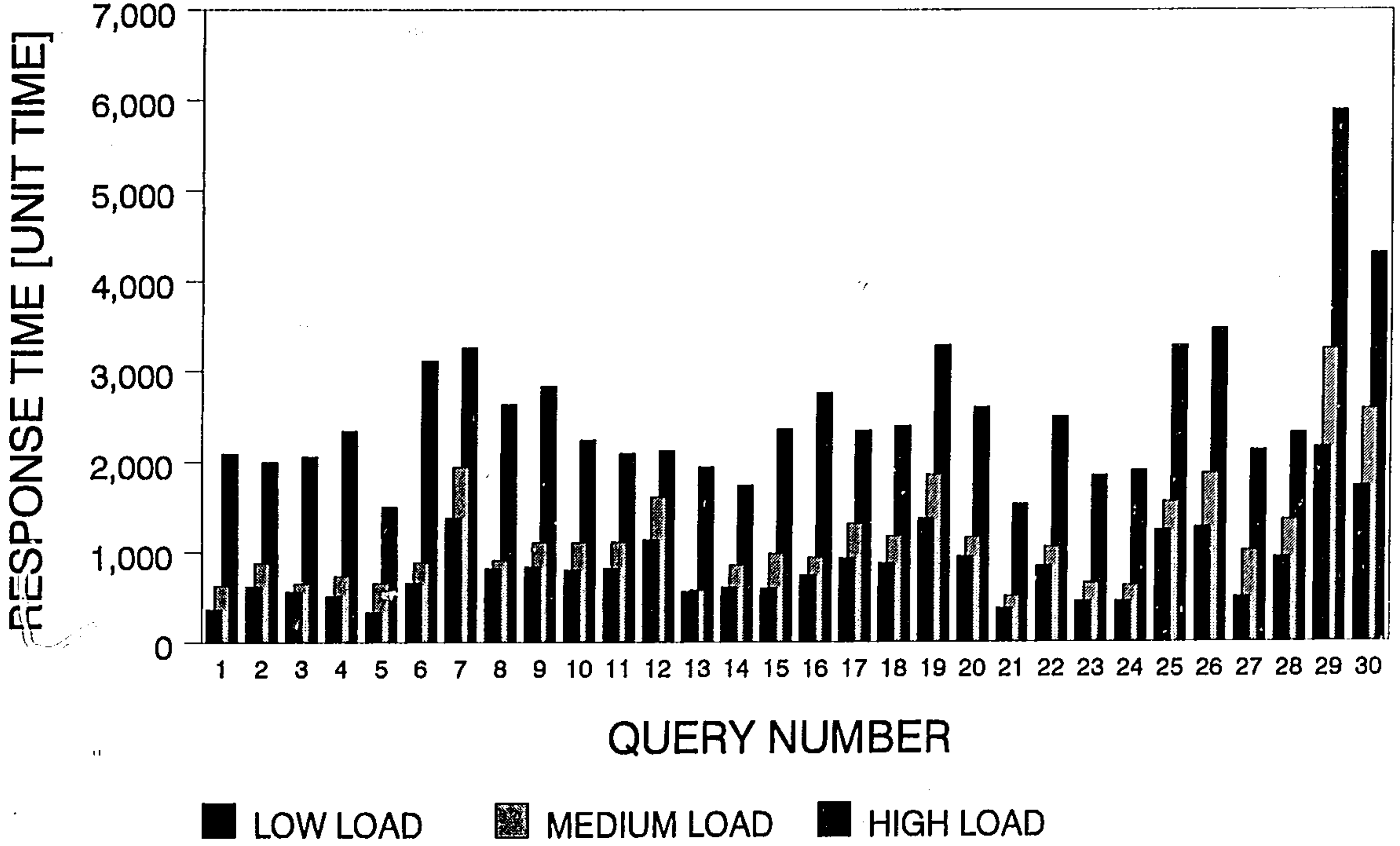
5.3.1 The Results for AHY (Response Time Version)

Table 61 is the response time for the AHY (Response Time Version). The graph of Table 61 is shown on next page.

Query	LOW LOAD	MED. LOAD	HIGH LOAD
Num.	(25% - 30%)	(50% - 55%)	(75% - 80%)
1	360.48	626.46	2091.13
2	620.76	877.17	2000.76
3	562.42	649.87	2056.80
4	512.82	735.64	2338.14
5	331.52	656.85	1501.66
6	654.66	881.91	3121.82
7	1382.28	1944.79	3263.16
8	819.50	904.43	2631.98
9	832.89	1107.34	2838.19
10	801.16	1103.24	2240.02
11	818.60	1111.64	2094.86
12	1134.13	1607.31	2125.86
13	559.45	576.37	1942.45
14	607.43	853.67	1743.83
15	598.35	978.68	2357.73
16	741.99	940.82	2760.47
17	926.70	1314.45	2341.99
18	873.35	1171.51	2389.02
19	1370.95	1857.36	3290.84
20	943.99	1157.85	2591.63
21	371.05	505.95	1524.45
22	843.35	1049.43	2488.04
23	447.20	656.12	1845.15
24	444.71	626.74	1896.99
25	1238.39	1551.80	3282.04
26	1272.89	1868.55	3475.90
27	491.42	1007.00	2310.74
28	928.80	1354.28	2370.44
29	2155.70	3243.74	5886.06
30	1716.70	2568.14	4305.17

Table 61 The Results for AHY (Response Time Version)

THE COMPARSION OF ALGORITHM PARALLEL AT LOW LOAD, MEDIUM LOAD & HIGH LOAD



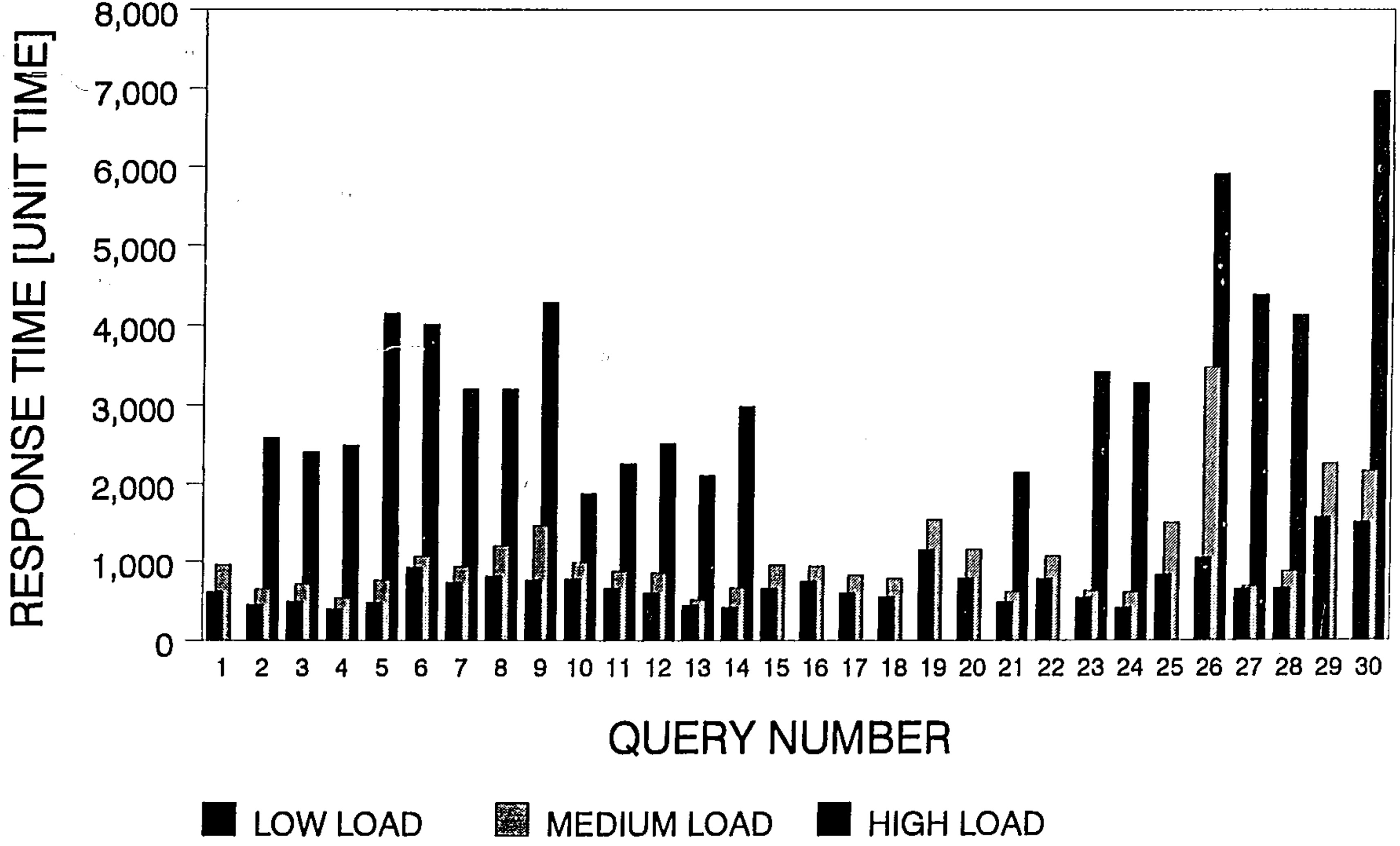
5.3.2 The Results for Response Time Heuristic

Table 62 is the response time for the Heuristic (Response Time Version). the graph of Table 62 is shown on next page.

Query Num.	LOW LOAD (25% - 30%)	MED. LOAD (50% - 55%)	HIGH LOAD (75% - 80%)
1	618.25	962.35	— — —
2	452.17	658.70	2592.43
3	499.32	723.90	2416.81
4	403.19	545.10	2492.72
5	482.29	771.36	4150.00
6	926.72	1070.01	4016.81
7	737.60	940.40	3210.70
8	821.35	1206.59	3210.71
9	772.24	1469.19	4289.03
10	785.00	990.54	1877.18
11	669.59	877.27	2262.39
12	608.17	860.13	2520.24
13	447.26	523.62	2109.72
14	418.76	675.51	2993.52
15	669.96	958.65	— — —
16	756.09	948.00	— — —
17	606.69	830.04	— — —
18	555.90	789.05	— — —
19	1154.94	1542.15	— — —
20	790.15	1159.56	— — —
21	487.43	617.46	2154.78
22	784.87	1069.99	— — —
23	545.96	629.92	3426.03
24	412.08	614.94	3284.19
25	831.67	1506.29	— — —
26	1042.13	1434.27	5917.80
27	656.42	690.87	4384.28
28	660.76	873.40	4130.72
29	1567.60	2256.92	— — —
30	1506.70	2158.86	6974.54

Table 62 The Results for Response Time Heuristic

THE COMPARSION OF HEURISTIC (RESP. TIME) AT LOW LOAD, MEDIUM LOAD & HIGH LOAD



5.4 CONCLUSIONS FOR THE TOTAL COST VERSION

This section compares the total cost between the AHY Algorithm (Total Cost Version) and the Total Cost Heuristic. The results are presented in table and graph format. Each table has six columns. The first column is the query number. The second column is the number of relations. The third is the number of attributes. The fourth is the total cost for the total cost AHY Algorithm. The fifth column is the total cost for the Heuristic Algorithm. The last column is the percentage different between the AHY Algorithm and the Heuristic.

If the value of the percentage difference is negative, it indicates the Heuristic reduces the cost by this percentage when compared to the AHY Algorithm. If the value of the percentage different is positive, it indicates that the Heuristic increases the cost by this percentage.

Section 5.4.1 compares the AHY Algorithm (Total Cost Version) and Total Cost Heuristic at the low load situation. Section 5.4.2 compares at the medium load situation. Section 5.4.3 compares at the high load situation.

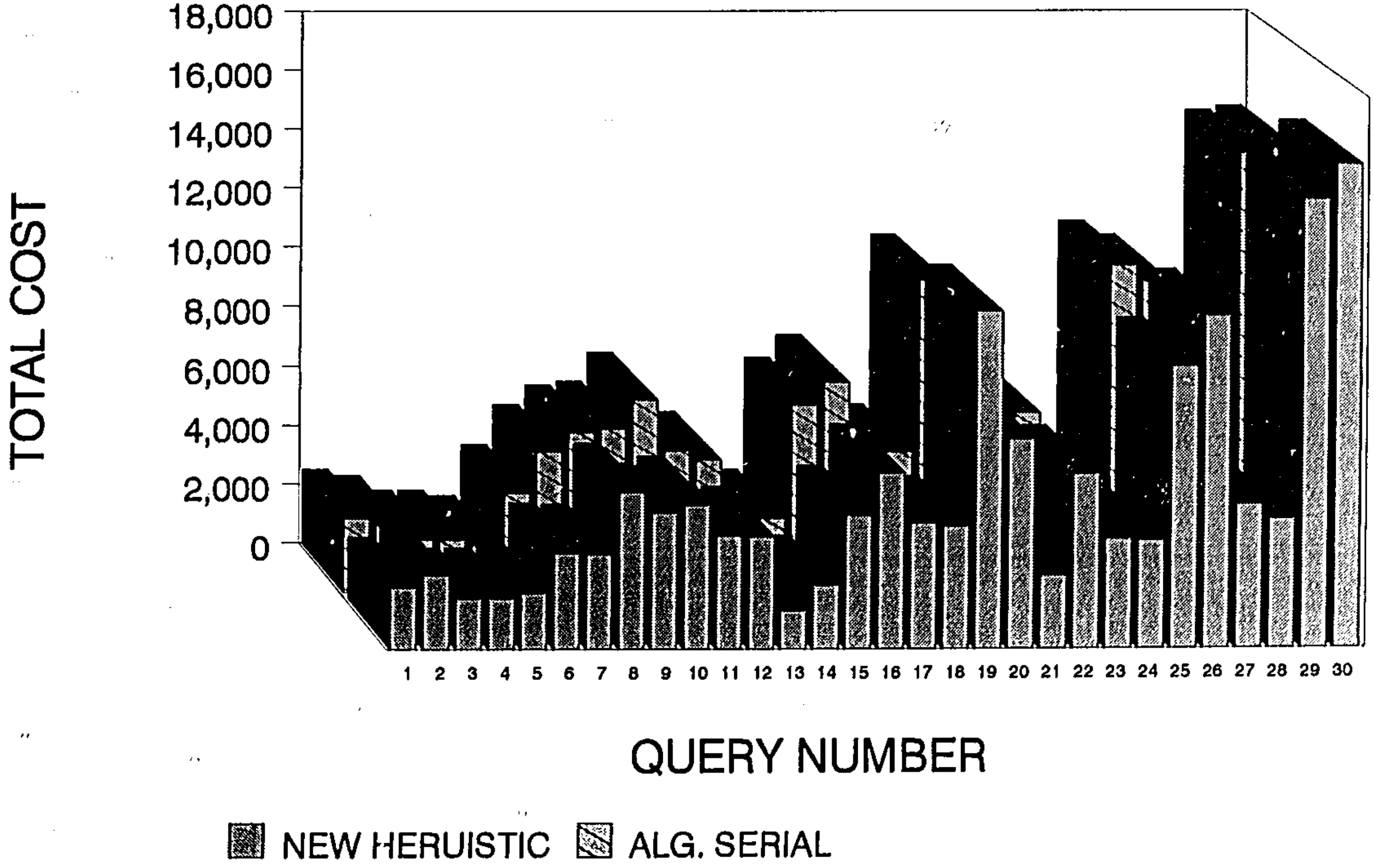
5.4.1 The Comparisons at Low Load Situation

Table 63 is the comparison between the AHY Algorithm (Total Cost Version) and the Total Cost Heuristic at the low load situation. The graph of Table 63 is shown on next page.

Query #	# of Rel.	# of Att.	AHY	HEURISTIC	% DIFF.
1	4	1	2506	2050	-18.20%
2	4	1	2300	2460	+6.96%
3	5	1	1789	1690	-5.53%
4	5	1	1789	1690	-5.53%
5	5	1	1607	1867	+16.18%
6	6	1	3341	3197	-4.31%
7	6	1	4705	3143	-33.20%
8	6	1	5368	5198	-3.22%
9	6	1	5506	4512	-18.05%
10	3	2	6460	4750	-26.47%
11	3	2	4740	3764	-20.59%
12	3	2	4452	3722	-16.40%
13	4	2	1527	1254	-17.88%
14	4	2	2475	2112	-14.67%
15	6	2	6275	4424	-29.50%
16	6	2	7037	5830	-17.15%
17	8	2	4722	4176	-11.56%
18	8	2	4696	4066	-13.42%
19	5	3	10396	11100	-6.77%
20	5	3	7142	6920	-3.11%
21	5	3	4423	2420	-45.29%
22	5	3	6013	5776	-3.94%
23	5	4	3475	3644	+4.86%
24	5	4	3803	3602	-5.29%
25	5	4	10870	9308	-14.36%
26	5	4	10390	10927	+5.17%
27	8	4	3534	4764	+34.80%
28	8	4	3822	4275	+11.85%
29	8	4	14606	14714	+0.74%
30	8	4	14760	15859	+7.45%

Table 63 The Comparisons between AHY Algorithm & Heuristic at Low Load

THE COMPARSION BETWEEN ALG. SERIAL & HEURISTIC AT LOW LOAD



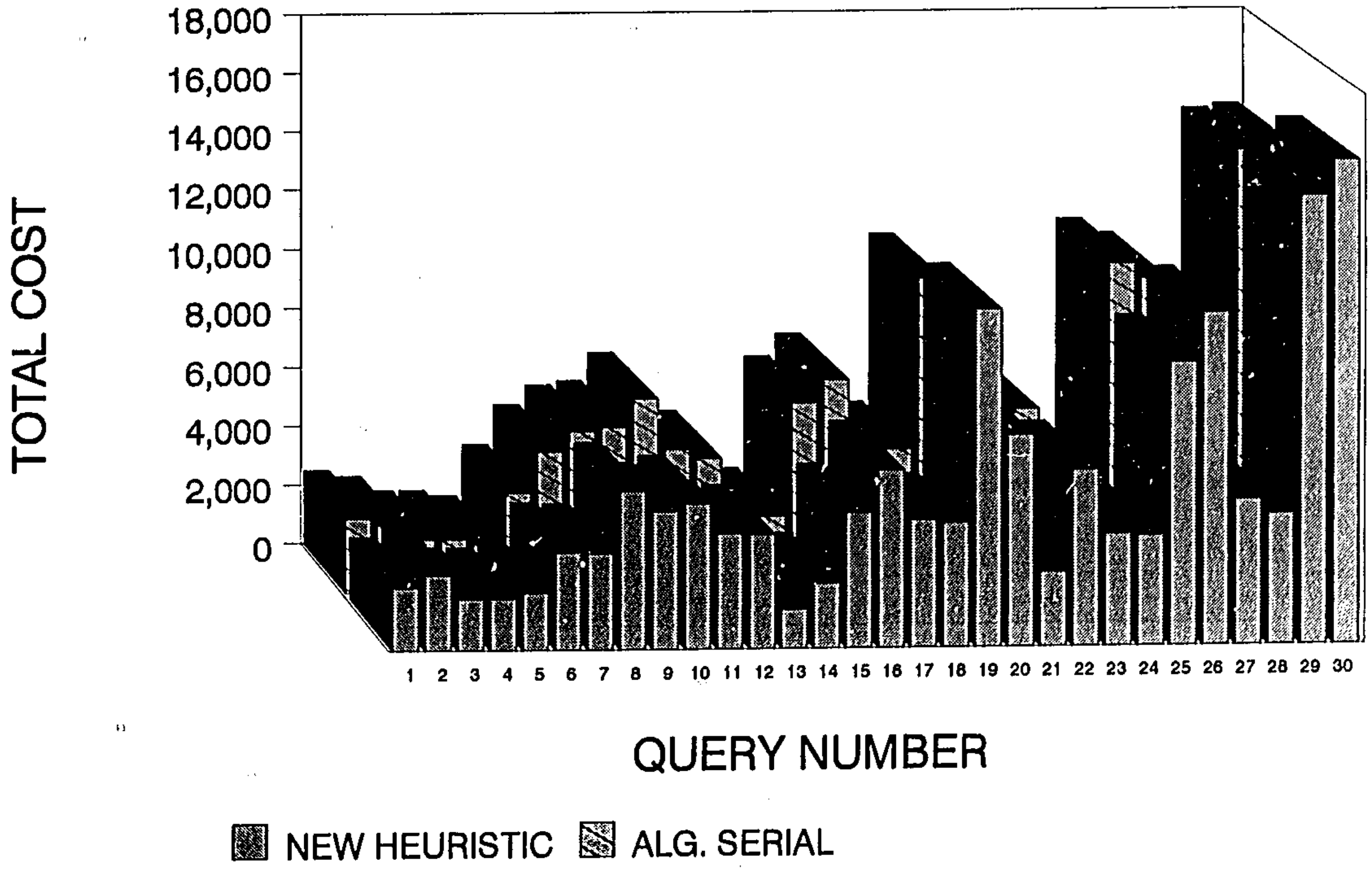
5.4.2 The Comparisons at Medium Load Situation

Table 64 is the comparison between AHY Algorithm (Total Cost Version) and the Total Cost Heuristic at the medium load situation. The graph of Table 64 is shown on next page.

Query #	# of Rel.	# of Att.	AHY	HEURISTIC	% DIFF.
1	4	1	2506	2050	-18.20%
2	4	1	2300	2460	+6.96%
3	5	1	1789	1690	-5.53%
4	5	1	1789	1690	-5.53%
5	5	1	1607	1867	+16.18%
6	6	1	3341	3197	-4.31%
7	6	1	4705	3143	-33.20%
8	6	1	5368	5195	-3.22%
9	6	1	5506	4512	-18.05%
10	3	2	6460	4750	-26.47%
11	3	2	4740	3764	-20.59%
12	3	2	4452	3722	-16.40%
13	4	2	1527	1254	-17.88%
14	4	2	2475	2112	-14.67%
15	6	2	6275	4424	-29.50%
16	6	2	7037	5830	-17.15%
17	8	2	4722	4176	-11.56%
18	8	2	4696	4066	-13.42%
19	5	3	10396	11100	+6.77%
20	5	3	7142	6920	-3.11%
21	5	3	4423	2420	-45.29%
22	5	3	6013	5776	-3.94%
23	5	4	3475	3644	+4.86%
24	5	4	3803	3602	-5.29%
25	5	4	10870	9308	-14.36%
26	5	4	10390	10927	+5.17%
27	8	4	3534	4764	+34.80%
28	8	4	3822	4275	+11.85%
29	8	4	14606	14714	+0.74%
30	8	4	14760	15859	+7.45%

Table 64 The Comparisons between AHY Algorithm & Heuristic at Medium Load

THE COMPARSION BETWEEN ALG. SERIAL & HEURISTIC AT MEDIUM LOAD

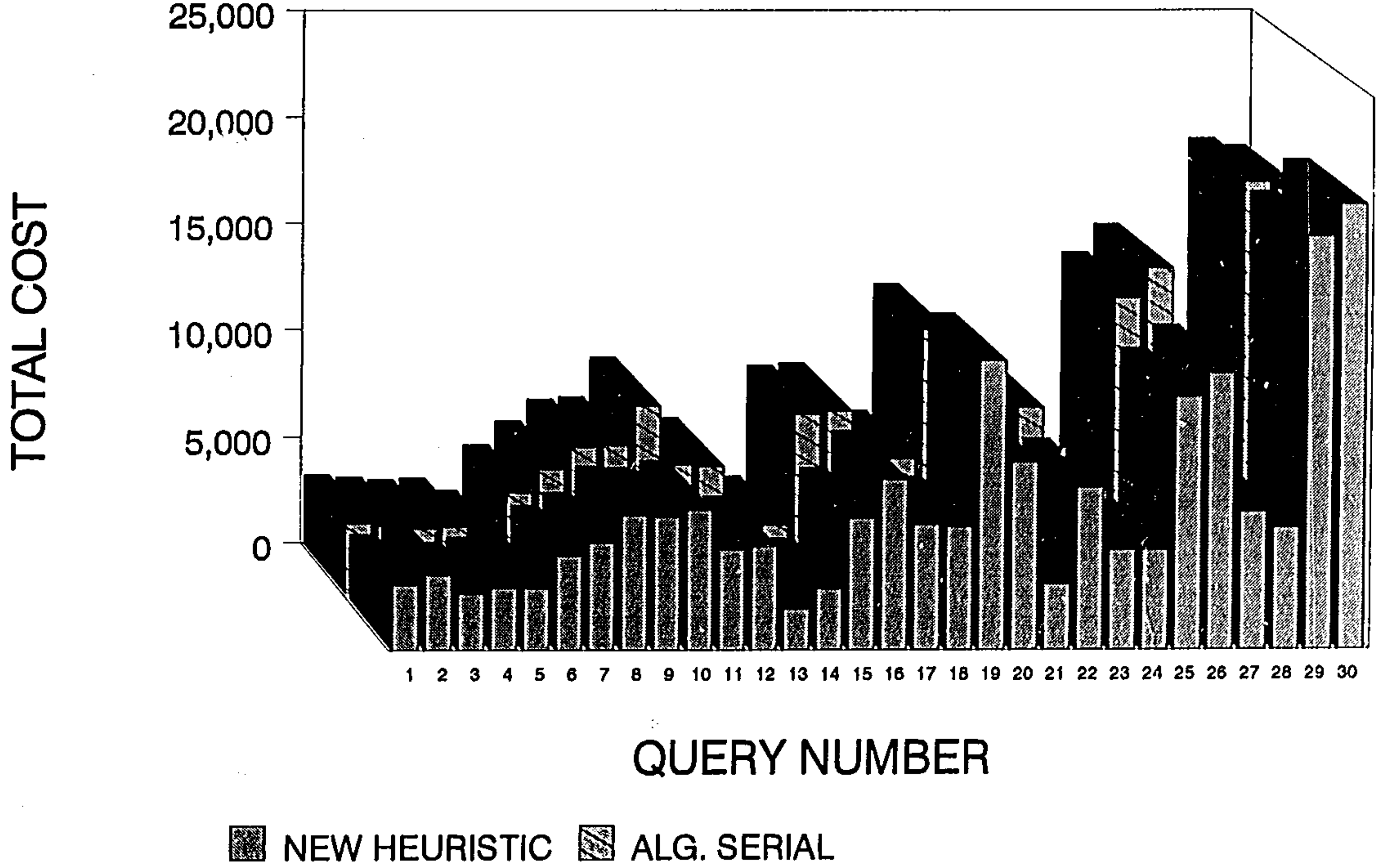


5.4.3 The Comparisons at High Load Situation

Table 65 is the comparison between AHY Algorithm (Total Cost Version) and the Total Cost Heuristic at the high load situation. The graph of Table 65 is shown on next page.

Query #	# of Rel.	# of Att.	AHY	HEURISTIC	% DIFF.
1	4	1	3251.20	2985.60	-8.17%
2	4	1	3130.00	3425.20	+9.43%
3	5	1	3130.40	2602.30	-16.87%
4	5	1	3091.20	2850.00	-7.80%
5	5	1	2492.40	2820.50	+13.16%
6	6	1	4667.60	4316.20	-7.53%
7	6	1	5798.00	4913.50	-15.26%
8	6	1	6792.40	6174.30	-9.10%
9	6	1	6889.40	6102.40	-11.42%
10	3	2	8720.00	6416.50	-26.42%
11	3	2	5988.00	4582.80	-23.47%
12	3	2	5910.40	4745.50	-19.71%
13	4	2	2266.60	1884.20	-16.87%
14	4	2	3156.90	2798.00	-11.37%
15	6	2	8327.60	6053.10	-27.31%
16	6	2	8455.00	7813.70	-7.58%
17	8	2	6279.80	5730.60	-8.75%
18	8	2	6270.00	5640.30	-10.04%
19	5	3	12138.40	13175.90	+8.55%
20	5	3	8909.20	8589.60	-3.59%
21	5	3	5721.20	3002.80	-47.51%
22	5	3	8594.20	7415.22	-13.72%
23	5	4	4425.20	4566.40	+3.19%
24	5	4	4711.40	4555.90	-3.30%
25	5	4	13612.00	11557.00	-15.10%
26	5	4	14954.00	12598.90	-15.75%
27	8	4	4770.20	6295.56	+31.98%
28	8	4	4960.00	5593.10	+12.76%
29	8	4	18992.80	18786.20	-1.09%
30	8	4	18635.20	20250.30	+8.67%

Table 65 The Comparisons between AHY Algorithm & Heuristic at High Load



THE COMPARSION BETWEEN
ALG. SERIAL & HEURISTIC AT HIGH LOAD

5.5 CONCLUSIONS FOR THE RESPONSE TIME VERSION

Section 5.5.1 compares the AHY Algorithm (Response Time Version) and the Response Time Heuristic at a low load. Section 5.5.2 compares at a medium load. Section 5.5.3 compares at a high load.

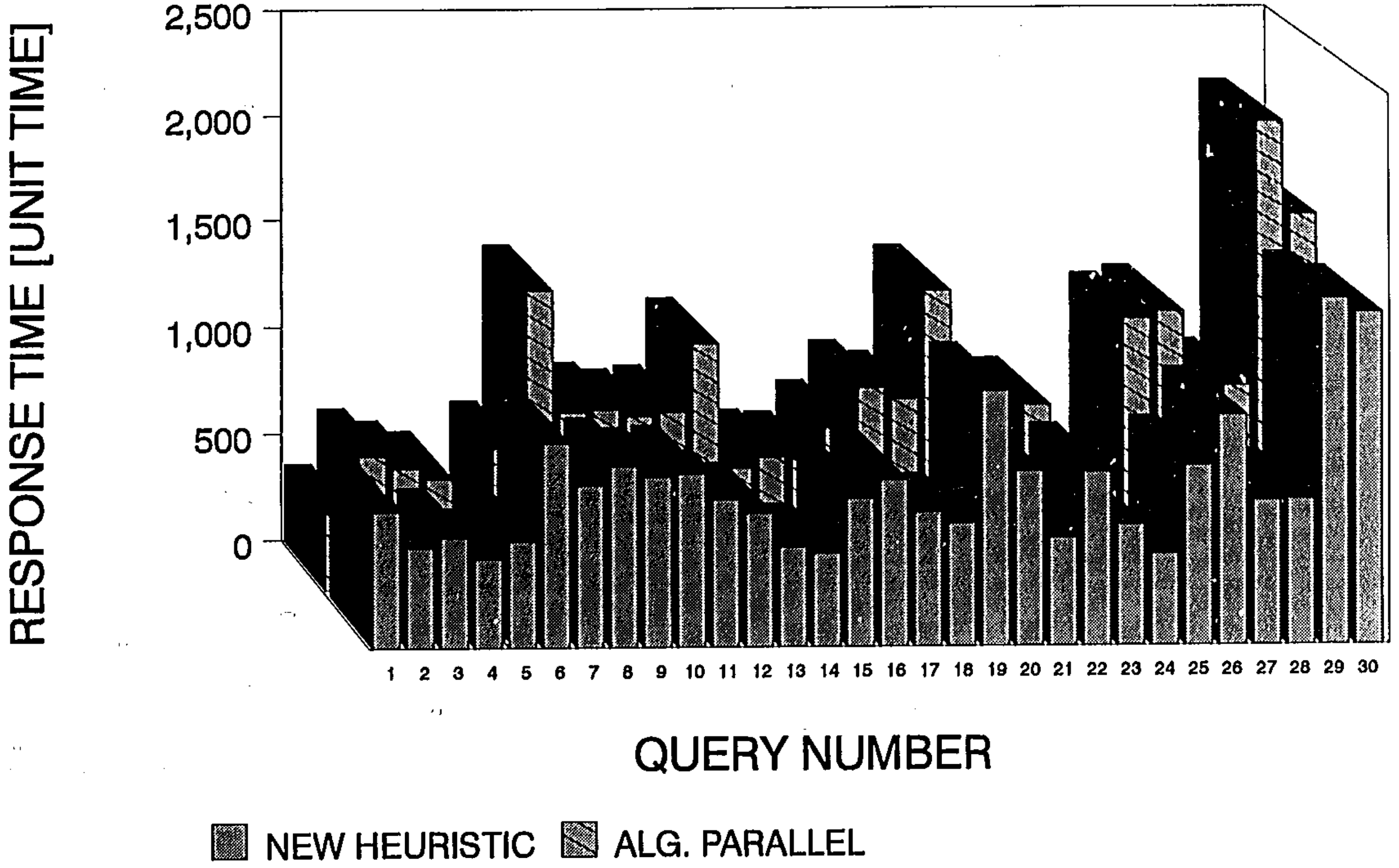
5.5.1 The Comparisons at Low Load Situation

Table 66 is the comparison between AHY Algorithm (Response Time Version) and the Response Time Heuristic at a low load. The graph of Table 66 is shown on next page.

Query #	# of Rel.	# of Att.	AHY	HEURISTIC	% DIFF.
1	4	1	360.48	618.25	+71.50%
2	4	1	620.76	452.17	-27.16%
3	5	1	562.42	499.32	-20.11%
4	5	1	512.82	403.19	-21.38%
5	5	1	331.52	482.29	+45.48%
6	6	1	654.66	926.72	+41.56%
7	6	1	1382.28	737.60	-46.64%
8	6	1	819.50	821.35	+0.23%
9	6	1	832.89	772.24	-7.28%
10	3	2	801.16	785.00	-2.02%
11	3	2	818.60	669.59	-18.20%
12	3	2	1134.13	608.17	-46.38%
13	4	2	559.45	447.26	-20.05%
14	4	2	607.43	418.78	-31.06%
15	6	2	598.35	669.96	+11.97%
16	6	2	741.99	756.09	+1.90%
17	8	2	926.70	606.69	-34.53%
18	8	3	873.35	555.90	-36.35%
19	5	3	1370.95	1154.94	-15.76%
20	5	3	943.99	790.15	-16.30%
21	5	3	371.05	487.43	+31.37%
22	5	3	843.35	784.87	-6.93%
23	5	4	447.20	545.96	+22.08%
24	5	4	444.71	412.08	-7.34%
25	5	4	1238.39	831.67	-32.84%
26	5	4	1272.89	1042.13	-18.13%
27	8	4	491.42	656.42	+33.58%
28	8	4	928.80	660.76	-28.86%
29	8	4	2155.70	1567.60	-27.28%
30	8	4	1716.70	1506.70	-12.23%

Table 66 The Comparison between AHY Algorithm & Heuristic at Low Load

THE COMPARSION BETWEEN ALG. PARALLEL & HEURISTIC AT LOW LOAD



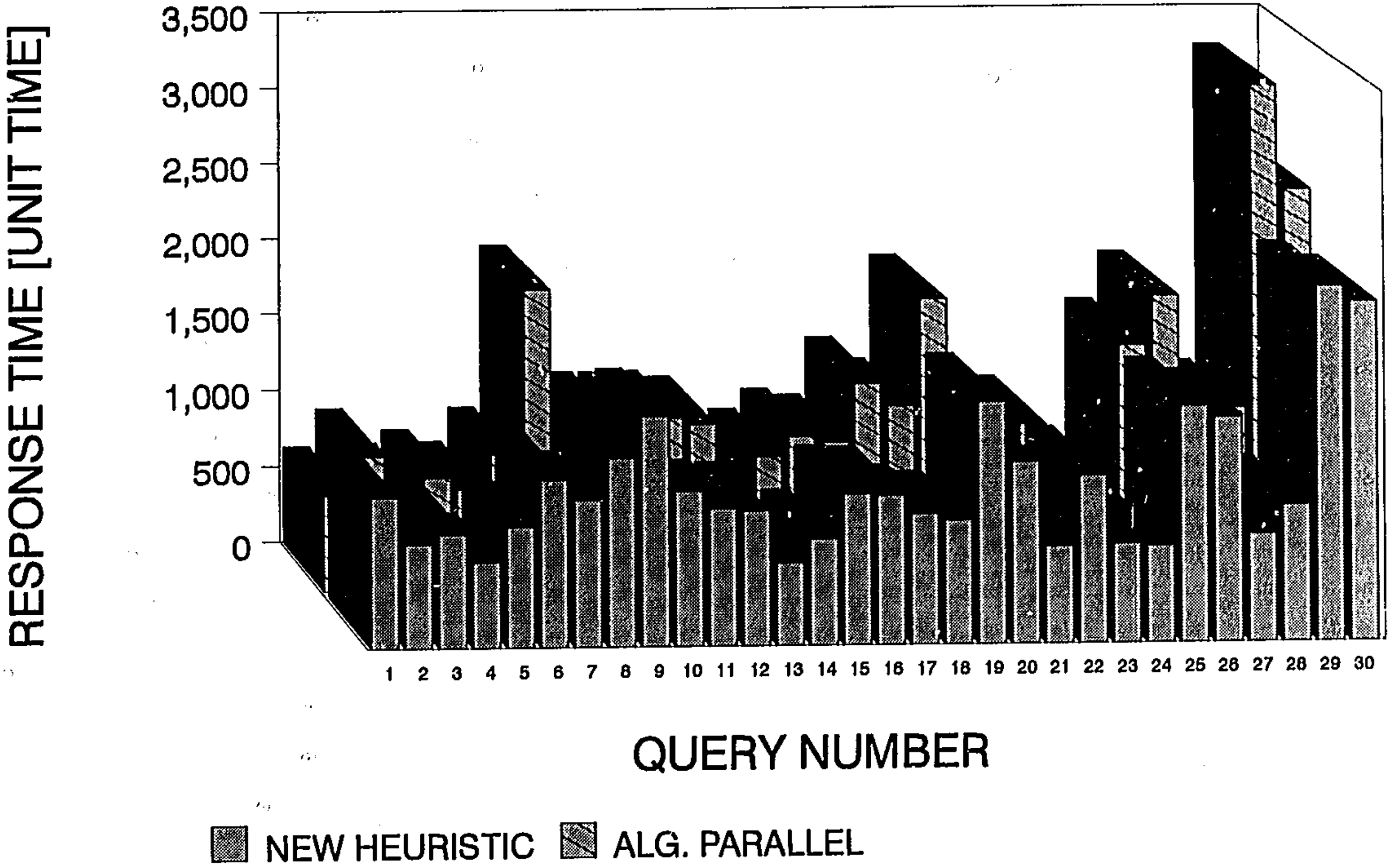
5.5.2 The Comparisons at Medium Load Situation

Table 67 is the comparison between AHY Algorithm (Response Time Version) and the Response Time Heuristic at a medium load. The graph of Table 67 is shown on next page.

Query #	# of Rel.	# of Att.	AHY	HEURISTIC	% DIFF.
1	4	1	626.46	962.35	+53.62%
2	4	1	877.17	658.70	-24.91%
3	5	1	649.87	723.90	+11.39%
4	5	1	735.64	545.10	-25.90%
5	5	1	656.85	771.36	+17.43%
6	6	1	881.91	1070.01	+21.33%
7	6	1	1944.79	940.40	-51.65%
8	6	1	904.43	1206.59	+33.41%
9	6	1	1107.34	1469.19	+32.68%
10	3	2	1103.24	990.54	-10.22%
11	3	2	1111.64	877.27	-21.08%
12	3	2	1607.31	860.13	-46.49%
13	4	2	576.37	523.62	-9.15%
14	4	2	853.67	675.51	-20.89%
15	6	2	978.68	958.65	-2.05%
16	6	2	940.82	948.00	+0.76%
17	8	2	1314.45	830.04	-36.85%
18	8	2	1171.51	789.05	-32.65%
19	5	3	1857.36	1542.15	-16.97%
20	5	3	1157.85	1159.56	+0.15%
21	5	3	505.95	617.46	+22.04%
22	5	3	1049.43	1069.99	+1.96%
23	5	4	656.12	629.92	-3.99%
24	5	4	626.74	614.94	-1.88%
25	5	4	1551.80	1506.29	-2.93%
26	5	4	1868.55	1434.22	-23.25%
27	8	4	1007.60	690.87	-31.43%
28	8	4	1354.28	873.40	-35.50%
29	8	4	3243.74	2256.92	-30.42%
30	8	4	2568.14	2158.86	-15.94%

Table 67 The Comparison between AHY Algorithm & Heuristic at Medium Load

THE COMPARSION BETWEEN ALG. PARALLEL & HEURISTIC AT MEDIUM LOAD



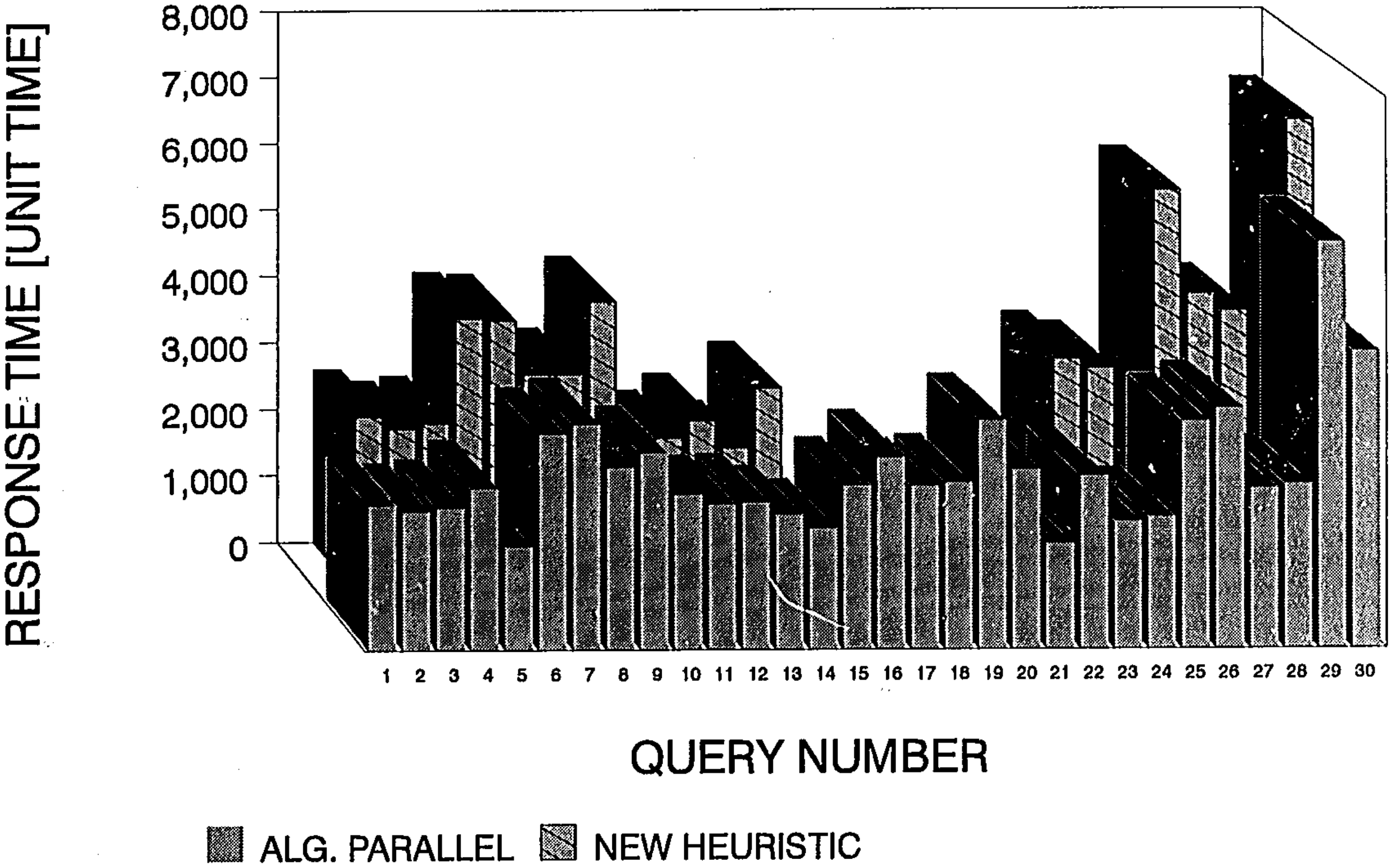
5.5.3 The Comparisons at High Load Situation

Table 68 is the comparison between AHY Algorithm (Response Time Version) and the Response Time Heuristic at the high load situation. The graph of Table 68 is shown on next page. Some queries were not executed since it was clear that the Response Time Heuristic was performing very badly at a high load.

Query #	# of Rel.	# of Att.	AHY	HEURISTIC	% DIFF.
1	4	1	2091.13	— — —	— — —
2	4	1	2000.76	2592.43	+29.57%
3	5	1	2056.80	2416.81	+17.50%
4	5	1	2338.14	2497.72	+6.83%
5	5	1	1501.66	4150.00	+176.36%
6	6	1	3121.82	4016.81	+28.67%
7	6	1	3262.16	3210.70	+1.61%
8	6	1	2631.98	3210.71	+22.00%
9	6	1	2838.19	4289.03	+51.12%
10	3	2	2240.02	1877.18	-16.20%
11	3	2	2094.86	2262.39	+8.00%
12	3	2	2125.86	2520.24	+18.55%
13	4	2	1942.45	2109.72	+8.61%
14	4	2	1743.83	2993.52	+71.66%
15	6	2	2357.73	— — —	— — —
16	6	2	2760.47	— — —	— — —
17	8	2	2341.99	— — —	— — —
18	8	2	2389.02	— — —	— — —
19	5	3	3290.84	— — —	— — —
20	5	3	2591.63	— — —	— — —
21	5	3	1524.45	2154.78	+41.45%
22	5	3	2488.04	— — —	— — —
23	5	4	1845.15	3426.03	+85.78%
24	5	4	1896.99	3284.19	+73.13%
25	5	4	3282.04	— — —	— — —
26	5	4	3475.90	5917.80	+70.25%
27	8	4	2310.74	4384.28	+89.73%
28	8	4	2370.44	4130.72	+74.26%
29	8	4	5886.06	— — —	— — —
30	8	4	4305.17	6974.54	+62.00%

Table 68 The Comparison between AHY Algorithm & Heuristic at High Load

THE COMPARSION BETWEEN ALG. PARALLE!. & HEURISTIC AT HIGH LOAD



CHAPTER 6 CONCLUSIONS

There are many possible strategies for processing a query, especially complex queries, and a substantial amount of time and effort is needed to select an optimal strategy. In the earlier distributed DBMSs, the objective was to minimize the transmission costs in terms of the message size. However, there are some other factors such as the ***Network Load*** which should also be considered.

6.1 FINDINGS RELATED TO THE TOTAL COST HEURISTIC

6.1.1 THE SAME QUERY PROCESS AT DIFFERENT INSTANT

At low and medium load the hop table remains unchanged as the routing remains the same. However, at high loads the routing and hop table change. If the traffic load does not change very much, then the total cost does not change very much. For example, the total cost is not changed very much from T_1 to T_4 in Query 1 in Table 58. However, the total cost is changed to 2946 at T_5 which means that the traffic has changed very much.

If the traffic of the network is increased, it may cause the Total Cost Heuristic to produce a higher total cost for that query.

6.1.2 EFFECT ON THRESHOLD VALUE

The ***Threshold Value*** plays a very important role in the Total Cost Heuristic. If the heuristic uses a bad threshold value, then the total cost of the query will be worse than Algorithm AHY. There is no unique way to find the best threshold value for a given query. The best method to obtain a threshold value which is close to the optimum one is by a "Trial and Error" method. The term "optimum threshold value" is a threshold value which gives the smallest total cost for the same query.

In general, there are five different cases to consider :

6.1.2.1 ONLY ONE OPTIMUM THRESHOLD VALUE

When there is only one optimum threshold value, then the total cost will be increased if the threshold value is bigger or smaller than the optimum threshold value. Some examples are given on Table 69.

6.1.2.2 SEVERAL OPTIMUM THRESHOLD VALUES

The second case is that there are several optimum threshold values in a query. Therefore, more than one threshold value can give the optimum total cost for that query. Some examples are given on Table 70.

6.1.2.3 MANY OPTIMUM THRESHOLD VALUES

The third case is that there are many threshold values that can produce the optimum total cost in a given query. However, this kind of case does not happen very often. It only happens in about 5% of the queries. Some examples are given on Table 71.

6.1.2.4 THE OPTIMUM THRESHOLD VALUE IS ZERO

The fourth case is that the optimum threshold value is zero. The total cost will increase exponentially if the threshold value is increased. Table 72 shows some examples of this case.

6.1.2.5 SOME SPECIAL CASES

The fifth case is where a threshold value can suddenly increase the total cost of a query. If that threshold value is used, then the total cost will suddenly increase. However, this kind of case does not happen very often. Some examples are shown on Table 73.

The threshold value plays a very important role in the Total Cost Heuristic. It can directly affect the result of the total cost of a given query. Therefore, the Total Cost Heuristic needs to use "Trial and Error" method to find the optimum threshold value for that query.

THRESHOLE VALUE	TOTAL COST for Query 3 at Table 7	TOTAL COST for Query 9 at Table 4	TOTAL COST for Query 17 at Table 3
0	2496	5910	7965
5	2496	5946	8197
10	2496	5946	8197
20	2451	5922	8197
30	2370	5922	8197
40	2370	5922	8251
50	2345	5922	7610
60	2422	5922	7958
70	2422	5922	7958
80	2440	5901	7958
100	2440	6612	7948
110	2440	6612	7948
120	2440	6608	7975
130	2440	6564	7965
140	2440	6564	7965
150	2440	6564	7945
160	2440	6564	7789
170	2440	6614	7876
180	2440	6614	7876
190	2440	6614	7876
200	2440	6614	7876
300	2415	7024	10943

Table 69 Showing Only One Optimum Threshold Value

THRESHOLE VALUE	TOTAL COST for Query 6 at Hop Table 3	TOTAL COST for Query 8 at Hop Table 5	TOTAL COST for Query 16 at Hop Table 4
0	4373	6333	7679
5	4373	6333	7679
10	4373	6333	7679
20	4373	6333	7679
30	4373	6333	7420
40	4373	6333	7420
50	4284	6333	7420
60	4284	6333	7420
70	4284	6333	7420
80	4284	6333	7420
100	4284	5974	7420
110	4284	5974	7420
120	4284	5974	7473
130	4284	5974	7473
140	4346	5974	7473
150	4346	5974	7728
160	4346	5974	7728
170	4346	5974	7728
180	4346	6046	7728
190	4396	6046	7728
200	4396	6046	7728
300	4414	6464	10549

Table 70 Showing Several Optimum Threshold Values

THRESHOLE VALUE	TOTAL COST for Query 2 at Hop Table 5	TOTAL COST for Query 8 at Hop Table 7	TOTAL COST for Query 20 at Hop Table 10
0	4084	7079	7336
5	4084	7079	7336
10	4084	7079	7336
20	4084	7079	7336
30	4084	7079	7336
40	4084	7079	7336
50	4084	7079	7336
60	4084	7079	7336
70	4084	7079	7336
80	4084	7079	7336
100	4084	7079	7336
110	4084	7079	7336
120	4084	7079	7336
130	4084	7079	7336
140	4084	7079	7336
150	4084	7079	7336
160	4084	7079	7336
170	4084	7079	7336
180	4084	7079	7336
190	4084	7079	7336
200	4084	7079	7336

Table 71 Showing Many Optimum Threshold Value

THRESHOLE VALUE	TOTAL COST for Query 11 at Hop Table 4	TOTAL COST for Query 10 at Hop Table 7	TOTAL COST for Query 19 at Hop Table 3
0	4404	7380	12600
5	5344	8630	16420
10	6284	9880	20240
20	7224	11130	24060
30	8164	12380	27880
40	9104	13630	31700
50	10044	14880	35520
60	10984	16130	39340
70	11924	17380	43160
80	12864	18630	46980
100	13804	19880	50800
110	14744	21130	54620
120	15684	22380	58440
130	16624	23630	62260
140	17564	24880	66080
150	18504	26130	69900
160	19444	27380	73720
170	20384	28630	77540
180	21324	29880	81360
190	22264	31130	85180
200	23204	32380	89120
300	24144	33995	93052

Table 72 Showing Zero is the Optimum Threshold Value

THRESHOLE VALUE	TOTAL COST for Query 3 at Hop Table 6	TOTAL COST for Query 16 at Hop Table 5
0	2578	6741
5	2578	6741
10	2578	6741
20	2573	6741
30	2528	6670
40	2528	6670
50	2532	6670
60	2532	6670
70	12573	6655
80	2457	6708
100	2457	17388
110	2457	8788
120	2457	8788
130	2457	8948
140	2457	8948
150	2457	8948
160	2457	8948
170	2457	8948
180	2457	9264
190	2457	6605
200	2457	6762

Table 73 Showing the Special Case on Threshold Value

6.1.3 FINDINGS IN LOW LOAD SITUATION

Table 74 shows some statistics for the Total Cost Heuristic in a low load situation. The first row shows that the total number of queries where the performance is between 0 and 5 percent worse than AHY. A negative percentage indicates that the heuristic performed better than AHY.

73.33% of queries have some improvement. About 46.67% queries have more than 10% improvement and about 16.67% queries have more than 20% improvement. There is only one query where the performance is very poor. The heuristic performs well at low load.

	TOTAL NUMBER OF QUERY
0% ≤ % DIFFERENT ≤ +5%	2
+ 5% < % DIFFERENT ≤ +10%	3
+10% < % DIFFERENT ≤ +20%	2
+20% < % DIFFERENT	1
0% ≤ % DIFFERENT ≤ -5%	4
-5% < % DIFFERENT ≤ -10%	4
-10% < % DIFFERENT ≤ -20%	9
-20% < % DIFFERENT	5

Table 74 Some Statistics for Total Cost Heuristic in Low Load Situation

6.1.4 FINDINGS IN MEDIUM LOAD SITUATION

Table 75 shows some statistics for the Total Cost Heuristic in a medium load situation.

70% of queries have some improvement. About 46.67% of queries have more than 10% improvement and about 20% of queries have more than 20% improvement. The result is even better than in the low load situation. There is only one query which does not perform well. The heuristic performs well at a medium load.

	TOTAL NUMBER OF QUERY
0% ≤ % DIFFERENT ≤ +5%	2
+ 5% < % DIFFERENT ≤ +10%	4
+10% < % DIFFERENT ≤ +20%	2
+20% < % DIFFERENT	1
0% ≤ % DIFFERENT ≤ -5%	4
-5% < % DIFFERENT ≤ -10%	3
-10% < % DIFFERENT ≤ -20%	8
-20% < % DIFFERENT	6

Table 75 Some Statistics for Total Cost Heuristic in Medium Load Situation

6.1.5 FINDINGS IN HIGH LOAD SITUATION

Table 76 shows some statistics for the Total Cost Heuristic in a high load situation.

76.67% of queries have some improvement. The result is even better than at the low and medium load situation. About 46.67% of queries have more than 10% improvement and about 13.33% of queries have more than 20% improvement. There is only one query which performs badly. Therefore, the performance of this heuristic is also quite good in a high load situation.

	TOTAL NUMBER OF QUERY
0% ≤ % DIFFERENT ≤ +5%	0
+ 5% < % DIFFERENT ≤ +10%	4
+10% < % DIFFERENT ≤ +20%	2
+20% < % DIFFERENT	1
0% ≤ % DIFFERENT ≤ -5%	2
-5% < % DIFFERENT ≤ -10%	7
-10% < % DIFFERENT ≤ -20%	10
-20% < % DIFFERENT	4

Table 76 Some Statistics for Total Cost Heuristic in High Load Situation

6.2 FINDINGS RELATED TO THE RESPONSE TIME HEURISTIC

6.2.1 FINDINGS IN LOW LOAD SITUATION

Table 77 shows some statistics for the Response Time Heuristic in low load situation.

70% of queries have some improvement. About 56.67% of queries have more than 10% improvement, about 40% of queries have more than 20% improvement and about 26.67% of queries have more than 30% improvement. However, there are five queries where the response time is higher. Therefore, some of the queries have a significance improvement in the response time, but a small number of the queries have a higher response time.

	TOTAL NUMBER OF QUERY
0% ≤ % DIFFERENT ≤ +5%	2
+ 5% < % DIFFERENT ≤ +10%	0
+10% < % DIFFERENT ≤ +20%	1
+20% < % DIFFERENT ≤ +30%	1
+30% < % DIFFERENT	5
0% ≤ % DIFFERENT ≤ -5%	1
-5% < % DIFFERENT ≤ -10%	3
-10% < % DIFFERENT ≤ -20%	5
-20% < % DIFFERENT ≤ -30%	4
-30% < % DIFFERENT	8

Table 77 Some Statistics for Response Time Heuristic in Low Load Situation

6.2.2 FINDINGS IN MEDIUM LOAD SITUATION

Table 78 shows some statistics for the Response Time Heuristic in medium load situation.

66.66% of queries have some improvement. About 50% of queries have more than 10% improvement, about 40% of queries have more than 20% improvement and about 23.33% of queries have more than 30% improvement. However, three queries have increased response time. Therefore, the performance of this heuristic at medium load is similar to the low load situation.

	TOTAL NUMBER OF QUERY
0% ≤ % DIFFERENT ≤ +5%	3
+ 5% < % DIFFERENT ≤ +10%	0
+10% < % DIFFERENT ≤ +20%	2
+20% < % DIFFERENT ≤ +30%	2
+30% < % DIFFERENT	3
0% ≤ % DIFFERENT ≤ -5%	4
-5% < % DIFFERENT ≤ -10%	1
-10% < % DIFFERENT ≤ -20%	3
-20% < % DIFFERENT ≤ -30%	5
-30% < % DIFFERENT	7

Table 78 Some Statistics for Response Time Heuristic in Medium Load Situation

6.2.3 FINDINGS IN HIGH LOAD SITUATION

Table 79 shows some statistics for the Response Time Heuristic in a high load situation.

About 95% of the queries have an increased response time, about 80% of the queries have a 20% increase in the response time and about 50% of the queries have a 30% increase in the response time. Therefore, the performance of this heuristic is much worst than the AHY Algorithm (Response Time Version).

	TOTAL NUMBER OF QUERY
0% ≤ % DIFFERENT ≤ +5%	0
+ 5% < % DIFFERENT ≤ +10%	3
+10% < % DIFFERENT ≤ +20%	3
+20% < % DIFFERENT ≤ +30%	3
+30% < % DIFFERENT	10
0% ≤ % DIFFERENT ≤ -5%	0
-5% < % DIFFERENT ≤ -10%	0
-10% < % DIFFERENT ≤ -20%	1
-20% < % DIFFERENT ≤ -30%	0
-30% < % DIFFERENT	0

Table 79 Some Statistics for Response Time Heuristic in High Load Situation

6.2.3.1 PROBLEMS WITH THE RESPONSE TIME HEURISTIC

Sometimes the heuristic generates a sequential schedule. Consider the following diagram as an example :

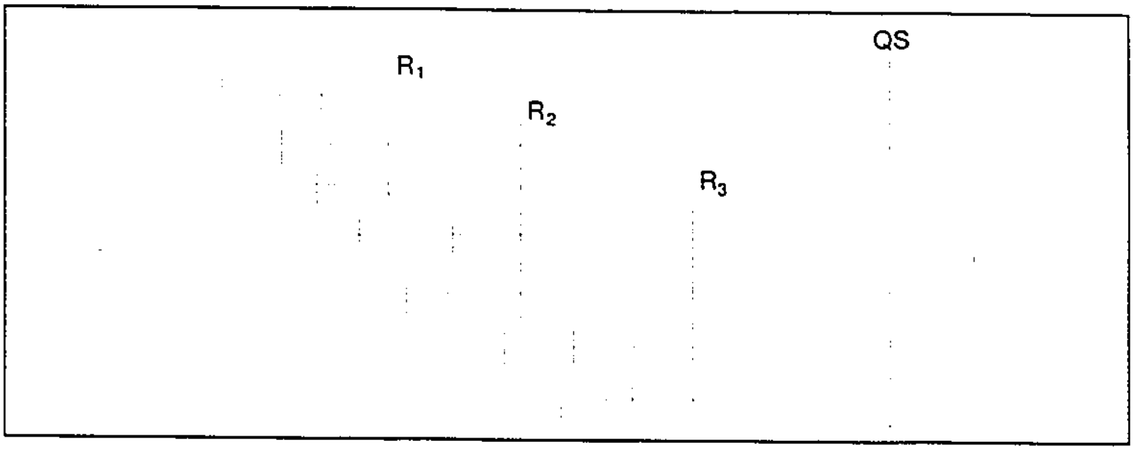


Figure 38 SEQUENTIAL SCHEDULE

Sometime the heuristic will use the updated attributes to reduce other relations. In the above schedule, R_2 will not start until R_1 is finished. R_3 will not start until R_2 is finished. If this kind of situation is carried on, then the response time of that query will be very long. That is why the performance is so bad in this heuristic.

Another problem of this heuristic is that the relations are not reduced encugh. When the heuristic finds that the delay to transmit the packets is very high. then the number of transmissions is reduced. When the number of transmissions is reduced then the number of Semi-Joins is also reduced. This results in large relations being sent to the Query Site. The following table shows the percentage differences on the number of transmissions to send the attributes of AHY Algorithm and the Response Time Heuristic :

Query Num.	AHY		HEURISTIC		% DIFF.
	RESPONSE TIME	# of Transmission	RESPONSE TIME	# of Transmission	
1	2091.13	15	— — —	8	-46.67%
2	2000.76	15	2592.43	8	-46.67%
3	2056.80	18	2416.81	19	+5.56%
4	2338.14	18	2497.72	18	0.00%
5	1501.66	18	— — —	5	-72.22%
6	3121.82	38	4016.81	23	-39.47%
7	3263.16	38	— — —	26	-31.58%
8	2631.98	38	— — —	18	-52.63%
9	2838.19	38	4289.03	13	-65.79%
10	2240.02	11	1877.18	4	-63.64%
11	2094.86	11	2262.39	6	-45.45%
12	2125.86	11	2520.24	6	-45.45%
13	1942.45	18	2109.72	12	-33.33%
14	1743.83	18	2993.52	8	-55.56%
15	2357.73	25	— — —	19	-24.00%
16	2760.47	25	— — —	14	-44.00%
17	2341.99	46	— — —	23	-50.00%
18	2389.02	46	— — —	17	-63.04%
19	3290.84	21	— — —	10	-52.40%
20	2591.63	18	— — —	10	-44.44%
21	1524.45	18	2154.78	15	-16.67%
22	2488.04	21	— — —	15	-28.57%
23	1845.15	22	— — —	10	-54.55%
24	1896.99	22	3284.19	6	-86.36%
25	3282.04	16	— — —	9	-43.75%
26	3475.90	15	— — —	16	+6.67%
27	2310.74	39	4384.28	31	-20.51%
28	2370.44	43	4130.72	34	-20.93%
29	5886.06	41	— — —	28	-31.71%
30	4305.17	39	— — —	22	-43.60%

Table 80 Percentage Different on The Number of Transmissions

6.3 FUTURE WORK

6.3.1 RECOMMENDATIONS ON TOTAL COST HEURISTIC

The Total Cost Heuristic shows some improvement in more than 3/4 of all queries in all load situations. The results are quite good. However, there are some suggestions on the Total Cost Heuristic :

1. Use the "Trial and Error" method to obtain the optimum threshold value.
2. Re-arrange the order of the four phases in order to generate a new sequence of transmissions. This new schedule may even have a lower total cost.

6.3.2 RECOMMENDATIONS ON RESPONSE TIME HEURISTIC

1. Modify the heuristic to avoid the problem of "sequential schedule".
2. Reduce the size of the relations by doing more semi-joins.

APPENDIX A
THE HYPERCUBE TESTBED

In this thesis, a simulator of a Hypercube network which is written in Simscript II.5 has been developed. In this simulator, it can simulate a hypercube network with up to 64 nodes. Some packets are generated to form the background traffic. The total cost and the response time of the schedules can be measured in this simulator. The description of this simulator is discussed in this appendix.

A.1 AN OVERVIEW OF THE HYPERCUBE TOPOLOGY

The term topology, in the context of a communications network refers to the way in which the end points or nodes of the network are interconnected.

The sites in the system can be connected in a number of different ways, such as star, hypercube, ring, mesh, tree and so on. The main differences between them are as follows:

1. Installation Cost : The cost to install the physical link between the sites.
2. Communication Cost : The Cost in time and money to send a message between two sites.
3. Reliability : The frequency with which a link or a site fails.
4. Availability : The portion of the data that can be accessed even when some links or sites fail.

A.1.1 NETWORK TOPOLOGY

It is not feasible to fully connect all sites together in most cases . In order to solve this problem, some other network topologies are suggested that can share the transmission links. Therefore, the installation cost can be reduced significantly.

One network topology that has been proposed is called the Hypercube Topology. The advantages of a Hypercube Topology are low communication diameter, high connectivity, fault tolerance. This architecture is scalable to thousands of nodes. Therefore, it is a good option for designing a distributed database systems. A disadvantage of this topology is the number of computing nodes must be a power of two.

A.1.2 DEFINITION OF A HYPERCUBE TOPOLOGY

An n-dimensional hypercube (n-cube) network consists of $N = 2^n$ nodes constructed as follows : the nodes are addressed distinctly by n-bit binary numbers, $b_{n-1}b_{n-2}...b_j...b_0$, from 0 to $2^n - 1$ [16]. The Hamming distance is the number of the corresponding bit positions in which the two addresses have different bit values. Therefore, a node x is connected to a node y if the hamming distance is equal to 1. For example, the hamming distance between node [000] and node [111] is three in a 3-dimensional cube. Node [000] is directly connected to node [001], [010] and [100] since its hamming distance is one. A 8 nodes hypercube with its node addresses is shown in Figure 55.

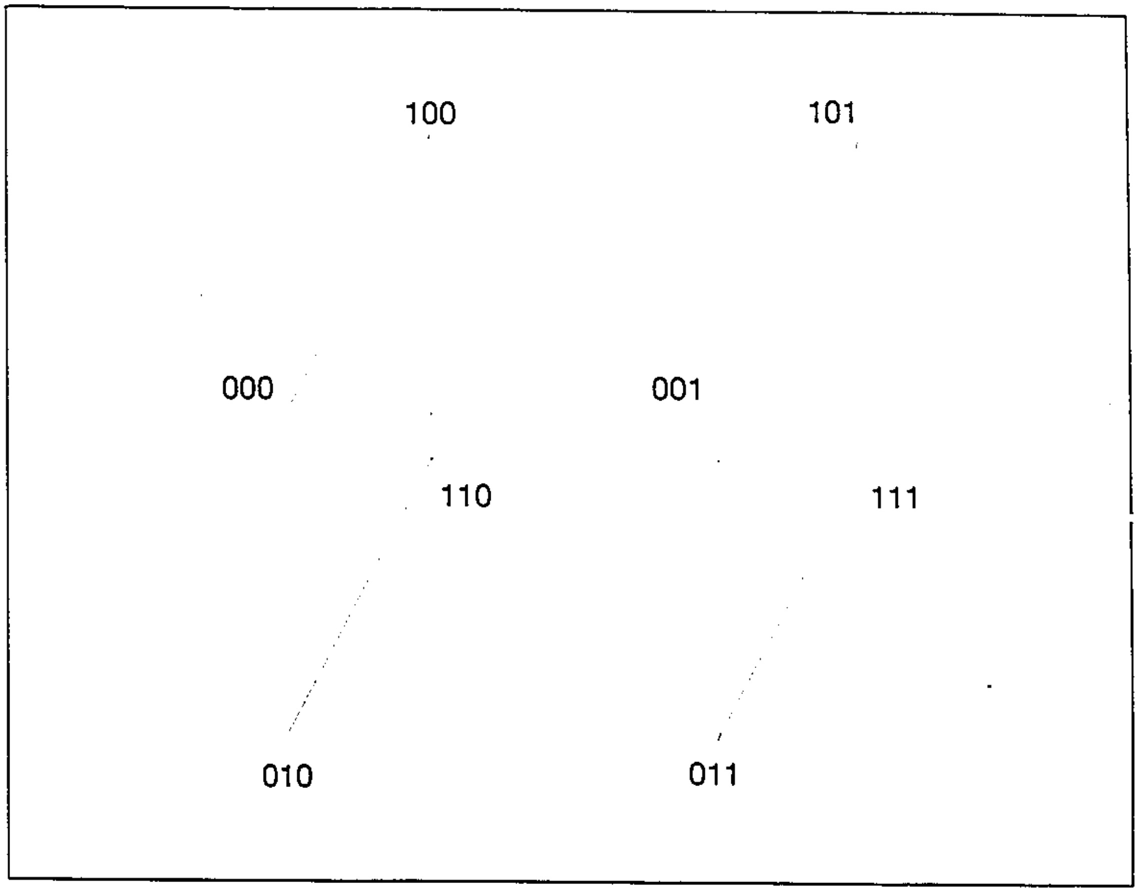


Figure 39 A 8 nodes Hypercube

Each of the query schedule will be executed ten times, and their total cost and response time will be measured individually. Finally, the average of the total cost and the response time will be calculated as the final result.

A.2 AN OVERVIEW OF THE OSI REFERENCE MODEL

The Open System Interconnection (OSI) is a seven-layered ISO compliant architecture for network operations that enables two or more OSI devices to communicate with each other. The philosophy guiding OSI is that an open system of interconnection is possible if an encompassing set of standards is created. With the implementation of standardized protocols, computers and related devices can exchange information despite a diversity in makes and models.

The seven OSI layers are depicted in Figure 40.

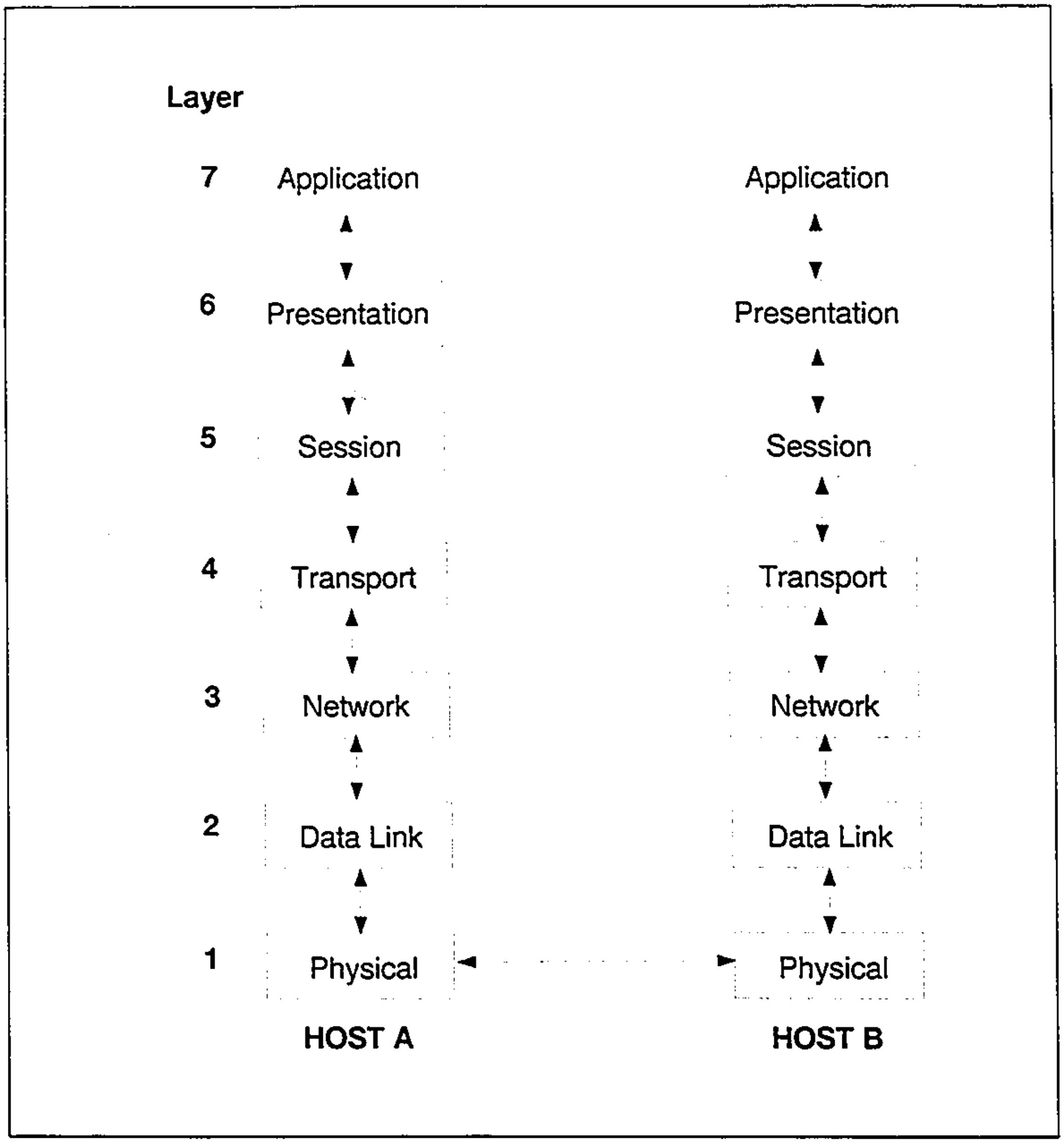


Figure 40 The Seven OSI Layers Model

The lowest layer is the **physical layer**. This layer is directly connected to the physical medium which connected the systems. It sends and receives a stream of bits across the medium. A common used medium is coaxial cable. but its bandwidth is limited.

However, optical technology has made it possible to transmit data by pulses of light. A light pulse can be used to signal a 1 bit; the absence of a pulse signals a 0 bit. Visible light has a frequency of about 108 MHZ, so the bandwidth of an optical transmission system is potentially enormous.

The **data link layer** is the second layer in the OSI model. It controls the flow of data, the correction and detection of errors, and sequencing. In other words, it ensures that the data arrives safely from the sending node to the receiving node. Therefore, the most important functions are to provide for the detection of transmission errors and provide mechanisms to recover from lost, duplicated, or erroneous data.

The **network layer** is the third layer in the OSI model. It is responsible for the addressing and routing in the network. If there is no direct connection between the two nodes which wish to communicate, the network layer finds out what intermediate nodes can relay the messages to their destination.

The **transport layer** is the fourth layer in the OSI model. It provides a transparent, reliable, and end-to-end data transfer mechanism. It ensures that each message arrives error free at its destination node. It is the hinge between the upper and lower parts of the model. It can send several messages down the same network connection at the same time. This is called multiplexing data streams.

The **session layer** is the fifth layer in the OSI model. It is responsible for establishing, managing and terminating connections for individual application programs. Therefore, it sets up a framework for dialogue between systems.

The **presentation layer** is the sixth layer in the OSI model. It is responsible for the format and code conversion. This layer concerns with how the data is represented. Therefore, it is used to code data from an internal format of a sending machine into a common transfer format, and then to decode this format to a required representation at the receiving end.

The application layer is the highest layer in the OSI model. It contains all user or application programs. Therefore, its function is to support the end-user application. Some services such as job management, file transfers and electronic mail are provided in this layer.

A.3 THE DESCRIPTION OF THE HYPERCUBE SIMULATION

The hypercube simulation is written in SIMSCRIPT II.5, which is a discrete-event simulation language. Discrete-event simulation describes a system in terms of logical relationships that cause changes of state at discrete points in time rather than continuously over time. For example, objects arrive and change the state of the system instantaneously. Some of the state variables are the number of objects waiting for service and the number being served.

Simscript is essentially event-oriented, with event routines containing the instructions necessary to change the status of the system. For example, a process is a special kind of temporary entity, and a resource is a special kind of permanent entity. Processes are initiated by an ACTIVATE statement, they can WAIT, and they can be interrupted and resumed. Processes can REQUEST and RELINQUISH resources; they are made to wait if they request more of a particular resource than is currently available.

A Hypercube network has been simulated in this simulator. Each node in this simulator represents a site. Each node contains two links to each of its adjacent nodes. One of the links is used to receive packets and the other link is used to send packets.

In this simulator, the routing of the packets is determined by the adaptive Dijkstra Algorithm. Some packets are generated exponentially in order to form the background traffic.

The input of this simulator is the schedule from the AHY Algorithm or the heuristics. The output of this simulator is the average of the total cost and the response time of ten executions.

In the simulator, "NODE" which represents a node in the hypercube, is a permanent entity in this program. Each node contains a value of in-degree and a value of counter to count the in-degree. "MESSAGE" and "PACKET" are two important processes in this program. Once the process "MESSAGE" is activated, it will put a predefined number of packets to the network. The purpose of the process "PACKET" it to send the packet until it reaches its destination node.

"LINK" is a resource in the simulator. Each node establishes two links to each of its adjacent nodes. One link is used to transfer the packets and the other link is to receive packets from its adjacent node. If the link is being occupied, then the request will be put into its corresponding queue to wait until that link is released.

A.3.1 AN OVERVIEW OF THE HYPERCUBE SIMULATOR

The OSI model is the reference model of this simulation model. However, it is not practical to simulate all seven layers. In order to reduce the complexity of the simulation, only four layers are modelled. These are the application layer, transport layer, network layer and physical layer.

A.3.2 SIMULATION OF THE PHYSICAL LAYER

The physical layer is mainly concerned with communication links. When the network layer of a node wants to send a packet, it first needs to request the corresponding link. If the link is occupied, then the packet will be queued until that link is released. In other words, once a link is used for transmitting a packet, it cannot transmit another packet until the current transmission is completed. All links are assumed to be unidirectional. The speed of all communication links is identical.

Once a packet is transmitted to another node, the physical layer checks whether or not it arrives at its destination. If the packet does not arrive at its destination, then the same procedure is repeated until it reaches its destination.

A.3.3 SIMULATION OF THE NETWORK LAYER

There are many algorithms to handle the routing of a packet from one node to another. In this simulator, the adaptive Dijkstra Algorithm, which can compute the shortest path, is used to transmit the packets from one node to another. This algorithm combines two routing techniques. One is adaptive and the other is the shortest path routing. The term adaptive means that the algorithm attempts to change its routing decisions to reflect changes in the current traffic. A detailed description on this algorithm can be found in [25].

In this simulator, the shortest path between two nodes is based on recent changes in the network traffic. The program has an array "DELAY_TABLE" which stores the delays from all source nodes to all destination nodes. This array is updated at regular intervals. Table 81 is an example of a DELAY_TABLE. The Dijkstra Algorithm uses this array to determine the routing. The results of these calculations are stored in another array called "SHORTEST_PATH". This array allows us to determine the shortest path between any two nodes is the hypercube. Table 82 is an example of the "SHORTEST_PATH". If node 1 wants to send a packet to node 8, the shortest path is : node 1 -> node 3-> node 7 –> node 8.

0.00	1.46	1.43	2.91	1.51	3.24	2.88	4.40
1.27	0.00	2.71	2.78	2.78	3.16	4.15	4.26
1.39	2.84	0.00	2.70	4.35	4.63	1.44	4.55
2.72	2.72	2.72	0.00	4.23	4.61	4.04	2.81
1.45	2.91	2.76	4.02	0.00	1.45	2.76	4.57
2.64	2.64	4.14	4.14	1.38	0.00	4.14	2.74
4.13	4.15	2.70	4.01	1.26	4.08	0.00	3.11
3.95	4.30	3.95	2.92	4.41	2.96	2.51	0.00

Table 81 An example of a DELAY_TABLE

0	2	3	2	5	2	3	3
1	0	1	4	1	6	1	4
1	1	0	4	1	1	7	7
2	2	3	0	2	2	3	8
1	1	7	7	0	6	7	7
2	2	5	2	5	0	5	8
3	3	3	3	5	5	0	8
7	4	7	4	6	6	7	0

Table 82 An example of a SHORTEST_PATH

When the network layer of a particular node receives a packet, the packet's address is checked against the current address. If the addresses do not match, the packet has not yet reached its destination. In this case, the packet is retransmitted.

A.3.4 SIMULATION OF THE TRANSPORT LAYER

The purpose of the transport layer is to provide a reliable mechanism to transmit the packets. In other words, it ensures the packets are delivered in sequence, error-free. To simplify the simulation, we assume that packets will always arrive at a destination and are error free. Therefore, this simulator provides an error free point-to-point connection to the network. In addition, each node is assumed to have a logical connection to every other node in the network.

A.3.5 SIMULATION OF THE USER/APPLICATION LAYER

This layer is responsible for generating the packets which form the background traffic of the simulator. The packets have an exponentially distributed mean generation period and are assigned to any node with equal probability. The process "BACKGROUND.PROCESS" is responsible for this job. This simulator can also change the rate of background traffic to low load, medium load and heavy load.

The schedules produced by Algorithm General and the heuristics are executed in this layer. The term schedule represents the sequence of the transmission which is generated using the AHY Algorithm or out heuristic. The schedule is formed by a number of segments. Each segment represents a transmission of a number of packets. The information of the source node, destination node and the total number of packets to transmit are stored in each segment.

Execution of a schedule starts after a warm up period. Each segment with an in-degree of zero will start to execute. A predefined number of packets are placed in the sending queue for transmission to the destination node. A counter at the destination node is used to ensure that all the packets arrive. Once the value of this counter is equal to the expected number of packets, the in-degree of that particular destination node will be incremented by one. In other words, the in-degree represents the expected number of transmissions at the destination node. If the counter for a particular node is equal to the expected number of transmission, then all transmissions have been received.

The transmissions are in the correct sequence and parallel transmissions are possible. The scheduler continually controls the sequencing of the distribution schedule until all the packets arrive at the query site. A limitation of this simulator is that it cannot handle cycles in a query schedule.

A.3.6 INPUT PARAMETERS FOR THE HYPERCUBE SIMULATOR

The hypercube simulator can handle 8, 16, 32 or 64 nodes. The first input parameter is the total number of relations plus the total number of attributes. The second input parameter is the total number of relations. The third input parameter is a set of ordered pairs [X, Y]. The first item X is the physical location of a relation. The second item Y is the common join attribute of that relation. The fourth input parameter is the physical location of the query site. The last input parameter contains the information of the transmission which are produced by the scheduling subsystem. The first element is the number of in degree. The second element is a set of triple [X, Y, Z]. The first item X is the address of the source node. The second item Y is the address of the destination node. The last item Z is the total number of packets to transmit. Finally, a end of line marker [-1] is put at the end. A sample input file is shown as below :

```
LINE  1 :  9
LINE  2 :  3
LINE  3 :  1 2
LINE  4 :  7 2
LINE  5 :  4 2
LINE  6 :  8
LINE  7 :  2 1 99 40 –1
LINE  8 :  2 2 99 45 –1
LINE  9 :  1 3 99 20 –1
LINE 10 :  0 4 3 25 –1
LINE 11 :  0 5 9 5 –1
LINE 12 :  0 6 8 10 –1
```

LINE 13 : 0 7 1 30 -1

LINE 14 : 1 8 1 20 8 2 20 -1

LINE 15 : 1 9 2 15 -1

The number 9 in line 1 is the sum of the total number of relations and the total number of attributes. The number 3 in line 2 is the total number of relations. Line 3 to line 5 is a set of ordered pairs. The number 1 in line 3 represents the physical location of the first relation. The number 2 in line 3 represents the total number of attributes. The number in line 6 indicates the physical location of the query site. Information about the number of the packets to transmit is in line 7 to line 15. The number 2 in line 7 is the in-degree. The following three numbers form a triple [1, 99, 40]. The number 1 is the logical address of the source. The number 99 is the logical address of the destination. The last number 40 represents the total number of packet to send.

A.3.7 THE OUTPUT OF THE HYPERCUBE SIMULATOR

The execution time for each segment is stored in the array "TRANSMIT_TABLE". From this array we can calculate the total cost of any query schedule. The array "RESPONSE.TIME" stores the response time for each schedule. From this array we can calculate the response time for a distributed query.

Each of the query schedule will be executed ten times. And their total cost and response time will be measured individually. Finally, the average of the total cost and the response time will be calculated as the final result.

APPENDIX B
THE DESCRIPTION OF THE ALGORITHM GENERAL PROGRAMS

Two programs will be discussed in this appendix. The first program uses the Algorithm Parallel to generate the transmission segments for the Hypercube simulator to use. The second program uses the Algorithm Serial to generate the transmission segments for the Hypercube simulator to use.

The data structure, the input parameters and the output parameters of these programs will be discussed.

B.1 THE DATA STRUCTURE OF THIS PROGRAM

A pointer array with four elements is used for each of the schedules. There are four items in each row of the pointer array. These items are the relation number, attribute number, response time and a pointer which points to the first segment of that schedule.

Each segment contains eight items. The first item is the logical address of the source node. The second item is the logical address of the destination node. The third item is the total number of packets to transmit. The fourth item is the selectivity factor of that common join attribute. The fifth item is the segment ID which is used to connect the segment together. The sixth item is a flag which is used to duplicate a same schedule. The seventh item is a pointer which points to the next level of the segment. The last item is also a pointer which points to the next segment. Figure 41 shows the data fields of each segment and Figure 42 is an example of the structure to store the segments.

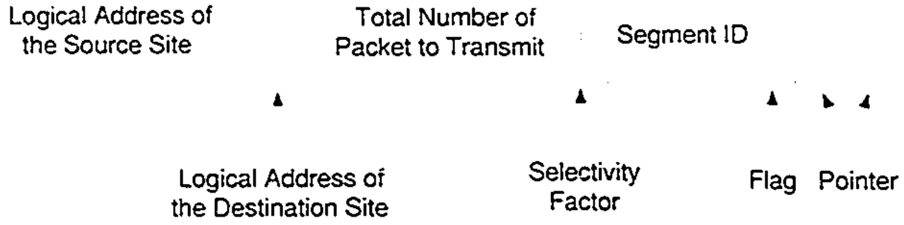

Logical Address of the Source Site
Total Number of Packet to Transmit
Segment ID
Logical Address of the Destination Site
Selectivity Factor
Flag
Pointer

Figure 41 The Data Field of the Segment

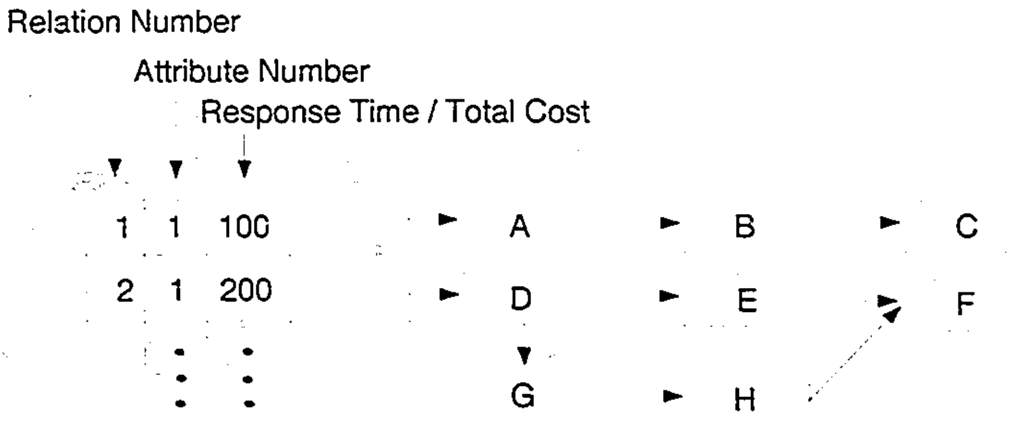


Figure 42 The Data Structure to Store the Schedules

The box with the label from A to C indicates the first schedule in this pointer array. The box with the label from D to H indicates the second schedule. The relation number, attribute number and the response time of that schedule is stored in this pointer array. For example, the first schedule has relation number 1, attribute number 1 and the response time is 100. For each schedule, it contains the number of segments.

B.2 THE DESCRIPTION OF THE PROGRAM IN ALGORITHM PARALLEL

This program is based on the Algorithm General. The following are the steps of this program.

1. Perform the following on each of the common join attributes in the query in order to generate the schedule for each of the attributes.

    a. The first schedule is to send the attribute directly to the query site.

    b. Find the schedule with the smallest response time.

    c. Find the next smallest common join attribute and do the semi-join with the previous schedule.

    d. Repeat step 1c until all of that attributes are considered.

    e. Find the schedule with minimum response time from step 1c as the result.

2. Perform the following on each of the relation in order to generate the schedule for each of the relation.

    a. Collect the *valid* common join attribute schedules from step 1. The term valid common join attribute means that :

        - If relation R_i is processed, then all the common join attributes with the relation number i will not be considered.

    b. Order the schedules which are generated in step 2a in ascending order.

    c. Find the schedule with the smallest response time from step 2b and do the semi-join with the current relation.

    d. Find the schedule with the next smallest response time from step 2b and do the semi-join with the previous schedule.

    e. Repeat step 2d until all the valid attributes of the current relation are considered.

    f. Find the schedule with the minimum respond time from step 2d as the result for the current relation.

The following relation table is the example for the illustration of Algorithm Parallel.

RELATION R_i	SIZE S_i	d_{i1}		d_{i2}	
		b_{i1}	p_{i1}	b_{i2}	p_{i2}
R1	1000	400	0.4	100	0.2
R2	2000	400	0.4	450	0.9
R3	3000	900	0.9	———	———

Table 83 The Relation Table

The complete result of the first step can be found in the example of Algorithm Parallel in section 2.7.

The iteration for common join attribute d_{31} is shown in Figure 43. In order to simplify the diagram, only three data fields are shown in each of the transmissions. The first data field is the relation number. The second data field is the attribute number and the last data field is the total number of packet to transmit. Each common join attribute is indicated by its logical address. The following table shows the relationship between the common join attribute and its logical address. It shows that the logical address of R_1 is 1, the logical address of d_{11} is 4, the query site is 99 and so on.

Logical Address	1	2	3	4	5	6	7	8	9	99
Join Attribute	R1	R2	R3	d_{11}	d_{12}	d_{21}	d_{22}	d_{31}	d_{32}	QS

Table 84 The Relationship between Logical Address & Common Join Attribute

The first row is to send d_{31} to the query site. The second row indicates that d_{11} is joined to d_{31}, since d_{11} is selected as minimum in the first iteration. The last row joins d_{21} to the previous schedule, since the schedule for d_{21} is the minimum in the second iteration. Finally, find the schedule with the smallest response time. The minimum response time for attribute d_{31} is the last row with RT = 584 and this schedule will be selected. This schedule will be stored in another pointer array for future use.

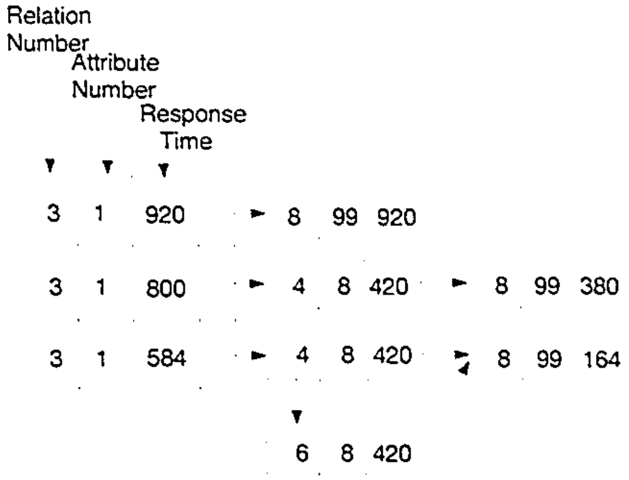


Figure 43 The Choice for the Attribute d_{31}

The following table shows all the valid attributes for each relation. The valid attributes for R_1 are d_{22}, d_{21} and d_{31}. The valid attributes for R_2 are d_{12}, d_{11} and d_{31}. The valid attributes for R_3 are d_{11} and d_{21}.

Relation 1		Relation 2		Relation 3	
Attribute	Response Time	Attribute	Response Time	Attribute	Response Time
d_{22}	230	d_{12}	120	d_{11}	420
d_{21}	420	d_{11}	420	d_{21}	420
d_{31}	584	d_{31}	584	———	———

Table 85 The Valid Attributes for each of the Relation

Figure 44 illustrates step 2 for R_1. The valid common join attributes for R_1 will be ordered in ascending order as shown in the diagram.

The schedule with the minimum response time will be selected and joined to the corresponding relation. The result is shown in row 1.

The previous schedule will be duplicated, then the schedule with the next smallest common join attribute will be selected and joined to it which is shown in row two.

Step 2d will continue until all the valid common join attributes have been considered. The schedule with the minimum response time will be chosen. The second schedule with the response time equal to 800 will be selected. Once all the relations have gone through all the above steps, the program is finished.

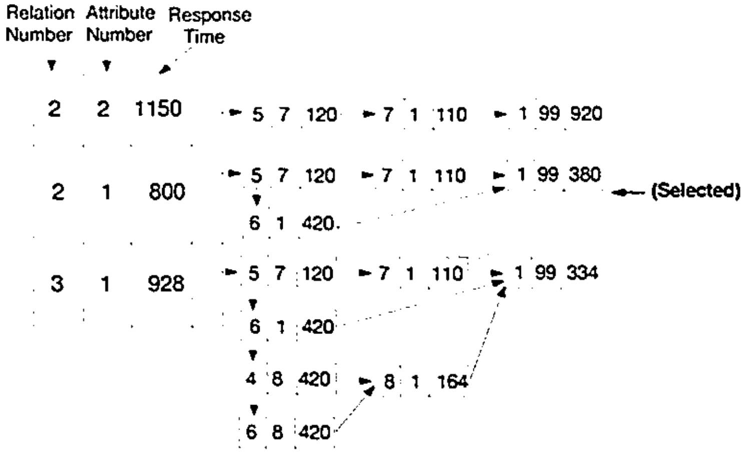


Figure 44 The options for R_1

B.3 THE DESCRIPTION OF THE PROGRAM IN ALGORITHM SERIAL

This program is based on Algorithm General for the Total Time Version. This algorithm has the following steps :

1. Create the schedules which transmit the relations from their own sites to the users site in ascending order.
2. Perform the following steps on each of the common join attributes in the query.
   a. The common join attribute with the smallest total time will be selected.
   b. Join to the next smallest common join attribute which is in the other relation.
3. Perform the following steps on each of the relations with the schedules from step 2.
   a. Use the schedules from step 2 to do the semi-join with a relation. For example, R_i.

b. Two considerations on doing the semi-join :

   - If the schedule contains a transmission of d_{ij} at the beginning of the schedule, then add another candidate schedule which is the same as that schedule except that the transmission d_{ij} is deleted and called this schedule d_{ij}'.
   - If the schedule contains only one transmission of d_{ij}, then this schedule will not be considered.

4. Find the schedule with the smallest total cost from step 3 on each of the common join attribute.
5. Create the integrated schedules[14] from the schedules which are generated from step 4.
6. Find the schedule with the smallest total cost from step 5 as the result for that relation.
7. Repeat step 3, 4, 5 and 6 until we have schedules for all relations.

The same example will be used here for illustration. Since the first two steps are very straight forward, it will not be discussed here. Figure 45 shows the schedules for relation 1.

[14] Let $BEST_{ij}$ be the schedules which are generated from step 3. For each $BEST_{ij}$ in ascending on of j, construct an integrated schedule to R_i that consists the parallel transmission of candidate schedule $BEST_{ij}$ and schedules $BEST_{ik}$ where k<j.

The options for attribute one

Relation Number	Attribute Number	Total Cost	Pointer				
2	1	1020	→ 4 6 420	→ 6 1 180	→ 1 99 420		
2	1'	840	→ 6 1 420	→ 1 99 420			← (Selected)
3	1	1144	→ 4 6 420	→ 6 8 180	→ 8 1 164	→ 1 99 380	
3	1'	1180	→ 6 8 420	→ 8 1 380	→ 1 99 380		

The options for attribute two

2	2	1150	→ 5 7 120	→ 7 1 110	→ 1 99 920	← (Selected)
2	2'	1390	→ 7 1 470	→ 1 99 920		

Figure 45 The Options for Relation 1

The attributes d_{21}' and d_{31}' are generated from d_{21} and d_{31} respectively. Since the first segment in the first row has a logical address 4 (ie. d_{11}) as the source site and a logical address 6 (ie. d_{21}) as the destination site, this segment can be deleted to produce the d_{21}'. The common join attributes d_{31}' and d_{22}' are generated in the same way.

The schedules with the smallest total cost on each common join attributes will be selected as discussed in step 4. Therefore, d_{21}' with the total cost of 840 and d_{22} with the total cost 1150 will be selected in this example.

Finally, those selected schedules will be sorted in ascending order and then integrated. The integrated schedule to R_i constitutes the parallel transmission. The integrated schedules for R_1 are shown in Figure 46. The second row is the integrated schedule.

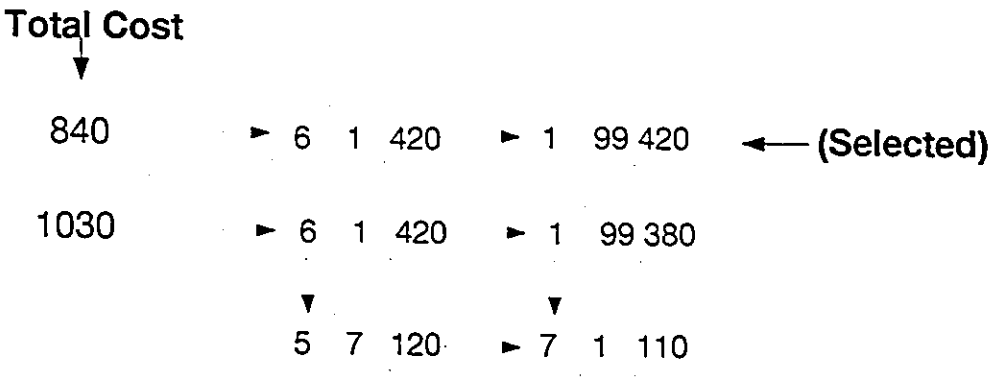


Figure 46 The integrated schedules for R_1

In step 6, the schedule with the minimum total time will be the final schedule for that relation. That is, the first option with the total cost 840. The same procedure is repeated for the rest of relations.

B.4 THE INPUT PARAMETERS OF THE PROGRAM

When the program is executed, the user has to provide the names of the input file and the output file. Once these two file names are entered, the program will start to execute.

The first two input parameters are the total number of relations and the total number of attributes. The second input parameter is the physical location of the query site. The last input parameter contains the details information of the relations. The first element is the physical location of the relation. The second element is the total number of packets of that relation. The third element is an ordered pair [X, Y]. X is the size of the common join attribute. Y is the selectivity factor of that common join attribute. If a common join attribute does not exist in a relation, then the value of X and Y will be zero. A sample input file is shown as below :

LINE 1 : 3 2

LINE 2 : 30

LINE 3 : 20 1000 400 0.4 100 0.2

LINE 4 : 25 2000 400 0.4 450 0.9

LINE 5 : 17 3000 900 0.9 0 0

The first number 3 in line 1 represents the total number of relations in this query. The second number 2 in line 1 represents the total number of attributes. The number 30 in line 2 represents the physical location of the query site. The information from line 3 to line 6 represents the details of each relation. The number 20 in line 3 represents the physical location of that relation. The number 1000 in the same line represents the total number of packets in that relation. The following two ordered pairs [400, 0.4] and [100, 0.2] represents the details of two common join attributes. The numbers 400 and 100 are the size of the first and the second common join attribute. The numbers 0.4 and 0.2 are their corresponding selectivity factors.

B.5 THE OUTPUT OF THE PROGRAM

The output file will be the input file produced of the Hypercube simulator program which is written in Simscript II.5.

The first output parameter is the total number of relations plus the total number of attributes. The second output parameter is the total number of relations. The third output parameter is a set of ordered pairs [X, Y]. X is the physical location of a relation. Y is the common join attribute of that relation. The fourth output parameter is the physical location of the query site. The last output parameter contains details of the transmissions produced by the scheduling subsystem. The first element is in-degree. The second element is a triple [X, Y, Z]. X is the address of the source node. Y is the address of the destination node. Z is the total number of packets to transmit. Finally, a end of line marker [-1] is placed at the end. An example output is shown below :

LINE 1 : 9

```
LINE  2 : 3
LINE  3 : 1 2
LINE  4 : 7 2
LINE  5 : 4 2
LINE  6 : 2
LINE  7 : 2 1 99 40 –1
LINE  8 : 2 2 99 45 -1
LINE  9 : 1 3 99 20 -1
LINE 10 : 0 4 3 25 -1
LINE 11 : 0 5 9 5 -1
LINE 12 : 0 6 8 10 -1
LINE 13 : 0 7 1 30 -1
LINE 14 : 1 8 1 20 8 2 20 -1
LINE 15 : 1 9 2 15 -1
```

The number 9 in line 1 represents the total number of relations plus the total number of attributes. The number 3 in line 2 represents the total number of relations. From line 3 to line 5 represents three sets of ordered pairs. The first number 1 on line 3 represents the physical location of relation one. The second number 2 on line 3 represents the total number of attributes. The number 2 on line 6 represents the physical location of the query site. Details of the transmission are shown from line 7 to line 15. The first number 2 in line 7 represents the in-degree. The following numbers [1, 99, 40] represents a triple. 1 represents the logical address of the source node. 99 represents the logical address of the destination node. 40 represents the destination node. The –1 is the end of line indicator in each case.

APPENDIX C
THE DESCRIPTION OF THE PROGRAM TO COMPUTE THE TOTAL COST

This program computes the total cost of the schedules generated by Algorithm Serial or the heuristic for the response time.

The total cost is the sum of each segment's size multiplied by the total number of hops that the packets travel from the source node to the destination node. The following is an example to compute the total cost from a hop table and a given schedule. The following table is an 8 x 8 hop table which is supplied by the simulator.

	T0 1	To 2	To 3	To 4	To 5	To 6	To 7	To 8
From 1	0	1	3	2	3	1	2	3
From 2	2	0	3	2	1	2	3	2
From 3	1	2	0	3	3	1	2	3
From 4	2	2	1	0	1	2	3	1
From 5	1	2	2	3	0	2	1	1
From 6	2	1	1	2	3	0	3	1
From 7	3	1	1	2	2	2	0	1
From 8	2	2	3	3	1	3	2	0

Table 86 Example of Hop Table

The following table is a schedule which contains 5 segments. Each segment contains a logical address of its source site, the destination site and the number of packets to transmit.

SEGMENT #	SOURCE ADDRESS	DESTINATION ADDRESS	SIZE
1	1	99	200
2	2	99	100
3	7	3	50
4	8	4	30
5	5	9	80

Table 87 Example of a Schedule .

For example, for the first segment the logical address of the source site is 1, the logical address of the destination site is 99 (ie. Query site) and the size of that segment is 200. The program will find out the physical address from its logical address. Let the physical address of the logical address 1, 3, 4, 5 and 6 be one. Let the physical address of the logical address 2, 7, 8, 9, 10 be two. Let the physical address of the logical address 99 be eight. The number of hops from 1 to 8 from the Hops Table is 3. The number of hops from 2 to 8 is 2 and the number of hops from 2 to 1 is 2.

Therefore, the cost to transmit this 200 packets is 3 multiplied by 200 and the result is 600. We repeat the above steps on each segment and add them up. The total cost of the schedule can be found. It is (3*200) + (2*100) + (2*50) + (2*30) + (1*80) = 1040.

C.1 THE INPUT/OUTPUT OF THE PROGRAM

This program can handle 8, 16, 32 or 64 nodes in the network. The program will ask the user to enter two input file names. The first input file name is for the network load (ie. Hop Table). The second input file name is for the schedule.

The input format for the second input file is the same as the output file which is produced by the Algorithm Serial or the heuristic for the response time version. There is an example in Appendix B.5.

The output of this program is simply the total cost of the given schedule with the corresponding hop table.

APPENDIX D
THE ALGORITHM OF TOTAL COST HEURISTIC

NUM_REL : Total Number of Relations

NUM_ATT : Total Number of Attributes

NUM_NODE : Total Number of Nodes on the Network

R_i : Relation i

$Site_i$: Physical Address of R_i

s_i : Size of R_i

b_{ij} : Size of Attribute j at R_i

p_{ij} : Selectivity Factor of Attribute j at R_i

d_{ij} : Attribute with b_{ij} and p_{ij}

Hop_Table[i, j] : Stores the number of Hops travels from site i to site j

Cost/Benefit Table : Stores the value of C[b_{ij} -> b_{xy}] and B[b_{ij} -> b_{xy}]

B[b_{ij} -> b_{xy}] : The Benefit to send b_{ij} from the site of R_i to the site of R_x

(ie. B[b_{ij} -> b_{xy}] = Hop_Table[x,QS] * ((1−p_{ij}) * R_x + Overhead Cost))

C[b_{ij} -> b_{xy}] : Hop_Table[i, x] * (b_{ij} + Overhead Cost)

OUTPUT LIST : The storage to store the selected schedules

OPTION LIST : The storage to store the intermediate schedules

EMPTY_OPTION_LIST : Boolean [True/False]

DONE : Boolean [True/False]

Figure 47 The Definition of the Symbols

```
Given a Relation Table contains [[[$Site_i$], [$S_i$] and [$b_{ij}$ and $p_{ij}$ where j = 1
to NUM_ATT]] where i = 1 to NUM_REL]

Given a Hop_Table[1..NUM_NODE, 1..NUM_NODE]

DONE = False
<<< Create a Cost/Benefit Table >>>
For j = 1 to NUM_ATT do
    For i = 1 to NUM_REL do
        {Compute the Cost & Benefit to send a attribute from its
          own site to every other site which contains a relation. }
        If $d_{ij}$ <> 0 then {Attribute Exist}
            For y = 1 to NUM_ATT do
                For x = 1 to NUM_REL do
                    If (i <> x) and (j = y ) then =>{Not the same
                                         relation & Same attribute}
                        <<< Compute the Cost >>>
                        C[$b_{ij}$ -> $b_{xy}$] = Hop_Table[i,x]
                                         * ($b_{ij}$ + Overhead Cost)
                        <<< Compute the Benefit >>>
                        B[$b_{ij}$ -> $b_{xy}$] = Hop_Table[i,x]
                              * ((1-$p_{ij}$) * $R_x$ + Overhead Cost)
                    End if
                End for
            End for
        End if
    End for
End for
  << Find the Transmission from the Cost/Benefit Table >>>
Loop
  EMPTY_OPTION_LIST = True
```

Figure 48 The Algorithm of the Total Cost Heuristic

```
<<<FIRST   PHASE>>>
  {The First Priority is to find the schedule with the smallest
    cost under the condition [ Cost < Benefit + Threshold
    Value] . If only one schedule is selected, it will be stored
    in the OUTPUT LIST. Otherwise, it will select those
    schedules and store them in OPTION LIST. }
   For j = 1 to NUM_ATT do
     For i = 1 to NUM_REL do
        If $d_{ij}$ <> 0 then
           For y = 1 to NUM_ATT do
              For x = 1 to NUM_REL do
                 If (i <> x) and (j = y ) and
                   (C[$b_{ij}$ -> $b_{xy}$] < B[$b_{ij}$ -> $b_{xy}$]
                                 + Threshold Value )then
                    If only one smallest C[$b_{ij}$ -> $b_{xy}$]
                      is found then
                        Schedule $d_{ij}$ -> $d_{xy}$ is selected
                          and added to the OUTPUT LIST
                        If any free transition is found on
                          OUTPUT LIST, then remove that
                          transition from the Cost/Benefit
                          Table
                    Else if any schedules with the
                          same smallest
                          C[$b_{ij}$ -> $b_{xy}$].
                          Storing those schedules in the
                          OPTION LIST
                        EMPTY_OPTION_LIST = False
                    Else
                        DONE = True
                    End if
                 End if
              End for
           End for
        End if
```

Figure 49 The Algorithm of the Total Cost Heuristic (Continuous...)

```
    End for
End for

<<<SECOND    PHASE>>>

{ The Second Priority is to find the schedule with the best
  sending attribute $d_{ij}$ from the OPTION LIST.
  If only one schedule is available, it will be stored in the
  OUTPUT LIST and by pass the following two pass.
  Otherwise, It will select those best schedules and store
  them in OPTION LIST. }



If EMPTY_OPTION_LIST = False then
    If only one smallest $p_{ij}$ is found in the
       OPTION LIST then
           Schedule $d_{ij}$ -> $d_{xy}$ is selected and
              added to the OUTPUT LIST
           If any free transition is found on
              OUTPUT LIST, then remove that
              transition from the Cost/Benefit Table
           EMPTY_OPTION_LIST = True
     Else
           Find out all the schedules with the same smallest
              $p_{ij}$ -> $b_{xy}$ and store them in the OPTION LIST
           EMPTY_OPTION_LIST = False
     End if
 End if
```

Figure 50 The Algorithm of the Total Cost Heuristic (Continuous...)

```
<<<THIRD  PHASE>>>

    { The Third Priority is to find the schedule with the best
      receiving attribute $d_{xy}$ from the OPTION
      LIST. If only one schedule is available, it will be stored in
      the OUTPUT LIST and by pass the following pass.
      Otherwise, It will select those best schedules and store
      them in OPTION LIST. }

    If EMPTY_OPTION_LIST = False then
        If only one smallest $p_{xy}$ is found in the
          OPTION LIST then
              Schedule $d_{ij}$ -> $d_{xy}$ is selected and
                added to the OUTPUT LIST
              If any free transition is found on
                OUTPUT LIST, then remove that
                transition from the Cost/Benefit Table
              EMPTY_OPTION_LIST = True
        Else
              Find out all the schedules with the same smallest
                $p_{ij}$ -> $b_{xy}$ and store them in the OPTION LIST
              EMPTY_OPTION_LIST = False
        End if
     End if
```

Figure 51 The Algorithm of the Total Cost Heuristic (Continuous...)

```
<<<FOURTH   PHASE>>>

   {The Fourth Priority is to find the schedule with maximum
     benefit from the OPTION LIST. If more than one
     schedules in the OPTION LIST, then it will select the
     first available schedule to the OUTPUT LIST.}

   If EMPTY_OPTION_LIST = False then
       If only one biggest B[$b_{ij}$ -> $b_{xy}$] is found
           in the OPTION LIST then
               Schedule $d_{ij}$ -> $d_{xy}$ is selected and
                  added to the OUTPUT LIST
               If any free transition is found on
                  OUTPUT LIST, then remove that
                  transition from the Cost/Benefit Table
               EMPTY_OPTION_LIST = True
        Else
               Find the first available schedule from the
                  OPTION LIST and added to the OUTPUT LIST
               EMPTY_OPTION_LIST = False
       End if
    End if
```

Figure 52 The Algorithm of the Total Cost Heuristic (Continuous...)

```
<<< Modify the Relation Table >>>
    $R_x = p_{ij} * R_x$
    $b_{xy} = p_{ij} * b_{xy}$

<<< Modify the Cost/Benefit Table >>>
    Update all the cost which contains the sending attribute $d_{ij}$.
         ==> (Horizontal of the Cost/Benefit Table)
    Update all the benefit which contains the $R_x$.
         ==> (Vertical of the Cost/Benefit Table)

    Delete the schedule $d_{ij}$ -> $d_{xy}$ in
      the Cost/Benefit Table

    Empty the OPTION LIST

    Do the same modification as shown on above for those
      free transition(s).

    Exit When (DONE = True)
End Loop

Add all schedules which send all Relations from its own site
to the Query Site to the OUTPUT LIST


                  <<<<< T H E     E N D >>>>>
```

Figure 53 The Algorithm of the Total Cost Heuristic (Continuous...)

APPENDIX E
THE ALGORITHM OF RESPONSE TIME HEURISTIC

NUM_REL : Total Number of Relations

NUM_ATT : Total Number of Attributes

NUM_NODE : Total Number of Nodes on the Network

NUM_SELECTION : Total Number of Relations don't have a schedule yet.

R_i : Relation i

s_i : Size of R_i

b_{ij} : Size of Attribute j at R_i

p_{ij} : Selectivity Factor of Attribute j at R_i

Delay_Table[i,j] : Stores the value of D_{ij}

D_{ij} : The delay from the site of R_i to the site of R_j.

$C[b_{ij} -> k]$: The Cost to send b_{ij} from the site of R_i to the site of R_k.

(ie. $C[b_{ij} -> k] = (b_{ij} + \text{Overhead Cost}) * D_{ik}$ where $i <> k$)

COUNT : Total Number of Cost in SELECTION_LIST for a relation

IFS : [Initial Feasible Schedule] The Schedule sends the whole relation
from its own site to the Query Site

INDEX : The index for Schedule_R_x

Schedule_R_i : Storage of the schedules for R_i

MIN_Schedule_R_i : Storage the schedules of R_i with Minimum Response Time in a iteration

SELECTION_LIST : Temporary Storage

Figure 54 The Definition of the Symbols

```
Given a Relation Table contains [[[$S_i$] and [$b_{ij}$ and $p_{ij}$ where j = 1 to
NUM_ATT]] where i = 1 to NUM_REL]

Given a Delay_Table[1..NUM_NODE, 1..NUM_NODE]

<<<Compute the Cost >>>
For j = 1 to NUM_ATT do
    For i = 1 to NUM_REL do
        For k = 1 to NUM_REL do
            If (i <> k) or ($b_{kj}$ <> 0) or ($b_{ij}$ <> 0) then
                C[$b_{ij}$ -> k] = ($b_{ij}$ + Overhead Cost) * $D_{ik}$
            Else
                C[$b_{ij}$ -> k] = "UNDEFINED"
            End if
        End for
    End for
End for
```

Figure 55 The Algorithm of the Response Time Heuristic

```
For i = 1 to NUM_REL do
    NUM_SELECTION = (NUM_REL + 1) - i
    For h = 1 to NUM_SELECTION do
         Find a $R_x$ that does not
            generate a schedule yet.

         {<<< Create the Selection List >>>}

         For i = 1 to NUM_REL do
             For j = 1 to NUM_ATT do
                 If $b_{ij}$ <> "UNDEFINED" then
                    Sorting the C[$b_{ij}$ -> x] in
                       ascending order
                 End if
             End for
         End for
         Storing the sorted C[$b_{ij}$ -> x] in
            the storage SELECTION_LIST
         Let COUNT = Total Number of cost in the
            SELECTION_LIST for $R_x$
```

Figure 56 The Algorithm of the Response Time Heuristic (Continuous...)

```
            Compute the Cost of IFS and stores in Schedule_$R_x$[1]
            Let INDEX = 1
            For k = 1 to COUNT do
                INDEX = INDEX + 1
                Find the Minimum cost from SELECTION_LIST
                Let the Minimum cost be C[$b_{ab}$ -> c]
                If k = 1 then
                     Use the $p_{ab}$ to do the Semi-Join
                        with $R_x$ and store the result in
                        Schedule_$R_x$[INDEX]
                 Else
                       Use the $p_{ab}$ to do the Semi-Join
                          with Schedule_$R_x$[INDEX-1] and
                          store the result in Schedule_$R_x$[INDEX]
                 End if

                 This minimum cost C[$b_{ab}$ -> c] will
                     be deleted from the SELECTION_LIST
             End for {k = 1 to COUNT}

             Find the schedule with the Minimum Cost in the
                 storage of Schedule_$R_x$ and store
                 it to MIN_Schedule_$R_x$
      End for {h = 1 to NUM_SELECTION}
```

Figure 57 The Algorithm of the Response Time Heuristic (Continuous...)

```
    For i = 1 to NUM_SELECTION do
        Find the Schedule with the Minimum Response time in
          the storage of MIN_Schedule_$R_i$
        (Let this schedule optimizes $R_y$)
    End for

    Consider the above Schedule be the ***Optimum Schedule***
      for $R_y$

    Do the Semi-Join for attributes ($b_{yj}$ where j = 1 to
      NUM_ATT) on the Relation Table

    Do the Semi-Join on the size of $R_y$ on the Relation Table

    Modify the cost on the Cost Table which has the sending
      attribute ($d_{yj}$) is stored in $R_y$ and is
      modified by the Optimum Schedule as follow :
         Cost to modify $R_y$ + Cost of sending $d_{yj}$

    C[$b_{ij}$ -> y] where (i = 1 to NUM_REL) and (j = 1
      to NUM_ATT) will **NOT** be considered again

End for {i = 1 to NUM_REL}

                    <<<THE    END>>>
```

Figure 58 The Algorithm of the Response Time Heuristic (Continuous...)

APPENDIX F
DEFINITION OF SEMIJOIN

Definition :

The semijoin of relation R, defined over the set of attributes A, by relation S, defines over the set of attributes B, is the subset of the tuples of R that participate in the join of R with S. It is denoted as $R \ltimes_F S$ (where F is a formula specifying the *join predicate*) and can be obtained as follows [21] :

$$\mathbf{R} \bowtie_{\mathbf{F}} \Pi_{A \cap B}(S)$$

Readers should note that $R \ltimes_F S$ is **Not equal to** $S \ltimes_F R$.

Example :

Table 88 Relation C

ID#	Name	Test_Mark	Assignment
92-100	J. Lee	89	82
92-123	M. Chu	33	76
90-782	K. Miller	64	87
92-683	I. Chan	78	65
91-343	Y. Smith	55	65
92-834	H. Wong	90	87
91-454	T. Casey	78	78

Table 89 Relation ER

ID#	Name	Telephone	Address
92-123	M. Chu	256-9833	ER 788
92-765	K. Ho	266-8755	ER 654
92-834	H. Wong	277-7878	ER 321
91-454	T. Casey	233-7877	ER 221
92-142	G. Miller	234-6727	ER 253

Table 90 Relation $C \times_{C.ID\#=ER.ID\#} ER$

ID#	Name	Test_Mark	Assignment
92-123	M. Chu	33	76
92-834	H. Wong	90	87
91-454	T. Casey	78	78

APPENDIX G
BIBLIOGRAPHY

[1] Ahn, J. K., and Moon, S. C. Optimizing joins between two fragmented relations on a broadcast local network. *Information Systems 16* (1991), 185–198.

[2] Apers, P. M., Hevner, A. R., and Yao, S. Optimization algorithms for distributed queries. *IEEE Transactions on Software Engineering 9* (1983), 57–68.

[3] Bernstein, P. A., and Chiu, D.-M. W. Using semi-joins to solve relational queries. *Journal of the Association for Computing Machinery 28* (1981), 25–40.

[4] Bhuyan, L. N., and Agrawal, D. P. Generalized hypercube and hyperbus structures for a computer network. *IEEE Transactions on Computers 33* (1984), 323–333.

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

[6] Chen, A. L. A localized approach to distributed query processing. In *International Conference on Extending Database Technology* (1990), pp. 188–202.

[7] Chen, A. L., and Li, V. O. An optimal algorithm for processing distributed star queries. *IEEE Transactions on Software Engineering 11* (1985), 1097–1107.

[8] Chen, J. S. J., and Li, V. O. K. Optimizing joins in fragmented database systems on a broadcast local network. *IEEE Transactions on Software Engineering 15* (1989), 26–38.

[9] Chen, M.-S., and Yu, P. S. Using join operations as reducers in distributed query processing. In *2nd International Symposium on Database in Parallel and distributed systems* (1990), pp. 116–123.

[10]Date, C. *An Introduction to Database Systems (Volume 1)*. Addison Wesley, 1990.

[11]Frank, Ariel J., W. L. D., and Bernstein, A. J. Multicast communication on network computers. *IEEE Transactions on Software Engineering 2* (1985), 49–61.

[12]Hevner, A. R., and Yao, S. B. Query processing in distributed database systems. *IEEE Transactions on Software Engineering 5* (1979), 177–187.

[13]Kambayashi, Y., Yoshikawa, M., and Yajima, S. Query processing for distributed databases using generalized semi-joins. In *Proceeding of ACM Sigmod International COnference on Management of Data* (1982), pp. 151–160.

[14]Kerschberg, L., and Ting, P. D. Query optimization in star computer networks. *ACM Transactions on database Systems 7* (1982), 678–711.

[15]Korth, H. F., and Silberschatz, A. *Database System Concepts*. McGraw Hill, 1991.

[16]Lan, Y. Multicast in hypercube multiprocessors. *Journal of Parallel and Distributed Computing 8* (1990), 30–41.

[17]Lee, J. M., Park, J. S., and Kim, M. A distributed join algorithm between two fragmented relations in distributed database systems. *INFOR 73* (1990), 1225–1232.

[18]McKinley, P. Multicast routing in spanning bus hypercubes.

[19]Navathe, S., Ceri, S., Wiederhold, G., and Dou, J. Vertical partitioning algorithms for database design. *ACM Transactions on Database Systems 9* (1984), 680–710.

[20]Ozsu, M. T., and Valduriez, P. Distrubuted data management: Unsolved problems and new issues. Tech. rep., University of Alberta, 1991.

[21]Ozsu, M. T., and Valduriez, P. *Principles of Distributed Database Systems*. Prentice Hall International, 1991.

[22]Pramanik, S., and Vineyard, D. Optimizing join queries in distributed databases. *IEEE Transactions on Software Engineering 14* (1988), 1319–7326.

[23]Sacco, G. M., and Yao, S. B. Query optimization in distributed database systems. *Advances in Computers 21* (1982), 225–273.

[24]Segev, A. Optimization of join operations in horizontally partitioned database systems. *ACM Transactions on Database Systems 11* (1986), 48–80.

[25]Tanenbaum, A. S. *Computer Networks*. Prentice Hall, 1988.

[26]Valduriez, G., and Valduriez, P. *Relational Databases and Knowledges Bases*. Addison-Wesley, 1989.

[27]Wang, C., and Chen, A. L. A parallel execution method for minimizing distributed query response time. In *IN Proc. 7th IEEE Data Engineering Conf.* (1991), p. Unknown.

[28]Wang, C., Li, V. O. K., and Chen, A. L. P. Distributed query optimization by one-shot fixed-precision semi-join execution. In *Proceeding IEEE Data Engineering Conference* (1991), pp. 1–8.

[29]Yoo, H., and Lafortune, S. An intelligent search method for query optimization by semijoins. *IEEE Transactions on Knowledge and Data Engineering 1* (1989), 226–237.

APPENDIX H
VITA AUCTORIS

George Mak was born in **Hong Kong**. He graduated from **Columbia Secondary School of Canada — Hamilton, Ontario** in 1985. From there he went on to the **University of Windsor** where he obtained a B.A.Sc. in Electrical Engineering in 1990. He also awarded the Windsor Entry Scholarship during his undergraduate studies. With his careful planning since his first year as an undergraduate, he was able to complete significant course work in Computer Science. From this achievement, he was awarded a tuition scholarship for his Masters studies in Computer Science. He is currently a candidate for the Master's degree in Science at the University of Windsor and hopes to graduate in the Winter of 1993. George intends to work in the area of Computer Network and Database Systems.